Modelling Financial Time Series with S-PLUS, Second Edition

Eric Zivot and Jiahui Wang

March 30, 2006

Preface

What Is the Book and Why Was It Written?

This book is a guide to analyzing and modeling financial time series using `S-PLUS` and `S+FinMetrics`. It is a unique blend of econometric theory, financial models, data analysis, and statistical programming. It serves as a user's guide for Insightful's `S+FinMetrics` module of statistical functions for financial time series analysis and financial econometrics as well as a general reference for models used in applied financial econometrics. The format of the chapters in the book is to give a reasonably complete description of a statistical model and how it works followed by illustrations of how to analyze the model using `S-PLUS` and the functions in `S+FinMetrics`. In this way, the book stands alone as an introduction to financial time series analysis as well as a user's guide for `S+FinMetrics`. It also highlights the general analysis of time series data using the new time series objects introduced in `S-PLUS` 6.

Intended Audience

This book is written for a wide audience of individuals who work, do research or study in the areas of empirical finance and financial econometrics. The field of financial econometrics has exploded over the last decade, and this book represents an integration of theory, methods and examples using the `S-PLUS` modeling language to facilitate the practice of financial econometrics. This audience includes researchers and practitioners in the finance industry, academic researchers in economics and finance, and advanced MBA and graduate students in economics and finance. Researchers and practitioners in the finance industry who already use `S-PLUS` and desire more functionality for analyzing and modeling financial data will find this text useful. It is also appropriate for financial analysts who may not be familiar with `S-PLUS` but who desire an integrated and open statistical modeling and programming environment for the analysis of financial data. This guide is useful for academic researchers interested in empirical finance and financial econometrics. Finally, this book may be used as a textbook or a textbook companion for advanced MBA and graduate level courses in time series analysis, empirical finance and financial econometrics.

Audience Background

It is assumed that the reader has a basic familiarity with `S-PLUS` at the level of Krause and Olson (2005) and a background in mathematical statistics at the level of Hogg and Craig (1994), is comfortable with linear algebra and linear regression, and has been exposed to basic time series concepts as presented in Harvey (1993) or Franses (1998). Most importantly, the book assumes that the reader is interested in modeling and analyzing financial time series.

Overview of the Book

The chapters in the book cover univariate and multivariate models for analyzing financial time series using `S-PLUS` and the functions in `S+FinMetrics`. Chapter one gives a general overview of the use of `S-PLUS` 6 and highlights certain aspects of the language for statistical modeling. Chapter two introduces the new time series objects in `S-PLUS` 6 and illustrates the specification, manipulation and visualization of these objects. Chapter three surveys time series concepts used throughout the book. Chapters four through eight cover a variety of topics in the modeling of univariate financial time series, including testing for unit roots, extreme value theory, time series regression models, GARCH models of volatility, and long memory models. Chapter nine introduces rolling analyses of time series models and covers related topics such as technical analysis of financial time series and moving average methods for high frequency data. Chapters ten through fifteen cover models for the analysis of multivariate financial time series. Topics include systems of regression equations, classical and Bayesian vector autoregressive models, cointegration, factor models, multivariate GARCH models, and state space models. Chapter 16 covers aspects of modeling time series arising from fixed income financial assets. Chapter 17, written by Victor Yohai and Jiahui Wang, describes robust REGARIMA models that allow for structural change. Chapters 18 through 23 are new to the Second Edition of the book. These new chapters cover nonlinear regime-switching models, copulas, continuous-time models, the generalized method of moments, semi-nonparametric conditional density models, and the efficient method of moments.

What Is `S+FinMetrics`?

`S+FinMetrics` is an `S-PLUS` module for the econometric modeling and prediction of economic and financial time series. With some 600 functions, version 1.0 of `S+FinMetrics` offers the following functionality:

- Easy-to-use Trellis plots for multivariate time series
- Time series manipulations such as missing value interpolation, disaggregation, differences, distributed lags and polynomial distributed lags
- Rolling sample statistics such as variance, maximum, and minimum
- Moving average operators for both regularly spaced and irregularly spaced time series
- Common technical analysis measures and indicators
- Statistical tests for normality, autocorrelation, heteroskedasticity, multicollinearity, GARCH effects, and long memory
- Extreme value theory models based on generalized extreme value and generalized Pareto distributions as well as copulas
- Autoregressive distributed lag regression models
- White and Newey-West corrections for heteroskedasticity and serial correlation
- Robust estimation of REG-ARIMA models and robust detection of level shifts, trend breaks, and outliers
- Rolling and recursive regression
- Generic rolling models for back-testing
- Long memory fractional ARIMA and SEMIFAR models
- Univariate GARCH models including long memory FIGARCH and FIEGARCH models
- Multivariate GARCH models
- Linear and nonlinear systems of regression equations
- Classical and Bayesian vector autoregression models

- Tests for unit roots and cointegration
- Vector error correction models
- State space models and efficient estimation, prediction, smoothing, and simulation using the Kalman filter
- Statistical multifactor models for large data sets based on asymptotic principal components
- Term structure interpolation

New features in version 2.0 of `S+FinMetrics` include:

- Variance ratio tests, efficient unit root tests and tests for nonlinearity
- Threshold AR, smooth transition AR and Markov switching AR models as well as Markov switching state space models
- Simulated solutions to systems of stochastic differential equations
- Generalized method of moments estimation
- Gallant and Tauchen's semi-nonparametric conditional density estimation and efficient method of moments estimation

`S+FinMetrics` incorporates functions from `S+GARCH`, the `EVIS` library of functions for modeling extreme values created by Alexander McNeil, the `EVANESCE` library of functions for modeling extreme values and bivariate copulas created by Rene Carmona and Julia Morrison, the *SsfPack* C library of state space modeling functions created by Siem Jan Koopman, and the SNP and EMM FORTRAN libraries created by Ronald Gallant and George Tauchen. `S+GARCH` was originally developed by Zhuanxin Ding, Hong-Ye Gao, Doug Martin, Jiahui Wang and Yihui Zhan. The `S+FinMetrics` function `arima.rob` was written by Ana Bianco, Marta Garcia Ben, Elena Martinez and Victor Yohai. The `S+FinMetrics` long memory modeling functions `FAR`, `FARIMA`, `SEMIFAR` and `fgarch` were developed by Jan Beran, Andrew Bruce, Don Percival, Alan Gibbs and Jiahui Wang and supported by NSF grant DMI-9801614 to Insightful Corporation (formerly MathSoft, Inc.). Much of the new functionality in version 2.0 of `S+FinMetrics` was supported by the NSF SBIR Phase II grant DMI-0132076 to Insightful Corporation. The `S-PLUS` implementation of Gallant and Tauchen's SNP and EMM FORTRAN libraries was accomplished by Jiahui Wang, Bob Thurman, Michael Sannella, Ying Gu and Eric Zivot, with the generous help and support of George Tauchen. Hu McCulloch kindly provided the term structure data included with `S+FinMetrics`, and James MacKinnon provided data sets for the response surface critical values for the Dickey-Fuller and Phillips-Ouliaris distributions.

Contact Information and Website

The authors are responsible for all of the material in the book except the material on robust change detection, which was written by Victor Yohai. Eric Zivot is primarily responsible for chapters 2-6, 9-12, 14-15, and 19-23, and Jiahui Wang is primarily responsible for chapters 1, 7-8, 13, 16 and 18. The authors may be contacted by electronic mail at

```
ezivot@u.washington.edu
jwang@svolatility.com
```

and welcome feedback and suggestions for improvements to the contents of the book. The website for the book is located on Eric Zivot's University of Washington web site at

```
http://faculty.washington.edu/ezivot/
       ModelingFinancialTimeSeries.htm
```

The website for version 2.0 of `S+FinMetrics` is located on the Insightful Corporation website at

```
http://www.insightful.com/support/finmetrics20
```

Acknowledgements

This book would not have been written without the support and encouragement from Insightful Corporation. The idea for the `S+FinMetrics` module was conceived by Douglas Martin and the authors. The development of `S+FinMetrics` was completed at Insightful by Jiahui Wang, Quan Wen and Hui Huang with help from many people. In particular, Jan Beran wrote many of the long memory functions while acting as a consultant to Insightful. Siem Jan Koopman helped to incorporate the *SsfPack* functions into `S-PLUS` and to write the chapter on state space models. Alexander McNeil and Rene Carmona graciously provided background material and `S-PLUS` examples for the material in the chapters on modeling extreme values and copulas. Bob Thurman helped to write the chapter on continuous-time models, and Ying Gu helped with the SNP and EMM examples. Ronald Gallant and George Tauchen graciously allowed the use of material from their unpublished 2001 survey paper "Efficient Method of Moments" for Chapters 22 and 23. A number of people were helpful in proofreading the book and testing the software. Particular thanks go to Eric Aldrich, Andrew Bruce, Chuck Curry, Zhuanxin Ding, Ruud Koning, Steve McKinney, Jun Ma, Scott Payseur, David Weitzel, Quan Wen and Bingcheng Yan.

Typographical Conventions

This book obeys the following typographic conventions:

- The *italic* font is used for emphasis, and also for user-supplied variables within `UNIX`, `DOS` and `S-PLUS` commands.
- The `typewriter font` is used for `S-PLUS` functions, the output of `S-PLUS` functions and examples of `S-PLUS` sessions.
- `S-PLUS` objects of a specified class are expressed in `typewriter font` enclosed in quotations " ". For example, the `S-PLUS` `timeSeries` function creates objects of class "`timeSeries`".

Displayed `S-PLUS` commands are shown with the prompt character `>`. For example

```
> summary(ols.fit)
```

`S-PLUS` commands that require more than one line of input are displayed with the continuation prompt indicated by `+` or `Continue string:`. The `S-PLUS` output and plots in this book were generated from a combination of `S+FinMetrics` Version 1.0 and `S-PLUS` Version 6.0 release 2 for Windows, and `S+FinMetrics` Versions 2.0 and `S-PLUS` Version 7.0 for Windows. The `S-PLUS` output and "`timeSeries`" objects were generated with the options settings

```
> options(width=60)
> options(time.zone="GMT")
```

In some cases, parts of long output from `S-PLUS` functions is omitted and these lines are indicated by

```
  ...
```

Some of the output has been hand edited to avoid line overflow.

Seattle, Washington, USA — Eric Zivot
Chicago, Illinois, USA — Jiahui Wang

References

Franses, P.H. (1998). *Time Series Models for Business and Economic Forecasting.* Cambridge University Press, Cambridge.

Harvey, A.C. (1993). *Time Series Models, Second Edition.* MIT Press, Cambridge.

Hogg, R.V. and A.T. Craig (1994). *Introduction to Mathematical Statistics, Fifth Edition.* Prentice Hall, New York.

Krause, A. and M. Olson (2002). *The Basics of S-PLUS, Fourth Edition.* Springer-Verlag, New York.

Contents

1
S and S-PLUS

1.1 Introduction

S-PLUS is a commercial software package developed by Insightful Corporation, based on the S language originally developed at Bell Laboratories (of AT&T and now Lucent Technologies) for statistical computation and visualization. Both S and S-PLUS have evolved through many versions. In 1999 John M. Chambers, the principal developer of S language, received the prestigious Software System Award from the Association for Computing Machinery (ACM), which has been awarded to UNIX, TEX, PostScript, TCP/IP and World Wide Web in the past.

The discussion of S language in this book is based on S-PLUS 6, which is supported on Microsoft Windows, Sun Solaris, and LINUX operating systems[1]. In addition to *S-PLUS 6 Programmer's Guide*, there are many excellent books available introducing different aspects of S and S-PLUS (see Section 1.4 for a list of them), and refer to these books if you are not familiar with S or S-PLUS. This chapter has a rather limited goal: to introduce the object oriented approach of S language and summarize some modeling conventions that will be followed in this book. Section 1.2 introduces the concept of objects in S language, and Section 1.3 summarizes the usage

[1]Some of the examples in the book have been updated to make use of new features in S-PLUS 7. All of the examples in Chapters 18 through 23 make use of S+FinMetrics 2.0, which is based on S-PLUS 7.

of modeling functions in S-PLUS and S+FinMetrics. Finally, Section 1.4 points out some useful resources for learning and using S-PLUS.

1.2 S Objects

1.2.1 Assignment

As the S language evolved over time, different assignment operators have been used, such as =, <-, <<-, and _ (underscore). This book will use the assignment operator = whenever possible, because it is more intuitive and requires only one key stroke. For example, in the command window of an S-PLUS session, use the following command to assign the value of 3 to a variable called a:

```
> a = 3
> a
[1] 3
```

When the name of the variable is typed at the command prompt, the value of the variable is printed on screen with an index [1]. Since _ is reserved as an assignment operator, it cannot be used in the names of any object. Avoid the use of _ as an assignment operator, because the code may look confusing to someone who is not familiar with S.

Although = has been chosen as the assignment operator whenever possible, only <- can be used as the assignment operator if the assignment is inside a function call.[2] For example, suppose the user wants to assign the value of 10 to the variable a, and use a to initialize a 5×5 matrix. If = is used as the assignment operator, an error message appears:

```
> matrix(a = 10, 5, 5)
Problem in matrix: argument a= not matched: matrix(a = 10, 5, 5)
Use traceback() to see the call stack
```

But if the assignment operator <- is used, the desired behavior is achieved:

```
> matrix(a <- 10, 5, 5)
     [,1] [,2] [,3] [,4] [,5]
[1,]   10   10   10   10   10
[2,]   10   10   10   10   10
[3,]   10   10   10   10   10
[4,]   10   10   10   10   10
[5,]   10   10   10   10   10
> a
```

[2]The reason is that S-PLUS functions allow optional arguments with default values, and = is used to set the default values in a function call.

```
[1] 10
```

and 10 is successfully assigned as the value of a.

1.2.2 Class

Since the S language is object oriented, everything in S-PLUS is an object with a *class*, and the class function can be used to find out the class of an object. For example:

```
> class(a)
[1] "integer"
```

thus the variable a has class "integer". Explicitly using the decimal point forces an integer number to be stored in double precision:

```
> b = 100000.
> class(b)
[1] "numeric"
```

A number with double precision in S-PLUS has class "numeric". In most situations S-PLUS is "smart" enough to perform computations in double precision if necessary. However, one has to be a little careful with integer arithmetic. For example, the following operation returns an NA:

```
> 100000 * 100000
[1] NA
```

because in this case, the multiplication is performed in integer mode, and the largest integer on a 32-bit machine is:

```
> 2^31 - 1
[1] 2147483647
```

which can be verified by querying the integer.max component of the machine constant object in S-PLUS:[3]

```
> .Machine$integer.max
[1] 2147483647
```

However, since the variable b created earlier is stored in double precision, the multiplication using b would return the desired result:

```
> b * b
[1] 1e+10
```

Together with "logical" and "character", "integer" and "numeric" objects are known as the *atomic* objects, upon which the user can build more complicated data structure, such as matrix, list, data frame, function,

[3]See the on-line help file for .Machine for other components in the list.

etc. For example, use the concatenation function `c` to combine the variables `a` and `b` into a vector, and use the `matrix` function to reshape it into a 2×1 matrix:

```
> abMat = matrix(c(a,b), nrow=2)
> class(abMat)
[1] "matrix"
> abMat
      [,1]
[1,] 1e+01
[2,] 1e+05
```

As another example, although `matrix` is a built-in function in S-PLUS, it is just another object in S-PLUS:

```
> class(matrix)
[1] "function"
> matrix
function(data = NA, nrow = 1, ncol = 1, byrow = F, dimnames)
{
  if(missing(nrow))
    nrow <- ceiling(length(data)/ncol)
  else if(missing(ncol))
    ncol <- ceiling(length(data)/nrow)
  dim <- c(nrow, ncol)
  if(length(dim) != 2)
    stop("nrow and ncol should each be of length 1")
  value <- if(byrow) t(array(data, dim[2:1])) else
           array(data, dim)
  if(!missing(dimnames))
    value@.Dimnames <- dimnames
  value
}
```

The preceding output shows that `matrix` is just a "`function`" object. When the name of this object is typed, S-PLUS prints its function definition on the screen.

Most complicated S-PLUS objects are constructed as a list. For example, combine the variables `a` and `b` into a list as follows:

```
> abList = list(aComponent=a, bComponent=b)
> class(abList)
[1] "list"
> abList
$aComponent:
[1] 10
```

```
$bComponent:
[1] 1e+05
```

where the names `aComponent` and `bComponent` are given to `a` and `b`, respectively. Use the `length` function to find out the number of components in a list and the `names` function to extract the names of those components:

```
> length(abList)
[1] 2
> names(abList)
[1] "aComponent" "bComponent"
```

A particular component of a list can be extracted using the `$` operator. For example:

```
> abList$aComponent
[1] 10
```

or the `[[` operator:

```
> abList[[2]]
[1] 1e+05
```

`S-PLUS` 6 is based on `S` language Version 4 (SV4). In SV4, a new class structure is introduced to build more complicated objects, as an alternative to using lists. One example is the "`timeDate`" objects in `S-PLUS`. For example, in the following example, use the `timeDate` function to parse a vector of character strings representing some dates:

```
> timeStamp = timeDate(c("1/1/2001", "1/2/2001", "1/3/2001"))
> timeStamp
[1] 01/01/2001 01/02/2001 01/03/2001
> class(timeStamp)
[1] "timeDate"
```

The `names` function cannot be used with these new class objects, which will be referred to as SV4 objects. Instead, use the `slotNames` function to extract the names of their components. For example:

```
> slotNames(timeStamp)
[1] ".Data"         ".Data.names"   ".Data.classes" "format"
[5] "time.zone"
```

A "`timeDate`" object has five slots. Instead of using the `$` operator as for lists, use the `@` operator to extract the component in a particular slot. For example:

```
> timeStamp@.Data
[[1]]:
[1] 14976 14977 14978
```

```
[[2]]:
[1] 0 0 0
```

The .Data slot of a "timeDate" object actually stores a list with two components.[4]

One difference between the list based objects and SV4 objects is that the list based objects are more flexible and thus prone to cause accidental programming errors if a programmer is not careful enough. In contrast, the SV4 objects are more stringently defined and can lead to robust software and computational efficiency. For example, the user can add or delete a component to a list at will:

```
> abList$anotherComponent = "a string component"
> abList
$aComponent:
[1] 10

$bComponent:
[1] 1e+05

$anotherComponent:
[1] "a string component"

> abList$aComponent = NULL
> abList
$bComponent:
[1] 1e+05

$anotherComponent:
[1] "a string component"
```

However, an SV4 object is strictly defined, and a component cannot be edited unless it is defined in its declaration:

```
> timeStamp@time.zone
[1] "GMT"
> timeStamp@time.zone = "Pacific"
> timeStamp@anotherSlot = "no way"
Problem in timeStamp@anotherSlot = "no way": Class "timeDate"
has no "anotherSlot" slot
Use traceback() to see the call stack
```

[4]The first component represents the Julian dates, and the second component represents the milliseconds elapsed since midnight of each day.

1.2.3 Method

Many S-PLUS functions are defined as *generic* in the sense that the user has the freedom of defining his or her own method for a particular class. For example, the print and summary functions in S-PLUS are so generic that they work with any object and may generate different types of results depending on the class of the object.[5] For example:

```
> summary(abMat)
   Min. 1st Qu. Median   Mean 3rd Qu.   Max.
     10   25008  50005  50005  75002 100000
> summary(abList)
                 Length Class      Mode
      bComponent 1            numeric
anotherComponent 1            character
```

For a numeric matrix object, the summary method generates some sample statistics, while for a list object, the summary method summarizes the length and mode of each component.

In the above example, S-PLUS is "smart" enough to figure out the appropriate method to use for the generic summary function. If the name of the method function is known, the user can also call the method function directly. For example, if the user types matrix at the command prompt, S-PLUS will dispatch the print method for "function" objects because matrix is a "function" object. However, it can also call the function print.list on a "function" object to view the object using another format:

```
> print.list(matrix)
$data:
NA

$nrow:
[1] 1

$ncol:
[1] 1

$byrow:
F

$dimnames:
```

[5]In fact, typing the name of an object at the command prompt, S-PLUS calls the print method of that object automatically. So any print methods rarely need to be called explicitly, except for Trellis graphics objects.

```
$"":
{
  if(missing(nrow))
    nrow <- ceiling(length(data)/ncol)
  else if(missing(ncol))
    ncol <- ceiling(length(data)/nrow)
  dim <- c(nrow, ncol)
  if(length(dim) != 2)
    stop("nrow and ncol should each be of length 1")
  value <- if(byrow) t(array(data, dim[2:1])) else
           array(data, dim)
  if(!missing(dimnames))
    value@.Dimnames <- dimnames
  value
}
```

1.3 Modeling Functions in S+FinMetrics

In this book, many statistical and econometric examples are illustrated using modeling functions in S+FinMetrics. Some modeling functions in S+FinMetrics are named using upper case acronyms as they are known in the literature, because S is case sensitive and it distinguishes between upper case and lower case letters.

1.3.1 Formula Specification

For many modeling functions in S+FinMetrics, S formulas are used to specify the model to be estimated. Chambers and Hastie (1993) and *S-PLUS Guide to Statistics* provide detailed examples of how to specify models using formulas in S. This section points out some restrictions in formula specification so that the user can avoid some errors in using these functions. For illustrations, use the S-PLUS lm function as an example of modeling function.

If a formula is used to specify models in a modeling function, usually at least two arguments are supplied to the function: a formula object and a data object. The args function can always be used to find out the argument names of any function:

```
> args(lm)
function(formula, data, weights, subset, na.action, method =
        "qr", model = F, x = F, y = F, contrasts = NULL, ...)
NULL
```

The data object must be a "data.frame" object, or a "timeSeries" object with a "data.frame" in its data slot. First create a data frame using the S-PLUS data objects stack.x and stack.loss:

```
> stack.df = data.frame(Loss=stack.loss, stack.x)
> colIds(stack.df)
[1] "Loss"       "Air.Flow"   "Water.Temp" "Acid.Conc."
```

so the data frame stack.df has four columns with variable names as shown above.

To regress the variable Loss on Air.Flow, and Water.Temp using least squares, use the lm function as follows:

```
> test.mod = lm(Loss~Air.Flow + Water.Temp, data=stack.df)
> test.mod
Call:
lm(formula = Loss ~ Air.Flow + Water.Temp, data = stack.df)

Coefficients:
 (Intercept)  Air.Flow Water.Temp
   -50.35884 0.6711544   1.295351

Degrees of freedom: 21 total; 18 residual
Residual standard error: 3.238615
```

Notice that in the first formula object, Loss is on the left hand side of ~, so it represents the endogenous or response variable of the model; Air.Flow and Water.Temp are on the right hand side of ~, so they represent two independent or explanatory variables. An intercept or a constant term is also included automatically, as can be seen from the coefficient estimates in the output, which is generated by a call to the print method for "lm" objects:

```
> class(test.mod)
[1] "lm"
> oldClass(test.mod)
[1] "lm"
```

Note that since an "lm" object is a list based object, the user can also use the oldClass function to obtain its class. However, oldClass function does not work with SV4 objects. For example:

```
> oldClass(timeStamp)
NULL
```

The data argument can also be a "timeSeries" object with a data frame in its data slot. To illustrate this possibility, turn stack.df into a "timeSeries" object and call it stack.ts:

```
> stack.ts = timeSeries(stack.df)
> stack.ts
  Positions Loss Air.Flow Water.Temp Acid.Conc.
 01/01/1960 42   80       27         89
 01/02/1960 37   80       27         88
 01/03/1960 37   75       25         90
 01/04/1960 28   62       24         87
 01/05/1960 18   62       22         87
 ...
```

Again, a linear model can be estimated using this data object just like in the previous example:

```
> test.mod = lm(Loss~Air.Flow + Water.Temp, data=stack.ts)
```

However, the `data` argument must have a data frame representation. The same function call will generate an error if the `data` argument is represented by a matrix:

```
> stack.mat = as.matrix(stack.df)
> lm(Loss~Air.Flow+Water.Temp, data=stack.mat)
Warning messages:
  Numerical expression has 84 elements: only the first used in:
  model.frame(formula, data, na.action, dots)
Problem: Invalid frame number, 42
Use traceback() to see the call stack
```

For most modeling functions such as `lm`, the `data` argument is actually an optional argument, which is not required. If the `data` argument is not supplied by the user, then the variables specified in the `formula` object must be on the search path. For example:

```
> lm(stack.loss~stack.x)
Call:
lm(formula = stack.loss ~ stack.x)

Coefficients:
 (Intercept) stack.xAir Flow stack.xWater Temp stack.xAcid Conc.
   -39.91967       0.7156402          1.295286        -0.1521225

Degrees of freedom: 21 total; 17 residual
Residual standard error: 3.243364
```

In addition, if the data argument is not supplied, the variables specified in the formula object must be either a vector or a matrix, and they cannot be a data frame nor a "timeSeries" object. For example:[6]

```
> stack.x.df = as.data.frame(stack.x)
> lm(stack.loss~stack.x.df)
Problem: Length of stack.x.df (variable 2) is 3 != length of
others (21)
Use traceback() to see the call stack

> stack.loss.ts = timeSeries(stack.loss)
> lm(stack.loss.ts~stack.x)
Problem: Length of stack.loss.ts (variable 1) is 11 != length
of others (21)
Use traceback() to see the call stack
```

In S+FinMetrics, the formula is extended to support autoregressive specification, moving average specification, distributed lags and polynomial distributed lags for many modeling functions. These formulas will be illustrated in the appropriate chapters.

1.3.2 Method

In addition to print and summary functions, many other functions in S-PLUS are defined to be generic to work with modeling functions and objects, such as plot for diagnostic plots, coefficients or simply coef for extracting model coefficients, residuals for extracting model residuals, fitted.values or simply fitted for extracting model fitted values, predict for out of sample prediction, etc. For example, for the "lm" object test.mod, if the generic functions coef, predict or plot are applied, S-PLUS will figure out the appropriate method to use:

```
> coef(test.mod)
 (Intercept)  Air.Flow Water.Temp
   -50.35884 0.6711544   1.295351

> predict(test.mod, matrix(1, 5, 3))
[1] -48.39233 -48.39233 -48.39233 -48.39233 -48.39233

> plot(test.mod, ask=T)

Make a plot selection (or 0 to exit):
```

[6]In fact, many modeling functions in S+FinMetrics actually does allow a "timeSeries" object on the left hand side of the formula, but not the right hand side of the formula, if the data argument is not supplied. One example is the garch function.

```
1: plot: All
2: plot: Residuals vs Fitted Values
3: plot: Sqrt of abs(Residuals) vs Fitted Values
4: plot: Response vs Fitted Values
5: plot: Normal QQplot of Residuals
6: plot: r-f spread plot
7: plot: Cook's Distances
Selection:
```

In addition to the above generic functions, `S+FinMetrics` defines three new generic functions for working with model objects: `vcov` for extracting the variance-covariance matrix of estimated parameters, `simulate` for generating simulations according to the estimated model, and `cpredict` for obtaining conditional forecasts of multivariate time series models.

1.4 S-PLUS Resources

1.4.1 Books

In addition to the S-PLUS manuals, there are a number of good books on using and programming in S and S-PLUS as well as data and statistical analysis using S-PLUS. The Insightful web page

`http://www.insightful.com/support/splusbooks.asp`

contains a listing of these books.

Using and Programming S-PLUS

Gentle introductions to S and S-PLUS are given in Spector (1994), Lam (2001) and Krause and Olson (2005). The details of version four of the S language are described in Chambers (1998), also known as the "green book". An indispensable guide to programming in the S language is Venables and Ripley (2000).

Data and Statistical Analysis in S-PLUS

S-PLUS provides extensive functionality for the statistical analysis of a wide variety of data, and many books have been written on the use of S-PLUS for particular applications. The following books describe statistical techniques that are useful for the analysis of financial data. Carmona (2004) and Chan (2002) describe the use of S-PLUS for the analysis of financial time series. Scherer and Martin (2005) cover portfolio optimization and related topic. An excellent guide to modern applied statistics using S-PLUS is Venables and Ripley (2002). Harrell (2001) gives a thorough treatment of regression models, including generalized linear models and survival model. Heiberger and Holland (2004) emphasize the importance of graphical techniques in statistical analysis. Pinheiro and Bates (2000) detail the analysis of mixed effects (panel data) models. Therneau and Grambsch (2000) survey survival analysis models. Wilcox (1997), and Atkinson and Riani (2000) discuss robust statistical methods. Bruce and Gao (1996) describe wavelet analysis. Hastie, Tibshirani and Friedman (2001) cover aspects of statistical learning and data mining. Davison and Hinkley (1997) survey bootstrap methods, and Bowman and Azzalini (1997) disucss nonparametric and smoothing methods.

1.4.2 Internet

There is a wealth of information about `S-PLUS` available on the internet. The obvious place to start is the Insightful website at

`http://www.insightful.com`

S-News is an electronic mail discussion list for `S` and `S-PLUS` users. Information about S-News may be found at

`http://www.biostat.wustl.edu/s-news/s-news-intro.html`

StatLib is a repository for software and extensions for the S language, including many useful additions to `S-PLUS`. It can be found at

`http://lib.stat.cmu.edu/S`

Eric Zivot maintains a website containing links to websites for `S-PLUS` applications in econometrics and finance at

`http://faculty.washington.edu/ezivot/splus.htm`

1.5 References

Atkinson, A. and M. Riani (2000). *Robust Diagnostic Regression Analysis.* Springer-Verlag, New York.

Bowman, A.W., and A. Azzalini (1997). *Applied Smoothing Techniques for Data Analysis:The Kernel Approach with S-PLUS Illustrations.* Oxford University Press, Oxford.

Bruce, A. and H.-Y. Gao (1996). *Applied Wavelet Analysis with S-PLUS.* Springer-Verlag, New York.

Carmona, R. (2004). *Statistical Analysis of Financial Data in S-PLUS.* Springer-Verlag, New York.

Chambers, J.M. (1998). *Programming with Data.* Springer-Verlag, New York.

Chambers, J. M., and Hastie, T. J. (1993). *Statistical Models in S.* Chapman & Hall.

Chan, N.H. (2002). *Time Series: Applications to Finance.* John Wiley & Sons, New York.

Davidson, A.C. and D.V. Hinkley (1997). *Bootstrap Methods and Their Application.* Cambridge University Press, Cambridge, UK.

Harrell, F.E. (2001). *Regression Modeling Strategies with Applications to Linear Models, Logistic Regression, and Survival Analysis.* Springer-Verlag, New York.

Hastie, T., R. Tibshirani and J. Friedman (2001). *The Elements of Statistical Learning: Data Mining, Inference and Prediction.* Springer-Verlag, New York.

Heiberger, R.M. and B. Holland (2004). *Statistical Analysis and Data Display.* Springer-Verlag, New York.

Krause, A. and M. Olson (2005). *The Basics of S-PLUS, Fourth Edition.* Springer-Verlag, New York.

Lam, L. (2001). *An Introduction to S-PLUS for Windows.* Candiensten, Amsterdam.

Pinheiro, J.C. and D.M. Bates (2000). *Mixed-Effects Models in S and S-PLUS.* Springer-Verlag, New York.

Scherer, B. and R.D. Martin (2005). *Modern Portfolio Optimization with NuOPT, S-PLUS and S+Bayes.* Springer-Verlag, New York.

Spector, P. (1994). *An Introduction to S and S-PLUS.* Duxbury Press, Belmont, CA.

Therneau, T.M. and P.M Grambsch (2000). *Modeling Survival Data.* Springer-Verlag, New York.

Venables, W.N. and B.D. Ripley (2002). *Modern Applied Statistics with S-PLUS, Fourth Edition.* Springer-Verlag, New York.

Venables, W.N. and B.D. Ripley (1999). *S Programming.* Springer-Verlag, New York.

Wilcox, P. (1997). *Introduction to Robust Estimation and Hypothesis Testing.* Academic Press, San Diego.

2 Time Series Specification, Manipulation, and Visualization in S-PLUS

2.1 Introduction

Time series data may be stored, manipulated and visualized in a variety of ways in S-PLUS[1]. This chapter discusses the basics of working with financial time series data in the form of S-PLUS "timeSeries" objects. It begins with a discussion of the specification of "timeSeries" and "timeDate" objects in S-PLUS and gives examples of how to specify common "timeDate" sequences for financial time series. Basic manipulations of financial time series are discussed and illustrated. These manipulations include aggregating and disaggregating time series, handling of missing values, creations of lags and differences and asset return calculations. The chapter ends with an overview of time series visualization tools and techniques, including the S-PLUS plotting functions for "timeSeries" as well as specialized plotting functions in S+FinMetrics.

2.2 The Specification of "timeSeries" Objects in S-PLUS

Financial time series data may be represented and analyzed in S-PLUS in a variety of ways. By far the most flexible way to analyze, manipulate

[1]Chapters 25-27 in the *S-PLUS Guide to Statistic (Vol. II)* discusses the analysis of time series in S-PLUS.

and visualize time series data is through the use of S-PLUS calendar-based "timeSeries" objects. A calendar-based "timeSeries" object, hereafter referred to as simply a "timeSeries" is an S version 4 (sv4) object that stores time and date information from a "timeDate" object in a positions slot and time series data from any rectangular data object (vector, matrix or data frame) in a data slot. Additionally, summary information about the time series may be stored in the title, documentation, units and attributes slots.

To illustrate a typical "timeSeries" object, consider the S+FinMetrics "timeSeries" object singleIndex.dat which contains monthly closing price data on Microsoft and the S&P 500 index over the period January 1990 through January 2001:

```
> class(singleIndex.dat)
[1] "timeSeries"

> slotNames(singleIndex.dat)
 [1] "data"              "positions"         "start.position"
 [4] "end.position"      "future.positions"  "units"
 [7] "title"             "documentation"     "attributes"
[10] "fiscal.year.start" "type"

> singleIndex.dat@title
[1] "Monthly prices on Microsoft and S&P 500 Index"

> singleIndex.dat@documentation
[1] "Monthly closing prices over the period January 1900"
[2] "through January 2001 adjusted for dividends and stock"
[3] "splits.

> singleIndex.dat@units
[1] "Monthly price"

> singleIndex.dat[1:5,]
 Positions   MSFT  SP500
 Jan 1990  1.2847 329.08
 Feb 1990  1.3715 331.89
 Mar 1990  1.5382 339.94
 Apr 1990  1.6111 330.80
 May 1990  2.0278 361.23
```

The date information in the positions slot may be extracted directly or by using the positions extractor function:

```
> singleIndex.dat@positions[1:5]
[1] Jan 1990 Feb 1990 Mar 1990 Apr 1990 May 1990
```

```
> positions(singleIndex.dat)[1:5]
[1] Jan 1990 Feb 1990 Mar 1990 Apr 1990 May 1990
```

The generic `start` and `end` functions may be used to extract the start and end dates of a "`timeSeries`" object:

```
> start(singleIndex.dat)
[1] Jan 1990
> end(singleIndex.dat)
[1] Jan 2001
```

The date information in the `positions` slot is an object of class "`timeDate`"

```
> class(positions(singleIndex.dat))
[1] "timeDate"
```

Details on "`timeDate`" objects are given later on in this chapter.

The time series data in the `data` slot may be accessed directly or through the `seriesData` extractor function:

```
> singleIndex.dat@data[1:5,]
    MSFT  SP500
1 1.2847 329.08
2 1.3715 331.89
3 1.5382 339.94
4 1.6111 330.80
5 2.0278 361.23

> seriesData(singleIndex.dat)[1:5,]
    MSFT  SP500
1 1.2847 329.08
2 1.3715 331.89
3 1.5382 339.94
4 1.6111 330.80
5 2.0278 361.23
```

In general, the time series data in the `data` slot is a "`rectangular`" data object and is usually a data frame or a matrix. For example,

```
> class(seriesData(singleIndex.dat))
[1] "data.frame"
```

In fact, "`timeSeries`" objects themselves are "`rectangular`" data objects and so the functions `numRows`, `numCols`, `colIds` and `rowIds` may be used to extract useful information:

```
> is.rectangular(singleIndex.dat)
[1] T
> numRows(singleIndex.dat)
[1] 133
```

```
> numCols(singleIndex.dat)
[1] 2
> colIds(singleIndex.dat)
[1] "MSFT"  "SP500"
> rowIds(singleIndex.dat)[1:5]
[1] Jan 1990 Feb 1990 Mar 1990 Apr 1990 May 1990
```

2.2.1 Basic Manipulations

Basic manipulation of "`timeSeries`" objects may be done in the same way as other `S-PLUS` objects. Mathematical operations may be applied to "`timeSeries`" objects in the usual way and the result will be a "`timeSeries`" object. Subscripting a "`timeSeries`" works in the same way as subscripting a data frame or matrix. For example, a "`timeSeries`" with the prices on Microsoft may be extracted from `singleIndex.dat` using

```
> msft.p = singleIndex.dat[,"MSFT"]
> msft.p = singleIndex.dat[,1]
> msft.p@title = "Monthly closing price on Microsoft"
> msft.p@documentation =
+ c("Monthly closing price adjusted for stock",
+ "splits and dividends.")
> msft.p@units = "US dollar price"
> class(msft.p)
[1] "timeSeries"
```

Subsamples from a "`timeSeries`" may be extracted by creating an index of logical values that are true for the times and dates of interest. For example, consider creating a subsample from the "`timeSeries`" `singleIndex.dat` over the period March 1992 through January 1993.

```
> smpl = (positions(singleIndex.dat) >= timeDate("3/1/1992") &
+ positions(singleIndex.dat) <= timeDate("1/31/1993"))
> singleIndex.dat[smpl,]
 Positions  MSFT SP500
 Mar 1992  4.938 403.7
 Apr 1992  4.594 414.9
 May 1992  5.042 415.4
 Jun 1992  4.375 408.1
 Jul 1992  4.547 424.2
 Aug 1992  4.656 414.0
 Sep 1992  5.031 417.8
 Oct 1992  5.547 418.7
 Nov 1992  5.820 431.4
 Dec 1992  5.336 435.7
 Jan 1993  5.406 438.8
```

S-PLUS 7 supports subscripting a "timeSeries" object directly with dates. For example, the subsample from singleIndex.dat over the period March 1992 through January 1993 may be produced using

```
> singleIndex.dat[timeEvent("3/1/1992","1/31/1993"),]
```

Most S-PLUS functions have methods to handle "timeSeries" objects. Some common examples are the S-PLUS functions colMeans, colVars and colStdevs which compute the mean, variance and standard deviation value for each column of data:

```
> colMeans(singleIndex.dat)
     MSFT    SP500
 26.74513 730.3805
```

For functions that do not have methods to handle "timeSeries" objects, the extractor function seriesData should be used to extract the data slot of the "timeSeries" prior to applying the function:

```
> colMeans(seriesData(singleIndex.dat))
     MSFT    SP500
 26.74513 730.3805
```

All of the S+FinMetrics modeling and support functions are designed to accept "timeSeries" objects in a uniform way.

2.2.2 S-PLUS "timeDate" Objects

Time and date information in S-PLUS may be stored in "timeDate" objects. The S-PLUS function timeDate is used to create "timeDate" objects. For example, to create a "timeDate" object for the date January 1, 2002 for the US Pacific time zone use

```
> td = timeDate("1/1/2002",in.format="%m/%d/%Y",
+ zone="Pacific")
```

The date information is specified in a character string and the optional arguments in.format and zone determine the input date format and the time zone, respectively. The input formats are single-element character vectors consisting of input fields which start with "%" and end with a letter. The default input date format may be viewed with

```
> options("time.in.format")
$time.in.format:
[1] "%m[/][.]%d[/][,]%y [%H[:%M[:%S[.%N]]][%p][[(]%3Z[)]]]"
```

and examples of common date formats can be found in the S-PLUS object format.timeDate

```
> names(format.timeDate)
```

```
 [1] "1/3/1998"
 [2] "3/1/1998"
...
[32] "03 Jan 1998 14:04:32 (PST)"
> format.timeDate[[1]]$input
[1] "%m/%d/%Y"
```

The result of `timeDate` is an object of class "`timeDate`"

```
> class(td)
[1] "timeDate"
> td
[1] 1/1/02 0:00:00 AM
> slotNames(td)
[1] ".Data"         ".Data.names"   ".Data.classes"
[4] "format"        "time.zone"
```

"`timeDate`" objects have a number of slots that are used to specify and control time and date information. Full details may be seen using

```
> ?class.timeDate
```

The `.Data` slot is a list with components giving the Julian date representation of the day and time within the day. The Julian day represents the number of days since January 1, 1960 and the Julian time within the day indicates the number of milliseconds since midnight Greenwich mean time (GMT)

```
> td@.Data
[[1]]:
[1] 15341

[[2]]:
[1] 28800000
```

Since the US Pacific Time Zone is 8 hours behind GMT, the number of milliseconds since Greenwich mean time is $8 * 60 * 60 * 1000 = 28,800,000$. The output display format of the date information is specified in the `format` slot

```
> td@format
[1] "%m/%d/%02y %H:%02M:%02S %p"
```

Like input formats, output formats are single-element character vectors consisting of output fields, which start with "%" and end with a letter, and other characters that are simply printed. The above format specifies printing the date as month/day/year and then hour:minute:second and AM or PM. The integers 02 before `y`, `M` and `S` fix the output width to 2 characters. All supported output fields are described in the help file for class.timeDate and a list of example output formats are given in the S-PLUS object format.timeDate. For example,

```
> names(format.timeDate)[18]
[1] "03 Jan 1998"
> format.timeDate[[18]]$output
[1] "%02d %b %Y"
```

Time Zone Issues

The time and date information stored in a "timeDate" object is aligned to the time zone specified in the time.zone slot

```
> td@time.zone
[1] "Pacific"
```

To modify the output format of a "timeDate" object to display time zone information simply add "%z"

```
> td@format = paste(td@format,"%z")
> td
[1] 1/1/02 0:00:00 AM Pacific
```

The object td is aligned to the US Pacific time zone. If the zone argument to timeDate is omitted when the "timeDate" object is created the default time zone in options(''time.zone") is used[2]. For example,

```
> options("time.zone")
$time.zone:
[1] "Pacific"
> td2 = timeDate("Mar 02, 1963 08:00 PM",
+ in.format="%m %d, %Y %H:%M %p",
+ format="%b %02d, %Y %02I:%02M %p %z")
> td2
[1] Mar 02, 1963 08:00 PM Pacific
```

Note that the above example shows that the output format of the "timeDate" object can be specified when the object is created using the argument format.

All of the time zone specifications supported by S-PLUS are described in the help file for class.timeZone and these specifications are defined relative to times and dates given in GMT. The time zone specifications include daylight savings time in various areas around the world. To see how a time zone specification affects a timeDate object, consider what

[2]On Windows platforms, the time zone specification is obtained from the Windows regional settings. The examples in this section were created on a Windows computer in the U.S. Pacific time zone. Therefore, the default time zone taken from the Windows regional settings is "Pacific".

happens when the time zone for the object `td` is changed to US Eastern Time:

```
> td@time.zone = "Eastern"
> td
[1] 1/1/02 3:00:00 AM Eastern
> td@.Data
[[1]]:
[1] 15341

[[2]]:
[1] 28800000
```

Since US Eastern Time is three hours ahead of US Pacific Time the displayed date is moved ahead three hours. That is, midnight US Pacific Time on January 1, 2002 is the same as 3 AM US Eastern Time on January 1, 2002. Notice that changing the time zone information does not alter the Julian date information in the `.Data` slot. To align the Julian date representation to reflect the number of milliseconds from GMT on US Eastern time the millisecond information in the second component of the `.Data` slot must be adjusted directly.

If a "`timeDate`" object is created in GMT then the S-PLUS function `timeZoneConvert` may be used to re-align the millisecond offset to a specified time zone. For example,

```
> tdGMT = timeDate("1/1/2002",zone="GMT",
+ format="%m/%d/%02y %H:%02M:%02S %p %z")
> tdGMT
[1] 1/1/02 0:00:00 AM GMT
> tdGMT@.Data
[[1]]:
[1] 15341

[[2]]:
[1] 0

> tdPST = timeZoneConvert(tdGMT,"PST")
> tdPST
[1] 1/1/02 0:00:00 AM PST
> tdPST@.Data
[[1]]:
[1] 15341

[[2]]:
[1] 28800000
```

Be aware that timeZoneConvert is not designed to convert the millisecond offsets from one arbitrary time zone other than GMT to another arbitrary time zone.

Mathematical Operations with “timeDate” Objects

Since “timeDate” objects have a Julian date representation, certain mathematical operations like addition and subtractions of numbers may be performed on them and the result will also be a “timeDate” object. For example,

```
> td1 = timeDate("1/1/2002",in.format="%m/%d/%Y",
+ zone="GMT",format="%m/%d/%04Y %H:%02M:%02S %p %z")
> td2 = timeDate("2/1/2002",in.format="%m/%d/%Y",
+ zone="GMT",format="%m/%d/%04Y %H:%02M:%02S %p %z")
> td1
[1] 1/1/2002 0:00:00 AM GMT
> td2
[1] 2/1/2002 0:00:00 AM GMT

> as.numeric(td1)
[1] 15341
> td1 + 1
[1] 1/2/2002 0:00:00 AM GMT
> td1 + 0.5
[1] 1/1/2002 12:00:00 PM GMT
> td1 - 1
[1] 12/31/2001 0:00:00 AM GMT
> 2*td1
[1] 30682
> td1+td2
[1] 2/2/2044 0:00:00 AM GMT
```

Adding two “timeDate” objects together creates another “timeDate” object with date given by the addition of the respective Julian dates. Subtraction of two “timeDate” objects, however, produces an sv4 object of class “timeSpan”

```
> td.diff = td2 - td1
> class(td.diff)
[1] "timeSpan"
> td.diff
[1] 31d 0h 0m 0s 0MS
> slotNames(td.diff)
[1] ".Data"          ".Data.names"    ".Data.classes"
[4] "format"
```

The "timeSpan" object td.diff gives the time difference between td1 and td2 - 31 days, 0 hours, 0 minutes, 0 seconds and 0 milliseconds. The Julian date information is kept in the .Data slot and the output format is in the format slot. Details about "timeSpan" objects is given in The *S-PLUS Guide to Statistics*, *Vol. II*, chapter 25.

2.2.3 Creating Common "timeDate" Sequences

Most historical financial time series are regularly spaced calendar-based time series; e.g. daily, monthly or annual time series. However, some financial time series are irregularly spaced. Two common examples of irregularly spaced financial time series are daily closing prices and intra-day transactions level data. There are a variety of time and date functions in S-PLUS that may be used to create regularly spaced and irregularly spaced "timeDate" sequences for essentially any kind of financial data. These functions are illustrated using the following examples[3].

Regularly and irregularly spaced sequences may be created using the S-PLUS functions timeCalendar, timeSeq and timeSequence. The function timeSeq is the most flexible. The following examples illustrate the use of these functions for creating common "timeDate" sequences.

Annual Sequences

Creating a "timeDate" sequence for an annual time series from 1900 to 1910 may be done in a variety of ways. Perhaps, the simplest way uses the S-PLUS timeCalendar function:

```
> td = timeCalendar(y=1900:1910,format="%Y")
> class(td)
[1] "timeDate"
> td
 [1] 1900 1901 1902 1903 1904 1905 1906 1907 1908 1909 1910
```

The timeCalendar function produces an object of class "timeDate". The argument format="%Y" specifies the output format of the "timeDate" object as a four digit year.

Since td contains a sequence of dates, the Julian date information for all of the dates is available in the .Data slot

```
> td@.Data
[[1]]:
 [1] -21914 -21549 -21184 -20819 -20454 -20088 -19723 -19358
 [9] -18993 -18627 -18262
```

[3] To avoid problems with time zone specifications, all examples in this sections were created after setting the default time zone to GMT using options(time.zone="GMT").

```
[[2]]:
 [1] 0 0 0 0 0 0 0 0 0 0 0
```

An annual sequence from 1900 to 1910 may also be computed using the S-PLUS function timeSeq:

```
> timeSeq(from="1/1/1900", to="1/1/1910", by="years",
+ format="%Y")
 [1] 1900 1901 1902 1903 1904 1905 1906 1907 1908 1909 1910
```

The argument by="years" specifies annual spacing between successive values in the sequence starting at 1/1/1900 and ending at 1/1/1910. The date formats for the starting and ending dates must conform to the default input format for "timeDate" objects (see options("time.in.format")).

Finally, an annual sequence from 1900 to 1910 may be created using the S-PLUS function timeSequence:

```
> tds = timeSequence("1/1/1900","1/1/1910",by="years",
+ format="%Y")
> class(tds)
[1] "timeSequence"
> tds
from:   1900
to:     1910
by:     +1yr
[1] 1900 1901 1902 ...  1910
```

timeSequence creates an object of class "timeSequence" which stores time and date information in a compact fashion. The "timeSequence" object may be converted to a "timeDate" object using the S-PLUS as function

```
> td = as(tds,"timeDate")
> td
 [1] 1900 1901 1902 1903 1904 1905 1906 1907 1908 1909 1910
```

Quarterly Sequences

A quarterly "timeDate" sequence from 1900:I through 1902:IV may be created using timeSeq with the by="quarters" option:

```
> timeSeq(from="1/1/1900", to="10/1/1902", by="quarters",
+ format="%Y:%Q")
 [1] 1900:I   1900:II  1900:III 1900:IV  1901:I   1901:II
 [7] 1901:III 1901:IV  1902:I   1902:II  1902:III 1902:IV
```

The output format character %Q displays the quarter information. Notice that the dates are specified as the first day of the quarter.

Monthly Sequences

Now consider creating a monthly "timeDate" sequence from January 1, 1900 through March 1, 1901. This may be done using timeCalendar

```
> timeCalendar(m=rep(1:12,length=15),y=rep(1900:1901,each=12,
+ length=15), format="%b %Y")
 [1] Jan 1900 Feb 1900 Mar 1900 Apr 1900 May 1900 Jun 1900
 [7] Jul 1900 Aug 1900 Sep 1900 Oct 1900 Nov 1900 Dec 1900
[13] Jan 1901 Feb 1901 Mar 1901
```

or timeSeq

```
> timeSeq(from="1/1/1900",to="3/1/1901",by="months",
+ format="%b %Y")
 [1] Jan 1900 Feb 1900 Mar 1900 Apr 1900 May 1900 Jun 1900
 [7] Jul 1900 Aug 1900 Sep 1900 Oct 1900 Nov 1900 Dec 1900
[13] Jan 1901 Feb 1901 Mar 1901
```

To create a monthly sequence of end of month values from December 31, 1899 through February 28, 1901, subtract 1 from the above calculation:

```
> timeSeq(from="1/1/1900",to="3/1/1901",by="months",
+ format="%b %Y") - 1
 [1] Dec 1899 Jan 1900 Feb 1900 Mar 1900 Apr 1900 May 1900
 [7] Jun 1900 Jul 1900 Aug 1900 Sep 1900 Oct 1900 Nov 1900
[13] Dec 1900 Jan 1901 Feb 1901
```

Weekly Sequences

Weekly sequences are best created using timeSeq with by="weeks". For example, a weekly sequence from Monday January 1, 1990 to Monday Feb 26, 1990 may be created using

```
> timeSeq(from="1/1/1990",to="3/1/1990",by="weeks",
+ format="%a %b %d, %Y")
[1] Mon Jan 1, 1990  Mon Jan 8, 1990  Mon Jan 15, 1990
[4] Mon Jan 22, 1990 Mon Jan 29, 1990 Mon Feb 5, 1990
[7] Mon Feb 12, 1990 Mon Feb 19, 1990 Mon Feb 26, 1990
```

To create a weekly sequence starting on a specific day, say Wednesday, make the starting date a Wednesday.

Daily Sequences

A regularly spaced daily sequence may be created using timeSeq with by = "days". For an irregularly spaced daily sequence of weekdays use timeSeq with by = "weekdays". For financial asset price data that trades on U.S. exchanges, the relevant "daily" sequence of dates is an irregularly spaced sequence based on business days. Business days are weekdays excluding certain holidays. For example, consider creating a daily "timeDate" sequence for the month of January, 2000 for a time series of asset prices that trade on the New York stock exchange (NYSE). The NYSE is not open on weekends and on certain holidays and these dates should be omitted from the "timeDate" sequence. The S-PLUS function holiday.NYSE returns the New York Stock Exchange holidays for a given year, 1885-present, according to the historical and current (as of 1998) schedule, not including special-event closure days or partial-day closures. The NYSE holidays for 2000 are

```
> holiday.NYSE(2000)
[1] 1/17/2000  2/21/2000  4/21/2000  5/29/2000  7/4/2000
[6] 9/4/2000   11/23/2000 12/25/2000
```

Martin Luther King day on Monday January 17^{th} is the only weekday holiday. A "timeDate" sequence of business days excluding the holiday 1/17/2000 may be created using

```
> timeSeq(from="1/3/2000",to="1/31/2000",by="bizdays",
+ holidays=holiday.NYSE(2000),format="%a %b %d, %Y")
 [1] Mon Jan 3, 2000  Tue Jan 4, 2000  Wed Jan 5, 2000
 [4] Thu Jan 6, 2000  Fri Jan 7, 2000  Mon Jan 10, 2000
 [7] Tue Jan 11, 2000 Wed Jan 12, 2000 Thu Jan 13, 2000
[10] Fri Jan 14, 2000 Tue Jan 18, 2000 Wed Jan 19, 2000
[13] Thu Jan 20, 2000 Fri Jan 21, 2000 Mon Jan 24, 2000
[16] Tue Jan 25, 2000 Wed Jan 26, 2000 Thu Jan 27, 2000
[19] Fri Jan 28, 2000 Mon Jan 31, 2000
```

The argument holidays=holiday.NYSE(2000) in conjunction with by = "bizdays" instructs timeSeq to exclude the weekday dates associated with the NYSE holidays for 2000. Notice that the date Mon Jan 17, 2000 has been omitted from the sequence.

Intra-day Irregularly Spaced Sequences

Sequences of irregularly spaced intra-day dates may be created using the function timeCalendar. For example, consider creating a sequence of hourly observations only during the hypothetical trading hours from 9:00 AM to 3:00 PM from Monday January 3, 2000 through Tuesday January 4, 2000. Such a sequence may be created using timeCalendar as follows

```
> timeCalendar(h=rep(9:15,2),d=rep(3:4,each=7),
+ y=2000,format="%a %b %d, %Y %02I:%02M %p")
 [1] Mon Jan 3, 2000 09:00 AM Mon Jan 3, 2000 10:00 AM
 [3] Mon Jan 3, 2000 11:00 AM Mon Jan 3, 2000 12:00 PM
 [5] Mon Jan 3, 2000 01:00 PM Mon Jan 3, 2000 02:00 PM
 [7] Mon Jan 3, 2000 03:00 PM Tue Jan 4, 2000 09:00 AM
 [9] Tue Jan 4, 2000 10:00 AM Tue Jan 4, 2000 11:00 AM
```

```
[11] Tue Jan 4, 2000 12:00 PM Tue Jan 4, 2000 01:00 PM
[13] Tue Jan 4, 2000 02:00 PM Tue Jan 4, 2000 03:00 PM
```

In a similar fashion, a sequence of minute observations from 9:00 AM to 3:00 PM on Monday January 3, 2000 and Tuesday January 4, 2000 may be created using

```
> timeCalendar(min=rep(rep(0:59,6),2),
+ h=rep(9:14,each=60,length=360*2),
+ d=rep(3:4,each=360,length=360*2),
+ y=2000,format="%a %b %d, %Y %02I:%02M %p")
 [1] Mon Jan 3, 2000 09:00 AM Mon Jan 3, 2000 09:01 AM
 [3] Mon Jan 3, 2000 09:02 AM Mon Jan 3, 2000 09:03 AM
...
[359] Mon Jan 3, 2000 02:58 PM Mon Jan 3, 2000 02:59 PM
[361] Tue Jan 4, 2000 09:00 AM Tue Jan 4, 2000 09:01 AM
...
[719] Tue Jan 4, 2000 02:58 PM Tue Jan 4, 2000 02:59 PM
```

2.2.4 Miscellaneous Time and Date Functions

In addition to the time and date functions discussed so far, S-PLUS has a number of miscellaneous time and date functions. In addition S+FinMetrics provides a few time and date functions. These are summarized in Table 2.1.

2.2.5 Creating "timeSeries" Objects

S-PLUS "timeSeries" objects are created with the timeSeries function. Typically a "timeSeries" is created from some existing data in a data frame or matrix and a "timeDate" object. For example,

```
> my.df = data.frame(x=abs(rnorm(10,mean=5)),
+ y=abs(rnorm(10,mean=10)))
> my.td = timeCalendar(y=1990:1999,format="%Y")
> my.ts = timeSeries(data=my.df,pos=my.td)
> my.ts
 Positions     x      y
 1990      4.250 11.087
 1991      5.290 11.590
 1992      5.594 11.848
 1993      5.138 10.426
 1994      5.205  9.678
 1995      4.804 11.120
 1996      5.726 11.616
 1997      6.124  9.781
 1998      3.981 10.725
```

S-PLUS **function**	**Description**
`month.day.year`	Converts calendar dates to Julian dates
`julian`	Converts Julian dates to calendar dates
`quarters`	Create an ordered factor corresponding to quarters
`months`	Create an ordered factor corresponding to months
`days`	Create an ordered factor corresponding to days
`weekdays`	Create an ordered factor corresponding to weekdays
`years`	Create an ordered factor corresponding to years
`yeardays`	Extract year day from date
`hours`	Extract hour from date
`minutes`	Extract minutes from date
`seconds`	Extract seconds from date
`hms`	Create data frame containing hours, minutes and seconds
`mdy`	Create data frame containing month, day and year
`wdydy`	Create data frame containing weekday, year day and year
`leap.year`	Determines if year number corresponds to a leap year
`holidays`	Generate a collection of holidays
`holiday.fixed`	Generate holidays that occur on fixed dates
`holiday.weekday.number`	Generate holidays that occur on weekdays
S+FinMetrics **function**	**Description**
`days.count`	Count number of days between two dates
`is.weekday`	Tests if date is a weekday
`is.weekend`	Tests if date is a weekend
`is.bizday`	Tests if date is a business day
`imm.dates`	Create International Monetary Market dates

TABLE 2.1. Miscellaneous time and date functions

```
 1999      6.006 10.341
```

Information about the “timeSeries” object may be added to the title, documentation and units slots:

```
> my.ts@title = "My timeSeries"
> my.ts@documentation = c("Simulated annual price data using ",
+ "the S-PLUS function rnorm")
> my.ts@units = c("US dollars","US dollars")
```

The title and units information is utilized in certain plot functions.

Creating “timeSeries” Objects from Time Series in Data Frames

Very often time series data that are in data frames have a date variable with a formatted date string. The S-PLUS function timeDate has a variety of input formats that may be used to convert such date strings into “timeDate” objects. For example, the S+FinMetrics data frame yhoo.df contains daily high, low, open and close prices as well as volume information for Yahoo stock for the month of February 2002

```
> yhoo.df[1:2,]
      Date  Open High   Low Close  Volume
1 1-Feb-02 17.26 17.3 16.35 16.68 6930100
2 4-Feb-02 16.55 16.6 15.60 15.75 8913700
```

The variable Date is a character vector containing the date strings. A “timeDate” sequence created from the date strings in Date is

```
> td = timeDate(yhoo.df[,1],in.format="%d-%m-%y",
+ format="%a %b %d, %Y")
> td[1:2]
[1] Fri Feb 1, 2002 Mon Feb 4, 2002
```

A “timeSeries” object containing the data from yhoo.df is created using

```
> yhoo.ts = timeSeries(pos=td,data=yhoo.df[,-1])
> yhoo.ts[1:2,]
       Positions  Open High   Low Close  Volume
 Fri Feb 1, 2002 17.26 17.3 16.35 16.68 6930100
 Mon Feb 4, 2002 16.55 16.6 15.60 15.75 8913700
```

High frequency data, however, is often recorded using nonstandard time formats. For example, consider the transactions level data for the month of December 1999 for 3M stock in the S+FinMetrics data frame highFreq3m.df

```
> highFreq3M.df[1:2,]
  trade.day trade.time trade.price
1         1      34412      94.688
2         1      34414      94.688
```

The variable trade.day contains the integer trading day of the month, the variable trade.time contains the integer trade time recorded as the number of seconds from midnight and the variable trade.price contains the transaction price in dollars. A "timeDate" sequence may be easily created from the trade day and trade time information as follows

```
> td = timeDate(julian=(highFreq3M.df$trade.day-1),
+ ms=highFreq3M.df$trade.time*1000,
+ in.origin=c(month=12,day=1,year=1999),zone="GMT")
> td[1:2]
[1] 12/1/99 9:33:32 AM 12/1/99 9:33:34 AM
```

The function timeDate can create a "timeDate" sequence using Julian date and millisecond information. The argument julian takes an integer vector containing the number of days since the date specified in the argument in.origin, and the argument ms takes an integer vector containing the number of milliseconds since midnight. In the above example, in.origin is specified as December 1, 1999 and the optional argument zone is used to set the time zone to GMT. A "timeSeries" object containing the high frequency data in highFreq3M.df is created using

```
> hf3M.ts = timeSeries(pos=td,data=highFreq3M.df)
```

2.2.6 Aggregating and Disaggregating Time Series

Often a regularly spaced financial time series of a given frequency may need to be aggregated to a coarser frequency or disaggregated to a finer frequency. In addition, aggregation and disaggregation may involve flow or stock variables. The S-PLUS functions aggregateSeries and align may be used for such purposes. To enhance and extend the disaggregation functionality in S-PLUS the S+FinMetrics function disaggregate is introduced.

Aggregating Time Series

Given a monthly "timeSeries" of end of month prices over a number of years, suppose one would like to create an annual time series consisting of the end of month December prices. Such a series may be easily constructed by subsetting using the S-PLUS function months:

```
> dec.vals = "Dec"==months(positions(singleIndex.dat))
> annual.p = singleIndex.dat[dec.vals,]
> annual.p
 Positions    MSFT  SP500
 Dec 1990    2.090  330.2
 Dec 1991    4.635  417.1
 Dec 1992    5.336  435.7
 Dec 1993    5.039  466.4
 Dec 1994    7.641  459.3
 Dec 1995   10.969  615.9
 Dec 1996   20.656  740.7
 Dec 1997   32.313  970.4
 Dec 1998   69.344 1229.2
 Dec 1999  116.750 1469.3
 Dec 2000   43.375 1320.3
```

Another way to create the above annual time series is to use the S-PLUS aggregateSeries function with a user-written function to pick off December values. One such function, based on the S-PLUS function hloc used to compute high, low, open and close values, is

```
pickClose = function(x)
{
# return closing values of a vector
     if(length(dim(x))) x = as.vector(as.matrix(x))
     len = length(x)
     if(!len)
          as(NA, class(x))
     else x[len]
}
```

The annual data is then constructed using aggregateSeries with optional arguments FUN=pickClose and by="years"

```
> annual.p = aggregateSeries(singleIndex.dat,
+ FUN=pickClose,by="years")
> positions(annual.p)@format = "%Y"
> annual.p
 Positions    MSFT  SP500
 1990        2.090  330.2
 1991        4.635  417.1
 1992        5.336  435.7
 1993        5.039  466.4
 1994        7.641  459.3
 1995       10.969  615.9
 1996       20.656  740.7
 1997       32.313  970.4
 1998       69.344 1229.2
 1999      116.750 1469.3
 2000       43.375 1320.3
 2001       61.063 1366.0
```

The function aggregateSeries passes to the function pickClose data from singleIndex.dat in blocks of year's length. The function pickClose simply picks off the last value for the year. Since singleIndex.dat only has data for January 2, 2001, the 2001 value for annual.p is this value.

The method described above may also be used to construct end-of-month closing price data from a "timeSeries" of daily closing price data. For example, the commands to create end of month closing prices from daily closing prices for Microsoft, taken from the S+FinMetrics "timeSeries" DowJones30, using aggregateSeries with FUN = pickClose and by = "months" are

```
> msft.daily.p = DowJones30[,"MSFT"]
> msft.daily.p@title = "Daily closing price on Microsoft"
> msft.daily.p@units = "Dollar price"
> msft.monthly.p = aggregateSeries(msft.daily.p,FUN=pickClose,
+ by="months",adj=0.99)
> msft.monthly.p[1:12]
  Positions  MSFT
  1/31/1991 2.726
  2/28/1991 2.882
  3/31/1991 2.948
  4/30/1991 2.750
  5/31/1991 3.049
  6/30/1991 2.838
  7/31/1991 3.063
  8/31/1991 3.552
  9/30/1991 3.708
 10/31/1991 3.912
 11/30/1991 4.052
 12/31/1991 4.635
```

The option adj=0.99 adjusts the positions of the monthly data to the end of the month. Notice that the end of month dates are not necessarily the last trading days of the month.

The monthly closing price data may be extracted from the daily closing price data by clever use of subscripting[4]. One way to do this is

```
> end.month.idx =
+ which(diff(as.numeric(months(positions(msft.daily.p)))) != 0)
> msft.monthly.p = msft.daily.p[end.month.idx]
> msft.monthly.p[1:12]
  Positions  MSFT
  1/31/1991 2.726
  2/28/1991 2.882
  3/28/1991 2.948
  4/30/1991 2.750
```

[4]This method was suggested by Steve McKinney.

```
  5/31/1991 3.049
  6/28/1991 2.838
  7/31/1991 3.063
  8/30/1991 3.552
  9/30/1991 3.708
 10/31/1991 3.912
 11/29/1991 4.052
 12/31/1991 4.635
```

A common aggregation operation with financial price data is to construct a *volume weighted average price* (vwap). This may be easily accomplished with `aggregateSeries` and a user-specified function to compute the vwap. For example, consider the daily open, high, low and close prices and volume on Microsoft stock from October 2, 2000 through August 31, 2001 in the `S+FinMetrics` “`timeSeries`” `msft.dat`.

```
> smpl = (positions(msft.dat) >= timeDate("10/1/2000") &
+ positions(msft.dat) <= timeDate("8/31/2001"))
> msft.dat[smpl,]
  Positions  Open  High   Low Close    Volume
  10/2/2000 60.50 60.81 58.25 59.13  29281200
  ...
  8/31/2001 56.85 58.06 56.30 57.05  28950400
```

A function that can be used to aggregate open, high, low and close prices, volume and compute the open and close vwap is

```
vol.wtd.avg.price = function(x) {
  VolumeSum = as.double(sum(x[, "Volume"]))
  nrowx = numRows(x)
  return(data.frame(Open = x[1, "Open"],
  High = max(x[, "High"]),
  Low = min(x[, "Low"]),
  Close = x[nrowx, "Close"],
  vwap.Open = sum(x[, "Open"] * x[, "Volume"])/VolumeSum,
  wap.Close = sum(x[, "Close"] * x[, "Volume"])/VolumeSum,
  Volume = VolumeSum))
}
```

Using `aggregateSeries` and the function `vol.wtd.avg.price` one can compute the monthly open, high, low, close prices, volume, and open and close vwap

```
> msft.vwap.dat = aggregateSeries(x = msft.dat[smpl,],
+ by = "months",FUN = vol.wtd.avg.price,
+ together = T)
> positions(msft.vwap.dat)@format="%b %Y"
> msft.vwap.dat[,-7]
```

```
  Positions  Open  High   Low Close vwap.Open vwap.Close
  Oct 2000  60.50 70.13 48.44 68.88 59.10     59.48
  Nov 2000  68.50 72.38 57.00 57.38 68.35     67.59
  ...
  Aug 2001  66.80 67.54 56.30 57.05 62.99     62.59
```

Disaggregating Time Series

Consider the problem of creating a daily “timeSeries” of inflation adjusted (real) prices on Microsoft stock over the period January 2, 1991 through January 2, 2001. To do this the daily nominal prices must be divided by a measure of the overall price level; e.g. the consumer price level (CPI). The daily nominal stock price data is in the “timeSeries” msft.daily.p created earlier and the CPI data is in the S+FinMetrics “timeSeries” CPI.dat. The CPI data, however, is only available monthly.

```
> start(CPI.dat)
[1] Jan 1913
> end(CPI.dat)
[1] Nov 2001
```

and represents the average overall price level during the month but is recorded at the end of the month. The CPI data from December 1990 through January 2001 is extracted using

```
> smpl = (positions(CPI.dat) >= timeDate("12/1/1990")
+ & positions(CPI.dat) <= timeDate("2/1/2001"))
> cpi = CPI.dat[smpl,]
> cpi[1:3]
 Positions   CPI
 Dec 1990  134.3
 Jan 1991  134.8
 Feb 1991  134.9
```

To compute real daily prices on Microsoft stock, the monthly CPI data in the “timeSeries” object cpi must be *disaggregated* to daily data. This disaggregation may be done in a number of ways. For example, the CPI for every day during the month of January, 1991 may be defined as the monthly CPI value for December, 1990 or the monthly CPI value for January, 1991. Alternatively, the daily values for January 1991 may be computed by linearly interpolating between the December, 1990 and January, 1991 values. The S-PLUS function align may be used to do each of these disaggregations.

The align function aligns a “timeSeries” object to a given set of positions and has options for the creation of values for positions in which the “timeSeries” does not have values. For example, the disaggregated

CPI using the previous month's value for the current month's daily data is constructed using

```
> cpi.daily.before =
+ align(cpi,positions(msft.daily.p),how="before")
> cpi.daily.before[c(1:3,21:23)]
 Positions   CPI
  1/2/1991 134.3
  1/3/1991 134.3
  1/4/1991 134.3
 1/30/1991 134.3
 1/31/1991 134.8
  2/1/1991 134.8
```

The new positions to align the CPI values are the daily positions of the "timeSeries" msft.daily.p, and the argument how="before" specifies that the previous month's CPI data is to be used for the current month's daily CPI values. Similarly, the disaggregated CPI using the next month's value for the current month's daily data is constructed using

```
> cpi.daily.after =
+ align(cpi,positions(msft.daily.p),how="after")
> cpi.daily.after[c(1:3,21:23)]
 Positions   CPI
  1/2/1991 134.8
  1/3/1991 134.8
  1/4/1991 134.8
 1/30/1991 134.8
 1/31/1991 134.8
  2/1/1991 134.9
```

Finally, the disaggregated daily CPI using linear interpolation between the monthly values is constructed using

```
> cpi.daily.interp = align(cpi,positions(msft.daily.p),
+ how="interp")
> cpi.daily.interp[c(1:3,21:23)]
 Positions   CPI
  1/2/1991 134.3
  1/3/1991 134.3
  1/4/1991 134.4
 1/30/1991 134.8
 1/31/1991 134.8
  2/1/1991 134.8
```

The daily real prices on Microsoft stock using the interpolated daily CPI values are then

```
> msft.daily.rp = (msft.daily.p/cpi.daily.interp)*100
```

Disaggregating Time Series using the S+FinMetrics disaggregate Function

With economic and financial time series, it is sometimes necessary to distribute a flow variable or time average a stock variable that is observed at a low frequency to a higher frequency. For example, a variable of interest may only be observed on an annual basis and quarterly or monthly values are desired such that their sum is equal to the annual observation or their average is equal to the annual observation. The S+FinMetrics function disaggregate performs such disaggregations using two methods. The first method is based on cubic spline interpolation and is appropriate if the only information is on the series being disaggregated. The second method utilizes a generalized least squares (gls) fitting method due to Chow and Lin (1971) and is appropriate if information is available on one or more related series that are observed at the desired disaggregated frequency. The arguments expected by disaggregate are

```
> args(disaggregate)
function(data, k, method = "spline", how = "sum", x = NULL,
+ out.positions = NULL, ...)
```

where data is a vector, matrix or “timeSeries” of low frequency data, k is the number of disaggregtion periods, method determines the disaggregation method (spline or gls), how specifies if the disaggregated values sum to the aggregated values or are equal on average to the disaggregated values, x respresents any related observed data at the disaggregated frequency and out.positions represents a “timeDate” sequence for the resulting output.

To illustrate the use of disaggregate, consider the problem of disaggregating the annual dividend on the S&P 500 index to a monthly dividend. Since the annual dividend is a flow variable, the sum of the monthly dividends should equal the annual dividend. The annual S&P 500 dividend information over the period 1871 - 2000 is in the S+FinMetrics “timeSeries” shiller.annual. The disaggregated monthly dividend values such that their sum is equal to the annual values is created using

```
> monthly.dates = timeSeq(from="1/1/1871",to="12/31/2000",
+ by="months",format="%b %Y")
> div.monthly =
+ disaggregate(shiller.annual[,"dividend"],12,
+ out.positions=monthly.dates)
> div.monthly[1:12]
  Positions dividend
  Jan 1871  0.02999
  Feb 1871  0.01867
  Mar 1871  0.01916
  Apr 1871  0.01963
  May 1871  0.02009
  Jun 1871  0.02054
  Jul 1871  0.02097
  Aug 1871  0.02140
  Sep 1871  0.02181
  Oct 1871  0.02220
  Nov 1871  0.02259
  Dec 1871  0.02296
> sum(div.monthly[1:12])
[1] 0.26
> shiller.annual[1,"dividend"]
  Positions dividend
  1871      0.26
```

For the S&P 500 index, the index price is available in the `S+FinMetrics` monthly "`timeSeries`" `shiller.dat`. This information may be utilized in the disaggregation of the annual dividend using the gls method as follows

```
> smpl = positions(shiller.dat) <= timeDate("12/31/2000")
> price.monthly = as.matrix(seriesData(shiller.dat[smpl,"price"]))
> div2.monthly =
+ disaggregate(shiller.annual[,"dividend"], 12,
+ method="gls", x=price.monthly, out.positions=monthly.dates)
> div2.monthly[1:12]
  Positions dividend
  Jan 1871  0.006177
  Feb 1871  0.010632
  Mar 1871  0.014610
  Apr 1871  0.018104
  May 1871  0.021104
  Jun 1871  0.023569
  Jul 1871  0.025530
  Aug 1871  0.027043
  Sep 1871  0.028063
  Oct 1871  0.028508
  Nov 1871  0.028548
  Dec 1871  0.028111
> sum(div2.monthly[1:12])
[1] 0.26
> shiller.annual[1,"dividend"]
  Positions dividend
  1871      0.26
```

2.2.7 Merging Time Series

Often one would like to combine several "`timeSeries`" objects into a single "`timeSeries`" object. The `S-PLUS` functions `c`, `concat` and `cbind` do not operate on "timeSeries" objects. Instead, the S-PLUS function seriesMerge is used to combine or merge a collection of "timeSeries". To illustrate, consider creating a new "timeSeries" object consisting of the S+FinMetrics "timeSeries" CPI.dat and IP.dat containing monthly observations on the U.S. consumer price index and U.S. industrial production index, respectively:

```
> CPI.dat
  Positions    CPI
  Jan 1913    9.80
  Feb 1913    9.80
  ...
  Nov 2001  177.60
> IP.dat
  Positions      IP
  Jan 1919    7.628
  Feb 1919    7.291
  ...
  Nov 2001  137.139
```

Notice that the start date for CPI.dat is earlier than the start date for IP.dat,but the end dates are the same. A new "timeSeries" containing both CPI.dat and IP.dat with positions aligned to those for IP.dat using seriesMerge is

```
> IP.CPI.dat = seriesMerge(IP.dat,CPI.dat,
+ pos=positions(IP.dat))
> IP.CPI.dat[1:2,]
  Positions    IP  CPI
  Jan 1919  7.628 16.5
  Feb 1919  7.291 16.2
```

To create a "timeSeries" with positions given by the union of the positions for CPI.dat and IP.dat set pos="union" in the call to seriesMerge. Since IP.dat does not have observations for the dates January 1913 through December 1918, NA values for IP for these dates will be inserted in the new "timeSeries".

2.2.8 *Dealing with Missing Values Using the **S+FinMetrics** Function **interpNA***

Occasionally, time series data contain missing or incorrect data values. One approach often used to fill-in missing values is interpolation[5]. The S-PLUS

[5]More sophisticated imputation methods for dealing with missing values are available in the library S+MISSINGDATA which is included with S-PLUS.

align function may be used for this purpose. The S+FinMetrics function interpNA performs similar missing value interpolation as align but is easier to use and is more flexible. The arguments expected by interpNA are

```
> args(interpNA)
function(x, method = "spline")
```

where x is a rectangular object and method sets the interpolation method. Valid interpolation methods are "before", "after", "nearest", "linear" and (cubic) "spline". To illustrate the use of interpNA, note that the closing price for the Dow Jones Industrial Average in the S-PLUS "timeSeries" djia has a missing value on January 18, 1990:

```
> djia.close = djia[positions(djia) >= timeDate("1/1/1990"),
+ "close"]
> djia.close[10:12,]
  Positions  close
 01/17/1990 2659.1
 01/18/1990     NA
 01/19/1990 2677.9
```

To replace the missing value with an interpolated value based on a cubic spline use

```
> djia.close = interpNA(djia.close)
> djia.close[10:12,]
  Positions      1
 01/17/1990 2659.1
 01/18/1990 2678.7
 01/19/1990 2677.9
```

2.3 Time Series Manipulation in S-PLUS

There are several types of common manipulations and transformations that often need to be performed before a financial time series is to be analyzed. The most important transformations are the creation of lagged and differenced variables and the creation of returns from asset prices. The following sections describe how these operations may be performed in S-PLUS.

2.3.1 Creating Lags and Differences

Three common operations on time series data are the creation of lags, leads, and differences. The S-PLUS function shift may be used to create leads and lags, and the generic function diff may be used to create differences. However, these functions do not operate on "timeSeries" objects in the most convenient way. Consequently, the S+FinMetrics module contains the functions tslag and diff.timeSeries for creating lags/leads and differences.

Creating Lags and Leads Using the S+FinMetrics Function tslag

The S+FinMetrics function tslag creates a specified number of lag/leads of a rectangular data object. The arguments expected by tslag are

```
> args(tslag)
function(x, k = 1, trim = F)
```

where x is any rectangular object, k specifies the number of lags to be created (negative values create leads) and trim determines if NA values are to be trimmed from the result. For example, consider the "timeSeries" singleIndex.dat containing monthly prices on Microsoft and the S&P 500 index. The first five values are

```
> singleIndex.dat[1:5,]
 Positions  MSFT SP500
 Jan 1990  1.285 329.1
 Feb 1990  1.371 331.9
 Mar 1990  1.538 339.9
 Apr 1990  1.611 330.8
 May 1990  2.028 361.2
```

The "timeSeries" of lagged values using tslag are

```
> tslag(singleIndex.dat[1:5,])
 Positions MSFT.lag1 SP500.lag1
 Jan 1990     NA        NA
 Feb 1990  1.285     329.1
 Mar 1990  1.371     331.9
 Apr 1990  1.538     339.9
 May 1990  1.611     330.8
```

Notice that tslag creates a "timeSeries" containing the lagged prices on Microsoft and the S&P 500 index. The variable names are adjusted to indicate the type of lag created and since trim=F, NA values are inserted for the first observations. To create a "timeSeries" without NA values in the first position, use tslag with trim=T:

```
> tslag(singleIndex.dat[1:5,],trim=T)
 Positions MSFT.lag1 SP500.lag1
 Feb 1990  1.285     329.1
 Mar 1990  1.371     331.9
 Apr 1990  1.538     339.9
 May 1990  1.611     330.8
```

Leads are created by setting k equal to a negative number:

```
> tslag(singleIndex.dat[1:5,],k=-1)
 Positions MSFT.lead1 SP500.lead1
 Jan 1990  1.371      331.9
 Feb 1990  1.538      339.9
 Mar 1990  1.611      330.8
 Apr 1990  2.028      361.2
 May 1990     NA         NA
```

To create a "timeSeries" with multiple lagged values, simply specify the lags to create in the call to tslag. For example, specifying k=c(1,3) creates the first and third lag

```
> tslag(singleIndex.dat[1:5,],k=c(1,3))
 Positions MSFT.lag1 SP500.lag1 MSFT.lag3 SP500.lag3
 Jan 1990     NA        NA         NA        NA
 Feb 1990  1.285     329.1         NA        NA
 Mar 1990  1.371     331.9         NA        NA
 Apr 1990  1.538     339.9      1.285     329.1
 May 1990  1.611     330.8      1.371     331.9
```

Similarly, specifying k=-1:1 creates

```
> tslag(singleIndex.dat[1:5,],k=-1:1)
 Positions MSFT.lead1 SP500.lead1 MSFT.lag0 SP500.lag0
 Jan 1990  1.371      331.9       1.285     329.1
 Feb 1990  1.538      339.9       1.371     331.9
 Mar 1990  1.611      330.8       1.538     339.9
 Apr 1990  2.028      361.2       1.611     330.8
 May 1990     NA         NA       2.028     361.2
 MSFT.lag1 SP500.lag1
    NA        NA
 1.285     329.1
 1.371     331.9
 1.538     339.9
 1.611     330.8
```

Creating Differences Using the S+FinMetrics Function diff.timeSeries

The S+FinMetrics function diff.timeSeries is a method function for the generic S-PLUS function diff for objects of class "timeSeries" and creates a specified number of differences of a "timeSeries" object. The arguments expected by diff.timeSeries are

```
> args(diff.timeSeries)
function(x, lag = 1, differences = 1, trim = T, pad = NA)
```

where x represents a "timeSeries" object, lag specifies the number of lagged periods used in the difference, differences specifies the number of times to difference the series, `trim` determines if the resulting series is to have `NA` values removed and trimmed and `pad` specifies the value to be padded to the series in the positions where the differencing operation exceeds the start or the end positions. For example, consider again the "`timeSeries`" `singleIndex.dat` containing monthly prices on Microsoft and the S&P 500 index. Let P_t denote the price at time t. To create the first difference $\Delta P_t = P_t - P_{t-1}$ use `diff` with `lag=1`:

```
> diff(singleIndex.dat[1:5,],lag=1,trim=F)
 Positions   MSFT  SP500
 Jan 1990      NA     NA
 Feb 1990  0.0868   2.81
 Mar 1990  0.1667   8.05
 Apr 1990  0.0729  -9.14
 May 1990  0.4167  30.43
```

To create the difference $P_t - P_{t-2}$ and pad the result with zeros instead of NAs use `diff` with `lag=2` and `pad=0`:

```
> diff(singleIndex.dat[1:5,],lag=2,trim=F,pad=0)
 Positions   MSFT  SP500
 Jan 1990  0.0000   0.00
 Feb 1990  0.0000   0.00
 Mar 1990  0.2535  10.86
 Apr 1990  0.2396  -1.09
 May 1990  0.4896  21.29
```

To create the 2nd difference $\Delta^2 P_t = \Delta(P_t - P_{t-1}) = P_t - 2P_{t-1} + P_{t-2}$ use `diff` with `lag=1` and `diff=2`:

```
> diff(singleIndex.dat[1:5,],lag=1,diff=2,trim=F)
 Positions     MSFT  SP500
 Jan 1990        NA     NA
 Feb 1990        NA     NA
 Mar 1990    0.0799   5.24
 Apr 1990   -0.0938 -17.19
 May 1990    0.3438  39.57
```

Unlike `tslag`, `diff.timeSeries` does not rename the variables to indicate the differencing operation performed. Additionally, `diff.timeSeries` will not accept a vector of values for the arguments `lag` and `differences`.

2.3.2 Return Definitions

Simple Returns

Let P_t denote the price at time t of an asset that pays no dividends and let P_{t-1} denote the price at time $t-1$. Then the *simple net return* on an investment in the asset between times $t-1$ and t is defined as

$$R_t = \frac{P_t - P_{t-1}}{P_{t-1}} = \%\Delta P_t. \tag{2.1}$$

Writing $\frac{P_t - P_{t-1}}{P_{t-1}} = \frac{P_t}{P_{t-1}} - 1$, we can define the *simple gross return* as

$$1 + R_t = \frac{P_t}{P_{t-1}}. \tag{2.2}$$

Unless otherwise stated, references to returns mean net returns.

The simple two-period return on an investment in an asset between times $t-2$ and t is defined as

$$\begin{aligned} R_t(2) &= \frac{P_t - P_{t-2}}{P_{t-2}} = \frac{P_t}{P_{t-2}} - 1 \\ &= \frac{P_t}{P_{t-1}} \cdot \frac{P_{t-1}}{P_{t-2}} - 1 \\ &= (1+R_t)(1+R_{t-1}) - 1. \end{aligned}$$

Then the simple two-period gross return becomes

$$1 + R_t(2) = (1+R_t)(1+R_{t-1}) = 1 + R_{t-1} + R_t + R_{t-1}R_t,$$

which is a *geometric* (multiplicative) sum of the two simple one-period gross returns and not the simple sum of the one period returns. If, however, R_{t-1} and R_t are small then $R_{t-1}R_t \approx 0$ and $1 + R_t(2) \approx 1 + R_{t-1} + R_t$ so that $R_t(2) \approx R_{t-1} + R_t$.

In general, the k-period gross return is defined as the geometric average of k one period gross returns

$$\begin{aligned} 1 + R_t(k) &= (1+R_t)(1+R_{t-1})\cdots(1+R_{t-k+1}) \qquad (2.3) \\ &= \prod_{j=0}^{k-1}(1+R_{t-j}) \end{aligned}$$

and the k-period net return is

$$R_t(k) = \prod_{j=0}^{k-1}(1+R_{t-j}) - 1. \tag{2.4}$$

Continuously Compounded Returns

Let R_t denote the simple one period return on an investment. The *continuously compounded one period return, r_t*, is defined as

$$r_t = \ln(1+R_t) = \ln\left(\frac{P_t}{P_{t-1}}\right) \tag{2.5}$$

where $\ln(\cdot)$ is the natural log function. To see why r_t is called the continuously compounded return, take exponentials of both sides of (2.5) to give

$$e^{r_t} = 1 + R_t = \frac{P_t}{P_{t-1}}.$$

Rearranging gives

$$P_t = P_{t-1}e^{r_t},$$

so that r_t is the continuously compounded growth rate in prices between periods $t-1$ and t. This is to be contrasted with R_t which is the simple growth rate in prices between periods $t-1$ and t without any compounding. Since $\ln\left(\frac{x}{y}\right) = \ln(x) - \ln(y)$ it follows that

$$\begin{aligned} r_t &= \ln\left(\frac{P_t}{P_{t-1}}\right) \\ &= \ln(P_t) - \ln(P_{t-1}) \\ &= p_t - p_{t-1} \end{aligned}$$

where $p_t = \ln(P_t)$. Hence, the continuously compounded one period return, r_t, can be computed simply by taking the first difference of the natural logarithms of prices between periods $t-1$ and t.

Given a one period continuously compounded return r_t, it is straightforward to solve back for the corresponding simple net return R_t:

$$R_t = e^{r_t} - 1$$

Hence, nothing is lost by considering continuously compounded returns instead of simple returns.

The computation of multi-period continuously compounded returns is considerably easier than the computation of multi-period simple returns. To illustrate, consider the two period continuously compounded return defined as

$$r_t(2) = \ln(1 + R_t(2)) = \ln\left(\frac{P_t}{P_{t-2}}\right) = p_t - p_{t-2}.$$

Taking exponentials of both sides shows that

$$P_t = P_{t-2}e^{r_t(2)}$$

so that $r_t(2)$ is the continuously compounded growth rate of prices between periods $t-2$ and t. Using $\frac{P_t}{P_{t-2}} = \frac{P_t}{P_{t-1}} \cdot \frac{P_{t-1}}{P_{t-2}}$ and the fact that $\ln(x \cdot y) = \ln(x) + \ln(y)$ it follows that

$$\begin{aligned} r_t(2) &= \ln\left(\frac{P_t}{P_{t-1}} \cdot \frac{P_{t-1}}{P_{t-2}}\right) \\ &= \ln\left(\frac{P_t}{P_{t-1}}\right) + \ln\left(\frac{P_{t-1}}{P_{t-2}}\right) \\ &= r_t + r_{t-1}. \end{aligned}$$

Hence the continuously compounded two period return is just the sum of the two continuously compounded one period returns.

The continuously compounded k-period return is defined as

$$r_t(k) = \ln(1 + R_t(k)) = \ln\left(\frac{P_t}{P_{t-k}}\right) = p_t - p_{t-k}. \tag{2.6}$$

Using similar manipulations to the ones used for the continuously compounded two period return the continuously compounded k-period return may be expressed as the sum of k continuously compounded one period returns:

$$r_t(k) = \sum_{j=0}^{k-1} r_{t-j}. \tag{2.7}$$

The additivitity of continuously compounded returns to form multiperiod returns is an important property for statistical modeling purposes.

2.3.3 *Computing Asset Returns Using the* `S+FinMetrics` *Function* `getReturns`

Given a data set with asset prices the `S+FinMetrics` function `getReturns` may be used to compute discrete and continuously compounded returns. The arguments to `getReturns` are

```
> args(getReturns)
function(x, type = "continuous", percentage = F, trim = T)
```

where `x` is any rectangular data object and `type` specifies the type of returns to compute (discrete or continuously compounded). To illustrate, the `S+FinMetrics` "`timeSeries`" `singleIndex.dat` contains monthly closing prices on Microsoft stock and the S&P 500 index, adjusted for stock splits and dividends, over the period January 1990 through January 2001.

```
> colIds(singleIndex.dat)
[1] "MSFT"  "SP500"
> singleIndex.dat[1:3,]
 Positions   MSFT  SP500
 Jan 1990  1.2847 329.08
 Feb 1990  1.3715 331.89
 Mar 1990  1.5382 339.94
```

A "`timeSeries`" of simple one-period discrete returns expressed as percentages is computed as

```
> ret.d = getReturns(singleIndex.dat,type="discrete",
+ percentage=T)
> ret.d[1:3,]
 Positions   MSFT   SP500
 Feb 1990   6.756  0.8539
 Mar 1990  12.155  2.4255
 Apr 1990   4.739 -2.6887
```

By default the first observation in the "timeSeries" is trimmed. To retain the first (NA) observation use the optional argument trim=F

```
> ret.d = getReturns(singleIndex.dat,type="discrete",trim=F)
> ret.d[1:3,]
 Positions     MSFT     SP500
 Jan 1990        NA        NA
 Feb 1990  0.067564  0.008539
 Mar 1990  0.121546  0.024255
```

Continuously compounded returns are created by specifying the optional argument type="continuous"

```
> ret.cc = getReturns(singleIndex.dat,type="continuous")
> ret.cc[1:3,]
 Positions     MSFT      SP500
 Feb 1990  0.065380  0.0085027
 Mar 1990  0.114708  0.0239655
 Apr 1990  0.046304 -0.0272552
```

Multiperiod returns may be computed from a "timeSeries" of one period returns using the S-PLUS function aggregateSeries. Multiperiod returns may be either overlapping or non-overlapping. For example, consider computing a monthly "timeSeries" of overlapping annual continuously compounded returns from the monthly continuously compounded returns in the "timeSeries" ret.cc using aggregateSeries:

```
> ret12.cc = aggregateSeries(ret.cc,moving=12,FUN=sum)
> ret12.cc[1:3,]
 Positions    MSFT    SP500
 Feb 1990  0.75220 0.044137
 Mar 1990  0.74254 0.100749
 Apr 1990  0.65048 0.098743
> colSums(seriesData(ret.cc[1:12,]))
   MSFT    SP500
 0.7522 0.044137
```

The argument moving=12 and FUN=sum tells aggregateSeries to compute a moving sum of twelve returns. Hence, the annual return reported for Feb 1990 is the sum of the twelve monthly returns from February 1990 through January 1991. Non-overlapping annual returns are computed from the monthly returns using aggregateSeries with the option by="years"

```
> ret12.cc = aggregateSeries(ret.cc,by="years",FUN=sum)
> ret12.cc[1:3,]
```

```
 Positions    MSFT     SP500
 Jan 1990  0.48678 0.0034582
 Jan 1991  0.79641 0.2335429
 Jan 1992  0.14074 0.0436749
> colSums(seriesData(ret.cc[1:11,]))
   MSFT     SP500
 0.48678 0.0034582
```

The "timeSeries" ret12.cc is now an annual series of non-overlapping annual returns. Notice that the annual return for January 1990 is computed using only the eleven returns from February 1990 through December 1990.

Multiperiod discrete returns (2.4) may be computed using the function aggregateSeries with FUN=prod. For example, a monthly "timeSeries" of overlapping annual discrete returns is computed as

```
> ret12.d = aggregateSeries((1+ret.d),moving=12,FUN=prod)-1
> ret12.d[1:3,]
 Positions    MSFT    SP500
 Feb 1990  1.12166 0.045126
 Mar 1990  1.10128 0.105999
 Apr 1990  0.91646 0.103783
> prod(seriesData(1+ret.d[1:12,1]))-1
[1] 1.1217
```

Notice that 1 is added to the return data and 1 is subtracted from the result in order to compute (2.4) properly. Non-overlapping multiperiod discrete returns may be computed using

```
> ret12.d = aggregateSeries((1+ret.d),by="years",FUN=prod)-1
> ret12.d[1:3,]
 Positions   MSFT   SP500
 Jan 1990      NA      NA
 Jan 1991  1.2176 0.26307
 Jan 1992  0.1511 0.04464
```

2.4 Visualizing Time Series in S-PLUS

Time series data in "timeSeries" objects may be visualized by using the S-PLUS generic plot function, the S-PLUS trellisPlot function, or by using the S+FinMetrics plotting functions based on Trellis graphics.

2.4.1 *Plotting "timeSeries" Using the S-PLUS Generic plot Function*

The S-PLUS generic plot function has a method function, plot.timeSeries, for plotting "timeSeries" objects. To illustrate, consider the monthly clos-

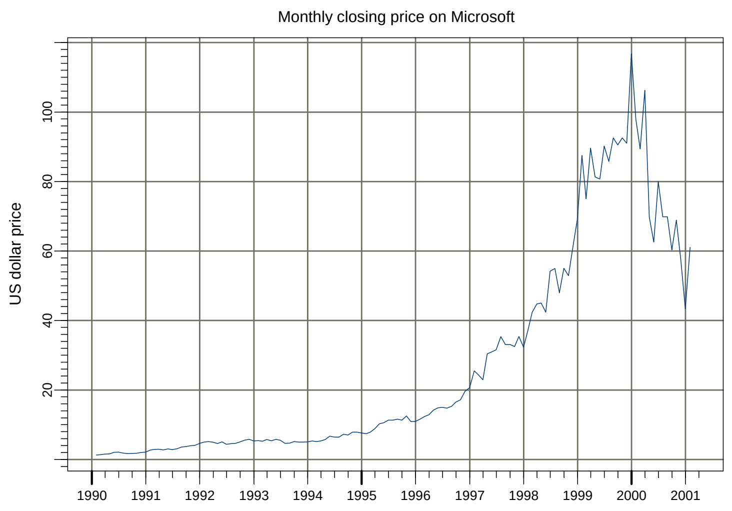


FIGURE 2.1. Monthly closing prices on Microsoft stock created using `plot.timeSeries`.

ing prices of Microsoft stock over the period January 1990 to January 2001 in the "`timeSeries`" object `msft.p` created earlier:

```
> msft.p@title
[1] "Monthly closing price on Microsoft"
> msft.p@units
[1] "US dollar price"
```

Figure 2.1 shows the output produced by the generic `plot` function

```
> plot(msft.p)
```

Notice how the information in the `title` and `units` slots is utilized in the plot. To eliminate the horizontal and vertical grid lines specify `reference.grid=F` in the call to `plot`. To show the price data on a logarithmic scale specify `log.axes="y"` in the call to `plot`.

Multiple series (on the same scale) may also be plotted together on the same plot using `plot`[6]. For example, the prices for Microsoft and the S&P 500 index in the "`timeSeries`" `singleIndex.dat` may be plotted together using

[6]To create a scatterplot of two "`timeSeries`" use the extractor function `seriesData` possibly in conjunction with the coersion function `as.matrix` on the "`timeSeries`" objects in the call to `plot`. Alternatively, the `S+FinMetrics` function `rvfPlot` may be used.

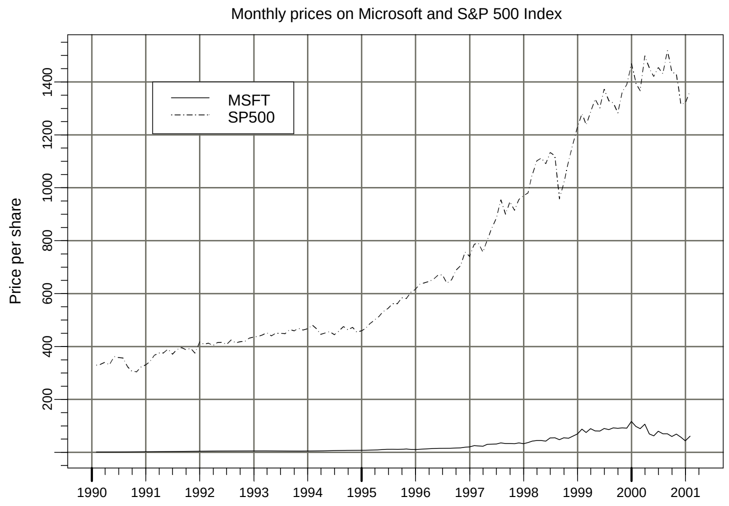


FIGURE 2.2. Monthly closing prices on Microsoft and the S&P 500 index created using `plot.timeSeries`.

```
> plot(singleIndex.dat,plot.args=list(lty=c(1,3)))
> legend(0.1,1400,legend=colIds(singleIndex.dat),lty=c(1,3))
```

The plot is illustrated in Figure 2.2. Notice how the line types are specified as a list argument to the optional argument `plot.args`. In the placement of the legend, the x-axis units are treated as values in the unit interval.

Multipanel plots may be created by specifying the plot layout using the S-PLUS function `par`. Figure 2.3 shows a two panel plot of the price data in `singleIndex.dat` produced using

```
> par(mfrow=c(2,1))
> plot(singleIndex.dat[,"MSFT"],
+ main="Monthly price on Microsoft")
> plot(singleIndex.dat[,"SP500"],
+ main="Monthly price on S&P 500 index")
```

Two specialized plot types for financial data can be made with the function `plot.timeSeries`. The first is a high/low/open/close (hloc) plot and the second is a stackbar plot. These plots are made by setting `plot.type = "hloc"` or `plot.type = "stackbar"` in the call to `plot.timeSeries`. For a hloc plot, the "`timeSeries`" to be plotted must have hloc information or such information must be created using `aggregateSeries` with the S-PLUS function `hloc`. Stackbar plots are generally used for plotting asset volume

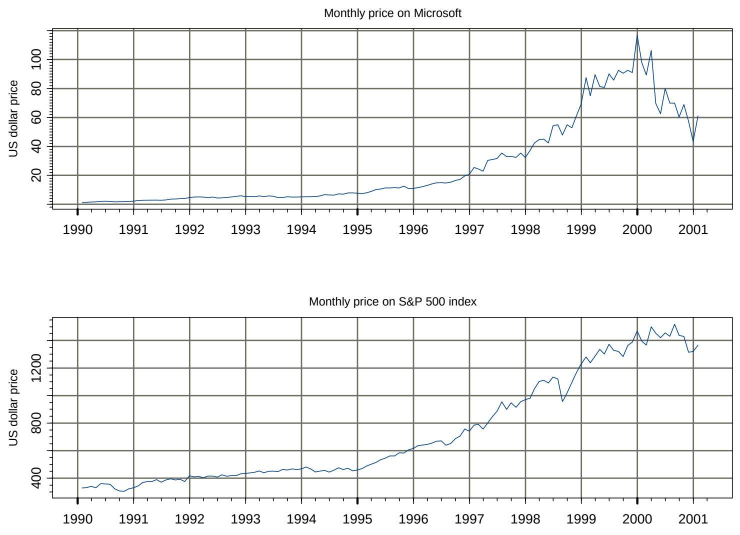


FIGURE 2.3. Two panel plot created using `par(mfrow=c(2,1))` in conjunction with `plot.timeSeries`.

information. To illustrate these plot types, consider the monthly data from the Dow Jones Industrial Averages in the S-PLUS “`timeSeries`” `djia`:

```
> colIds(djia)
[1] "open"   "high"   "low"    "close"  "volume"
```

Figure 2.4 gives a multipanel plot showing high, low, open, close and volume information created by

```
> smpl = (positions(djia) >= timeDate("9/1/1987") &
+ positions(djia) <= timeDate("11/30/1987"))
> par(mfrow=c(2,1))
> plot(djia[smpl,1:4],plot.type="hloc")
> plot(djia[smpl,5],plot.type="stackbar")
```

Lines may be added to an existing time series plot using the S-PLUS function `lines.render` and stackbar information may be added using the S-PLUS function `stackbar.render`. See chapter 26 in the *S-PLUS Guide to Statistics Vol. II* for details on using these functions.

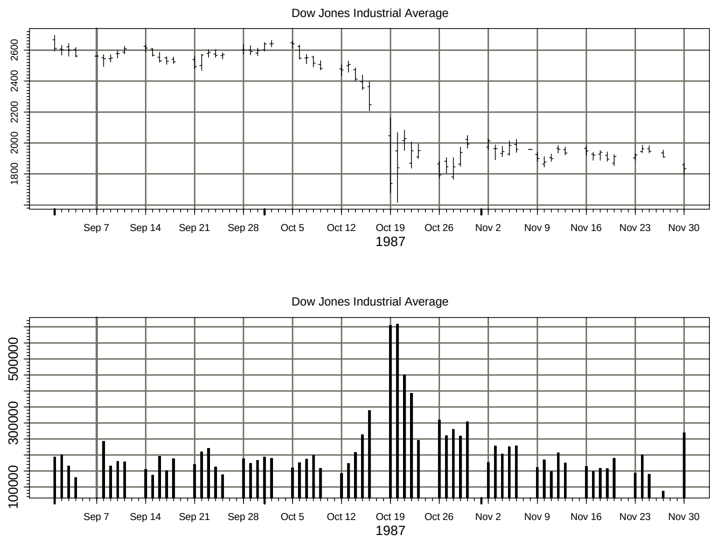


FIGURE 2.4. Monthly high, low, open, close and volume information for the Dow Jones Industrial Average using `plot.timeSeries` with `type="hloc"` and `type="stackbar"`.

Function	**Description**
`seriesPlot`	Trellis time series plot
`histPlot`	Trellis histogram plot
`qqPlot`	Trellis qq-plot for various distributions

TABLE 2.2. `S+FinMetrics` Trellis plotting functions

2.4.2 Plotting "`timeSeries`" Using the `S+FinMetrics` Trellis Plotting Functions

`S+FinMetrics` provides several specialized Trellis-based plotting functions for "`timeSeries`" objects. These functions extend the `S-PLUS` function `TrellisPlot.timeSeries` and are summarized in Table 2.2.

All of the functions in the table can create multi-panel plots with text labels in the panel strips. For the following examples, monthly return data on six stocks from the `S+FinMetrics` "`timeSeries`" `DowJones30` will be used. This data is created using

```
> DJ.ret = getReturns(DowJones30[,1:6], percentage=T)
> colIds(DJ.ret)
[1] "AA"  "AXP" "T"   "BA"  "CAT" "C"
```

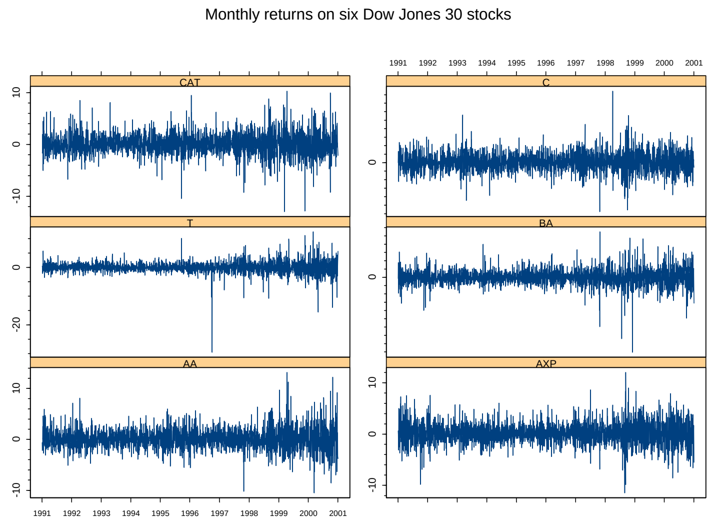


FIGURE 2.5. Multi-panel time plot created using the S+FinMetrics function seriesPlot.

The function seriesPlot may be used to create single panel or multi-panel time plots. To create the multi-panel time plot of the six Dow Jones 30 assets shown in Figure 2.5 use

```
> seriesPlot(DJ.ret,one.plot=F,strip.text=colIds(DJ.ret),
+ main="Monthly returns on six Dow Jones 30 stocks")
```

Notice that each time plot has a different scale.

The function histPlot may be used to create either a single panel histogram of one data series or a multi-panel plot of histograms for multiple series. The multi-panel plot in Figure 2.6 is created using

```
> histPlot(DJ.ret,strip.text=colIds(DJ.ret),
+ main="Histograms of returns on six Dow Jones 30 stocks")
```

Notice that each histogram uses the same bins.

Single panel or multi-panel Trellis-based qq-plots using Gaussian, Student-t, and double exponential distributions may be created using the function qqPlot. To illustrate, consider computing qq-plots for the six Dow Jones 30 assets using six Student-t reference distributions with degrees of freedom equal to 5, 6, 7, 8, 9 and 10. These qq-plots, shown in Figure 2.7, are created using

```
> s.text = paste(colIds(DJ.ret),5:10,sep=" ","df")
```

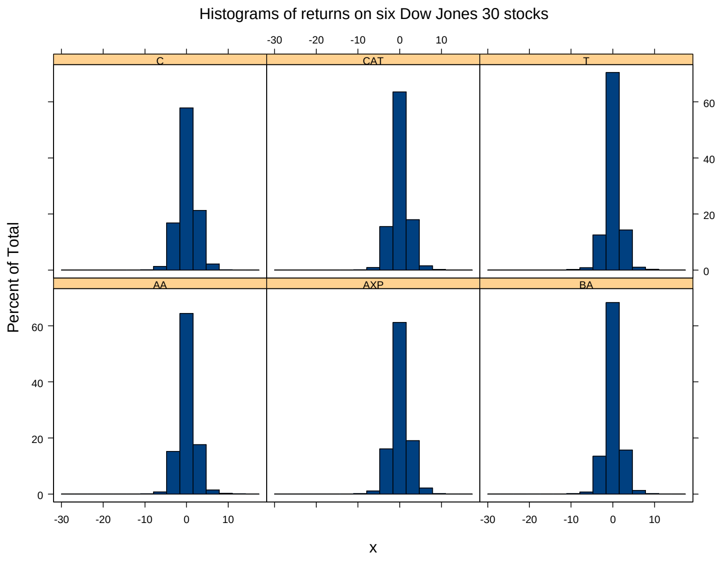


FIGURE 2.6. Multi-panel histogram plot created using the S+FinMetrics function histPlot.

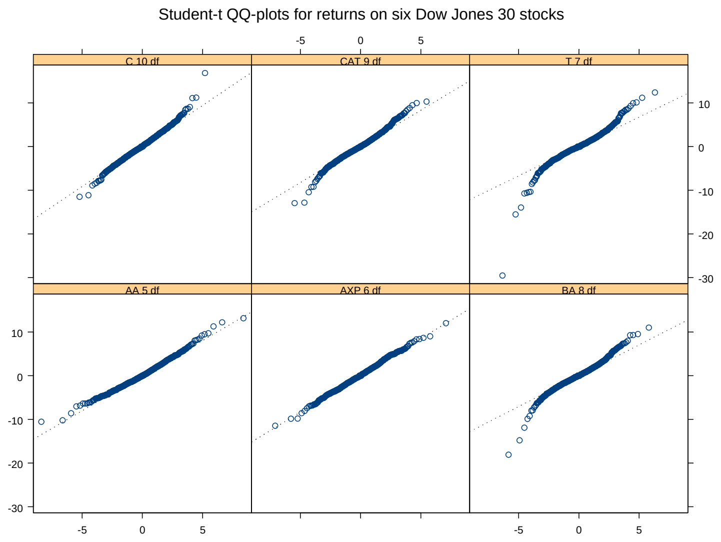


FIGURE 2.7. Multi-panel qq-plots created using the S+FinMetrics function qqPlot.

```
> qqPlot(DJ.ret,strip.text=s.text,
+ distribution="t",dof=c(5,6,7,8,9,10), id.n=FALSE,
+ main="Student-t QQ-plots for returns on six Dow Jones 30 stocks")
```

Notice how the degress of freedom for each Student-t distribution along with the asset name is indicated in the strip text. The optional argument `id.n=FALSE` suppresses the identification of outliers on the qq-plots.

2.5 References

Chow, G., and Lin, A. (1971). "Best Linear Unbiased Interpolation, Distribution, and Extrapolation of Time Series by Related Series," *Review of Economics & Statistics*, 53, 372-375.

3
Time Series Concepts

3.1 Introduction

This chapter provides background material on time series concepts that are used throughout the book. These concepts are presented in an informal way, and extensive examples using `S-PLUS` are used to build intuition. Section 3.2 discusses time series concepts for stationary and ergodic univariate time series. Topics include testing for white noise, linear and autoregressive moving average (ARMA) process, estimation and forecasting from ARMA models, and long-run variance estimation. Section 3.3 introduces univariate nonstationary time series and defines the important concepts of $I(0)$ and $I(1)$ time series. Section 3.4 explains univariate long memory time series. Section 3.5 covers concepts for stationary and ergodic multivariate time series, introduces the class of vector autoregression models, and discusses long-run variance estimation.

Rigorous treatments of the time series concepts presented in this chapter can be found in Fuller (1996) and Hamilton (1994). Applications of these concepts to financial time series are provided by Campbell, Lo, and MacKinlay (1997), Mills (1999), Gourieroux and Jasiak (2001), Tsay (2001), Alexander (2001), and Chan (2002).

3.2 Univariate Time Series

3.2.1 Stationary and Ergodic Time Series

Let $\{y_t\} = \{\ldots y_{t-1}, y_t, y_{t+1}, \ldots\}$ denote a sequence of random variables indexed by some time subscript t. Call such a sequence of random variables a *time series.*

The time series $\{y_t\}$ is *covariance stationary* if

$$\begin{aligned} E[y_t] &= \mu \text{ for all } t \\ \operatorname{cov}(y_t, y_{t-j}) &= E[(y_t-\mu)(y_{t-j}-\mu)] = \gamma_j \text{ for all } t \text{ and any } j \end{aligned}$$

For brevity, call a covariance stationary time series simply a *stationary* time series. Stationary time series have time invariant first and second moments. The parameter γ_j is called the j^{th} order or lag j *autocovariance* of $\{y_t\}$ and a plot of γ_j against j is called the *autocovariance function.* The *autocorrelations* of $\{y_t\}$ are defined by

$$\rho_j = \frac{\operatorname{cov}(y_t, y_{t-j})}{\sqrt{\operatorname{var}(y_t)\operatorname{var}(y_{t-j})}} = \frac{\gamma_j}{\gamma_0}$$

and a plot of ρ_j against j is called the *autocorrelation function* (ACF). Intuitively, a stationary time series is defined by its mean, variance and ACF. A useful result is that any function of a stationary time series is also a stationary time series. So if $\{y_t\}$ is stationary then $\{z_t\} = \{g(y_t)\}$ is stationary for any function $g(\cdot)$.

The lag j *sample autocovariance* and lag j *sample autocorrelation* are defined as

$$\begin{aligned} \hat{\gamma}_j &= \frac{1}{T}\sum_{t=j+1}^{T}(y_t-\bar{y})(y_{t-j}-\bar{y}) & (3.1) \\ \hat{\rho}_j &= \frac{\hat{\gamma}_j}{\hat{\gamma}_0} & (3.2) \end{aligned}$$

where $\bar{y} = \frac{1}{T}\sum_{t=1}^{T} y_t$ is the sample mean. The sample ACF (SACF) is a plot of $\hat{\rho}_j$ against j.

A stationary time series $\{y_t\}$ is *ergodic* if sample moments converge in probability to population moments; i.e. if $\bar{y} \xrightarrow{p} \mu, \hat{\gamma}_j \xrightarrow{p} \gamma_j$ and $\hat{\rho}_j \xrightarrow{p} \rho_j$.

Example 1 *Gaussian white noise (GWN) processes*

Perhaps the most simple stationary time series is the *independent Gaussian white noise* process $y_t \sim$ iid $N(0,\sigma^2) \equiv GWN(0,\sigma^2)$. This process has $\mu = \gamma_j = \rho_j = 0$ $(j \neq 0)$. To simulate a $GWN(0,1)$ process in `S-PLUS` use the `rnorm` function:

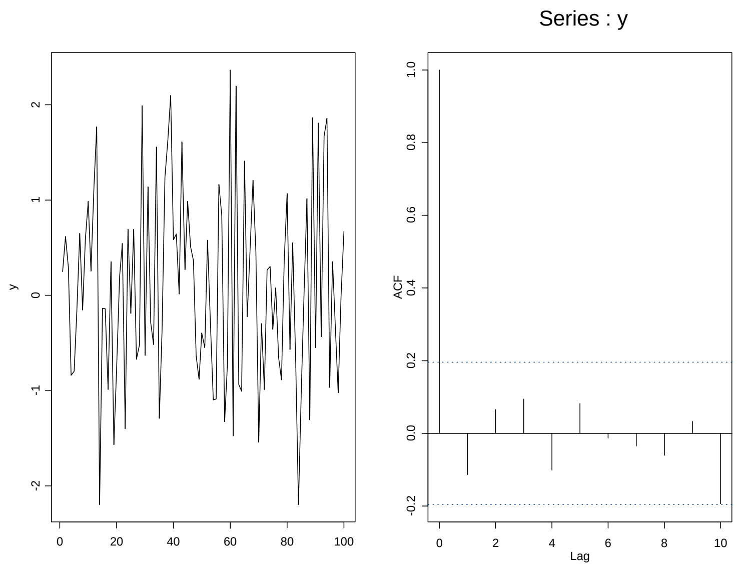


FIGURE 3.1. Simulated Gaussian white noise process and SACF.

```
> set.seed(101)
> y = rnorm(100,sd=1)
```

To compute the sample moments $\bar{y}$, $\hat{\gamma}_j$, $\hat{\rho}_j$ ($j = 1, \ldots, 10$) and plot the data and SACF use

```
> y.bar = mean(y)
> g.hat = acf(y,lag.max=10,type="covariance",plot=F)
> r.hat = acf(y,lag.max=10,type="correlation",plot=F)
> par(mfrow=c(1,2))
> tsplot(y,ylab="y")
> acf.plot(r.hat)
```

By default, as shown in Figure 3.1, the SACF is shown with 95% confidence limits about zero. These limits are based on the result (c.f. Fuller (1996) pg. 336) that if $\{y_t\} \sim$ iid $(0, \sigma^2)$ then

$$\hat{\rho}_j \overset{A}{\sim} N\left(0, \frac{1}{T}\right),\ j > 0.$$

The notation $\hat{\rho}_j \overset{A}{\sim} N\left(0, \frac{1}{T}\right)$ means that the distribution of $\hat{\rho}_j$ is approximated by normal distribution with mean 0 and variance $\frac{1}{T}$ and is based on the central limit theorem result $\sqrt{T}\hat{\rho}_j \xrightarrow{d} N(0,1)$. The 95% limits about zero are then $\pm\frac{1.96}{\sqrt{T}}$.

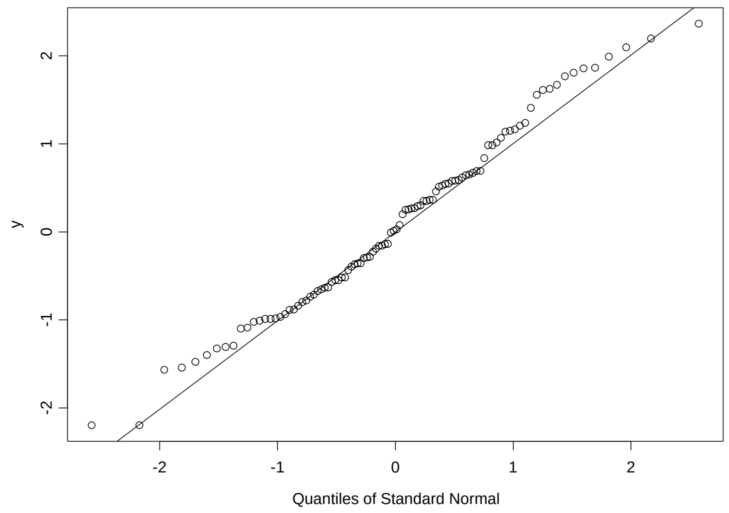


FIGURE 3.2. Normal qq-plot for simulated GWN.

Two slightly more general processes are the independent *white noise* (IWN) process, $y_t \sim IWN(0, \sigma^2)$, and the *white noise* (WN) process, $y_t \sim WN(0, \sigma^2)$. Both processes have mean zero and variance σ^2, but the IWN process has independent increments, whereas the WN process has uncorrelated increments.

Testing for Normality

In the previous example, $y_t \sim GWN(0, 1)$. There are several statistical methods that can be used to see if an iid process y_t is Gaussian. The most common is the normal quantile-quantile plot or *qq-plot*, a scatterplot of the standardized empirical quantiles of y_t against the quantiles of a standard normal random variable. If y_t is normally distributed, then the quantiles will lie on a 45 degree line. A normal qq-plot with 45 degree line for y_t may be computed using the `S-PLUS` functions `qqnorm` and `qqline`

```
> qqnorm(y)
> qqline(y)
```

Figure 3.2 shows the qq-plot for the simulated GWN data of the previous example. The quantiles lie roughly on a straight line. The `S+FinMetrics` function `qqPlot` may be used to create a Trellis graphics qq-plot.

The qq-plot is an informal graphical diagnostic. Two popular formal statistical tests for normality are the *Shapiro-Wilks* test and the *Jarque-*

Bera test. The Shapiro-Wilk's test is a well-known goodness of fit test for the normal distribution. It is attractive because it has a simple, graphical interpretation: one can think of it as an approximate measure of the correlation in a normal quantile-quantile plot of the data. The Jarque-Bera test is based on the result that a normally distributed random variable has skewness equal to zero and kurtosis equal to three. The Jarque-Bera test statistic is

$$\mathrm{JB} = \frac{T}{6}\left(\widehat{\mathrm{skew}}^2 + \frac{(\widehat{\mathrm{kurt}} - 3)^2}{4}\right) \tag{3.3}$$

where $\widehat{\mathrm{skew}}$ denotes the sample skewness and $\widehat{\mathrm{kurt}}$ denotes the sample kurtosis. Under the null hypothesis that the data is normally distributed

$$\mathrm{JB} \overset{A}{\sim} \chi^2(2).$$

Example 2 *Testing for normality using the* ***S+FinMetrics*** *function* ***normalTest***

The Shapiro-Wilks and Jarque-Bera statistics may be computed using the `S+FinMetrics` function `normalTest`. For the simulated GWN data of the previous example, these statistics are

```
> normalTest(y, method="sw")
Test for Normality: Shapiro-Wilks

Null Hypothesis: data is normally distributed

Test Statistics:

Test Stat 0.9703
  p.value 0.1449

Dist. under Null: normal
   Total Observ.: 100

> normalTest(y, method="jb")

Test for Normality: Jarque-Bera

Null Hypothesis: data is normally distributed

Test Statistics:

Test Stat 1.8763
  p.value 0.3914
```

```
Dist. under Null: chi-square with 2 degrees of freedom
   Total Observ.: 100
```

The null of normality is not rejected using either test.

Testing for White Noise

Consider testing the null hypothesis

$$H_0 : y_t \sim WN(0, \sigma^2)$$

against the alternative that y_t is not white noise. Under the null, all of the autocorrelations ρ_j for $j > 0$ are zero. To test this null, Box and Pierce (1970) suggested the *Q-statistic*

$$\mathrm{Q}(k) = T \sum_{j=1}^{k} \hat{\rho}_j^2 \tag{3.4}$$

where $\hat{\rho}_j$ is given by (3.2). Under the null, $\mathrm{Q}(k)$ is asymptotically distributed $\chi^2(k)$. In a finite sample, the Q-statistic (3.4) may not be well approximated by the $\chi^2(k)$. Ljung and Box (1978) suggested the *modified Q-statistic*

$$\mathrm{MQ}(k) = T(T+2) \sum_{j=1}^{k} \frac{\hat{\rho}_j^2}{T-j} \tag{3.5}$$

which is better approximated by the $\chi^2(k)$ in finite samples.

Example 3 *Daily returns on Microsoft*

Consider the time series behavior of daily continuously compounded returns on Microsoft for 2000. The following S-PLUS commands create the data and produce some diagnostic plots:

```
> r.msft = getReturns(DowJones30[,"MSFT"],type="continuous")
> r.msft@title = "Daily returns on Microsoft"
> sample.2000 = (positions(r.msft) > timeDate("12/31/1999")
+ & positions(r.msft) < timeDate("1/1/2001"))
> par(mfrow=c(2,2))
> plot(r.msft[sample.2000],ylab="r.msft")
> r.acf = acf(r.msft[sample.2000])
> hist(seriesData(r.msft))
> qqnorm(seriesData(r.msft))
```

The daily returns on Microsoft resemble a white noise process. The qq-plot, however, suggests that the tails of the return distribution are fatter than the normal distribution. Notice that since the hist and qqnorm functions do not have methods for "timeSeries" objects the extractor function seriesData is required to extract the data frame from the data slot of r.msft.

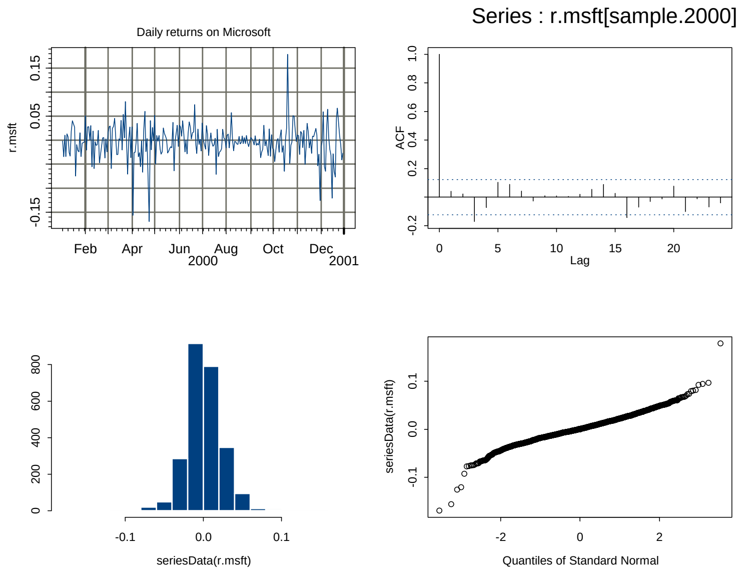


FIGURE 3.3. Daily returns on Microsoft with diagnostic plots.

The `S+FinMetrics` functions `histPlot` and `qqPlot` will produce a histogram and qq-plot for a "`timeSeries`" object using Trellis graphics. For example,

```
> histPlot(r.msft,strip.text="MSFT monthly return")
> qqPlot(r.msft,strip.text="MSFT monthly return")
```

However, Trellis plots cannot be displayed in a multipanel plot created using `par`.

The `S+FinMetrics` function `autocorTest` may be used to compute the Q-statistic and modified Q-statistic to test the null that the returns on Microsoft follow a white noise process:

```
> autocorTest(r.msft, lag.n=10, method="lb")

Test for Autocorrelation: Ljung-Box
Null Hypothesis: no autocorrelation

Test Statistics:

Test Stat 11.7746
  p.value  0.3004

Dist. under Null: chi-square with 10 degrees of freedom
```

```
Total Observ.: 2527
```

The argument `lag.n=10` specifies that $k = 10$ autocorrelations are used in computing the statistic, and `method="lb"` specifies that the modified Box-Pierce statistic (3.5) be computed. To compute the simple Box-Pierce statistic, specify `method="bp"`. The results indicate that the white noise null cannot be rejected.

3.2.2 Linear Processes and ARMA Models

Wold's decomposition theorem (c.f. Fuller (1996) pg. 96) states that any covariance stationary time series $\{y_t\}$ has a *linear process* or infinite order moving average representation of the form

$$\begin{aligned} y_t &= \mu + \sum_{k=0}^{\infty} \psi_k \varepsilon_{t-k} \\ \psi_0 &= 1, \ \sum_{k=0}^{\infty} \psi_k^2 < \infty \\ \varepsilon_t &\sim WN(0, \sigma^2) \end{aligned} \tag{3.6}$$

In the Wold form, it can be shown that

$$\begin{aligned} E[y_t] &= \mu \\ \gamma_0 &= \mathrm{var}(y_t) = \sigma^2 \sum_{k=0}^{\infty} \psi_k^2 \\ \gamma_j &= \mathrm{cov}(y_t, y_{t-j}) = \sigma^2 \sum_{k=0}^{\infty} \psi_k \psi_{k+j} \\ \rho_j &= \frac{\sum_{k=0}^{\infty} \psi_k \psi_{k+j}}{\sum_{k=0}^{\infty} \psi_k^2} \end{aligned}$$

Hence, the pattern of autocorrelations in any stationary and ergodic time series $\{y_t\}$ is determined by the moving average weights $\{\psi_j\}$ in its Wold representation. To ensure convergence of the linear process representation to a stationary and ergodic process with nice properties, it is necessary to further restrict the behavior of the moving average weights $\{\psi_j\}$. A standard assumption used in the econometrics literature (c.f. Hamilton (1994) pg. 504) is *1-summability*

$$\sum_{j=0}^{\infty} j|\psi_j| = 1 + 2|\psi_2| + 3|\psi_3| + \cdots < \infty.$$

The moving average weights in the Wold form are also called *impulse responses* since

$$\frac{\partial y_{t+s}}{\partial \varepsilon_t} = \psi_s, s = 1, 2, \ldots$$

For a stationary and ergodic time series $\lim_{s\to\infty}\psi_s = 0$ and the *long-run cumulative impulse response* $\sum_{s=0}^{\infty}\psi_s < \infty$. A plot of ψ_s against s is called the *impulse response function* (IRF).

The general Wold form of a stationary and ergodic time series is handy for theoretical analysis but is not practically useful for estimation purposes. A very rich and practically useful class of stationary and ergodic processes is the *autoregressive-moving average* (ARMA) class of models made popular by Box and Jenkins (1976). ARMA(p,q) models take the form of a *pth* order stochastic difference equation

$$\begin{aligned} y_t - \mu &= \phi_1(y_{t-1} - \mu) + \cdots + \phi_p(y_{t-p} - \mu) \\ &\quad + \varepsilon_t + \theta_1\varepsilon_{t-1} + \cdots + \theta_q\varepsilon_{t-q} \\ \varepsilon_t &\sim WN(0,\sigma^2) \end{aligned} \tag{3.7}$$

ARMA(p,q) models may be thought of as parsimonious approximations to the general Wold form of a stationary and ergodic time series. More information on the properties of ARMA(p,q) process and the procedures for estimating and forecasting these processes using `S-PLUS` are in the *S-PLUS Guide to Statistics Vol. II*, chapter 27, Venables and Ripley (2002) chapter 13, and Meeker (2001)[1].

Lag Operator Notation

The presentation of time series models is simplified using *lag operator* notation. The lag operator L is defined such that for any time series $\{y_t\}$, $Ly_t = y_{t-1}$. The lag operator has the following properties: $L^2y_t = L\cdot Ly_t = y_{t-2}$, $L^0 = 1$ and $L^{-1}y_t = y_{t+1}$. The operator $\Delta = 1 - L$ creates the first difference of a time series: $\Delta y_t = (1-L)y_t = y_t - y_{t-1}$. The ARMA$(p,q)$ model (3.7) may be compactly expressed using lag polynomials. Define $\phi(L) = 1 - \phi_1 L - \cdots - \phi_p L^p$ and $\theta(L) = 1 + \theta_1 L + \cdots + \theta_q L^q$. Then (3.7) may be expressed as

$$\phi(L)(y_t - \mu) = \theta(L)\varepsilon_t$$

Similarly, the Wold representation in lag operator notation is

$$\begin{aligned} y_t &= \mu + \psi(L)\varepsilon_t \\ \psi(L) &= \sum_{k=0}^{\infty}\psi_k L^k,\ \psi_0 = 1 \end{aligned}$$

and the long-run cumulative impulse response is $\psi(1)$ (i.e. evaluate $\psi(L)$ at $L = 1$). With ARMA(p,q) models the Wold polynomial $\psi(L)$ is approx-

[1]William Meeker also has a library of time series functions for the analysis of ARMA models available for download at
`http://www.public.iastate.edu/~stat451/splusts/splusts.html`.

imated by the ratio of the AR and MA polynomials

$$\psi(L) = \frac{\theta(L)}{\phi(L)}$$

3.2.3 Autoregressive Models

AR(1) Model

A commonly used stationary and ergodic time series in financial modeling is the AR(1) process

$$y_t - \mu = \phi(y_{t-1} - \mu) + \varepsilon_t,\ t = 1, \ldots, T$$

where $\varepsilon_t \sim WN(0, \sigma^2)$ and $|\phi| < 1$. The above representation is called the *mean-adjusted form*. The *characteristic equation* for the AR(1) is

$$\phi(z) = 1 - \phi z = 0 \tag{3.8}$$

so that the root is $z = \frac{1}{\phi}$. Stationarity is satisfied provided the absolute value of the root of the characteristic equation (3.8) is greater than one: $|\frac{1}{\phi}| > 1$ or $|\phi| < 1$. In this case, it is easy to show that $E[y_t] = \mu$, $\gamma_0 = \frac{\sigma^2}{1-\phi^2}$, $\psi_j = \rho_j = \phi^j$ and the Wold representation is

$$y_t = \mu + \sum_{j=0}^{\infty} \rho^j \varepsilon_{t-j}.$$

Notice that for the AR(1) the ACF and IRF are identical. This is not true in general. The long-run cumulative impulse response is $\psi(1) = \frac{1}{1-\phi}$.

The AR(1) model may be re-written in *components form* as

$$\begin{aligned} y_t &= \mu + u_t \\ u_t &= \phi u_{t-1} + \varepsilon_t \end{aligned}$$

or in *autoregression form* as

$$\begin{aligned} y_t &= c + \phi y_{t-1} + \varepsilon_t \\ c &= \mu(1 - \phi) \end{aligned}$$

An AR(1) with $\mu = 1$, $\phi = 0.75$, $\sigma^2 = 1$ and $T = 100$ is easily simulated in S-PLUS using the components form:

```
> set.seed(101)
> e = rnorm(100,sd=1)
> e.start = rnorm(25,sd=1)
> y.ar1 = 1 + arima.sim(model=list(ar=0.75), n=100,
```

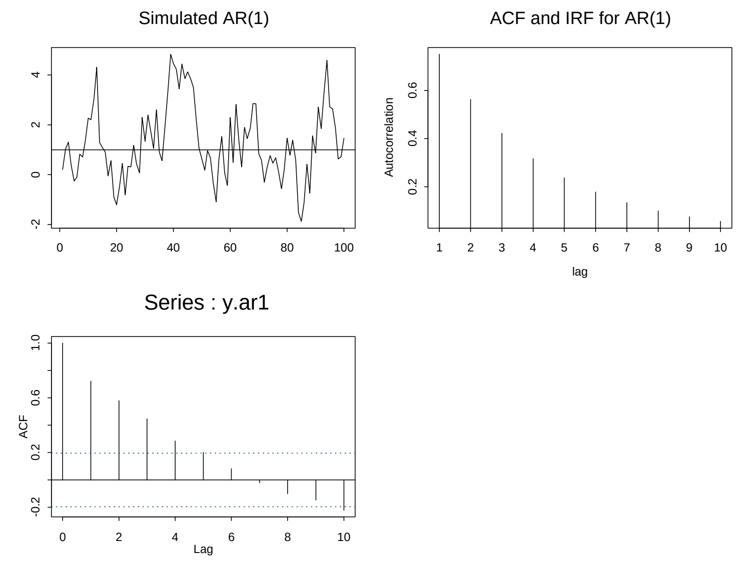


FIGURE 3.4. Simulated AR(1), ACF, IRF and SACF.

```
+ innov=e, start.innov=e.start)
> mean(y.ar1)
[1] 1.271
> var(y.ar1)
[1] 2.201
```

The ACF and IRF may be computed as

```
> gamma.j = rep(0.75,10)^seq(10)
```

The simulated data, ACF and SACF are illustrated in Figure 3.4 using

```
> par(mfrow=c(2,2))
> tsplot(y.ar1,main="Simulated AR(1)")
> abline(h=1)
> tsplot(gamma.j, type="h", main="ACF and IRF for AR(1)",
+ ylab="Autocorrelation", xlab="lag")
> tmp = acf(y.ar1, lag.max=10)
```

Notice that $\{y_t\}$ exhibits *mean-reverting* behavior. That is, $\{y_t\}$ fluctuates about the mean value $\mu = 1$. The ACF and IRF decay at a geometric rate. The decay rate of the IRF is sometimes reported as a *half-life* – the lag j^{half} at which the IRF reaches $\frac{1}{2}$. For the AR(1) with positive ϕ, it can be shown that $j^{\text{half}} = \ln(0.5)/\ln(\phi)$. For $\phi = 0.75$, the half-life is

```
> log(0.5)/log(0.75)
```

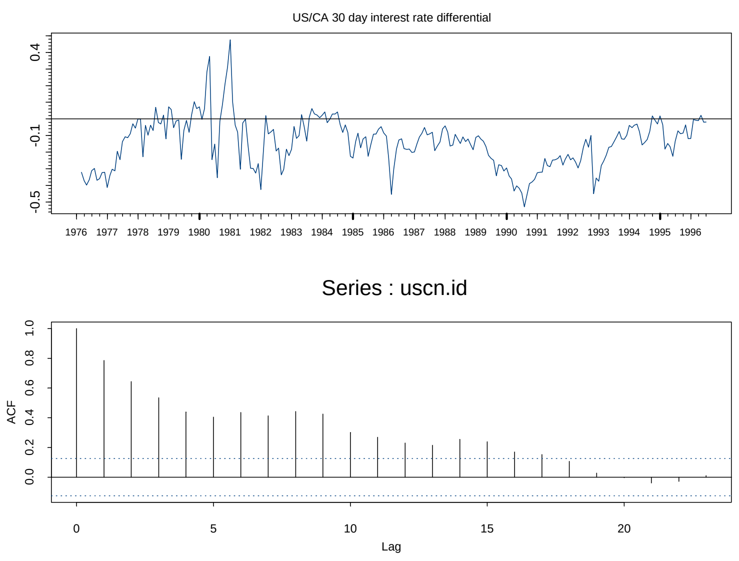


FIGURE 3.5. US/CA 30 day interest rate differential and SACF.

```
[1] 2.409
```

Many economic and financial time series are well characterized by an AR(1) process. Leading examples in finance are valuation ratios (dividend-price ratio, price-earning ratio etc), real exchange rates, interest rates, and interest rate differentials (spreads). To illustrate, consider the 30-day US/CA interest rate differential[2] constructed from the `S+FinMetrics` "`timeSeries`" object `lexrates.dat`:

```
> uscn.id = 100*(lexrates.dat[,"USCNF"]-
+ lexrates.dat[,"USCNS"])
> colIds(uscn.id) = "USCNID"
> uscn.id@title = "US/CA 30 day interest rate differential"
> par(mfrow=c(2,1))
> plot(uscn.id,reference.grid=F)
> abline(h=0)
> tmp = acf(uscn.id)
```

The interest rate differential is clearly persistent: autocorrelations are significant at the 5% level up to 15 months.

[2]By covered interest rate parity, the nominal interest rate differential between risk free bonds from two countries is equal to the difference between the nominal forward and spot exchange rates.

AR(p) Models

The AR(p) model in mean-adjusted form is

$$y_t - \mu = \phi_1(y_{t-1} - \mu) + \cdots + \phi_p(y_{t-p} - \mu) + \varepsilon_t$$

or, in lag operator notation,

$$\phi(L)(y_t - \mu) = \varepsilon_t$$

where $\phi(L) = 1 - \phi_1 L - \cdots - \phi_p L^p$. The autoregressive form is

$$\phi(L)y_t = c + \varepsilon_t.$$

It can be shown that the AR(p) is stationary and ergodic provided the roots of the *characteristic equation*

$$\phi(z) = 1 - \phi_1 z - \phi_2 z^2 - \cdots - \phi_p z^p = 0 \tag{3.9}$$

lie outside the complex unit circle (have modulus greater than one). A necessary condition for stationarity that is useful in practice is that $|\phi_1 + \cdots + \phi_p| < 1$. If (3.9) has complex roots then y_t will exhibit sinusoidal behavior. In the stationary AR(p), the constant in the autoregressive form is equal to $\mu(1 - \phi_1 - \cdots - \phi_p)$.

The moments of the AR(p) process satisfy the *Yule-Walker equations*

$$\begin{aligned} \gamma_0 &= \phi_1\gamma_1 + \phi_2\gamma_2 + \cdots + \phi_p\gamma_p + \sigma^2 \\ \gamma_j &= \phi_1\gamma_{j-1} + \phi_2\gamma_{j-2} + \cdots + \phi_p\gamma_{j-p} \end{aligned} \tag{3.10}$$

A simple recursive algorithm for finding the Wold representation is based on matching coefficients in $\phi(L)$ and $\psi(L)$ such that $\phi(L)\psi(L) = 1$. For example, in the AR(2) model

$$(1 - \phi_1 L - \phi_2 L^2)(1 + \psi_1 L + \psi_2 L^2 + \cdots) = 1$$

implies

$$\begin{aligned} \psi_1 &= 1 \\ \psi_2 &= \phi_1\psi_1 + \phi_2 \\ \psi_3 &= \phi_1\psi_2 + \phi_2\psi_1 \\ &\vdots \\ \psi_j &= \phi_1\psi_{j-1} + \phi_2\psi_{j-2} \end{aligned}$$

Partial Autocorrelation Function

The *partial autocorrelation function* (PACF) is a useful tool to help identify AR(p) models. The PACF is based on estimating the sequence of AR

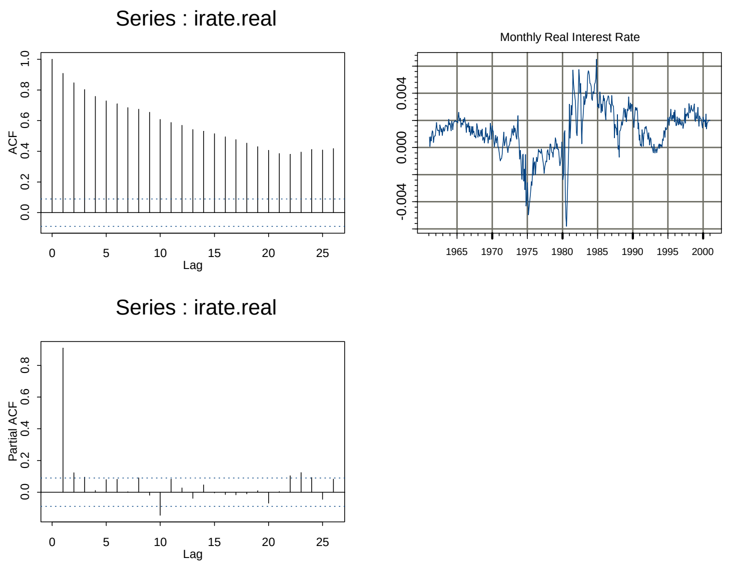


FIGURE 3.6. Monthly U.S. real interest rate, SACF and SPACF.

models

$$\begin{aligned}
z_t &= \phi_{11} z_{t-1} + \varepsilon_{1t} \\
z_t &= \phi_{21} z_{t-1} + \phi_{22} z_{t-2} + \varepsilon_{2t} \\
&\vdots \\
z_t &= \phi_{p1} z_{t-1} + \phi_{p2} z_{t-2} + \cdots + \phi_{pp} z_{t-p} + \varepsilon_{pt}
\end{aligned}$$

where $z_t = y_t - \mu$ is the demeaned data. The coefficients ϕ_{jj} for $j = 1, \ldots, p$ (i.e., the last coefficients in each AR(p) model) are called the partial autocorrelation coefficients. In an AR(1) model the first partial autocorrelation coefficient ϕ_{11} is non-zero, and the remaining partial autocorrelation coefficients ϕ_{jj} for $j > 1$ are equal to zero. Similarly, in an AR(2), the first and second partial autocorrelation coefficients ϕ_{11} and ϕ_{22} are non-zero and the rest are zero for $j > 2$. For an AR(p) all of the first p partial autocorrelation coefficients are non-zero, and the rest are zero for $j > p$. The sample partial autocorrelation coefficients up to lag p are essentially obtained by estimating the above sequence of p AR models by least squares and retaining the estimated coefficients $\hat{\phi}_{jj}$.

Example 4 *Monthly real interest rates*

The "timeSeries" object varex.ts in the S+FinMetrics module contains monthly data on real stock returns, real interest rates, inflation and real output growth.

```
> colIds(varex.ts)
[1] "MARKET.REAL" "RF.REAL"     "INF"         "IPG"
```

Figure 3.6 shows the real interest rate, RF.REAL, over the period January 1961 through December 2000 produced with the S-PLUS commands

```
> smpl = (positions(varex.ts) > timeDate("12/31/1960"))
> irate.real = varex.ts[smpl,"RF.REAL"]
> par(mfrow=c(2,2))
> acf.plot(acf(irate.real, plot=F))
> plot(irate.real, main="Monthly Real Interest Rate")
> tmp = acf(irate.real, type="partial")
```

The SACF and SPACF indicate that the real interest rate might be modeled as an AR(2) or AR(3) process.

3.2.4 Moving Average Models

MA(1) Model

The MA(1) model has the form

$$y_t = \mu + \varepsilon_t + \theta\varepsilon_{t-1},\ \varepsilon_t \sim WN(0, \sigma^2)$$

For any finite θ the MA(1) is stationary and ergodic. The moments are $E[y_t] = \mu$, $\gamma_0 = \sigma^2(1+\theta^2)$, $\gamma_1 = \sigma^2\theta$, $\gamma_j = 0$ for $j > 1$ and $\rho_1 = \theta/(1+\theta^2)$. Hence, the ACF of an MA(1) process cuts off at lag one, and the maximum value of this correlation is ± 0.5.

There is an identification problem with the MA(1) model since $\theta = 1/\theta$ produce the same value of ρ_1. The MA(1) is called *invertible* if $|\theta| < 1$ and is called *non-invertible* if $|\theta| \geq 1$. In the invertible MA(1), the error term ε_t has an infinite order AR representation of the form

$$\varepsilon_t = \sum_{j=0}^{\infty} \theta^{*j}(y_{t-j} - \mu)$$

where $\theta^* = -\theta$ so that ε_t may be thought of as a prediction error based on past values of y_t. A consequence of the above result is that the PACF for an invertible MA(1) process decays towards zero at an exponential rate.

Example 5 *Signal plus noise model*

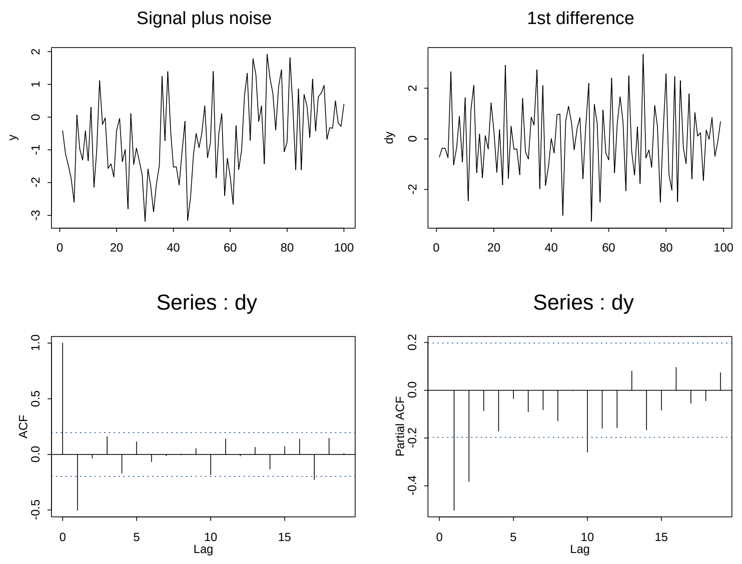


FIGURE 3.7. Simulated data, SACF and SPACF from signal plus noise model.

MA(1) models often arise through data transformations like aggregation and differencing[3]. For example, consider the signal plus noise model

$$\begin{aligned} y_t &= z_t + \varepsilon_t,\ \varepsilon_t \sim WN(0, \sigma_\varepsilon^2) \\ z_t &= z_{t-1} + \eta_t,\ \eta_t \sim WN(0, \sigma_\eta^2) \end{aligned}$$

where ε_t and η_t are independent. For example, z_t could represent the fundamental value of an asset price and ε_t could represent an iid deviation about the fundamental price. A stationary representation requires differencing y_t:

$$\Delta y_t = \eta_t + \varepsilon_t - \varepsilon_{t-1}$$

It can be shown, e.g. Harvey (1993), that Δy_t is an MA(1) process with $\theta = \frac{-(q+2)+\sqrt{q^2+4q}}{2}$ where $q = \frac{\sigma_\eta^2}{\sigma_\varepsilon^2}$ is the signal-to-noise ratio and $\rho_1 = \frac{-1}{q+2} < 0$.

Simulated data with $\sigma_\varepsilon^2 = 1$ and $\sigma_\eta^2 = (0.5)^2$ created with the S-PLUS commands

```
> set.seed(112)
> eps = rnorm(100,sd=1)
> eta = rnorm(100,sd=0.5)
```

[3]MA(1) type models for asset returns often occur as the result of no-trading effects or bid-ask bounce effects. See Campbell, Lo and MacKinlay (1997) chapter 3 for details.

```
> z = cumsum(eta)
> y = z + eps
> dy = diff(y)
> par(mfrow=c(2,2))
> tsplot(y, main="Signal plus noise",ylab="y")
> tsplot(dy, main="1st difference",ylab="dy")
> tmp = acf(dy)
> tmp = acf(dy,type="partial")
```

are illustrated in Figure 3.7. The signal-to-noise ratio $q = 0.25$ implies a first lag autocorrelation of $\rho_1 = -0.444$. This negative correlation is clearly reflected in the SACF.

MA(q) Model

The MA(q) model has the form

$$y_t = \mu + \varepsilon_t + \theta_1\varepsilon_{t-1} + \cdots + \theta_q\varepsilon_{t-q}, \text{ where } \varepsilon_t \sim WN(0, \sigma^2)$$

The MA(q) model is stationary and ergodic provided $\theta_1, \ldots, \theta_q$ are finite. It is *invertible* if all of the roots of the MA characteristic polynomial

$$\theta(z) = 1 + \theta_1 z + \cdots \theta_q z^q = 0 \tag{3.11}$$

lie outside the complex unit circle. The moments of the MA(q) are

$$\begin{aligned} E[y_t] &= \mu \\ \gamma_0 &= \sigma^2(1 + \theta_1^2 + \cdots + \theta_q^2) \\ \gamma_j &= \begin{cases} (\theta_j + \theta_{j+1}\theta_1 + \theta_{j+2}\theta_2 + \cdots + \theta_q\theta_{q-j})\,\sigma^2 \text{ for } j = 1, 2, \ldots, q \\ 0 \text{ for } j > q \end{cases} \end{aligned}$$

Hence, the ACF of an MA(q) is non-zero up to lag q and is zero afterwards. As with the MA(1), the PACF for an invertible MA(q) will show exponential decay and possibly pseudo cyclical behavior if the roots of (3.11) are complex.

Example 6 *Overlapping returns and MA(q) models*

MA(q) models often arise in finance through data aggregation transformations. For example, let $R_t = \ln(P_t/P_{t-1})$ denote the monthly continuously compounded return on an asset with price P_t. Define the annual return at time t using monthly returns as $R_t(12) = \ln(P_t/P_{t-12}) = \sum_{j=0}^{11} R_{t-j}$. Suppose $R_t \sim WN(\mu, \sigma^2)$ and consider a sample of monthly returns of size T, $\{R_1, R_2, \ldots, R_T\}$. A sample of annual returns may be created using *overlapping* or *non-overlapping* returns. Let $\{R_{12}(12), R_{13}(12), \ldots, R_T(12)\}$ denote a sample of $T^* = T - 11$ monthly overlapping annual returns and $\{R_{12}(12), R_{24}(12), \ldots, R_T(12)\}$ denote a sample of $T/12$ non-overlapping annual returns. Researchers often use overlapping returns in analysis due to the apparent larger sample size. One must be careful using overlapping returns because the monthly annual return sequence $\{R_t(12)\}$ is not a white noise process even if the monthly return sequence $\{R_t\}$ is. To see this, straightforward calculations give

$$\begin{aligned} E[R_t(12)] &= 12\mu \\ \gamma_0 &= \mathrm{var}(R_t(12)) = 12\sigma^2 \\ \gamma_j &= \mathrm{cov}(R_t(12), R_{t-j}(12)) = (12-j)\sigma^2 \text{ for } j < 12 \\ \gamma_j &= 0 \text{ for } j \geq 12 \end{aligned}$$

Since $\gamma_j = 0$ for $j \geq 12$ notice that $\{R_t(12)\}$ behaves like an MA(11) process

$$\begin{aligned} R_t(12) &= 12\mu + \varepsilon_t + \theta_1\varepsilon_{t-1} + \cdots + \theta_{11}\varepsilon_{t-11} \\ \varepsilon_t &\sim WN(0, \sigma^2) \end{aligned}$$

To illustrate, consider creating annual overlapping continuously compounded returns on the S&P 500 index over the period February 1990 through January 2001. The `S+FinMetrics` "`timeSeries`" `singleIndex.dat` contains the S&P 500 price data and the continuously compounded monthly returns are computed using the `S+FinMetrics` function `getReturns`

```
> sp500.mret = getReturns(singleIndex.dat[,"SP500"],
+ type="continuous")
> sp500.mret@title = "Monthly returns on S&P 500 Index"
```

The monthly overlapping annual returns are easily computed using the `S-PLUS` function `aggregateSeries`

```
> sp500.aret = aggregateSeries(sp500.mret,moving=12,FUN=sum)
> sp500.aret@title = "Monthly Annual returns on S&P 500 Index"
```

The optional argument `moving=12` specifies that the `sum` function is to be applied to moving blocks of size 12. The data together with the SACF and SPACF of the monthly annual returns are displayed in Figure 3.8.

The SACF has non-zero values up to lag 11. Interestingly, the SPACF is very small at all lags except the first.

3.2.5 ARMA(p,q) Models

The general ARMA(p, q) model in mean-adjusted form is given by (3.7). The regression formulation is

$$y_t = c + \phi_1 y_{t-1} + \cdots + \phi_p y_{t-p} + \varepsilon_t + \theta\varepsilon_{t-1} + \cdots + \theta\varepsilon_{t-q} \tag{3.12}$$

It is stationary and ergodic if the roots of the characteristic equation $\phi(z) = 0$ lie outside the complex unit circle, and it is invertible if the roots of the

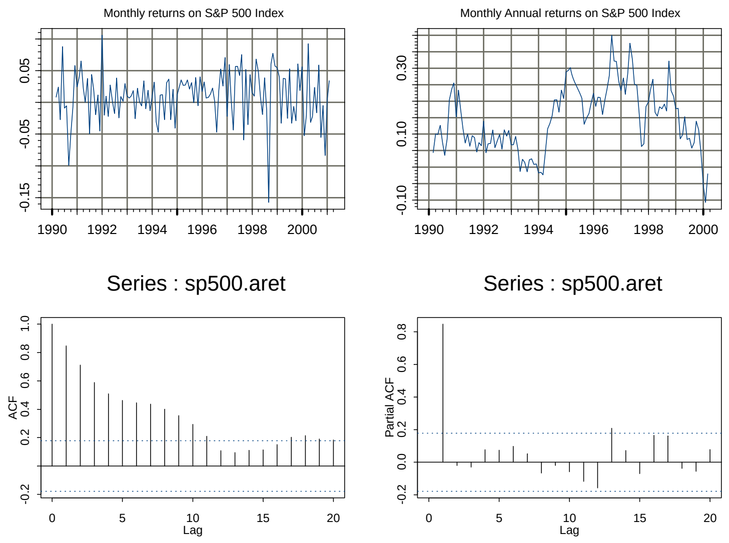


FIGURE 3.8. Monthly non-overlapping and overlapping annual returns on the S&P 500 index.

MA characteristic polynomial $\theta(z) = 0$ lie outside the unit circle. It is assumed that the polynomials $\phi(z) = 0$ and $\theta(z) = 0$ do not have canceling or common factors. A stationary and ergodic ARMA(p, q) process has a mean equal to

$$\mu = \frac{c}{1 - \phi_1 - \cdots - \phi_p} \tag{3.13}$$

and its autocovariances, autocorrelations and impulse response weights satisfy the recursive relationships

$$\begin{aligned}
\gamma_j &= \phi_1\gamma_{j-1} + \phi_2\gamma_{j-2} + \cdots + \phi_p\gamma_{j-p} \\
\rho_j &= \phi_1\rho_{j-1} + \phi_2\rho_{j-2} + \cdots + \phi_p\rho_{j-p} \\
\psi_j &= \phi_1\psi_{j-1} + \phi_2\psi_{j-2} + \cdots + \phi_p\psi_{j-p}
\end{aligned}$$

The general form of the ACF for an ARMA(p, q) process is complicated. See Hamilton (1994) chapter five for details. In general, for an ARMA(p, q) process, the ACF behaves like the ACF for an AR(p) process for $p > q$, and the PACF behaves like the PACF for an MA(q) process for $q > p$. Hence, both the ACF and PACF eventually show exponential decay.

ARMA(p, q) models often arise from certain aggregation transformations of simple time series models. An important result due to Granger and Morris (1976) is that if y_{1t} is an ARMA(p_1, q_1) process and y_{2t} is an ARMA(p_2, q_2) process, which may be contemporaneously correlated with y_{1t}, then $y_{1t} + y_{2t}$ is an ARMA(p, q) process with $p = p_1 + p_2$ and $q = \max(p_1 + q_2, q_1 + p_2)$. For example, if y_{1t} is an AR(1) process and y_2 is a AR(1) process, then $y_1 + y_2$ is an ARMA(2,1) process.

High order ARMA(p, q) processes are difficult to identify and estimate in practice and are rarely used in the analysis of financial data. Low order ARMA(p, q) models with p and q less than three are generally sufficient for the analysis of financial data.

ARIMA(p, d, q) Models

The specification of the ARMA(p, q) model (3.7) assumes that y_t is stationary and ergodic. If y_t is a trending variable like an asset price or a macroeconomic aggregate like real GDP, then y_t must be transformed to stationary form by eliminating the trend. Box and Jenkins (1976) advocate removal of trends by differencing. Let $\Delta = 1 - L$ denote the *difference operator*. If there is a linear trend in y_t then the first difference $\Delta y_t = y_t - y_{t-1}$ will not have a trend. If there is a quadratic trend in y_t, then Δy_t will contain a linear trend but the second difference $\Delta^2 y_t = (1 - 2L + L^2)y_t = y_t - 2y_{t-1} + y_{t-2}$ will not have a trend. The class of ARMA(p, q) models where the trends have been transformed by differencing d times is denoted ARIMA(p, d, q)[4].

3.2.6 Estimation of ARMA Models and Forecasting

ARMA(p, q) models are generally estimated using the technique of maximum likelihood, which is usually accomplished by putting the ARMA(p, q) in state-space form from which the prediction error decomposition of the log-likelihood function may be constructed. Details of this process are given in Harvey (1993). An often ignored aspect of the maximum likelihood estimation of ARMA(p, q) models is the treatment of initial values. These initial values are the first p values of y_t and q values of ε_t in (3.7). The *exact likelihood* utilizes the stationary distribution of the initial values in the construction of the likelihood. The *conditional likelihood* treats the p initial values of y_t as fixed and often sets the q initial values of ε_t to zero. The exact maximum likelihood estimates (MLEs) maximize the exact log-likelihood, and the conditional MLEs maximize the conditional log-likelihood. The exact and conditional MLEs are asymptotically equivalent but can differ substantially in small samples, especially for models that are close to being nonstationary or noninvertible.[5]

[4]More general ARIMA(p, d, q) models allowing for seasonality are discussed in chapter 27 of the *S-PLUS Guide to Statistics, Vol. II.*

[5]As pointed out by Venables and Ripley (1999) page 415, the maximum likelihood estimates computed using the `S-PLUS` function `arima.mle` are conditional MLEs. Exact MLEs may be easily computed using the `S+FinMetrics` state space modeling functions.

For pure AR models, the conditional MLEs are equivalent to the least squares estimates from the model

$$y_t = c + \phi_1 y_{t-1} + \cdots + \phi_p y_{t-p} + \varepsilon_t \tag{3.14}$$

Notice, however, that c in (3.14) is not an estimate of $E[y_t] = \mu$. The least squares estimate of μ is given by plugging in the least squares estimates of $c, \phi_1, \ldots, \phi_p$ into (3.13).

Model Selection Criteria

Before an ARMA(p, q) may be estimated for a time series y_t, the AR and MA orders p and q must be determined by visually inspecting the SACF and SPACF for y_t. Alternatively, statistical *model selection criteria* may be used. The idea is to fit all ARMA(p, q) models with orders $p \leq p_{\max}$ and $q \leq q_{\max}$ and choose the values of p and q which minimizes some model selection criteria. Model selection criteria for ARMA(p, q) models have the form

$$\mathrm{MSC}(p, q) = \ln(\tilde{\sigma}^2(p, q)) + c_T \cdot \varphi(p, q)$$

where $\tilde{\sigma}^2(p, q)$ is the MLE of $\mathrm{var}(\varepsilon_t) = \sigma^2$ *without a degrees of freedom correction* from the ARMA(p, q) model, c_T is a sequence indexed by the sample size T, and $\varphi(p, q)$ is a penalty function which penalizes large ARMA(p, q) models. The two most common information criteria are the Akaike (AIC) and Schwarz-Bayesian (BIC):

$$\begin{aligned} \mathrm{AIC}(p, q) &= \ln(\tilde{\sigma}^2(p, q)) + \frac{2}{T}(p + q) \\ \mathrm{BIC}(p, q) &= \ln(\tilde{\sigma}^2(p, q)) + \frac{\ln T}{T}(p + q) \end{aligned}$$

The AIC criterion asymptotically overestimates the order with positive probability, whereas the BIC estimate the order consistently under fairly general conditions if the true orders p and q are less than or equal to $p_{\max}$ and $q_{\max}$. However, in finite samples the BIC generally shares no particular advantage over the AIC.

Forecasting Algorithm

Forecasts from an ARIMA(p, d, q) model are straightforward. The model is put in state space form, and optimal h-step ahead forecasts along with forecast standard errors (not adjusted for parameter uncertainty) are produced using the Kalman filter algorithm. Details of the method are given in Harvey (1993).

Estimation and Forecasting ARIMA(p, d, q) Models Using the S-PLUS Function arima.mle

Conditional MLEs may be computed using the S-PLUS function `arima.mle`. The form of the ARIMA(p, d, q) assumed by `arima.mle` is

$$\begin{aligned} y_t &= \phi_1 y_{t-1} + \cdots + \phi_p y_{t-p} \\ &\quad + \varepsilon_t - \theta_1 \varepsilon_{t-1} - \cdots - \theta_q \varepsilon_{t-q} \\ &\quad + \boldsymbol{\beta}' \mathbf{x}_t \end{aligned}$$

where $\mathbf{x}_t$ represents additional explanatory variables. It is assumed that y_t has been differenced d times to remove any trends and that the unconditional mean μ has been subtracted out so that y_t is demeaned. Notice that `arima.mle` assumes that the signs on the MA coefficients θ_j are the opposite to those in (3.7).

The arguments expected by `arima.mle` are

```
> args(arima.mle)
function(x, model = NULL, n.cond = 0, xreg = NULL, ...)
```

where `x` is a univariate "`timeSeries`" or vector, `model` is a list object describing the specification of the ARMA model, `n.cond` sets the number of initial observations on which to condition in the formation of the log-likelihood, and `xreg` is a "`timeSeries`", vector or matrix of additional explanatory variables. By default, `arima.mle` assumes that the ARIMA(p, d, q) model is stationary and in mean-adjusted form with an estimate of μ subtracted from the observed data y_t. To estimate the regression form (3.12) of the ARIMA(p, q) model, simply set `xreg=1`. ARIMA(p, d, q) models are specified using list variables the form

```
> mod.list = list(order=c(1,0,1))
> mod.list = list(order=c(1,0,1),ar=0.75,ma=0)
> mod.list = list(ar=c(0.75,-0.25),ma=c(0,0))
```

The first list simply specifies an ARMA(1,0,1)/ARMA(1,1) model. The second list specifies an ARIMA(1,0,1) as well as starting values for the AR and MA parameters ϕ and θ. The third list implicitly determines an ARMA(2,2) model by giving the starting values for the AR and MA parameters. The function `arima.mle` produces an object of class "`arima`" for which there are `print` and `plot` methods. Diagnostics from the fit can be created with the S-PLUS function `arima.diag`, and forecasts may be produced using `arima.forecast`.

Example 7 *Estimation of ARMA model for US/CA interest rate differential*

Consider estimating an ARMA(p, q) for the monthly US/CA interest rate differential data in the "`timeSeries`" `uscn.id` used in a previous example. To estimate an ARMA(1,1) model for the demeaned interest rate differential with starting values $\phi = 0.75$ and $\theta = 0$ use

```
> uscn.id.dm = uscn.id - mean(uscn.id)
> arma11.mod = list(ar=0.75,ma=0)
> arma11.fit = arima.mle(uscn.id.dm,model=arma11.mod)
> class(arma11.fit)
[1] "arima"
```

The components of `arma11.fit` are

```
> names(arma11.fit)
 [1] "model"     "var.coef"  "method"    "series"
 [5] "aic"       "loglik"    "sigma2"    "n.used"
 [9] "n.cond"    "converged" "conv.type" "call"
```

To see the basic fit simply type

```
> arma11.fit
Call: arima.mle(x = uscn.id.dm, model = arma11.mod)
Method:  Maximum Likelihood
Model :  1 0 1

Coefficients:
     AR : 0.82913
     MA : 0.11008

Variance-Covariance Matrix:
         ar(1)    ma(1)
ar(1) 0.002046 0.002224
ma(1) 0.002224 0.006467

Optimizer has  converged
Convergence Type: relative function convergence
AIC: -476.25563
```

The conditional MLEs are $\hat{\phi}_{cmle} = 0.829$ and $\hat{\theta}_{cmle} = -0.110$. Standard errors for these parameters are given by the square roots of the diagonal elements of variance-covariance matrix

```
> std.errs = sqrt(diag(arma11.fit$var.coef))
> names(std.errs) = colIds(arma11.fit$var.coef)
> std.errs
   ar(1)   ma(1)
 0.04523 0.08041
```

It appears that the $\hat{\theta}_{cmle}$ is not statistically different from zero.

To estimate the ARMA(1,1) for the interest rate differential data in regression form (3.12) with an intercept use

```
> arma11.fit2 = arima.mle(uscn.id,model=arma11.mod,xreg=1)
> arma11.fit2
Call: arima.mle(x = uscn.id, model = arma11.mod, xreg = 1)
Method:  Maximum Likelihood
Model :  1 0 1

Coefficients:
     AR : 0.82934
     MA : 0.11065

Variance-Covariance Matrix:
         ar(1)    ma(1)
ar(1) 0.002043 0.002222
ma(1) 0.002222 0.006465
Coeffficients for regressor(s): intercept
[1] -0.1347

Optimizer has  converged
Convergence Type: relative function convergence
AIC: -474.30852
```

The conditional MLEs for ϕ and θ are essentially the same as before, and the MLE for c is $\hat{c}_{cmle} = -0.1347$. Notice that the reported variance-covariance matrix only gives values for the estimated ARMA coefficients $\hat{\phi}_{cmle}$ and $\hat{\theta}_{cmle}$.

Graphical diagnostics of the fit produced using the `plot` method

```
> plot(arma11.fit)
```

are illustrated in Figure 3.9. There appears to be some high order serial correlation in the errors as well as heteroskedasticity.

The h-step ahead forecasts of future values may be produced with the S-PLUS function `arima.forecast`. For example, to produce monthly forecasts for the demeaned interest rate differential from July 1996 through June 1997 use

```
> fcst.dates = timeSeq("7/1/1996", "6/1/1997",
+ by="months", format="%b %Y")
> uscn.id.dm.fcst = arima.forecast(uscn.id.dm, n=12,
+ model=arma11.fit$model, future.positions=fcst.dates)
> names(uscn.id.dm.fcst)
[1] "mean"    "std.err"
```

The object `uscn.id.dm.fcst` is a list whose first component is a "`timeSeries`" containing the h-step forecasts, and the second component is a "`timeSeries`" containing the forecast standard errors:

```
> uscn.id.dm.fcst[[1]]
```

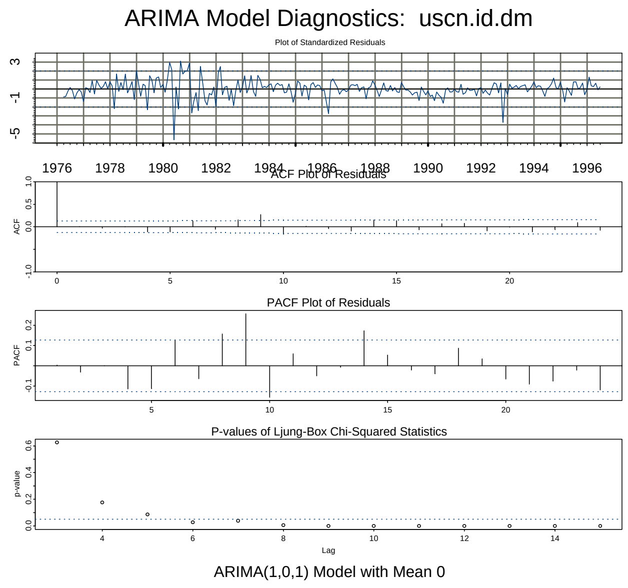


FIGURE 3.9. Residual diagnostics from ARMA(1,1) fit to US/CA interest rate differentials.

```
Positions       1
Jul 1996  0.09973
Aug 1996  0.08269
Sep 1996  0.06856
Oct 1996  0.05684
Nov 1996  0.04713
Dec 1996  0.03908
Jan 1997  0.03240
Feb 1997  0.02686
Mar 1997  0.02227
Apr 1997  0.01847
May 1997  0.01531
Jun 1997  0.01270
```

The data, forecasts and 95% forecast confidence intervals shown in Figure 3.10 are produced by

```
> smpl = positions(uscn.id.dm) >= timeDate("6/1/1995")
> plot(uscn.id.dm[smpl,],uscn.id.dm.fcst$mean,
+ uscn.id.dm.fcst$mean+2*uscn.id.dm.fcst$std.err,
+ uscn.id.dm.fcst$mean-2*uscn.id.dm.fcst$std.err,
+ plot.args=list(lty=c(1,4,3,3)))
```

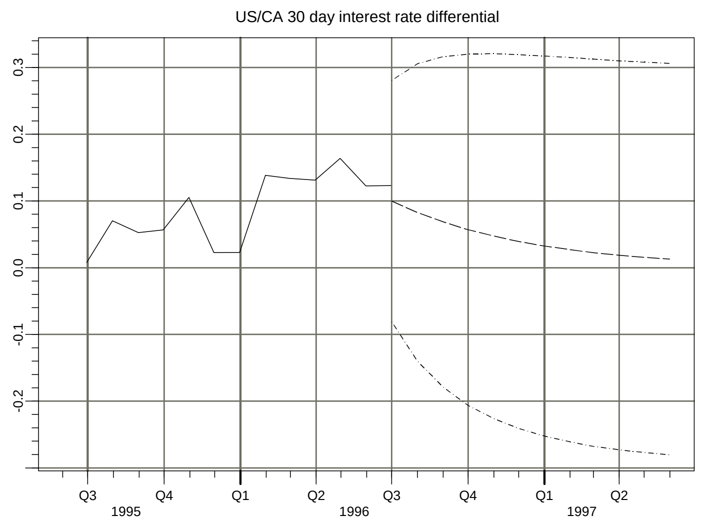


FIGURE 3.10. Forecasts for 12 months for the series `uscn.id.dm`.

Estimating AR(p) by Least Squares Using the `S+FinMetrics` Function `OLS`

As previously mentioned, the conditional MLEs for an AR(p) model may be computed using least squares. The `S+FinMetrics` function `OLS`, which extends the `S-PLUS` function `lm` to handle general time series regression, may be used to estimate an AR(p) in a particularly convenient way. The general use of OLS is discussed in Chapter 6, and its use for estimating an AR(p) is only mentioned here. For example, to estimate an AR(2) model for the US/CA interest rate differential use

```
> ar2.fit = OLS(USCNID~ar(2), data=uscn.id)
> ar2.fit

Call:
OLS(formula = USCNID ~ar(2), data = uscn.id)

Coefficients:
 (Intercept)    lag1    lag2
 -0.0265      0.7259  0.0758

Degrees of freedom: 243 total; 240 residual
Time period: from Apr 1976 to Jun 1996
Residual standard error: 0.09105
```

The least squares estimates of the AR coefficients are $\hat{\phi}_1 = 0.7259$ and $\hat{\phi}_2 = 0.0758$. Since $\hat{\phi}_1 + \hat{\phi}_2 < 1$ the estimated AR(2) model is stationary. To be sure, the roots of $\phi(z) = 1 - \hat{\phi}_1 z - \hat{\phi}_2 z^2 = 0$ are

```
> abs(polyroot(c(1,-ar2.fit$coef[2:3])))
[1]  1.222 10.798
```

are outside the complex unit circle.

3.2.7 Martingales and Martingale Difference Sequences

Let $\{y_t\}$ denote a sequence of random variables and let $I_t = \{y_{t,}y_{t-1}, \ldots\}$ denote a set of conditioning information or *information set* based on the past history of y_t. The sequence $\{y_t, I_t\}$ is called a *martingale* if

- $I_{t-1} \subset I_t$ (I_t is a filtration)
- $E[|y_t|] < \infty$
- $E[y_t|I_{t-1}] = y_{t-1}$ (martingale property)

The most common example of a martingale is the random walk model

$$y_t = y_{t-1} + \varepsilon_t, \; \varepsilon_t \sim WN(0, \sigma^2)$$

where y_0 is a fixed initial value. Letting $I_t = \{y_t, \ldots, y_0\}$ implies $E[y_t|I_{t-1}] = y_{t-1}$ since $E[\varepsilon_t|I_{t-1}] = 0$.

Let $\{\varepsilon_t\}$ be a sequence of random variables with an associated information set I_t. The sequence $\{\varepsilon_t, I_t\}$ is called a *martingale difference sequence* (MDS) if

- $I_{t-1} \subset I_t$
- $E[\varepsilon_t|I_{t-1}] = 0$ (MDS property)

If $\{y_t, I_t\}$ is a martingale, a MDS $\{\varepsilon_t, I_t\}$ may be constructed by defining

$$\varepsilon_t = y_t - E[y_t|I_{t-1}]$$

By construction, a MDS is an uncorrelated process. This follows from the *law of iterated expectations.* To see this, for any $k > 0$

$$\begin{aligned} E[\varepsilon_t \varepsilon_{t-k}] &= E[E[\varepsilon_t \varepsilon_{t-k}|I_{t-1}]] \\ &= E[\varepsilon_{t-k} E[\varepsilon_t|I_{t-1}]] \\ &= 0 \end{aligned}$$

In fact, if z_n is any function of the past history of ε_t so that $z_n \in I_{t-1}$ then

$$E[\varepsilon_t z_n] = 0$$

Although a MDS is an uncorrelated process, it does not have to be an independent process. That is, there can be dependencies in the higher order moments of ε_t. The *autoregressive conditional heteroskedasticity* (ARCH) process in the following example is a leading example in finance.

MDSs are particularly nice to work with because there are many useful convergence results (laws of large numbers, central limit theorems etc.). White (1984), Hamilton (1994) and Hayashi (2000) describe the most useful of these results for the analysis of financial time series.

Example 8 *ARCH process*

A well known stylized fact about high frequency financial asset returns is that volatility appears to be autocorrelated. A simple model to capture such volatility autocorrelation is the ARCH process due to Engle (1982). To illustrate, let r_t denote the daily return on an asset and assume that $E[r_t] = 0$. An ARCH(1) model for r_t is

$$\begin{aligned} r_t &= \sigma_t z_t & (3.15) \\ z_t &\sim \text{iid } N(0,1) \\ \sigma_t^2 &= \omega + \alpha r_{t-1}^2 & (3.16) \end{aligned}$$

where $\omega > 0$ and $0 < \alpha < 1$. Let $I_t = \{r_t, \ldots\}$. The S+FinMetrics function simulate.garch may be used to generate simulations from above ARCH(1) model. For example, to simulate 250 observations on r_t with $\omega = 0.1$ and $\alpha = 0.8$ use

```
> rt = simulate.garch(model=list(a.value=0.1, arch=0.8),
+      n=250, rseed=196)
> class(rt)
[1] "structure"
> names(rt)
[1] "et"      "sigma.t"
```

Notice that the function simulate.garch produces simulated values of both r_t and σ_t. These values are shown in Figure 3.11.

To see that $\{r_t, I_t\}$ is a MDS, note that

$$\begin{aligned} E[r_t|I_{t-1}] &= E[z_t\sigma_t|I_{t-1}] \\ &= \sigma_t E[z_t|I_{t-1}] \\ &= 0 \end{aligned}$$

Since r_t is a MDS, it is an uncorrelated process. Provided $|\alpha| < 1$, r_t is a mean zero covariance stationary process. The unconditional variance of r_t is given by

$$\begin{aligned} \text{var}(r_t) &= E[r_t^2] = E[E[z_t^2\sigma_t^2|I_{t-1}]] \\ &= E[\sigma_t^2 E[z_t^2|I_{t-1}] = E[\sigma_t^2] \end{aligned}$$

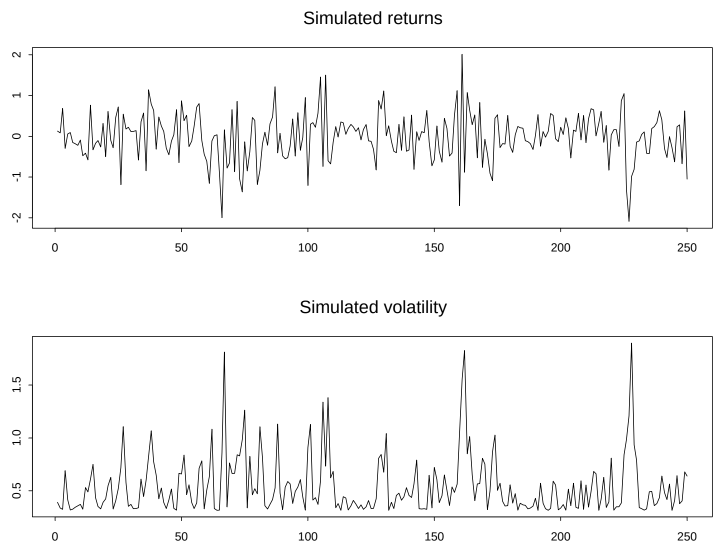


FIGURE 3.11. Simulated values from ARCH(1) process with $\omega = 1$ and $\alpha = 0.8$.

since $E[z_t^2|I_{t-1}] = 1$. Utilizing (3.16) and the stationarity of r_t, $E[\sigma_t^2]$ may be expressed as

$$E[\sigma_t^2] = \frac{\omega}{1-\alpha}$$

Furthermore, by adding ε_t^2 to both sides of (3.16) and rearranging it follows that r_t^2 has an AR(1) representation of the form

$$\varepsilon_t^2 = \omega + \alpha\varepsilon_{t-1}^2 + v_t$$

where $v_t = \varepsilon_t^2 - \sigma_t^2$ is a MDS.

3.2.8 Long-run Variance

Let y_t be a stationary and ergodic time series. Anderson's central limit theorem for stationary and ergodic processes (c.f. Hamilton (1994) pg. 195) states

$$\sqrt{T}(\bar{y} - \mu) \xrightarrow{d} N(0, \sum_{j=-\infty}^{\infty} \gamma_j)$$

or

$$\bar{y} \overset{A}{\sim} N\left(\mu, \frac{1}{T}\sum_{j=-\infty}^{\infty} \gamma_j\right)$$

The sample size, T, times the *asymptotic variance* of the sample mean is often called the *long-run variance* of y_t[6] :

$$\mathrm{lrv}(y_t) = T \cdot \mathrm{avar}(\bar{y}) = \sum_{j=-\infty}^{\infty} \gamma_j.$$

Since $\gamma_{-j} = \gamma_j$, $\mathrm{lrv}(y_t)$ may be alternatively expressed as

$$\mathrm{lrv}(y_t) = \gamma_0 + 2\sum_{j=1}^{\infty} \gamma_j.$$

Using the long-run variance, an asymptotic 95% confidence interval for μ takes the form

$$\bar{y} \pm 1.96 \cdot \sqrt{T^{-1}\widehat{\mathrm{lrv}}(y_t)}$$

where $\widehat{\mathrm{lrv}}(y_t)$ is a consistent estimate of $\mathrm{lrv}(y_t)$.

Estimating the Long-Run Variance

If y_t is a linear process, it may be shown that

$$\sum_{j=-\infty}^{\infty} \gamma_j = \sigma^2 \left(\sum_{j=0}^{\infty} \psi_j \right)^2 = \sigma^2 \psi(1)^2$$

and so

$$\mathrm{lrv}(y_t) = \sigma^2 \psi(1)^2 \tag{3.17}$$

Further, if $y_t \sim \mathrm{ARMA}(p, q)$ then

$$\psi(1) = \frac{1 + \theta_1 + \cdots + \theta_q}{1 - \phi_1 - \cdots - \phi_p} = \frac{\theta(1)}{\phi(1)}$$

so that

$$\mathrm{lrv}(y_t) = \frac{\sigma^2 \theta(1)^2}{\phi(1)^2}. \tag{3.18}$$

A consistent estimate of $\mathrm{lrv}(y_t)$ may then be computed by estimating the parameters of the appropriate $\mathrm{ARMA}(p, q)$ model and substituting these estimates into (3.18). Alternatively, the $\mathrm{ARMA}(p, q)$ process may be approximated by a high order $\mathrm{AR}(p^*)$ process

$$y_t = c + \phi_1 y_{t-1} + \cdots + \phi_{p^*} y_{t-p^*} + \varepsilon_t$$

[6] Using spectral methods, $lrv(\bar{y})$ has the alternative representation

$$lrv(\bar{y}) = \frac{1}{T} 2\pi f(0)$$

where $f(0)$ denotes the spectral density of y_t evaluated at frequency 0.

where the lag length p^* is chosen such that ε_t is uncorrelated. This gives rise to the *autoregressive long-run variance* estimate

$$\mathrm{lrv}_{\mathrm{AR}}(y_t) = \frac{\sigma^2}{\phi^*(1)^2}. \tag{3.19}$$

A consistent estimate of $\mathrm{lrv}(y_t)$ may also be computed using some non-parametric methods. An estimator made popular by Newey and West (1987) is the weighted autocovariance estimator

$$\widehat{\mathrm{lrv}}_{\mathrm{NW}}(y_t) = \hat{\gamma}_0 + 2\sum_{j=1}^{M_T} w_{j,T} \cdot \hat{\gamma}_j \tag{3.20}$$

where $w_{j,T}$ are weights which sum to unity and M_T is a truncation lag parameter that satisfies $M_T = O(T^{1/3})$. For MA(q) processes, $\gamma_j = 0$ for $j > q$ and Newey and West suggest using the *rectangular* weights $w_{j,T} = 1$ for $j \leq M_T = q$; 0 otherwise. For general linear processes, Newey and West suggest using the *Bartlett* weights $w_{j,T} = 1 - \frac{j}{M_T+1}$ with M_T equal to the integer part of $4(T/100)^{2/9}$.

Example 9 *Long-run variance of AR(1)*

Let y_t be an AR(1) process created using

```
> set.seed(101)
> e = rnorm(100,sd=1)
> y.ar1 = 1 + arima.sim(model=list(ar=0.75),innov=e)
```

Here $\psi(1) = \frac{1}{\phi(1)} = \frac{1}{1-\phi}$ and

$$\mathrm{lrv}(y_t) = \frac{\sigma^2}{(1-\phi)^2}.$$

For $\phi = 0.75$, $\sigma^2 = 1$, $\mathrm{lrv}(y_t) = 16$ implies for $T = 100$ an asymptotic standard error for $\bar{y}$ equal to $\mathrm{SE}(\bar{y}) = 0.40$. If $y_t \sim WN(0,1)$, then the asymptotic standard error for $\bar{y}$ is $\mathrm{SE}(\bar{y}) = 0.10$.

$\mathrm{lrv}_{\mathrm{AR}}(y_t)$ may be easily computed in `S-PLUS` using OLS to estimate the AR(1) parameters:

```
> ar1.fit = OLS(y.ar1~ar(1))
> rho.hat = coef(ar1.fit)[2]
> sig2.hat = sum(residuals(ar1.fit)^2)/ar1.fit$df.resid
> lrv.ar1 = sig2.hat/(1-rho.hat)^2
> as.numeric(lrv.ar1)
[1] 13.75
```

Here $\mathrm{lrv}_{\mathrm{AR}}(y_t) = 13.75$, and an estimate for $\mathrm{SE}(\bar{y})$ is $\widehat{\mathrm{SE}}_{\mathrm{AR}}(\bar{y}) = 0.371$.

The `S+FinMetrics` function `asymp.var` may be used to compute the nonparameteric Newey-West estimate $\mathrm{lrv}_{\mathrm{NW}}(y_t)$. The arguments expected by `asymp.var` are

```
 > args(asymp.var)
function(x, bandwidth, window = "bartlett", na.rm = F)
```

where `x` is a "`timeSeries`", `bandwidth` sets the truncation lag M_T in (3.20) and `window` specifies the weight function. Newey and West suggest setting the `bandwidth` using the sample size dependent rule

$$M_T = 4(T/100)^{2/9}$$

which is equal to 4 in the present case. The Newey-West long-run variance estimate is then

```
> lrv.nw = asymp.var(y.ar1, bandwidth=4)
> lrv.nw
[1] 7.238
```

and the Newey-West estimate of $\mathrm{SE}(\bar{y})$ is $\widehat{\mathrm{SE}}_{\mathrm{NW}}(\bar{y}) = 0.269$.

3.2.9 Variance Ratios

There has been considerable interest in testing the so-called *random walk* (RW) model for log stock prices (see chapter 2 in Campbell, Lo and MacKinlay (1997) for an extensive review). The RW model for log prices p_t has the form

$$p_t = \mu + p_{t-1} + \varepsilon_t,\ t = 1, \ldots, T$$

where ε_t is a random error term. Using $r_t = \Delta p_t$, the RW model may be rewritten as

$$r_t = \mu + \varepsilon_t$$

Campbell, Lo and MacKinlay distinguish three forms of the random walk model:

RW1 $\varepsilon_t \sim \mathrm{iid}(0, \sigma^2)$

RW2 ε_t is an independent process (allows for heteroskedasticity)

RW3 ε_t is an uncorrelated process (allows for dependence in higher order moments)

For asset returns, RW1 and RW2 are not very realistic and, therefore, most attention has been placed on testing the model RW3.

Some commonly used tests for RW3 are based on constructing *variance ratios*. To illustrate, consider the simple two-period variance ratio

$$\mathrm{VR}(2) = \frac{\mathrm{var}(r_t(2))}{2 \cdot \mathrm{var}(r_t)}$$

The numerator of the variance ratio is the variance of the two-period return, $r_t(2) = r_{t-1} + r_t$, and the deminator is two times the variance of the one-period return, r_t. Under RW1, is easy to see that $\mathrm{VR}(2) = 1$. If $\{r_t\}$ is an ergodic-stationary process then

$$\begin{aligned}\mathrm{VR}(2) &= \frac{\mathrm{var}(r_{t-1}) + \mathrm{var}(r_t) + 2 \cdot \mathrm{cov}(r_t, r_{t-1})}{2 \cdot \mathrm{var}(r_t)} \\ &= \frac{2\gamma_0 + 2\gamma_1}{2\gamma_0} = 1 + \rho_1\end{aligned}$$

There are three cases of interest depending on the value of ρ_1. If $\rho_1 = 0$ then $\mathrm{VR}(2) = 1$; if $\rho_1 > 1$ then $\mathrm{VR}(2) > 1$; if $\rho_1 < 1$ then $\mathrm{VR}(2) < 1$.

The general $q-$period variance ratio is

$$\mathrm{VR}(q) = \frac{\mathrm{var}(r_t(q))}{q \cdot \mathrm{var}(r_t)} \tag{3.21}$$

where $r_t(q) = r_{t-q+1} + \cdots + r_t$. Under RW1, $\mathrm{VR}(q) = 1$. For ergodic stationary returns, some algebra shows that

$$\mathrm{VR}(q) = 1 + 2 \cdot \sum_{k=1}^{q} \left(1 - \frac{k}{q}\right) \rho_k$$

When the variance ratio is greater than one, returns are called *mean averting* due to the dominating presence of positive autocorrelations. When the variance ratio is less than one, returns are called *mean reverting* due to the dominating presence of negative autocorrelations. Using the Wold representation (3.6), it can be shown that

$$\lim_{q \to \infty} \mathrm{VR}(q) = \frac{\sigma^2 \psi(1)^2}{\gamma_0} = \frac{\mathrm{lrv}(r_t)}{\mathrm{var}(r_t)}$$

That is, as q becomes large the variance ratio approaches the ratio of the long-run variance to the short-run variance. Furthermore, Under RW2 and RW3 it can be shown that $\mathrm{VR}(q) \to 1$ as $q \to \infty$ provided

$$\frac{1}{T} \sum_{t=1}^{T} \mathrm{var}(r_t) \to \bar{\sigma}^2 > 0$$

Test Statistics

Let $\{p_0, p_1, \ldots, p_{Tq}\}$ denote a sample of $Tq + 1$ log prices, which produces a sample of Tq one-period returns $\{r_1, \ldots, r_{Tq}\}$. Lo and MacKinlay (1988, 1989) develop a number of test statistics for testing the random walk hypothesis based on the estimated variance ratio

$$\widehat{\mathrm{VR}}(q) = \frac{\widehat{\mathrm{var}}(r_t(q))}{q \cdot \widehat{\mathrm{var}}(r_t)} \tag{3.22}$$

The form of the statistic depends on the particular random walk model (RW1, RW2 or RW3) assumed under the null hypothesis.

Under RW1, (3.22) is computed using

$$\widehat{\mathrm{VR}}(q) = \frac{\hat{\sigma}^2(q)}{\hat{\sigma}^2}$$

where

$$\begin{aligned}
\hat{\sigma}^2 &= \frac{1}{Tq}\sum_{k=1}^{Tq}(r_k - \hat{\mu})^2 \\
\hat{\sigma}^2(q) &= \frac{1}{Tq^2}\sum_{k=q}^{Tq}(r_k(q) - q\hat{\mu})^2 \\
\hat{\mu} &= \frac{1}{Tq}\sum_{k=1}^{Tq} r_k = \frac{1}{Tq}(p_{Tq} - p_0)
\end{aligned}$$

Lo and MacKinlay show that, under RW1,

$$\sqrt{Tq}(\widehat{\mathrm{VR}}(q) - 1) \overset{A}{\sim} N(0, 2(q-1))$$

Therefore, the *variance ratio test statistic*

$$\hat{\psi}(q) = \left(\frac{Tq}{2(q-1)}\right)^{1/2}(\widehat{\mathrm{VR}}(q) - 1) \tag{3.23}$$

has a limiting standard normal distribution under RW1.

Lo and MacKinlay also derive a modified version of (3.23) based on the following bias corrected estimates of σ^2 and $\sigma^2(q)$:

$$\begin{aligned}
\bar{\sigma}^2 &= \frac{1}{Tq-1}\sum_{k=1}^{Tq}(r_k - \hat{\mu})^2 \\
\bar{\sigma}^2(q) &= \frac{1}{m}\sum_{k=q}^{Tq}(r_k(q) - q\hat{\mu})^2 \\
m &= q(Tq - q + 1)\left(1 - \frac{q}{Tq}\right)
\end{aligned}$$

Defining $\overline{\mathrm{VR}}(q) = \bar{\sigma}^2(q)/\bar{\sigma}^2$, the biased corrected version of (3.23) has the form

$$\bar{\psi}(q) = \left(\frac{3Tq^2}{2(2q-1)(q-1)}\right)^{1/2}(\overline{\mathrm{VR}}(q) - 1) \tag{3.24}$$

which has a limiting standard normal distribution under RW1.

The variance ratio statistics (3.23) and (3.24) are not valid under the empirically relevant RW3 and RW3 models. For this model, Lo and MacKinlay derived the heteroskedasticity robust variance ratio statistic

$$\psi^*(q) = \hat{\Omega}(q)^{-1/2}(\overline{\mathrm{VR}}(q) - 1) \tag{3.25}$$

where

$$\begin{aligned}
\hat{\Omega}(q) &= \sum_{j=1}^{q-1} \left(\frac{2(q-j)}{j} \right) \hat{\delta}_j \\
\hat{\delta}_j &= \frac{\sum_{t=j+1}^{Tq} \hat{\alpha}_{0t} \hat{\alpha}_{jt}}{\left(\sum_{j=1}^{Tq} \hat{\alpha}_{0t} \right)^2} \\
\hat{\alpha}_{jt} &= (r_{t-j} - r_{t-j-1} - \hat{\mu})
\end{aligned}$$

Under RW2 or RW3, Lo and MacKinlay show that (3.25) has a limiting stardard normal distribution.

Example 10 *Testing the random walk hypothesis using variance ratios*

The variance ratio statistics (3.23), (3.24) and (3.25) may be computed using the `S+FinMetrics` function `varRatioTest`. The arguments for `varRatioTest` are

```
> args(varRatioTest)
function(x, n.periods, unbiased = T, hetero = F)
```

where `x` is the log return series (which may contain more than one series) and `n.periods` denotes the number of periods q in the variance ratio. If `unbiased=T` and `hetero=F` the bias corrected test statistic (3.24) is computed. If `unbiased=T` and `hetero=T` then the heteroskedasticity robust statistic (3.25) is computed. The function `varRatioTest` returns an object of class "`varRatioTest`" for which there are `print` and `plot` methods.

Consider testing the model RW3 for the daily log closing prices of the Dow Jones Industrial Average over the period 1/1/1960 through 1/1/1990. To compute the variance ratio (3.21) and the heteroskedasticity robust test (3.25) for $q = 1, \ldots, 60$ use

```
> VR.djia = varRatioTest(djia[timeEvent("1/1/1960","1/1/1990"),
+            "close"], n.periods=60, unbiased=T, hetero=T)
> class(VR.djia)
[1] "varRatioTest"
> names(VR.djia)
[1] "varRatio" "std.err"  "stat"     "hetero"
> VR.djia

Variance Ratio Test
```

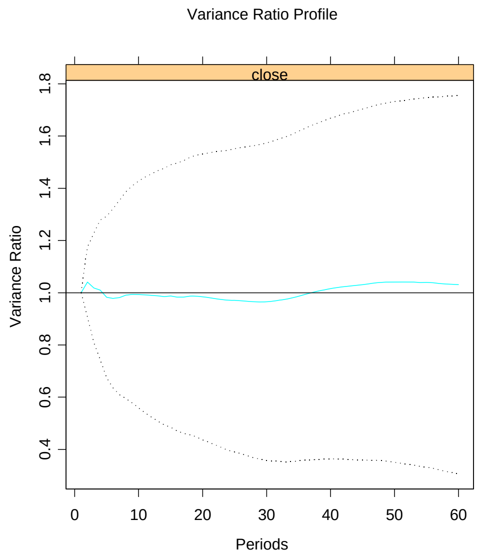


FIGURE 3.12. Variance ratios for the daily log prices of the Dow Jones Industrial Average.

```
Null Hypothesis: random walk with heteroskedastic errors

Variable: close
   var.ratio std.err       stat
 2    1.0403 0.06728  0.5994871
 3    1.0183 0.10527  0.1738146
...
60    1.0312 0.36227  0.0861747

 * : significant at 5% level
** : significant at 1% level
```

None of the variance ratios are statistically different from unity at the 5% level.

Figure 3.12 shows the results of the variance ratio tests based on `plot` method

```
> plot(VR.djia)
```

The variance ratios computed for different values of q hover around unity, and the $\pm$ 2 $\times$ standard error bands indicate that the model RW3 is not rejected at the 5% level.

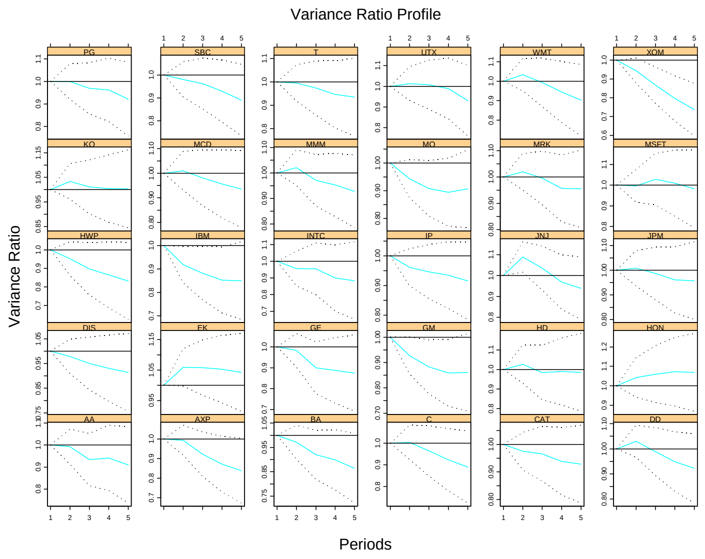


FIGURE 3.13. Variance ratio statistics for daily log prices on individual Dow Jones index stocks.

The RW3 model appears to hold for the Dow Jones index. To test the RW3 model for the top thirty stocks in the index individually, based on $q = 1, \ldots, 5$, use

```
> VR.DJ30 = varRatioTest(DowJones30, n.periods=5, unbiased=T,
+                         hetero=T)
> plot(VR.DJ30)
```

The results, illustrated in Figure 3.13, indicate that the RW3 model may not hold for some individual stocks.

3.3 Univariate Nonstationary Time Series

A univariate time series process $\{y_t\}$ is called *nonstationary* if it is not stationary. Since a stationary process has time invariant moments, a nonstationary process must have some time dependent moments. The most common forms of nonstationarity are caused by time dependence in the mean and variance.

Trend Stationary Process

$\{y_t\}$ is a *trend stationary* process if it has the form

$$y_t = TD_t + x_t$$

where TD_t are deterministic trend terms (constant, trend, seasonal dummies etc) that depend on t and $\{x_t\}$ is stationary. The series y_t is nonstationary because $E[TD_t] = TD_t$ which depends on t. Since x_t is stationary, y_t never deviates too far away from the deterministic trend TD_t. Hence, y_t exhibits *trend reversion.* If TD_t were known, y_t may be transformed to a stationary process by subtracting off the deterministic trend terms:

$$x_t = y_t - TD_t$$

Example 11 *Trend stationary AR(1)*

A trend stationary AR(1) process with $TD_t = \mu + \delta t$ may be expressed in three equivalent ways

$$\begin{aligned} y_t &= \mu + \delta t + u_t, u_t = \phi u_{t-1} + \varepsilon_t \\ y_t - \mu - \delta t &= \phi(y_{t-1} - \mu - \delta(t-1)) + \varepsilon_t \\ y_t &= c + \beta t + \phi y_{t-1} + \varepsilon_t \end{aligned}$$

where $|\phi| < 1$, $c = \mu(1-\phi) + \delta$, $\beta = \delta(1-\phi)t$ and $\varepsilon_t \sim WN(0, \sigma^2)$. Figure 3.14 shows $T = 100$ observations from a trend stationary AR(1) with $\mu = 1$, $\delta = 0.25$, $\phi = 0.75$ and $\sigma^2 = 1$ created with the S-PLUS commands

```
> set.seed(101)
> y.tsar1 = 1 + 0.25*seq(100) +
+ arima.sim(model=list(ar=0.75),n=100)
> tsplot(y.tsar1,ylab="y")
> abline(a=1,b=0.25)
```

The simulated data show clear trend reversion.

Integrated Processes

$\{y_t\}$ is an *integrated process* of order 1, denoted $y_t \sim I(1)$, if it has the form

$$y_t = y_{t-1} + u_t \tag{3.26}$$

where u_t is a stationary time series. Clearly, the first difference of y_t is stationary

$$\Delta y_t = u_t$$

Because of the above property, $I(1)$ processes are sometimes called *difference stationary* processes. Starting at y_0, by recursive substitution y_t has

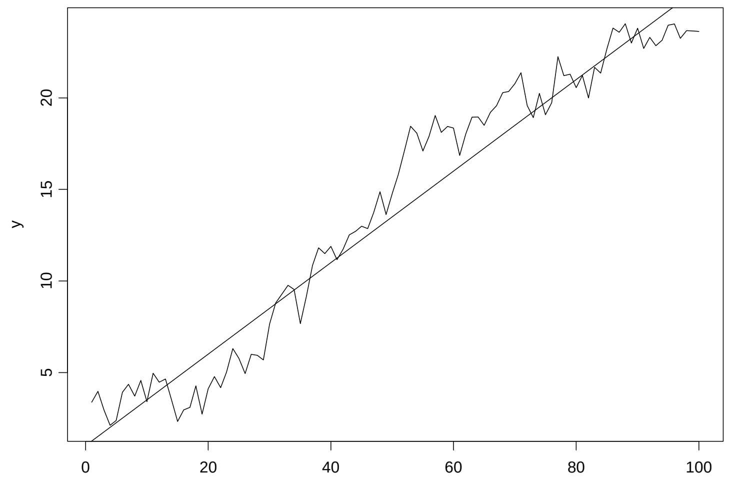


FIGURE 3.14. Simulated trend stationary process.

the representation of an *integrated sum* of stationary innovations

$$y_t = y_0 + \sum_{j=1}^{t} u_j. \tag{3.27}$$

The integrated sum $\sum_{j=1}^{t} u_j$ is called a *stochastic trend* and is denoted TS_t. Notice that

$$TS_t = TS_{t-1} + u_t$$

where $TS_0 = 0$. In contrast to a deterministic trend, changes in a stochastic trend are not perfectly predictable.

Since the stationary process u_t does not need to be differenced, it is called an integrated process of order zero and is denoted $u_t \sim I(0)$. Recall, from the Wold representation (3.6) a stationary process has an infinite order moving average representation where the moving average weights decline to zero at a geometric rate. From (3.27) it is seen that an $I(1)$ process has an infinite order moving average representation where all of the weights on the innovations are equal to 1.

If $u_t \sim IWN(0, \sigma^2)$ in (3.26) then y_t is called a *random walk*. In general, an $I(1)$ process can have serially correlated and heteroskedastic innovations u_t. If y_t is a random walk and assuming y_0 is fixed then it can be shown that

$$\begin{aligned}\gamma_0 &= \sigma^2 t \\ \gamma_j &= (t-j)\sigma^2 \\ \rho_j &= \sqrt{\frac{t-j}{t}}\end{aligned}$$

which clearly shows that y_t is nonstationary. Also, if t is large relative to j then $\rho_j \approx 1$. Hence, for an $I(1)$ process, the ACF does not decay at a geometric rate but at a linear rate as j increases.

An $I(1)$ process with drift has the form

$$y_t = \mu + y_{t-1} + u_t, \text{ where } u_t \sim I(0)$$

Starting at $t = 0$ an $I(1)$ process with drift μ may be expressed as

$$\begin{aligned}y_t &= y_0 + \mu t + \sum_{j=1}^{t} u_t \\ &= TD_t + TS_t\end{aligned}$$

so that it may be thought of as being composed of a deterministic linear trend $TD_t = y_0 + \mu t$ as well as a stochastic trend $TS_t = \sum_{j=1}^{t} u_j$.

An $I(d)$ process $\{y_t\}$ is one in which $\Delta^d y_t \sim I(0)$. In finance and economics data series are rarely modeled as $I(d)$ process with $d > 2$. Just as an $I(1)$ process with drift contains a linear deterministic trend, an $I(2)$ process with drift will contain a quadratic trend.

Example 12 *Simulated I(1) processes*

Consider the simulation of $T = 100$ observations from various $I(1)$ processes where the innovations u_t follow an AR(1) process $u_t = 0.75u_{t-1} + \varepsilon_t$ with $\varepsilon_t \sim GWN(0, 1)$.

```
> set.seed(101)
> u.ar1 = arima.sim(model=list(ar=0.75), n=100)
> y1 = cumsum(u.ar1)
> y1.d = 1 + 0.25*seq(100)+ y1
> y2 = rep(0,100)
> for (i in 3:100) {
+   y2[i] = 2*y2[i-1] - y2[i-2] + u.ar1[i]
+ }
```

The simulated data are illustrated in Figure 3.15 .

Example 13 *Financial time series*

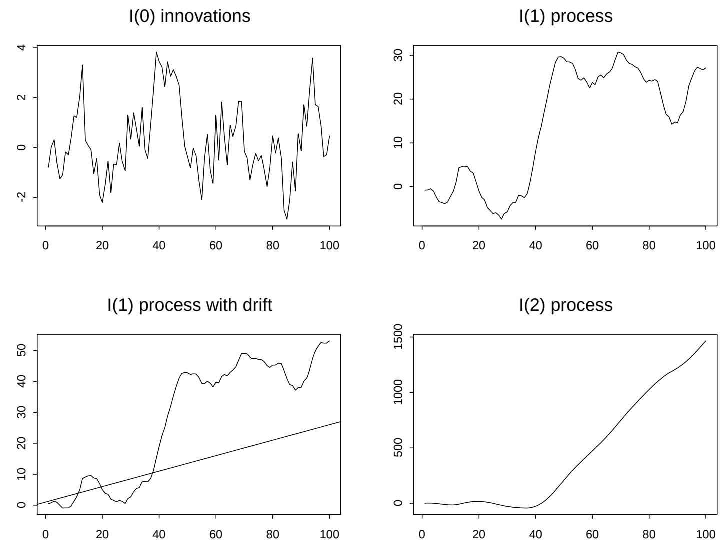


FIGURE 3.15. Simulated $I(d)$ processes for $d = 0$, 1 and 2.

Many financial time series are well characterized by $I(1)$ processes. The leading example of an $I(1)$ process with drift is the logarithm of an asset price. Common examples of $I(1)$ processes without drifts are the logarithms of exchange rates, nominal interest rates, and inflation rates. Notice that if inflation is constructed as the the difference in the logarithm of a price index and is an $I(1)$ process, then the logarithm of the price index is an $I(2)$ process. Examples of these data are illustrated in Figure 3.16. The exchange rate is the monthly log of the US/CA spot exchange rate taken from the `S+FinMetrics` "`timeSeries`" `lexrates.dat`, the asset price of the monthly S&P 500 index taken from the `S+FinMetrics` "`timeSeries`" object `singleIndex.dat`, the nominal interest rate is the 30 day T-bill rate taken from the `S+FinMetrics` "`timeSeries`" object `rf.30day,` and the monthly consumer price index is taken from the `S+FinMetrics` "`timeSeries`" object `CPI.dat`.

3.4 Long Memory Time Series

If a time series y_t is $I(0)$ then its ACF declines at a geometric rate. As a result, $I(0)$ process have *short memory* since observations far apart in time are essentially independent. Conversely, if y_t is $I(1)$ then its ACF declines at a linear rate and observations far apart in time are not independent. In

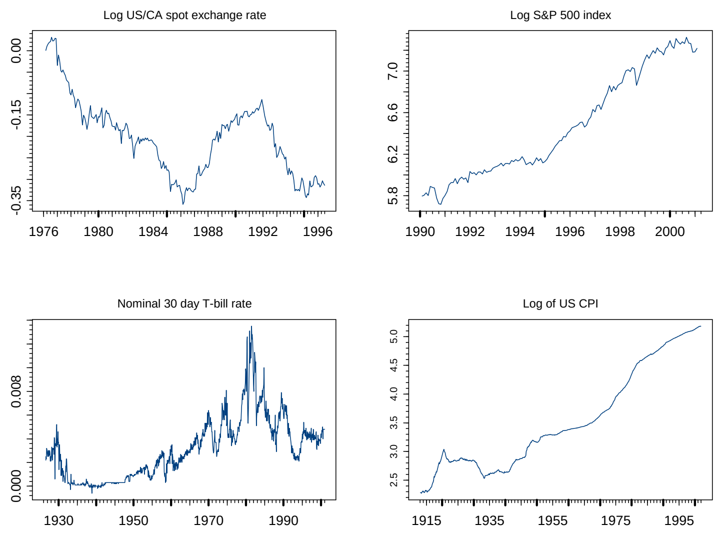


FIGURE 3.16. Monthly financial time series.

between $I(0)$ and $I(1)$ processes are so-called *fractionally integrated* $I(d)$ process where $0 < d < 1$. The ACF for a fractionally integrated processes declines at a polynomial (hyperbolic) rate, which implies that observations far apart in time may exhibit weak but non-zero correlation. This weak correlation between observations far apart is often referred to as *long memory*.

A fractionally integrated white noise process y_t has the form

$$(1-L)^d y_t = \varepsilon_t,\ \varepsilon_t \sim WN(0,\sigma^2) \tag{3.28}$$

where $(1-L)^d$ has the binomial series expansion representation (valid for any $d > -1$)

$$\begin{aligned}(1-L)^d &= \sum_{k=0}^{\infty}\binom{d}{k}(-L)^k \\ &= 1 - dL + \frac{d(d-1)}{2!}L^2 - \frac{d(d-1)(d-2)}{3!}L^3 + \cdots\end{aligned}$$

If $d = 1$ then y_t is a random walk and if $d = 0$ then y_t is white noise. For $0 < d < 1$ it can be shown that

$$\rho_k \propto k^{2d-1}$$

as $k \to \infty$ so that the ACF for y_t declines hyperbolically to zero at a speed that depends on d. Further, it can be shown y_t is stationary and ergodic for $0 < d < 0.5$ and that the variance of y_t is infinite for $0.5 \leq d < 1$.

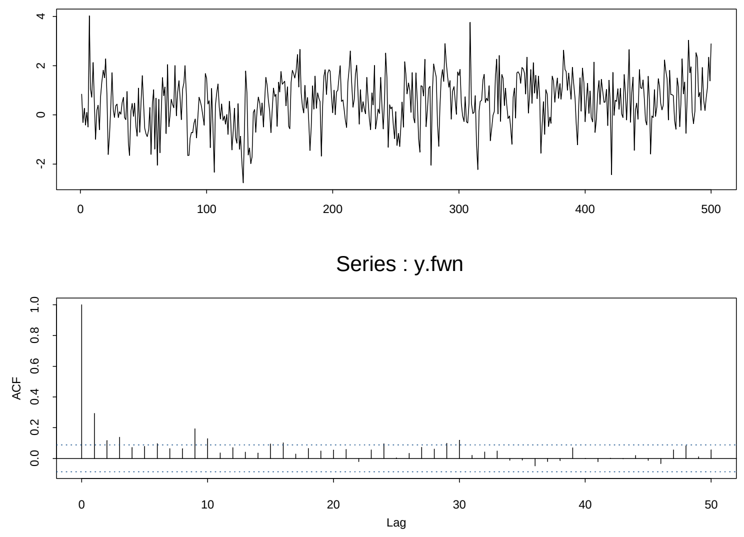


FIGURE 3.17. Simulated values from a fractional white noise process with $d = 0.3$ and $\sigma = 1$.

Example 14 *Simulated fractional white noise*

The `S+FinMetrics` function `simulate.FARIMA` may be used to generate simulated values from a fractional white noise process. To simulate 500 observations from (3.28) with $d = 0.3$ and $\sigma^2 = 1$ use

```
> set.seed(394)
> y.fwn = simulate.FARIMA(list(d=0.3), 500)
```

Figure 3.17 shows the simulated data along with the sample ACF created using

```
> par(mfrow=c(2,1))
> tsplot(y.fwn)
> tmp = acf(y.fwn,lag.max=50)
```

Notice how the sample ACF slowly decays to zero.

A fractionally integrated process with stationary and ergodic ARMA(p,q) errors

$$(1 - L)^d y_t = u_t,\ u_t \sim \text{ARMA}(p,q)$$

is called an *autoregressive fractionally integrated moving average* (ARFIMA) process. The modeling of long memory process is described in detail in Chapter 8.

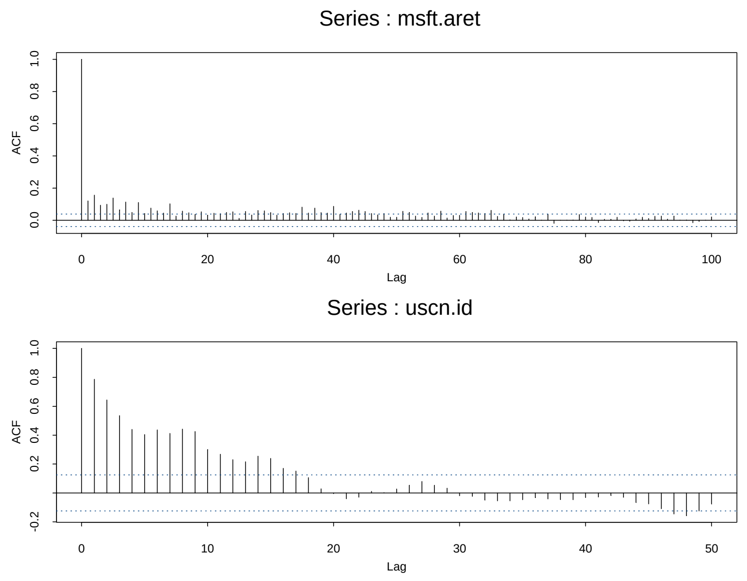


FIGURE 3.18. SACFs for the absolute value of daily returns on Microsoft and the monthly 30-day interest rate differential between U.S. bonds and Canadian bonds.

Example 15 *Long memory in financial time series*

Long memory behavior has been observed in certain types of financial time series. Ding, Granger and Engle (1993) find evidence of long memory in the absolute value of daily stock returns. Baillie and Bollerslev (1994) find evidence for long memory in the monthly interest rate differentials between short term U.S. government bonds and short term foreign government bonds. To illustrate, consider the absolute values of the daily returns on Microsoft over the 10 year period 1/2/1991 - 1/2/2001 taken from the `S+FinMetrics` “`timeSeries`” `DowJones30`

```
> msft.aret = abs(getReturns(DowJones30[,"MSFT"]))
```

Consider also the monthly US/CA 30-day interest rate differential over the period February 1976 through June 1996 in the “`timeSeries`” `uscn.id` constructed earlier and taken from the `S+FinMetrics` “`timeSeries`” object `lexrates.dat`. Figure 3.18 shows the SACFs these series create by

```
> par(mfrow=c(2,1))
> tmp = acf(msft.aret, lag.max=100)
> tmp = acf(uscn.id, lag.max=50)
```

For the absolute return series, notice the large number of small but apparently significant autocorrelations at very long lags. This is indicative of long memory. For the interest rate differential series, the ACF appears to decay fairly quickly, so the evidence for long memory is not as strong.

3.5 Multivariate Time Series

Consider n time series variables $\{y_{1t}\}, \ldots, \{y_{nt}\}$. A *multivariate time series* is the $(n \times 1)$ vector time series $\{\mathbf{Y}_t\}$ where the i^{th} row of $\{\mathbf{Y}_t\}$ is $\{y_{it}\}$. That is, for any time t, $\mathbf{Y}_t = (y_{1t}, \ldots, y_{nt})'$. Multivariate time series analysis is used when one wants to model and explain the interactions and co-movements among a group of time series variables. In finance, multivariate time series analysis is used to model systems of asset returns, asset prices and exchange rates, the term structure of interest rates, asset returns/prices, and economic variables etc. Many of the time series concepts described previously for univariate time series carry over to multivariate time series in a natural way. Additionally, there are some important time series concepts that are particular to multivariate time series. The following sections give the details of these extensions and provide examples using `S-PLUS` and `S+FinMetrics`.

3.5.1 Stationary and Ergodic Multivariate Time Series

A multivariate time series $\mathbf{Y}_t$ is covariance stationary and ergodic if all of its component time series are stationary and ergodic. The mean of $\mathbf{Y}_t$ is defined as the $(n \times 1)$ vector

$$E[\mathbf{Y}_t] = \boldsymbol{\mu} = (\mu_1, \ldots, \mu_n)'$$

where $\mu_i = E[y_{it}]$ for $i = 1, \ldots, n$. The variance/covariance matrix of $\mathbf{Y}_t$ is the $(n \times n)$ matrix

$$\begin{aligned}\mathrm{var}(\mathbf{Y}_t) &= \Gamma_0 = E[(\mathbf{Y}_t - \boldsymbol{\mu})(\mathbf{Y}_t - \boldsymbol{\mu})'] \\ &= \begin{pmatrix} \mathrm{var}(y_{1t}) & \mathrm{cov}(y_{1t}, y_{2t}) & \cdots & \mathrm{cov}(y_{1t}, y_{nt}) \\ \mathrm{cov}(y_{2t}, y_{1t}) & \mathrm{var}(y_{2t}) & \cdots & \mathrm{cov}(y_{2t}, y_{nt}) \\ \vdots & \vdots & \ddots & \vdots \\ \mathrm{cov}(y_{nt}, y_{1t}) & \mathrm{cov}(y_{nt}, y_{2t}) & \cdots & \mathrm{var}(y_{nt}) \end{pmatrix}\end{aligned}$$

The matrix $\mathbf{\Gamma}_0$ has elements $\gamma_{ij}^0 = \mathrm{cov}(y_{it}, y_{jt})$. The correlation matrix of Y_t is the $(n \times n)$ matrix

$$\mathrm{corr}(\mathbf{Y}_t) = \mathbf{R}_0 = \mathbf{D}^{-1}\mathbf{\Gamma}_0\mathbf{D}^{-1}$$

where $\mathbf{D}$ is an $(n \times n)$ diagonal matrix with j^{th} diagonal element $(\gamma_{jj}^0)^{1/2} = \mathrm{SD}(y_{jt})$. The parameters μ, Γ_0 and $\mathbf{R}_0$ are estimated from data $(\mathbf{Y}_1, \ldots, \mathbf{Y}_T)$

using the sample moments

$$\begin{aligned}\bar{\mathbf{Y}} &= \frac{1}{T}\sum_{t=1}^{T}\mathbf{Y}_t \\ \hat{\mathbf{\Gamma}}_0 &= \frac{1}{T}\sum_{t=1}^{T}(\mathbf{Y}_t - \bar{\mathbf{Y}})(\mathbf{Y}_t - \bar{\mathbf{Y}})' \\ \hat{\mathbf{R}}_0 &= \hat{\mathbf{D}}^{-1}\hat{\mathbf{\Gamma}}_0\hat{\mathbf{D}}^{-1}\end{aligned}$$

where $\mathbf{D}$ is the $(n\times n)$ diagonal matrix with the sample standard deviations of y_{jt} along the diagonal. In order for the sample variance matrix $\hat{\mathbf{\Gamma}}_0$ and correlation matrix $\hat{\mathbf{R}}_0$ to be positive definite, the sample size T must be greater than the number of component time series n.

Example 16 *System of asset returns*

The `S+FinMetrics` "`timeSeries`" object `DowJones30` contains daily closing prices on the 30 assets in the Dow Jones index. An example of a stationary and ergodic multivariate time series is the continuously compounded returns on the first four assets in this index:

```
> Y = getReturns(DowJones30[,1:4],type="continuous")
> colIds(Y)
[1] "AA"  "AXP" "T"   "BA"
```

The `S-PLUS` function `colMeans` may be used to efficiently compute the mean vector of $\mathbf{Y}$

```
> colMeans(Y)
        AA       AXP           T        BA
 0.0006661 0.0009478 -0.00002873 0.0004108
```

In S-PLUS 7, the function `colMeans` has a method for "`timeSeries`" objects so the extractor function `seriesData` is not needed to extract the data slot of the variable `Y`. The `S-PLUS` functions `var` and `cor`, which also have methods for "`timeSeries`" objects, may be used to compute $\hat{\mathbf{\Gamma}}_0$ and $\hat{\mathbf{R}}_0$

```
> var(Y)
            AA        AXP          T         BA
 AA 0.00041096 0.00009260 0.00005040 0.00007301
AXP 0.00009260 0.00044336 0.00008947 0.00009546
  T 0.00005040 0.00008947 0.00040441 0.00004548
 BA 0.00007301 0.00009546 0.00004548 0.00036829
> cor(Y)
        AA    AXP      T     BA
 AA 1.0000 0.2169 0.1236 0.1877
AXP 0.2169 1.0000 0.2113 0.2362
```

```
  T 0.1236 0.2113 1.0000 0.1179
 BA 0.1877 0.2362 0.1179 1.0000
```

If only the variances or standard deviations of $\mathbf{Y}_t$ are needed the S-PLUS functions colVars and colStdevs may be used

```
> colVars(seriesData(Y))
       AA       AXP         T        BA
 0.000411 0.0004434 0.0004044 0.0003683
> colStdevs(seriesData(Y))
       AA      AXP       T       BA
 0.020272 0.021056 0.02011 0.019191
```

Cross Covariance and Correlation Matrices

For a univariate time series y_t the autocovariances γ_k and autocorrelations ρ_k summarize the linear time dependence in the data. With a multivariate time series $\mathbf{Y}_t$ each component has autocovariances and autocorrelations but there are also cross lead-lag covariances and correlations between all possible pairs of components. The autocovariances and autocorrelations of y_{jt} for $j = 1, \ldots, n$ are defined as

$$\begin{aligned}\gamma_{jj}^k &= \text{cov}(y_{jt}, y_{jt-k}),\\ \rho_{jj}^k &= \text{corr}(y_{jt}, y_{jt-k}) = \frac{\gamma_{jj}^k}{\gamma_{jj}^0}\end{aligned}$$

and these are symmetric in k: $\gamma_{jj}^k = \gamma_{jj}^{-k}$, $\rho_{jj}^k = \rho_{jj}^{-k}$. The *cross lag covariances* and *cross lag correlations* between y_{it} and y_{jt} are defined as

$$\begin{aligned}\gamma_{ij}^k &= \text{cov}(y_{it}, y_{jt-k}),\\ \rho_{ij}^k &= \text{corr}(y_{jt}, y_{jt-k}) = \frac{\gamma_{ij}^k}{\sqrt{\gamma_{ii}^0 \gamma_{jj}^0}}\end{aligned}$$

and they are not necessarily symmetric in k. In general,

$$\gamma_{ij}^k = \text{cov}(y_{it}, y_{jt-k}) \neq \text{cov}(y_{it}, y_{jt+k}) = \text{cov}(y_{jt}, y_{it-k}) = \gamma_{ij}^{-k}$$

If $\gamma_{ij}^k \neq 0$ for some $k > 0$ then y_{jt} is said to *lead* y_{it}. Similarly, if $\gamma_{ij}^{-k} \neq 0$ for some $k > 0$ then y_{it} is said to *lead* y_{jt}. It is possible that y_{it} leads y_{jt} and vice-versa. In this case, there is said to be *feedback* between the two series.

All of the lag k cross covariances and correlations are summarized in the $(n \times n)$ lag k cross covariance and lag k cross correlation matrices

$$\begin{aligned}
\mathbf{\Gamma}_k &= E[(\mathbf{Y}_t - \boldsymbol{\mu})(\mathbf{Y}_{t-k} - \boldsymbol{\mu})'] \\
&= \begin{pmatrix} \text{cov}(y_{1t}, y_{1t-k}) & \text{cov}(y_{1t}, y_{2t-k}) & \cdots & \text{cov}(y_{1t}, y_{nt-k}) \\ \text{cov}(y_{2t}, y_{1t-k}) & \text{cov}(y_{2t}, y_{2t-k}) & \cdots & \text{cov}(y_{2t}, y_{nt-k}) \\ \vdots & \vdots & \ddots & \vdots \\ \text{cov}(y_{nt}, y_{1t-k}) & \text{cov}(y_{nt}, y_{2t-k}) & \cdots & \text{cov}(y_{nt}, y_{nt-k}) \end{pmatrix} \\
\mathbf{R}_k &= \mathbf{D}^{-1}\mathbf{\Gamma}_k\mathbf{D}^{-1}
\end{aligned}$$

The matrices $\mathbf{\Gamma}_k$ and $\mathbf{R}_k$ are not symmetric in k but it is easy to show that $\mathbf{\Gamma}_{-k} = \mathbf{\Gamma}'_k$ and $\mathbf{R}_{-k} = \mathbf{R}'_k$. The matrices $\mathbf{\Gamma}_k$ and $\mathbf{R}_k$ are estimated from data $(\mathbf{Y}_1, \ldots, \mathbf{Y}_T)$ using

$$\begin{aligned}
\hat{\mathbf{\Gamma}}_k &= \frac{1}{T}\sum_{t=k+1}^{T}(\mathbf{Y}_t - \bar{\mathbf{Y}})(\mathbf{Y}_{t-k} - \bar{\mathbf{Y}})' \\
\hat{\mathbf{R}}_k &= \hat{\mathbf{D}}^{-1}\hat{\mathbf{\Gamma}}_k\hat{\mathbf{D}}^{-1}
\end{aligned}$$

Example 17 *Lead-lag covariances and correlations among asset returns*

Consider computing the cross lag covariances and correlations for $k = 0, \ldots, 5$ between the first two Dow Jones 30 asset returns in the "`timeSeries`" `Y`. These covariances and correlations may be computed using the `S-PLUS` function `acf`

```
> Ghat = acf(Y[,1:2],lag.max=5,type="covariance",plot=F)
> Rhat = acf(Y[,1:2],lag.max=5,plot=F)
```

`Ghat` and `Rhat` are objects of class "`acf`" for which there is only a `print` method. For example, the estimated cross lag autocorrelations are

```
> Rhat
Call: acf(x = Y[, 1:2], lag.max = 5, plot = F)

Autocorrelation matrix:
  lag   AA.AA  AA.AXP AXP.AXP
1   0  1.0000  0.2169  1.0000
2   1  0.0182  0.0604 -0.0101
3   2 -0.0556 -0.0080 -0.0710
4   3  0.0145 -0.0203 -0.0152
5   4 -0.0639  0.0090 -0.0235
6   5  0.0142 -0.0056 -0.0169

  lag  AXP.AA
1   0  0.2169
```

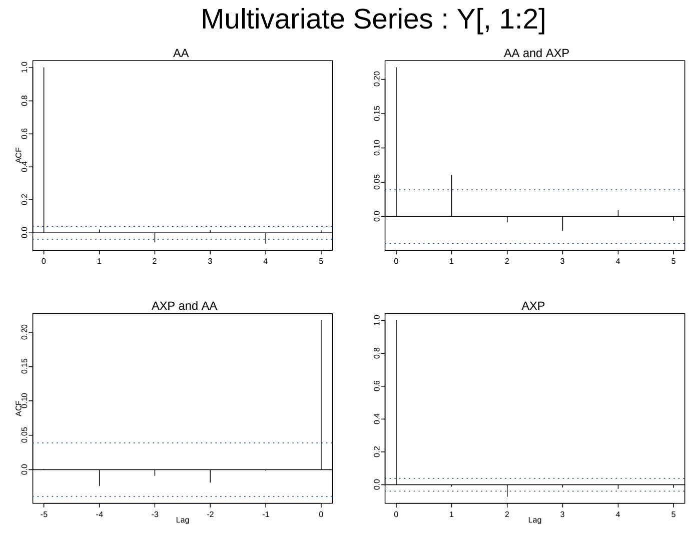


FIGURE 3.19. Cross lag correlations between the first two Dow Jones 30 asset returns.

```
2  -1 -0.0015
3  -2 -0.0187
4  -3 -0.0087
5  -4 -0.0233
6  -5  0.0003
```

The function `acf.plot` may be used to plot the cross lag covariances and correlations produced by `acf`.

```
> acf.plot(Rhat)
```

Figure 3.19 shows these cross lag correlations. The matrices $\hat{\mathbf{\Gamma}}_k$ and $\hat{\mathbf{R}}_k$ may be extracted from `acf` component of `Ghat` and `Rhat`, respectively. For example,

```
> Ghat$acf[1,,]
           [,1]       [,2]
[1,] 0.00041079 0.00009256
[2,] 0.00009256 0.00044318
> Rhat$acf[1,,]
       [,1]   [,2]
[1,] 1.0000 0.2169
[2,] 0.2169 1.0000
> Ghat$acf[2,,]
```

```
             [,1]        [,2]
[1,]  7.488e-006  2.578e-005
[2,] -6.537e-007 -4.486e-006
> Rhat$acf[2,,]
           [,1]     [,2]
[1,]  0.018229  0.06043
[2,] -0.001532 -0.01012
```

extracts $\hat{\mathbf{\Gamma}}_1$, $\hat{\mathbf{R}}_1$, $\hat{\mathbf{\Gamma}}_2$ and $\hat{\mathbf{R}}_2$.

3.5.2 Multivariate Wold Representation

Any $(n \times 1)$ covariance stationary multivariate time series $\mathbf{Y}_t$ has a Wold or linear process representation of the form

$$\begin{aligned} \mathbf{Y}_t &= \boldsymbol{\mu} + \boldsymbol{\varepsilon}_t + \mathbf{\Psi}_1 \boldsymbol{\varepsilon}_{t-1} + \mathbf{\Psi}_2 \boldsymbol{\varepsilon}_{t-2} + \cdots \\ &= \boldsymbol{\mu} + \sum_{k=0}^{\infty} \mathbf{\Psi}_k \boldsymbol{\varepsilon}_{t-k} \end{aligned} \tag{3.29}$$

where $\mathbf{\Psi}_0 = \mathbf{I}_n$ and $\boldsymbol{\varepsilon}_t$ is a multivariate white noise process with mean zero and variance matrix $E[\boldsymbol{\varepsilon}_t \boldsymbol{\varepsilon}_t'] = \mathbf{\Sigma}$. In (3.29), Ψ_k is an $(n \times n)$ matrix with (i,j)th element ψ_{ij}^k. In lag operator notation, the Wold form is

$$\begin{aligned} \mathbf{Y}_t &= \boldsymbol{\mu} + \mathbf{\Psi}(L) \boldsymbol{\varepsilon}_t \\ \mathbf{\Psi}(L) &= \sum_{k=0}^{\infty} \mathbf{\Psi}_k L^k \end{aligned}$$

The moments of $\mathbf{Y}_t$ are given by

$$\begin{aligned} E[\mathbf{Y}_t] &= \boldsymbol{\mu} \\ \text{var}(\mathbf{Y}_t) &= \sum_{k=0}^{\infty} \mathbf{\Psi}_k \mathbf{\Sigma} \mathbf{\Psi}_k' \end{aligned}$$

VAR Models

The most popular multivariate time series model is the *vector autoregressive* (VAR) model. The VAR model is a multivariate extension of the univariate autoregressive model. For example, a bivariate VAR(1) model has the form

$$\begin{pmatrix} y_{1t} \\ y_{2t} \end{pmatrix} = \begin{pmatrix} c_1 \\ c_2 \end{pmatrix} + \begin{pmatrix} \pi_{11}^1 & \pi_{12}^1 \\ \pi_{21}^1 & \pi_{22}^1 \end{pmatrix} \begin{pmatrix} y_{1t-1} \\ y_{2t-1} \end{pmatrix} + \begin{pmatrix} \varepsilon_{1t} \\ \varepsilon_{2t} \end{pmatrix}$$

or

$$\begin{aligned} y_{1t} &= c_1 + \pi_{11}^1 y_{1t-1} + \pi_{12}^1 y_{2t-1} + \varepsilon_{1t} \\ y_{2t} &= c_2 + \pi_{21}^1 y_{1t-1} + \pi_{22}^1 y_{2t-1} + \varepsilon_{2t} \end{aligned}$$

where

$$\begin{pmatrix} \varepsilon_{1t} \\ \varepsilon_{2t} \end{pmatrix} \sim \text{iid} \left(\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} \sigma_{11} & \sigma_{12} \\ \sigma_{12} & \sigma_{22} \end{pmatrix} \right)$$

In the equations for y_1 and y_2, the lagged values of both y_1 and y_2 are present.

The general VAR(p) model for $\mathbf{Y}_t = (y_{1t}, y_{2t}, \ldots, y_{nt})'$ has the form

$$\mathbf{Y}_t = \mathbf{c} + \mathbf{\Pi}_1 \mathbf{Y}_{t-1} + \mathbf{\Pi}_2 \mathbf{Y}_{t-2} + \cdots + \mathbf{\Pi}_p \mathbf{Y}_{t-p} + \boldsymbol{\varepsilon}_t, \; t = 1, \ldots, T$$

where $\mathbf{\Pi}_i$ are $(n \times n)$ coefficient matrices and $\boldsymbol{\varepsilon}_t$ is an $(n \times 1)$ unobservable zero mean white noise vector process with covariance matrix $\mathbf{\Sigma}$. VAR models are capable of capturing much of the complicated dynamics observed in stationary multivariate time series. Details about estimation, inference, and forecasting with VAR models are given in chapter eleven.

3.5.3 Long Run Variance

Let $\mathbf{Y}_t$ be an $(n \times 1)$ stationary and ergodic multivariate time series with $E[\mathbf{Y}_t] = \boldsymbol{\mu}$. Anderson's central limit theorem for stationary and ergodic process states

$$\sqrt{T}(\bar{\mathbf{Y}} - \boldsymbol{\mu}) \xrightarrow{d} N\left(\mathbf{0}, \sum_{j=-\infty}^{\infty} \mathbf{\Gamma}_j\right)$$

or

$$\bar{\mathbf{Y}} \overset{A}{\sim} N\left(\boldsymbol{\mu}, \frac{1}{T} \sum_{j=-\infty}^{\infty} \mathbf{\Gamma}_j\right)$$

Hence, the *long-run variance* of $\mathbf{Y}_t$ is T times the asymptotic variance of $\bar{\mathbf{Y}}$:

$$\text{lrv}(\mathbf{Y}_t) = T \cdot \text{avar}(\bar{\mathbf{Y}}) = \sum_{j=-\infty}^{\infty} \Gamma_j$$

Since $\mathbf{\Gamma}_{-j} = \mathbf{\Gamma}_j'$, lrv$(\mathbf{Y}_t)$ may be alternatively expressed as

$$\text{lrv}(\mathbf{Y}_t) = \mathbf{\Gamma}_0 + \sum_{j=1}^{\infty} (\mathbf{\Gamma}_j + \mathbf{\Gamma}_j')$$

Using the Wold representation of $\mathbf{Y}_t$ it can be shown that

$$\text{lrv}(\mathbf{Y}_t) = \mathbf{\Psi}(1) \mathbf{\Sigma} \mathbf{\Psi}(1)'$$

where $\mathbf{\Psi}(1) = \sum_{k=0}^{\infty} \mathbf{\Psi}_k$.

VAR Estimate of the Long-Run Variance

The Wold representation (3.29) may be approximated by high order VAR(p^*) model

$$\mathbf{Y}_t = \mathbf{c} + \mathbf{\Phi}_1 \mathbf{Y}_{t-1} + \cdots + \mathbf{\Phi}_{p^*} \mathbf{Y}_{t-p^*} + \boldsymbol{\varepsilon}_t$$

where the lag length p^* is chosen such $p^* = O(T^{1/3})$. This gives rise to the *autoregressive long-run variance matrix* estimate

$$\begin{aligned} \widehat{\mathrm{lrv}}_{\mathrm{AR}}(\mathbf{Y}_t) &= \hat{\mathbf{\Psi}}(1)\hat{\mathbf{\Sigma}}\hat{\mathbf{\Psi}}(1)' \\ \hat{\mathbf{\Psi}}(1) &= (\mathbf{I}_n - \hat{\mathbf{\Phi}}_1 - \cdots - \hat{\mathbf{\Phi}}_p)^{-1} \\ \hat{\mathbf{\Sigma}} &= \frac{1}{T}\sum_{t=1}^{T} \hat{\varepsilon}_t \hat{\varepsilon}_t' \end{aligned}$$

where $\hat{\mathbf{\Phi}}_k$ $(k = 1, \ldots, p^*)$ are estimates of the VAR parameter matrices.

Non-parametric Estimate of the Long-Run Variance

A consistent estimate of $\mathrm{lrv}(\mathbf{Y}_t)$ may be computed using non-parametric methods. A popular estimator is the Newey-West weighted autocovariance estimator

$$\widehat{\mathrm{lrv}}_{\mathrm{NW}}(\mathbf{Y}_t) = \hat{\mathbf{\Gamma}}_0 + \sum_{j=1}^{M_T} w_{j,T} \cdot \left(\hat{\mathbf{\Gamma}}_j + \hat{\mathbf{\Gamma}}_j'\right) \tag{3.30}$$

where $w_{j,T}$ are weights which sum to unity and M_T is a truncation lag parameter that satisfies $M_T = O(T^{1/3})$.

Example 18 *Newey-West estimate of long-run variance matrix for stock returns*

The `S+FinMetrics` function `asymp.var` may be used to compute the Newey-West long-run variance estimate (3.30) for a multivariate time series. The long-run variance matrix for the first four Dow Jones assets in the "`timeSeries`" `Y` is

```
> M.T = floor(4*(nrow(Y)/100)^(2/9))
> lrv.nw = asymp.var(Y,bandwidth=M.T)
> lrv.nw
            AA        AXP         T        BA
 AA 0.00037313 0.00008526 3.754e-005 6.685e-005
AXP 0.00008526 0.00034957 7.937e-005 1.051e-004
  T 0.00003754 0.00007937 3.707e-004 7.415e-006
 BA 0.00006685 0.00010506 7.415e-006 3.087e-004
```

3.6 References

ALEXANDER, C. (2001). *Market Models. A Guide to Financial Data Analysis.* John Wiley & Sons, Chichester, UK.

BAILLE, R.T. AND T. BOLLERSLEV (1994). "The Long Memory of the Forward Premium," *Journal of International Money and Finance*, 13, 555-571.

BOX, G.E.P. AND G.M. JENKINS (1976). *Time Series Analysis, Forecasting and Control. Revised Edition.* Holden Day, San Francisco.

CAMPBELL, J.Y., A.W. LO, A.C. MACKINLAY (1997). *The Econometrics of Financial Markets.* Princeton University Press, Princeton, NJ.

BOX, G.E.P., AND D.A. PIERCE (1970). "Distribution of Residual Autocorrelations in Autoregressive-integrated Moving Average Time Series Models," *Journal of the American Statistical Association*, 65, 1509-1526.

CHAN, N.H. (2002). *Time Series: Applicatios to Finance.* John Wiley & Sons, New York.

DING, Z., C.W.J. GRANGER AND R.F. ENGLE (1993). "A Long Memory Property of Stock Returns and a New Model," *Journal of Empirical Finance*, 1, 83-106.

ENGLE, R.F. (1982). "Autoregressive Conditional Heteroskedasticity with Estimates of the Variance of United Kingdom Inflations," *Econometrica*, 50, 987-1097.

FULLER, W.A. (1996). *Introduction to Statistical Time Series, Second Edition.* John Wiley & Sons, New York.

GOURIEROUX, C AND J. JASIAK (2001). *Financial Econometrics.* Princeton University Press, Princeton, NJ.

GRANGER, C.W.J. AND M.J. MORRIS (1976). "Time Series Modeling and Interpretation," *Journal of the Royal Statistical Society*, Series A, 139, 246-257.

HAMILTON, J.D. (1994). *Time Series Analysis.* Princeton University Press, Princeton, NJ.

HARVEY, A.C. (1993). *Time Series Models, Second Edition.* MIT Press, Cambridge, MA.

Hayashi, F. (2000). *Econometrics.* Princeton University Press, Princeton, NJ.

Ljung, T. and G.E.P. Box (1979). "The Likelihood Function for a Stationary Autoregressive Moving Average Process," *Biometrika*, 66, 265-270.

Mills, T.C. (1999). *The Econometric Modeling of Financial Time Series, Second Edition.* Cambridge University Press, Cambridge, UK.

Newey, W.K. and K.D. West (1987). "A Simple Positive Semidefinite Heteroskedasticity and Autocorrelation Consistent Covariance Matrix," *Econometrica*, 55, 703-708.

Tsay, R.S. (2001). *Analysis of Financial Time Series*, John Wiley & Sons, New York.

Venables, W.N. and B.D. Ripley (2002). *Modern Applied Statistics with S-PLUS,* Fourth Edition. Springer-Verlag, New York.

White, H. (1984). *Asymptotic Theory for Econometrians.* Academic Press, San Diego.

4
Unit Root Tests

4.1 Introduction

Many economic and financial time series exhibit trending behavior or nonstationarity in the mean. Leading examples are asset prices, exchange rates and the levels of macroeconomic aggregates like real GDP. An important econometric task is determining the most appropriate form of the trend in the data. For example, in ARMA modeling the data must be transformed to stationary form prior to analysis. If the data are trending, then some form of trend removal is required.

Two common trend removal or de-trending procedures are first differencing and time-trend regression. First differencing is appropriate for $I(1)$ time series and time-trend regression is appropriate for trend stationary $I(0)$ time series. Unit root tests can be used to determine if trending data should be first differenced or regressed on deterministic functions of time to render the data stationary. Moreover, economic and finance theory often suggests the existence of long-run equilibrium relationships among nonstationary time series variables. If these variables are $I(1)$, then cointegration techniques can be used to model these long-run relations. Hence, pre-testing for unit roots is often a first step in the cointegration modeling discussed in Chapter 12. Finally, a common trading strategy in finance involves exploiting mean-reverting behavior among the prices of pairs of assets. Unit root tests can be used to determine which pairs of assets appear to exhibit mean-reverting behavior.

This chapter is organized as follows. Section 4.2 reviews $I(1)$ and trend stationary $I(0)$ time series and motivates the unit root and stationary tests described in the chapter. Section 4.3 describes the class of autoregressive unit root tests made popular by David Dickey, Wayne Fuller, Pierre Perron and Peter Phillips. Section 4.4 describes the stationarity tests of Kwiatkowski, Phillips, Schmidt and Shinn (1992). Section 4.5 discusses some problems associated with traditional unit root and stationarity tests, and Section 4.6 presents some recently developed so-called "efficient unit root tests" that overcome some of the deficiencies of traditional unit root tests.

In this chapter, the technical details of unit root and stationarity tests are kept to a minimum. Excellent technical treatments of nonstationary time series may be found in Hamilton (1994), Hatanaka (1995), Fuller (1996) and the many papers by Peter Phillips. Useful surveys on issues associated with unit root testing are given in Stock (1994), Maddala and Kim (1998) and Phillips and Xiao (1998).

4.2 Testing for Nonstationarity and Stationarity

To understand the econometric issues associated with unit root and stationarity tests, consider the stylized trend-cycle decomposition of a time series y_t:

$$\begin{aligned} y_t &= TD_t + z_t \\ TD_t &= \kappa + \delta t \\ z_t &= \phi z_{t-1} + \varepsilon_t,\ \varepsilon_t \sim WN(0, \sigma^2) \end{aligned}$$

where TD_t is a deterministic linear trend and z_t is an AR(1) process. If $|\phi| < 1$ then y_t is $I(0)$ about the deterministic trend TD_t. If $\phi = 1$, then $z_t = z_{t-1} + \varepsilon_t = z_0 + \sum_{j=1}^{t} \varepsilon_j$, a stochastic trend and y_t is $I(1)$ with drift. Simulated $I(1)$ and $I(0)$ data with $\kappa = 5$ and $\delta = 0.1$ are illustrated in Figure 4.1. The $I(0)$ data with trend follows the trend $TD_t = 5 + 0.1t$ very closely and exhibits trend reversion. In contrast, the $I(1)$ data follows an upward drift but does not necessarily revert to TD_t.

Autoregressive *unit root tests* are based on testing the null hypothesis that $\phi = 1$ (difference stationary) against the alternative hypothesis that $\phi < 1$ (trend stationary). They are called unit root tests because under the null hypothesis the autoregressive polynomial of z_t, $\phi(z) = (1 - \phi z) = 0$, has a root equal to unity.

Stationarity tests take the null hypothesis that y_t is trend stationary. If y_t is then first differenced it becomes

$$\begin{aligned} \Delta y_t &= \delta + \Delta z_t \\ \Delta z_t &= \phi \Delta z_{t-1} + \varepsilon_t - \varepsilon_{t-1} \end{aligned}$$

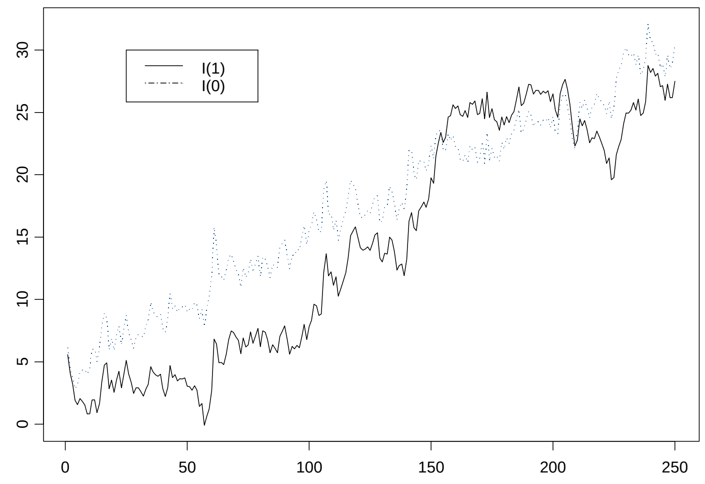


FIGURE 4.1. Simulated trend stationary ($I(0)$) and difference stationary ($I(1)$) processes.

Notice that first differencing y_t, when it is trend stationary, produces a unit moving average root in the ARMA representation of Δz_t. That is, the ARMA representation for Δz_t is the non-invertible ARMA(1,1) model

$$\Delta z_t = \phi \Delta z_{t-1} + \varepsilon_t + \theta \varepsilon_{t-1}$$

with $\theta = -1$. This result is known as *overdifferencing*. Formally, stationarity tests are based on testing for a unit moving average root in Δz_t.

Unit root and stationarity test statistics have nonstandard and nonnormal asymptotic distributions under their respective null hypotheses. To complicate matters further, the limiting distributions of the test statistics are affected by the inclusion of deterministic terms in the test regressions. These distributions are functions of standard Brownian motion (Wiener process), and critical values must be tabulated by simulation techniques. MacKinnon (1996) provided response surface algorithms for determining these critical values, and various S+FinMetrics functions use these algorithms for computing critical values and p-values.

4.3 Autoregressive Unit Root Tests

To illustrate the important statistical issues associated with autoregressive unit root tests, consider the simple AR(1) model

$$y_t = \phi y_{t-1} + \varepsilon_t, \text{ where } \varepsilon_t \sim WN(0, \sigma^2)$$

The hypotheses of interest are

$$\begin{aligned} H_0 &: \phi = 1 \text{ (unit root in } \phi(z) = 0) \Rightarrow y_t \sim I(1) \\ H_1 &: |\phi| < 1 \Rightarrow y_t \sim I(0) \end{aligned}$$

The test statistic is

$$t_{\phi=1} = \frac{\hat{\phi} - 1}{\mathrm{SE}(\hat{\phi})}$$

where $\hat{\phi}$ is the least squares estimate and $\mathrm{SE}(\hat{\phi})$ is the usual standard error estimate[1]. The test is a one-sided left tail test. If $\{y_t\}$ is stationary (i.e., $|\phi| < 1$) then it can be shown (c.f. Hamilton 1994, p. 216)

$$\sqrt{T}(\hat{\phi} - \phi) \xrightarrow{d} N(0, (1 - \phi^2))$$

or

$$\hat{\phi} \overset{A}{\sim} N\left(\phi, \frac{1}{T}(1 - \phi^2)\right)$$

and it follows that $t_{\phi=1} \overset{A}{\sim} N(0, 1)$. However, under the null hypothesis of nonstationarity the above result gives

$$\hat{\phi} \overset{A}{\sim} N(1, 0)$$

which clearly does not make any sense. The problem is that under the unit root null, $\{y_t\}$ is not stationary and ergodic, and the usual sample moments do not converge to fixed constants. Instead, Phillips (1987) showed that the sample moments of $\{y_t\}$ converge to random functions of Brownian

[1]The AR(1) model may be re-written as $\Delta y_t = \pi y_{t-1} + u_t$ where $\pi = \phi - 1$. Testing $\phi = 1$ is then equivalent to testing $\pi = 0$. Unit root tests are often computed using this alternative regression and the `S+FinMetrics` function `unitroot` follows this convention.

motion[2]:

$$T^{-3/2}\sum_{t=1}^{T} y_{t-1} \xrightarrow{d} \sigma \int_0^1 W(r)dr$$
$$T^{-2}\sum_{t=1}^{T} y_{t-1}^2 \xrightarrow{d} \sigma^2 \int_0^1 W(r)^2 dr$$
$$T^{-1}\sum_{t=1}^{T} y_{t-1}\varepsilon_t \xrightarrow{d} \sigma^2 \int_0^1 W(r)dW(r)$$

where $W(r)$ denotes a standard Brownian motion (Wiener process) defined on the unit interval. Using the above results Phillips showed that under the unit root null $H_0 : \phi = 1$

$$T(\hat{\phi}-1) \xrightarrow{d} \frac{\int_0^1 W(r)dW(r)}{\int_0^1 W(r)^2 dr} \tag{4.1}$$

$$t_{\phi=1} \xrightarrow{d} \frac{\int_0^1 W(r)dW(r)}{\left(\int_0^1 W(r)^2 dr\right)^{1/2}} \tag{4.2}$$

The above yield some surprising results:

- $\hat{\phi}$ is *super-consistent*; that is, $\hat{\phi} \xrightarrow{p} \phi$ at rate T instead of the usual rate $T^{1/2}$.
- $\hat{\phi}$ is not asymptotically normally distributed and $t_{\phi=1}$ is not asymptotically standard normal.
- The limiting distribution of $t_{\phi=1}$ is called the Dickey-Fuller (DF) distribution and does not have a closed form representation. Consequently, quantiles of the distribution must be computed by numerical approximation or by simulation[3].
- Since the *normalized bias* $T(\hat{\phi}-1)$ has a well defined limiting distribution that does not depend on nuisance parameters it can also be used as a test statistic for the null hypothesis $H_0 : \phi = 1$.

[2] A Wiener process $W(\cdot)$ is a continuous-time stochastic process, associating each date $r \in [0,1]$ a scalar random variable $W(r)$ that satisfies: (1) $W(0) = 0$; (2) for any dates $0 \leq t_1 \leq \cdots \leq t_k \leq 1$ the changes $W(t_2)-W(t_1), W(t_3)-W(t_2), \ldots, W(t_k)-W(t_{k-1})$ are independent normal with $W(s)-W(t) \sim N(0,(s-t))$; (3) $W(s)$ is continuous in s.

[3] Dickey and Fuller (1979) first considered the unit root tests and derived the asymptotic distribution of $t_{\phi=1}$. However, their representation did not utilize functions of Wiener processes.

4.3.1 Simulating the DF and Normalized Bias Distributions

As mentioned above, the DF and normalized bias distributions must be obtained by simulation methods. To illustrate, the following S-PLUS function wiener produces one random draw from the functions of Brownian motion that appear in the limiting distributions of $t_{\phi=1}$ and $T(\hat{\phi}-1)$:

```
wiener = function(nobs) {
   e = rnorm(nobs)
   y = cumsum(e)
   ym1 = y[1:(nobs-1)]
   intW2 = nobs^(-2) * sum(ym1^2)
   intWdW = nobs^(-1) * sum(ym1*e[2:nobs])
   ans = list(intW2=intW2,
              intWdW=intWdW)
   ans
}
```

A simple loop then produces the simulated distributions:

```
> nobs = 1000
> nsim = 1000
> NB = rep(0,nsim)
> DF = rep(0,nsim)
> for (i in 1:nsim) {
+    BN.moments = wiener(nobs)
+    NB[i] = BN.moments$intWdW/BN.moments$intW2
+    DF[i] = BN.moments$intWdW/sqrt(BN.moments$intW2)
}
```

Figure 4.2 shows the histograms and density estimates of the simulated distributions. The DF density is slightly left-skewed relative to the standard normal, and the normalized bias density is highly left skewed and non-normal. Since the alternative is one-sided, the test statistics reject if they are sufficiently negative. For the DF and normalized bias densities the empirical 1%, 5% and 10% quantiles are

```
> quantile(DF,probs=c(0.01,0.05,0.1))
     1%      5%     10%
 -2.451 -1.992 -1.603
> quantile(NB,probs=c(0.01,0.05,0.1))
     1%    5%    10%
 -11.94 -8.56 -5.641
```

For comparison purposes, note that the 5% quantile from the standard normal distribution is -1.645.

The simulation of critical values and p-values from (4.1) and (4.2) is straightforward but time consuming. The punitroot and qunitroot func-

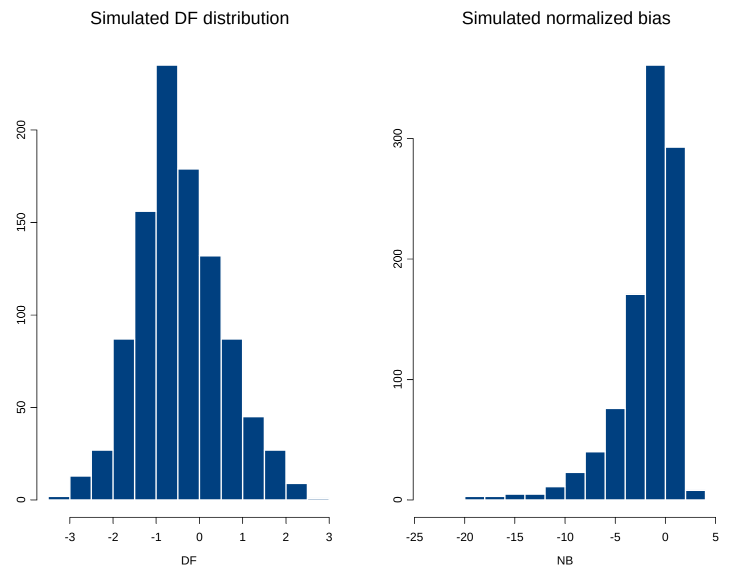


FIGURE 4.2. Histograms of simulated DF and normalized bias distributions.

tions in S+FinMetrics produce p-values and quantiles of the DF and normalized bias distributions based on MacKinnon's (1996) response surface methodology. The advantage of the response surface methodology is that accurate p-values and quantiles appropriate for a given sample size can be produced. For example, the 1%, 5% and 10% quantiles for (4.2) and (4.1) based on a sample size of 100 are

```
> qunitroot(c(0.01,0.05,0.10), trend="nc", statistic="t",
+ n.sample=100)
[1] -2.588 -1.944 -1.615
> qunitroot(c(0.01,0.05,0.10), trend="nc", statistic="n",
+ n.sample=100)
[1] -13.086  -7.787  -5.565
```

The argument `trend="nc"` specifies that no constant is included in the test regression. Other valid options are `trend="c"` for constant only and `trend="ct"` for constant and trend. These trend cases are explained below. To specify the normalized bias distribution, set `statistic="n"`. For asymptotic quantiles set `n.sample=0`.

Similarly, the *p*-value of -1.645 based on the DF distribution for a sample size of 100 is computed as

```
> punitroot(-1.645, trend="nc", statistic="t")
[1] 0.0945
```

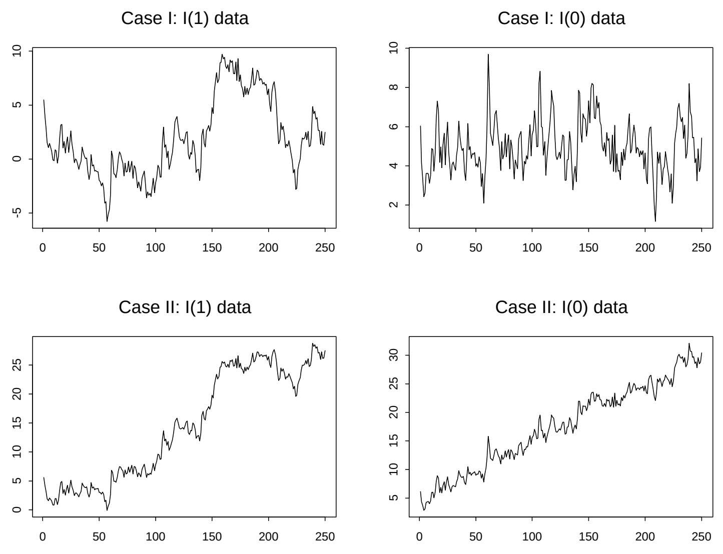


FIGURE 4.3. Simulated $I(1)$ and $I(0)$ data under trend cases I and II.

4.3.2 Trend Cases

When testing for unit roots, it is crucial to specify the null and alternative hypotheses appropriately to characterize the trend properties of the data at hand. For example, if the observed data does not exhibit an increasing or decreasing trend, then the appropriate null and alternative hypotheses should reflect this. The trend properties of the data *under the alternative hypothesis* will determine the form of the test regression used. Furthermore, the type of deterministic terms in the test regression will influence the asymptotic distributions of the unit root test statistics. The two most common trend cases are summarized below and illustrated in Figure 4.3.

Case I: Constant Only

The test regression is

$$y_t = c + \phi y_{t-1} + \varepsilon_t$$

and includes a constant to capture the nonzero mean under the alternative. The hypotheses to be tested are

$$\begin{aligned} H_0 &: \phi = 1 \Rightarrow y_t \sim I(1) \text{ without drift} \\ H_1 &: |\phi| < 1 \Rightarrow y_t \sim I(0) \text{ with nonzero mean} \end{aligned}$$

This formulation is appropriate for non-trending financial series like interest rates, exchange rates, and spreads. The test statistics $t_{\phi=1}$ and $T(\hat{\phi} - 1)$

are computed from the above regression. Under $H_0 : \phi = 1$ the asymptotic distributions of these test statistics are different from (4.2) and (4.1) and are influenced by the presence but not the coefficient value of the constant in the test regression. Quantiles and p-values for these distributions can be computed using the S+FinMetrics functions punitroot and qunitroot with the trend="c" option:

```
> qunitroot(c(0.01,0.05,0.10), trend="c", statistic="t",
+ n.sample=100)
[1] -3.497 -2.891 -2.582
> qunitroot(c(0.01,0.05,0.10), trend="c", statistic="n",
+ n.sample=100)
[1] -19.49 -13.53 -10.88
> punitroot(-1.645, trend="c", statistic="t", n.sample=100)
[1] 0.456
> punitroot(-1.645, trend="c", statistic="n", n.sample=100)
[1] 0.8172
```

For a sample size of 100, the 5% left tail critical values for $t_{\phi=1}$ and $T(\hat{\phi} - 1)$ are -2.891 and -13.53, respectively, and are quite a bit smaller than the 5% critical values computed when trend="nc". Hence, inclusion of a constant pushes the distributions of $t_{\phi=1}$ and $T(\hat{\phi} - 1)$ to the left.

Case II: Constant and Time Trend

The test regression is

$$y_t = c + \delta t + \phi y_{t-1} + \varepsilon_t$$

and includes a constant and deterministic time trend to capture the deterministic trend under the alternative. The hypotheses to be tested are

$$\begin{aligned} H_0 &: \phi = 1 \ \Rightarrow y_t \sim I(1) \text{ with drift} \\ H_1 &: |\phi| < 1 \Rightarrow y_t \sim I(0) \text{ with deterministic time trend} \end{aligned}$$

This formulation is appropriate for trending time series like asset prices or the levels of macroeconomic aggregates like real GDP. The test statistics $t_{\phi=1}$ and $T(\hat{\phi} - 1)$ are computed from the above regression. Under $H_0 : \phi = 1$ the asymptotic distributions of these test statistics are different from (4.2) and (4.1) and are influenced by the presence but not the coefficient values of the constant and time trend in the test regression. Quantiles and p-values for these distributions can be computed using the S+FinMetrics functions punitroot and qunitroot with the trend="ct" option:

```
> qunitroot(c(0.01,0.05,0.10), trend="ct", statistic="t",
+ n.sample=100)
[1] -4.052 -3.455 -3.153
```

```
> qunitroot(c(0.01,0.05,0.10), trend="ct", statistic="n",
+ n.sample=100)
[1] -27.17 -20.47 -17.35
> punitroot(-1.645, trend="ct", statistic="t", n.sample=100)
[1] 0.7679
> punitroot(-1.645, trend="ct", statistic="n", n.sample=100)
[1] 0.9769
```

Notice that the inclusion of a constant and trend in the test regression further shifts the distributions of $t_{\phi=1}$ and $T(\hat{\phi}-1)$ to the left. For a sample size of 100, the 5% left tail critical values for $t_{\phi=1}$ and $T(\hat{\phi}-1)$ are now -3.455 and -20.471.

4.3.3 Dickey-Fuller Unit Root Tests

The unit root tests described above are valid if the time series y_t is well characterized by an AR(1) with white noise errors. Many financial time series, however, have a more complicated dynamic structure than is captured by a simple AR(1) model. Said and Dickey (1984) augment the basic autoregressive unit root test to accommodate general ARMA(p, q) models with unknown orders and their test is referred to as the *augmented Dickey-Fuller* (ADF) test. The ADF test tests the null hypothesis that a time series y_t is $I(1)$ against the alternative that it is $I(0)$, assuming that the dynamics in the data have an ARMA structure. The ADF test is based on estimating the test regression

$$y_t = \boldsymbol{\beta}'\mathbf{D}_t + \phi y_{t-1} + \sum_{j=1}^{p} \psi_j \Delta y_{t-j} + \varepsilon_t \tag{4.3}$$

where $\mathbf{D}_t$ is a vector of deterministic terms (constant, trend etc.). The p lagged difference terms, Δy_{t-j}, are used to approximate the ARMA structure of the errors, and the value of p is set so that the error ε_t is serially uncorrelated. The error term is also assumed to be homoskedastic. The specification of the deterministic terms depends on the assumed behavior of y_t under the alternative hypothesis of trend stationarity as described in the previous section. Under the null hypothesis, y_t is $I(1)$ which implies that $\phi = 1$. The ADF t-statistic and normalized bias statistic are based on the least squares estimates of (4.3) and are given by

$$\begin{aligned} \text{ADF}_t &= t_{\phi=1} = \frac{\hat{\phi} - 1}{\text{SE}(\phi)} \\ \text{ADF}_n &= \frac{T(\hat{\phi} - 1)}{1 - \hat{\psi}_1 - \cdots - \hat{\psi}_p} \end{aligned}$$

An alternative formulation of the ADF test regression is

$$\Delta y_t = \boldsymbol{\beta}'\mathbf{D}_t + \pi y_{t-1} + \sum_{j=1}^{p} \psi_j \Delta y_{t-j} + \varepsilon_t \tag{4.4}$$

where $\pi = \phi - 1$. Under the null hypothesis, Δy_t is $I(0)$ which implies that $\pi = 0$. The ADF t-statistic is then the usual t-statistic for testing $\pi = 0$ and the ADF normalized bias statistic is $T\hat{\pi}/(1 - \hat{\psi}_1 - \cdots - \hat{\psi}_p)$. The test regression (4.4) is often used in practice because the ADF t-statistic is the usual t-statistic reported for testing the significance of the coefficient y_{t-1}. The S+FinMetrics function unitroot follows this convention.

Choosing the Lag Length for the ADF Test

An important practical issue for the implementation of the ADF test is the specification of the lag length p. If p is too small then the remaining serial correlation in the errors will bias the test. If p is too large then the power of the test will suffer. Ng and Perron (1995) suggest the following data dependent lag length selection procedure that results in stable size of the test and minimal power loss. First, set an upper bound $p_{\max}$ for p. Next, estimate the ADF test regression with $p = p_{\max}$. If the absolute value of the t-statistic for testing the significance of the last lagged difference is greater than 1.6 then set $p = p_{\max}$ and perform the unit root test. Otherwise, reduce the lag length by one and repeat the process.

A useful rule of thumb for determining $p_{\max}$, suggested by Schwert (1989), is

$$p_{\max} = \left[12 \cdot \left(\frac{T}{100} \right)^{1/4} \right] \tag{4.5}$$

where $[x]$ denotes the integer part of x. This choice allows $p_{\max}$ to grow with the sample so that the ADF test regressions (4.3) and (4.4) are valid if the errors follow an ARMA process with unknown order.

Example 19 *Testing for a unit root in exchange rate data using ADF tests*

To illustrate the ADF test procedure, consider testing for a unit root in the logarithm of the US/CA monthly spot exchange rate, denoted s_t, over the 30 year period 1976 - 1996. Figure 4.4 shows $s_t, \Delta s_t$ as well as the sample autocorrelations for these series. The data and plots are created with the S-PLUS commands

```
> uscn.spot = lexrates.dat[,"USCNS"]
> uscn.spot@title = "Log US/CN spot exchange rate"
> par(mfrow=c(2,2))
> plot.timeSeries(uscn.spot, reference.grid=F,
+ main="Log of US/CN spot exchange rate")
```

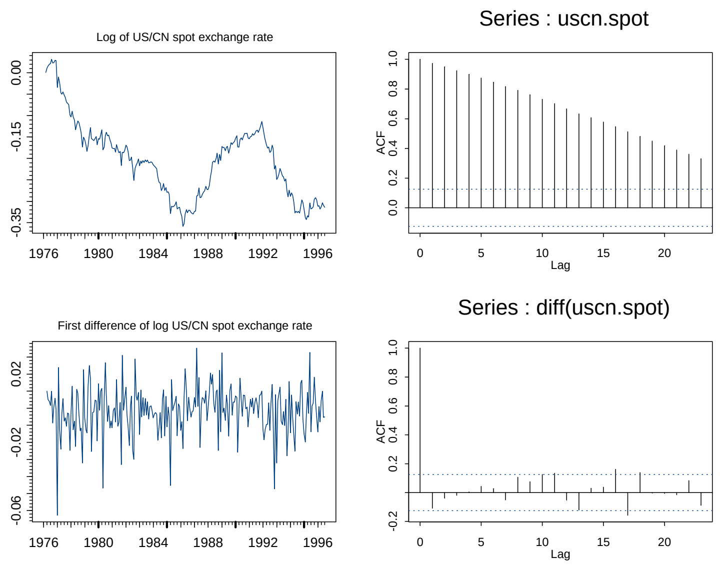


FIGURE 4.4. US/CN spot rate, first difference and SACF.

```
> xx = acf(uscn.spot)
> plot.timeSeries(diff(uscn.spot), reference.grid=F,
+ main="First difference of log US/CN spot exchange rate")
> xx = acf(diff(uscn.spot))
```

Clearly, s_t exhibits random walk like behavior with no apparent positive or negative drift. However, Δs_t behaves very much like a white noise process. The appropriate trend specification is to include a constant in the test regression. Regarding the maximum lag length for the Ng-Perron procedure, given the lack of serial correlation in Δs_t a conservative choice is $p_{\max} = 6$. The ADF t-statistic computed from the test regression with a constant and $p = 6$ lags can be computed using the S+FinMetrics function unitroot as follows

```
> adft.out = unitroot(uscn.spot, trend="c", statistic="t",
+ method="adf", lags=6)
> class(adft.out)
[1] "unitroot"
```

The output of unitroot is an object of class “unitroot” for which there are print and summary methods. Typing the name of the object invokes the print method and displays the basic test result

```
> adft.out
Test for Unit Root: Augmented DF Test
```

```
Null Hypothesis: there is a unit root
   Type of Test: t-test
 Test Statistic: -2.6
        P-value: 0.09427

Coefficients:
    lag1    lag2    lag3    lag4    lag5    lag6 constant
 -0.0280 -0.1188 -0.0584 -0.0327 -0.0019  0.0430 -0.0075

Degrees of freedom: 239 total; 232 residual
Time period: from Aug 1976 to Jun 1996
Residual standard error: 0.01386
```

With $p = 6$ the ADF t-statistic is -2.6 and has a p-value (computed using `punitroot`) of 0.094. Hence we do not reject the unit root null at the 9.4% level. The small p-value here may be due to the inclusion of superfluous lags. To see the significance of the lags in the test regression, use the `summary` method

```
> summary(adft.out)
Test for Unit Root: Augmented DF Test

Null Hypothesis: there is a unit root
   Type of Test: t test
 Test Statistic: -2.6
        P-value: 0.09427

Coefficients:
           Value Std. Error t value Pr(>|t|)
    lag1 -0.0280  0.0108    -2.6004  0.0099
    lag2 -0.1188  0.0646    -1.8407  0.0669
    lag3 -0.0584  0.0650    -0.8983  0.3700
    lag4 -0.0327  0.0651    -0.5018  0.6163
    lag5 -0.0019  0.0651    -0.0293  0.9766
    lag6  0.0430  0.0645     0.6662  0.5060
constant -0.0075  0.0024    -3.0982  0.0022

Regression Diagnostics:

         R-Squared 0.0462
Adjusted R-Squared 0.0215
Durbin-Watson Stat 2.0033

Residual standard error: 0.01386 on 235 degrees of freedom
F-statistic: 1.874 on 6 and 232 degrees of freedom, the
```

```
p-value is 0.08619
Time period: from Aug 1976 to Jun 1996
```

The results indicate that too many lags have been included in the test regression. Following the Ng-Perron backward selection procedure $p = 2$ lags are selected. The results are

```
> adft.out = unitroot(uscn.spot, trend="c", lags=2)
> summary(adft.out)
Test for Unit Root: Augmented DF Test

Null Hypothesis: there is a unit root
   Type of Test: t test
 Test Statistic: -2.115
        P-value: 0.2392

Coefficients:
           Value Std. Error t value Pr(>|t|)
    lag1 -0.0214  0.0101    -2.1146  0.0355
    lag2 -0.1047  0.0635    -1.6476  0.1007
constant -0.0058  0.0022    -2.6001  0.0099

Regression Diagnostics:

         R-Squared 0.0299
Adjusted R-Squared 0.0218
Durbin-Watson Stat 2.0145

Residual standard error: 0.01378 on 239 degrees of freedom
F-statistic: 3.694 on 2 and 240 degrees of freedom, the
p-value is 0.02629
Time period: from Apr 1976 to Jun 1996
```

With 2 lags the ADF t-statistic is -2.115, the p-value 0.239 and we have greater evidence for a unit root in s_t. A similar result is found with the ADF normalized bias statistic

```
> adfn.out = unitroot(uscn.spot, trend="c", lags=2,
+ statistic="n")
> adfn.out
Test for Unit Root: Augmented DF Test

Null Hypothesis: there is a unit root
   Type of Test: normalized test
 Test Statistic: -5.193
        P-value: 0.4129
```

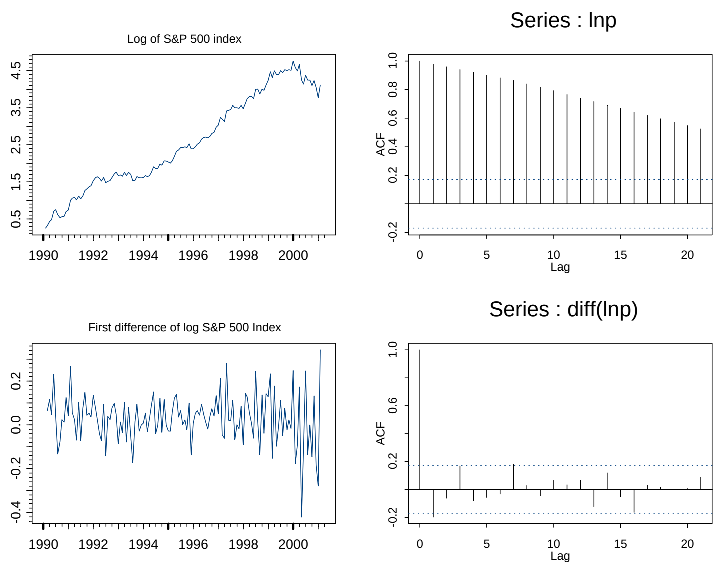


FIGURE 4.5. Log prices on the S&P 500 index, first difference and SACF.

```
Coefficients:
    lag1    lag2 constant
 -0.0214 -0.1047 -0.0058

Degrees of freedom: 243 total; 240 residual
Time period: from Apr 1976 to Jun 1996
Residual standard error: 0.01378
```

Example 20 *Testing for a unit root in log stock prices*

The log levels of asset prices are usually treated as $I(1)$ with drift. Indeed, the random walk model of stock prices is a special case of an $I(1)$ process. Consider testing for a unit root in the log of the monthly S&P 500 index, p_t, over the period January 1990 through January 2001. The data is taken from the `S+FinMetrics` "`timeSeries`" `singleIndex.dat`. The data and various plots are created with the `S-PLUS` commands

```
> lnp = log(singleIndex.dat[,1])
> lnp@title = "Log of S&P 500 Index"
> par(mfrow=c(2,2))
> plot.timeSeries(lnp, reference.grid=F,
+ main="Log of S&P 500 index")
> acf.plot(acf(lnp,plot=F))
> plot.timeSeries(diff(lnp), reference.grid=F,
+ main="First difference of log S&P 500 Index")
> acf.plot(acf(diff(lnp),plot=F))
```

and are illustrated in Figure 4.5. Clearly, the p_t is nonstationary due to the positive trend. Also, there appears to be some negative autocorrelation at lag one in Δp_t. The null hypothesis to be tested is that p_t is $I(1)$ with drift, and the alternative is that the p_t is $I(0)$ about a deterministic time trend. The ADF t-statistic to test these hypotheses is computed with a constant and time trend in the test regression and four lags of Δp_t (selecting using the Ng-Perron backward selection method)

```
> adft.out = unitroot(lnp, trend="ct", lags=4)
> summary(adft.out)

Test for Unit Root: Augmented DF Test

Null Hypothesis: there is a unit root
   Type of Test: t test
 Test Statistic: -1.315
        P-value: 0.8798

Coefficients:
           Value Std. Error t value Pr(>|t|)
    lag1 -0.0540  0.0410    -1.3150  0.1910
    lag2 -0.1869  0.0978    -1.9111  0.0583
    lag3 -0.0460  0.0995    -0.4627  0.6444
    lag4  0.1939  0.0971     1.9964  0.0481
constant  0.1678  0.1040     1.6128  0.1094
    time  0.0015  0.0014     1.0743  0.2848

Regression Diagnostics:

         R-Squared 0.1016
Adjusted R-Squared 0.0651
Durbin-Watson Stat 1.9544

Residual standard error: 0.1087 on 125 degrees of freedom
F-statistic: 2.783 on 5 and 123 degrees of freedom, the
p-value is 0.0204
Time period: from May 1990 to Jan 2001
```

$\text{ADF}_t = -1.315$ and has a p-value of 0.8798, so one clearly does not reject the null that p_t is $I(1)$ with drift.

4.3.4 Phillips-Perron Unit Root Tests

Phillips and Perron (1988) developed a number of unit root tests that have become popular in the analysis of financial time series. The Phillips-Perron (PP) unit root tests differ from the ADF tests mainly in how they deal with serial correlation and heteroskedasticity in the errors. In particular, where the ADF tests use a parametric autoregression to approximate the ARMA structure of the errors in the test regression, the PP tests ignore any serial correlation in the test regression. The test regression for the PP tests is

$$\Delta y_t = \boldsymbol{\beta}'\mathbf{D}_t + \pi y_{t-1} + u_t$$

where u_t is $I(0)$ and may be heteroskedastic. The PP tests correct for any serial correlation and heteroskedasticity in the errors u_t of the test regression by directly modifying the test statistics $t_{\pi=0}$ and $T\hat{\pi}$. These modified statistics, denoted Z_t and Z_π, are given by

$$\begin{aligned} Z_t &= \left(\frac{\hat{\sigma}^2}{\hat{\lambda}^2}\right)^{1/2} \cdot t_{\pi=0} - \frac{1}{2}\left(\frac{\hat{\lambda}^2 - \hat{\sigma}^2}{\hat{\lambda}^2}\right) \cdot \left(\frac{T \cdot \mathrm{SE}(\hat{\pi})}{\hat{\sigma}^2}\right) \\ Z_\pi &= T\hat{\pi} - \frac{1}{2}\frac{T^2 \cdot \mathrm{SE}(\hat{\pi})}{\hat{\sigma}^2}(\hat{\lambda}^2 - \hat{\sigma}^2) \end{aligned}$$

The terms $\hat{\sigma}^2$ and $\hat{\lambda}^2$ are consistent estimates of the variance parameters

$$\begin{aligned} \sigma^2 &= \lim_{T\to\infty} T^{-1}\sum_{t=1}^{T} E[u_t^2] \\ \lambda^2 &= \lim_{T\to\infty} \sum_{t=1}^{T} E\left[T^{-1}S_T^2\right] \end{aligned}$$

where $S_T = \sum_{t=1}^{T} u_t$. The sample variance of the least squares residual $\hat{u}_t$ is a consistent estimate of σ^2, and the Newey-West long-run variance estimate of u_t using $\hat{u}_t$ is a consistent estimate of λ^2.

Under the null hypothesis that $\pi = 0$, the PP Z_t and Z_π statistics have the same asymptotic distributions as the ADF t-statistic and normalized bias statistics. One advantage of the PP tests over the ADF tests is that the PP tests are robust to general forms of heteroskedasticity in the error term u_t. Another advantage is that the user does not have to specify a lag length for the test regression.

Example 21 *Testing for a unit root in exchange rates using the PP tests*

Recall the arguments for the `S+FinMetrics` `unitroot` function are

```
> args(unitroot)
function(x, trend = "c", method = "adf",
statistic = "t",lags = 1, bandwidth = NULL,
window = "bartlett", asymptotic = F, na.rm = F)
```

The PP statistics may be computed by specifying the optional argument `method="pp"`. When `method="pp"` is chosen, the argument `window` specifies the weight function and the argument `bandwidth` determines the lag truncation parameter used in computing the long-run variance parameter λ^2. The default bandwidth is the integer part of $(4 \cdot (T/100))^{2/9}$ where T is the sample size.

Now, consider testing for a unit root in the log of the US/CN spot exchange rate using the PP Z_t and Z_π statistics:

```
> unitroot(uscn.spot, trend="c", method="pp")
Test for Unit Root: Phillips-Perron Test

Null Hypothesis: there is a unit root
   Type of Test: t-test
 Test Statistic: -1.97
        P-value: 0.2999

Coefficients:
    lag1 constant
 -0.0202 -0.0054

Degrees of freedom: 244 total; 242 residual
Time period: from Mar 1976 to Jun 1996
Residual standard error: 0.01383

> unitroot(uscn.spot, trend="c", method="pp", statistic="n")
Test for Unit Root: Phillips-Perron Test

Null Hypothesis: there is a unit root
   Type of Test: normalized test
 Test Statistic: -4.245
        P-value: 0.5087

Coefficients:
    lag1 constant
 -0.0202 -0.0054

Degrees of freedom: 244 total; 242 residual
Time period: from Mar 1976 to Jun 1996
Residual standard error: 0.01383
```

As with the ADF tests, the PP tests do not reject the null that the log of the US/CN spot rate is $I(1)$ at any reasonable significance level.

4.4 Stationarity Tests

The ADF and PP unit root tests are for the null hypothesis that a time series y_t is $I(1)$. Stationarity tests, on the other hand, are for the null that y_t is $I(0)$. The most commonly used stationarity test, the KPSS test, is due to Kwiatkowski, Phillips, Schmidt, and Shin (1992). They derived their test by starting with the model

$$\begin{aligned} y_t &= \boldsymbol{\beta}'\mathbf{D}_t + \mu_t + u_t \\ \mu_t &= \mu_{t-1} + \varepsilon_t,\ \varepsilon_t \sim WN(0, \sigma_\varepsilon^2) \end{aligned} \tag{4.6}$$

where $\mathbf{D}_t$ contains deterministic components (constant or constant plus time trend), u_t is $I(0)$ and may be heteroskedastic. Notice that μ_t is a pure random walk with innovation variance σ_ε^2. The null hypothesis that y_t is $I(0)$ is formulated as $H_0 : \sigma_\varepsilon^2 = 0$, which implies that μ_t is a constant. Although not directly apparent, this null hypothesis also implies a unit moving average root in the ARMA representation of Δy_t. The KPSS test statistic is the Lagrange multiplier (LM) or score statistic for testing $\sigma_\varepsilon^2 = 0$ against the alternative that $\sigma_\varepsilon^2 > 0$ and is given by

$$\text{KPSS} = \left(T^{-2} \sum_{t=1}^{T} \hat{S}_t^2 \right) / \hat{\lambda}^2 \tag{4.7}$$

where $\hat{S}_t = \sum_{j=1}^{t} \hat{u}_j$, $\hat{u}_t$ is the residual of a regression of y_t on $\mathbf{D}_t$ and $\hat{\lambda}^2$ is a consistent estimate of the long-run variance of u_t using $\hat{u}_t$. Under the null that y_t is $I(0)$, Kwiatkowski, Phillips, Schmidt, and Shin showed that KPSS converges to a function of standard Brownian motion that depends on the form of the deterministic terms $\mathbf{D}_t$ but not their coefficient values $\boldsymbol{\beta}$. In particular, if $\mathbf{D}_t = 1$ then

$$\text{KPSS} \xrightarrow{d} \int_0^1 V_1(r)dr \tag{4.8}$$

where $V_1(r) = W(r) - rW(1)$ and $W(r)$ is a standard Brownian motion for $r \in [0, 1]$. If $\mathbf{D}_t = (1, t)'$ then

$$\text{KPSS} \xrightarrow{d} \int_0^1 V_2(r)dr \tag{4.9}$$

where $V_2(r) = W(r) + r(2 - 3r)W(1) + 6r(r^2 - 1)\int_0^1 W(s)ds$. Critical values from the asymptotic distributions (4.8) and (4.9) must be obtained by simulation methods, and these are summarized in Table 4.1.

The stationary test is a one-sided right-tailed test so that one rejects the null of stationarity at the $100 \cdot \alpha\%$ level if the KPSS test statistic (4.7) is greater than the $100 \cdot (1 - \alpha)\%$ quantile from the appropriate asymptotic distribution (4.8) or (4.9).

	Right tail quantiles				
Distribution	0.90	0.925	0.950	0.975	0.99
$\int_0^1 V_1(r)dr$	0.349	0.396	0.446	0.592	0.762
$\int_0^1 V_2(r)dr$	0.120	0.133	0.149	0.184	0.229

TABLE 4.1. Quantiles of the distribution of the KPSS statistic

4.4.1 Simulating the KPSS Distributions

The distributions in (4.8) and (4.9) may be simulated using methods similar to those used to simulate the DF distribution. The following S-PLUS code is used to create the quantiles in Table 4.1:

```
wiener2 = function(nobs) {
   e = rnorm(nobs)
# create detrended errors
   e1 = e - mean(e)
   e2 = residuals(OLS(e~seq(1,nobs)))
# compute simulated Brownian Bridges
   y1 = cumsum(e1)
   y2 = cumsum(e2)
   intW2.1 = nobs^(-2) * sum(y1^2)
   intW2.2 = nobs^(-2) * sum(y2^2)
   ans = list(intW2.1=intW2.1,
              intW2.2=intW2.2)
   ans
}
#
# simulate KPSS distributions
#
> nobs = 1000
> nsim = 10000
> KPSS1 = rep(0,nsim)
> KPSS2 = rep(0,nsim)
> for (i in 1:nsim) {
   BN.moments = wiener2(nobs)
   KPSS1[i] = BN.moments$intW2.1
   KPSS2[i] = BN.moments$intW2.2
}
#
# compute quantiles of distribution
#
> quantile(KPSS1, probs=c(0.90,0.925,0.95,0.975,0.99))
   90.0%   92.5%   95.0%   97.5%   99.0%
 0.34914 0.39634 0.46643 0.59155 0.76174
> quantile(KPSS2, probs=c(0.90,0.925,0.95,0.975,0.99))
```

```
  90.0%  92.5%   95.0%  97.5%   99.0%
0.12003 0.1325 0.14907 0.1841 0.22923
```

Currently, only asymptotic critical values are available for the KPSS test.

4.4.2 *Testing for Stationarity Using the* `S+FinMetrics` *Function* `stationaryTest`

The `S+FinMetrics` function `stationaryTest` may be used to test the null hypothesis that a time series y_t is $I(0)$ based on the KPSS statistic (4.7). The function `stationaryTest` has arguments

```
> args(stationaryTest)
function(x, trend = "c", bandwidth = NULL, na.rm = F)
```

where `x` represents a univariate vector or "`timeSeries`". The argument `trend` specifies the deterministic trend component in (4.6) and valid choices are `"c"` for a constant and `"ct"` for a constant and time trend. The argument `bandwidth` determines the lag truncation parameter used in computing the long-run variance parameter λ^2. The default bandwidth is the integer part of $(4 \cdot (T/100))^{2/9}$ where T is the sample size. The output of `stationaryTest` is an object of class "`stationaryTest`" for which there is only a `print` method. The use of `stationaryTest` is illustrated with the following example.

Example 22 *Testing for stationarity in exchange rates*

Consider the US/CN spot exchange data used in the previous examples. To test the null that s_t is $I(0)$, the KPSS statistic is computed using a constant in (4.6):

```
> kpss.out = stationaryTest(uscn.spot, trend="c")
> class(kpss.out)
[1] "stationaryTest"
> kpss.out

Test for Stationarity: KPSS Test

Null Hypothesis: stationary around a constant

Test Statistics:
    USCNS
 1.6411**

 * : significant at 5% level
** : significant at 1% level
```

```
 Total Observ.: 245
    Bandwidth : 5
```

The KPSS statistic is 1.641 and is greater than the 99% quantile, 0.762, from Table.4.1. Therefore, the null that s_t is $I(0)$ is rejected at the 1% level.

4.5 Some Problems with Unit Root Tests

The ADF and PP tests are asymptotically equivalent but may differ substantially in finite samples due to the different ways in which they correct for serial correlation in the test regression. In particular, Schwert (1989) found that if Δy_t has an ARMA representation with a large and negative MA component, then the ADF and PP tests are severely size distorted (reject $I(1)$ null much too often when it is true) and that the PP tests are more size distorted than the ADF tests. Recently, Perron and Ng (1996) have suggested useful modifications to the PP tests to mitigate this size distortion. Caner and Killian (2001) have found similar problems with the KPSS test.

In general, the ADF and PP tests have very low power against $I(0)$ alternatives that are close to being $I(1)$. That is, unit root tests cannot distinguish highly persistent stationary processes from nonstationary processes very well. Also, the power of unit root tests diminish as deterministic terms are added to the test regressions. That is, tests that include a constant and trend in the test regression have less power than tests that only include a constant in the test regression. For maximum power against very persistent alternatives the recent tests proposed by Elliot, Rothenberg, and Stock (1996), and Ng and Perron (2001) should be used. These tests are described in the next section.

4.6 Efficient Unit Root Tests

Assume that T observations are generated by

$$y_t = \boldsymbol{\beta}'\mathbf{D}_t + u_t,\ u_t = \phi u_{t-1} + v_t$$

where $\mathbf{D}_t$ represents a vector of deterministic terms, $E[u_0] < \infty$, and v_t is a 1-summable linear process with long-run variance λ^2. Typically $D_t = 1$ or $\mathbf{D}_t = [1, t]$. Consider testing the null hypothesis $\phi = 1$ versus $|\phi| < 1$. If the distribution of the data were known then the Neyman-Pearson Lemma gives the test with best power against any point alternative $\bar{\phi}$. The power of this optimal test plotted as a function of $\bar{\phi}$ gives an upper bound (envelope) for the power of any test based on the same distribution of the data. An undesirable feature of this power envelope is that it depends on the specific value of $\bar{\phi}$, so that there is no uniformly most power full test that can be used against all alternatives $|\phi| < 1$. However, using asymptotic approximations based on the local-to-unity alternative $\phi = 1 + c/T$, for $c < 0$, Elliot, Rothenberg, and Stock (2001) (hereafter ERS) derived a class of test statistics that come very close to the power envelope for a wide range of alternatives. These tests are referred to as *efficient unit root tests*, and they can have substantially higher power than the ADF or PP unit root tests especially when ϕ is close to unity.

4.6.1 Point Optimal Tests

The starting point for the class of efficient tests is the feasible test statistic that is optimal (asymptotically) for the point alternative $\bar{\phi} = 1-\bar{c}/T$, $\bar{c} < 0$. This test is constructed as follows. Define the $T-$dimensional column vector $\mathbf{y}_\phi$ and $T \times q$ dimensional matrix $\mathbf{D}_\phi$ by

$$\begin{aligned} \mathbf{y}_\phi &= (y_1, y_2 - \phi y_1, \ldots, y_T - \phi y_{T-1})' \\ \mathbf{D}_\phi &= (\mathbf{D}_1', \mathbf{D}_2' - \phi \mathbf{D}_1', \ldots, \mathbf{D}_T' - \phi \mathbf{D}_{T-1}')' \end{aligned}$$

All of the elements of $\mathbf{y}_\phi$ and $\mathbf{D}_\phi$, except the first, are quasi-differenced using the operator $1 - \phi L$. Next, for any value of ϕ, define $S(\phi)$ as the sum of squared residuals from a least squares regression of $\mathbf{y}_\phi$ on $\mathbf{D}_\phi$. That is,

$$S(\phi) = \tilde{\mathbf{y}}_\phi' \tilde{\mathbf{y}}_\phi$$

where $\tilde{\mathbf{y}}_\phi = \mathbf{y}_\phi - \mathbf{D}_\phi \hat{\boldsymbol{\beta}}_\phi$ and $\hat{\boldsymbol{\beta}}_\phi = (\mathbf{D}_\phi' \mathbf{D}_\phi)^{-1} \mathbf{D}_\phi' \mathbf{y}_\phi$. ERS showed that the feasible *point optimal unit root test* against the alternative $\bar{\phi} = 1 - \bar{c}/T$ has the form

$$P_T = \left[S(\bar{\phi}) - \bar{\phi} S(1)\right] / \hat{\lambda}^2 \tag{4.10}$$

where $\hat{\lambda}^2$ is a consistent estimate of λ^2. ERS derived the asymptotic distribution of P_T for $D_t = 1$ and $D_t = (1, t)$ and provided asymptotic and finite sample critical values for tests of size 1%, 2.5%, 5% and 10%[4].

Through a series of simulation experiments, ERS discovered that if $\bar{\phi} = 1+\bar{c}/T$ is chosen such that the power of P_T is tangent to the power envelope at 50% power then the overall power of P_T, for a wide range of $\bar{\phi}$ values less than unity, is close to the power envelope. For a given sample size T, the value of $\bar{\phi}$ that results in P_T having 50% power depends on $\bar{c}$ and the form of the deterministic terms in $\mathbf{D}_t$. ERS showed that if $D_t = 1$ then $\bar{c} = -7$, and if $\mathbf{D}_t = (1, t)$ then $\bar{c} = -13.5$.

The ERS P_T statistic may be computed using the function `unitroot` with `method="ers"`.

[4]These critical values are given in ERS Table I panels A and B.

4.6.2 DF-GLS Tests

In the construction of the ERS feasible point optimal test (4.10), the unknown parameters $\boldsymbol{\beta}$ of the trend function are efficiently estimated under the alternative model with $\bar{\phi} = 1 + \bar{c}/T$. That is, $\hat{\boldsymbol{\beta}}_{\bar{\phi}} = (\mathbf{D}'_{\bar{\phi}}\mathbf{D}_{\bar{\phi}})^{-1}\mathbf{D}'_{\bar{\phi}}\mathbf{y}_{\bar{\phi}}$. ERS used this insight to derive an efficient version of the ADF t-statistic, which they called the *DF-GLS test*. They constructed this t-statistic as follows. First, using the trend parameters $\hat{\boldsymbol{\beta}}_{\bar{\phi}}$ estimated under the alternative, define the detrended data

$$y_t^d = y_t - \hat{\boldsymbol{\beta}}'_{\bar{\phi}}\mathbf{D}_t$$

ERS called this detrending procedure *GLS detrending*[5]. Next, using the GLS detrended data, estimate by least squares the ADF test regression which omits the deterministic terms

$$\Delta y_t^d = \pi y_{t-1}^d + \sum_{j=1}^{p} \psi_j \Delta y_{t-j}^d + \varepsilon_t \tag{4.11}$$

and compute the t-statistic for testing $\pi = 0$. When $D_t = 1$, ERS showed that the asymptotic distribution of the DF-GLS test is the same as the ADF t-test, but has higher asymptotic power (against local alternatives) than the DF t-test. Furthermore, ERS show that the DF-GLS test has essentially the same asymptotic power as the ERS point optimal test when $\bar{c} = -7$. When $\mathbf{D}_t = (1, t)$ the asymptotic distribution of the DF-GLS test, however, is different from the ADF t-test. ERS and Ng and Perron (2001) provided critical values for the DF-GLS test in this case. ERS showed that the DF-GLS test has the same asymptotic power as the ERS point optimal test with $c = -13.5$, and has higher power than the DF t-test against local alternatives.

The DF-GLS t-test may be computed using the function `unitroot` with `method="dfgls"`.

4.6.3 Modified Efficient PP Tests

Ng and Perron (2001) used the GLS detrending procedure of ERS to create efficient versions of the modified PP tests of Perron and Ng (1996). These *efficient modified PP tests* do not exhibit the severe size distortions of the PP tests for errors with large negative MA or AR roots, and they can have substantially higher power than the PP tests especially when ϕ is close to unity.

[5]For deterministicly trending trend data with ergodic-stationary deviations from trend, Grenander's Theorem gives the result that least squares estimates of the trend parameters ignoring serial correlation are asymptotically equivalent to the generalized least squares estimates that incorporate serial correlation.

Using the GLS detrended data y_t^d, the efficient modified PP tests are defined as

$$\begin{aligned}
\overline{\text{MZ}}_\alpha &= (T^{-1}y_T^d - \hat{\lambda}^2)\left(2T^{-2}\sum_{t=1}^T y_{t-1}^d\right)^{-1} \\
\overline{\text{MSB}} &= \left(T^{-2}\sum_{t=1}^T y_{t-1}^d/\hat{\lambda}^2\right)^{1/2} \\
\overline{\text{MZ}}_t &= \overline{\text{MZ}}_\alpha \times \overline{\text{MSB}}
\end{aligned}$$

The statistics $\overline{\text{MZ}}_\alpha$ and $\overline{\text{MZ}}_t$ are efficient versions of the PP Z_α and Z_t tests that have much smaller size distortions in the presence of negative moving average errors. Ng and Perron derived the asymptotic distributions of these statistics under the local alternative $\phi = 1 - c/T$ for $D_t = 1$ and $\mathbf{D}_t = (1, t)$. In particular, they showed that the asymptotic distribution of $\overline{\text{MZ}}_t$ is the same as the DF-GLS t-test.

The statistic $\overline{\text{MZ}}_t$ may be computed using the function `unitroot` with `method="mpp"`.

4.6.4 Estimating λ^2

Ng and Perron (2001) emphasized that the estimation of the long-run variance λ^2 has important implications for the finite sample behavior of the ERS point optimal test and the efficient modified PP tests. They stressed that an autoregressive estimate of λ^2 should be used to achieve stable finite sample size. They recommended estimating λ^2 from the ADF test regression (4.11) based on the GLS detrended data:

$$\hat{\lambda}_{\text{AR}} = \frac{\hat{\sigma}_p^2}{\left(1 - \hat{\psi}(1)\right)^2}$$

where $\hat{\psi}(1) = \sum_{j=1}^p \hat{\psi}_j$ and $\hat{\sigma}_p^2 = (T-p)^{-1}\sum_{t=p+1}^T \hat{\varepsilon}_t^2$ are obtained from (4.11) by least squares estimation.

4.6.5 Choosing Lag Lengths to Achieve Good Size and Power

Ng and Perron also stressed that good size and power properties of the all the efficient unit root tests rely on the proper choice of the lag length p used for specifying the test regression (4.11). They argued, however, that traditional model selection criteria such as AIC and BIC are not well suited for determining p with integrated data. Ng and Perron suggested the *modified information criteria* (MIC) that selects p as $p_{mic} = \arg\min_{p \le p_{\max}} \text{MIC}(p)$

where

$$\begin{aligned}
\mathrm{MIC}(p) &= \ln(\hat{\sigma}_p^2) + \frac{C_T(\tau_T(p)+p)}{T-p_{\max}} \\
\tau_T(p) &= \frac{\hat{\pi}^2 \sum_{t=p_{\max}+1}^{T} y_{t-1}^d}{\hat{\sigma}_p^2} \\
\hat{\sigma}_p^2 &= \frac{1}{T-p_{\max}} \sum_{t=p_{\max}+1}^{T} \hat{\varepsilon}_t^2
\end{aligned}$$

with $C_T > 0$ and $C_T/T \to 0$ as $T \to \infty$. The maximum lag, $p_{\max}$, may be set using (4.5). The modified AIC (MAIC) results when $C_T = 2$, and the modified BIC (MBIC) results when $C_T = \ln(T - p_{\max})$. Through a series of simulation experiments, Ng and Perron recommended selecting the lag length p by minimizing the MAIC.

Example 23 *Efficient unit root tests*

To illustrate the efficient unit root tests, consider testing for a unit root in the 30-day interest rate differential formed from the difference between monthly US and UK spot exchange rates:

```
> fd = lexrates.dat[,"USUKS"] - lexrates.dat[,"USUKF"]
> colIds(fd) = "USUKFD"
> fd@title = "US/UK 30-day interest rate differential"
```

The interest rate differential, its SACF, and the SACF of its first difference are depicted in Figure 4.6. The graphs clearly show that the interest rate differential has a high degree of persistence, and that there is little persistence in the first difference.

The ERS P_T test, DF-GLS t-test and Ng-Perron $\overline{\mathrm{MZ}}_t$ test all with $D_t = 1$ may be computed using the function `unitroot` as follows:

```
> ers = unitroot(fd,trend="c",method="ers",max.lags=12)
> dfgls = unitroot(fd,trend="c",method="dfgls",max.lags=12)
> mpp = unitroot(fd,trend="c",method="mpp",max.lags=12)
```

Since the optional argument `lags` is omitted, the lag length for the test regression (4.11) is determined by minimizing the MAIC with $p_{\max} = 12$ set by the optional argument `max.lags=12`. The results of the efficient unit root tests are:

```
> ers.test

Test for Unit Root: Elliott-Rothenberg-Stock Test

Null Hypothesis: there is a unit root
 Test Statistic: 1.772**
```

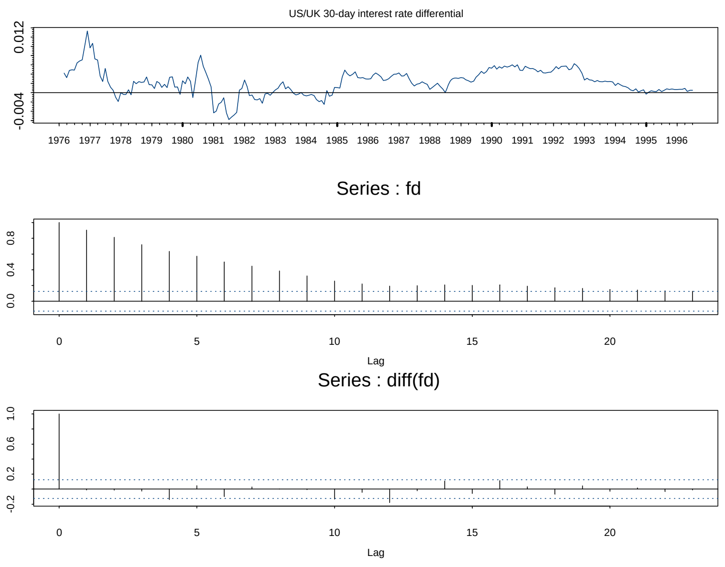


FIGURE 4.6. 30-day US/UK interest rate differential.

```
 * : significant at 5% level
** : significant at 1% level

Coefficients:
  lag1
 -0.07

Degrees of freedom: 244 total; 243 residual
Time period: from Mar 1976 to Jun 1996
Residual standard error: 0.00116

> dfgls.test

Test for Unit Root: DF Test with GLS detrending

Null Hypothesis: there is a unit root
   Type of Test: t-test
 Test Statistic: -2.9205**
 * : significant at 5% level
** : significant at 1% level

Coefficients:
  lag1
 -0.07

Degrees of freedom: 244 total; 243 residual
Time period: from Mar 1976 to Jun 1996
Residual standard error: 0.00116

> mpp.test

Test for Unit Root: Modified Phillips-Perron Test

Null Hypothesis: there is a unit root
   Type of Test: t-test
 Test Statistic: -2.8226**
 * : significant at 5% level
** : significant at 1% level

Coefficients:
  lag1
 -0.07

Degrees of freedom: 244 total; 243 residual
Time period: from Mar 1976 to Jun 1996
Residual standard error: 0.00116
```

Minimizing the MAIC gives $p = 0$, and with this lag length all tests reject the null hypothesis of a unit root at the 1% level.

4.7 References

Caner, M. and L. Kilian (2001). "Size Distortions of Tests of the Null Hypothesis of Stationarity: Evidence and Implications for the PPP Debate," *Journal of International Money and Finance*, 20, 639-657.

Dickey, D. and W. Fuller (1979). "Distribution of the Estimators for Autoregressive Time Series with a Unit Root," *Journal of the American Statistical Association,* 74, 427-431.

Dickey, D. and W. Fuller (1981). "Likelihood Ratio Statistics for Autoregressive Time Series with a Unit Root," *Econometrica*, 49, 1057-1072.

Elliot, G., T.J. Rothenberg, and J.H. Stock (1996). "Efficient Tests for an Autoregressive Unit Root," *Econometrica*, 64, 813-836.

Fuller, W. (1996). *Introduction to Statistical Time Series, Second Edition.* John Wiley, New York.

Hamilton, J. (1994). *Time Series Analysis.* Princeton University Press, Princeton, NJ.

Hatanaka, T. (1995). *Time-Series-Based Econometrics: Unit Roots and Co-Integration.* Oxford University Press, Oxford.

Kwiatkowski, D., P.C.B. Phillips, P. Schmidt and Y. Shin (1992). "Testing the Null Hypothesis of Stationarity Against the Alternative of a Unit Root," *Journal of Econometrics,* 54, 159-178.

MacKinnon, J. (1996). "Numerical Distribution Functions for Unit Root and Cointegration Tests," *Journal of Applied Econometrics*, 11, 601-618.

Maddala, G.S. and I.-M. Kim (1998). *Unit Roots, Cointegration and Structural Change.* Oxford University Press, Oxford.

Ng, S., and P. Perron (1995). "Unit Root Tests in ARMA Models with Data-Dependent Methods for the Selection of the Truncation Lag," *Journal of the American Statistical Association*, 90, 268-281.

Ng, S., and P. Perron (2001). "Lag Length Selection and the Construction of Unit Root Tests with Good Size and Power," *Econometrica*, 69, 1519-1554.

Perron, P. and S. Ng. (1996). "Useful Modifications to Some Unit Root Tests with Dependent Errors and their Local Asymptotic Properties," *Review of Economic Studies*, 63, 435-463.

Phillips, P.C.B. (1987). "Time Series Regression with a Unit Root," *Econometrica*, 55, 227-301.

Phillips, P.C.B. and P. Perron (1988). "Testing for Unit Roots in Time Series Regression," *Biometrika*, 75, 335-346.

Phillips, P.C.B. and Z. Xiao (1998). "A Primer on Unit Root Testing," *Journal of Economic Surveys*, 12, 423-470.

Schwert, W. (1989). "Test for Unit Roots: A Monte Carlo Investigation," *Journal of Business and Economic Statistics*, 7, 147-159.

Said, S.E. and D. Dickey (1984). "Testing for Unit Roots in Autoregressive Moving-Average Models with Unknown Order," *Biometrika*, 71, 599-607.

Stock, J.H. (1994). "Units Roots, Structural Breaks and Trends," in R.F. Engle and D.L. McFadden (eds.), *Handbook of Econometrics, Volume IV.* North Holland, New York.

5 Modeling Extreme Values

5.1 Introduction

One of the goals of financial risk management is the accurate calculation of the magnitudes and probabilities of large potential losses due to extreme events such as stock market crashes, currency crises, trading scandals, or large bond defaults. In statistical terms, these magnitudes and probabilities are *high quantiles* and *tail probabilities* of the probability distribution of losses. The importance of risk management in finance cannot be overstated. The catastrophes of October 17, 1987, Long-Term Capital Management, Barings PLC, Metallgesellschaft, Orange County and Daiwa clearly illustrate the losses that can occur as the result of extreme market movements[1]. The objective of extreme value analysis in finance is to quantify the probabilistic behavior of unusually large losses and to develop tools for managing extreme risks.

Traditional parametric and nonparametric methods for estimating distributions and densities work well in areas of the empirical distribution where there are many observations, but they often give very poor fits to the extreme tails of the distribution. This result is particularly troubling because the management of extreme risk often requires estimating quantiles and tail probabilities beyond those observed in the data. The methods of extreme value theory focus on modeling the tail behavior of a loss distribution using only extreme values rather than all of the data.

[1]See Jorian (2001) for a detailed description of these financial disasters.

This chapter is organized as follows. Section 5.2 covers the modeling of block maximum and minimum values using the generalized extreme value (GEV) distribution. The maximum likelihood estimator for the parameters of the GEV distribution is derived and analyzed, and graphical diagnostics for evaluating the fit are discussed. The use of the GEV distribution is illustrated with examples from finance, and the concept of return level is introduced. Section 5.3 discusses the modeling of extremes over high thresholds or “peaks over thresholds”. This technique is well suited for the estimation of common risk measures like value-at-risk and expected shortfall. Parametric models utilizing the generalized Pareto distribution as well as non-parametric models are presented.

Two excellent textbooks on extreme value theory are Embrechts, Klüppelberg and Mikosch (1997) and Coles (2001). Both books provide many examples utilizing `S-PLUS`. Less rigorous treatments of extreme value theory with many examples in finance are given in Alexander (2001), Jorian (2001) and Tsay (2001). Useful surveys of extreme value theory applied to finance and risk management are given in Diebold, Schuermann and Stroughair (1997), Danielsson and de Vries (2001), McNeil (1998) and Longin (2000).

The `S+FinMetrics` functions for modeling extreme values described in this chapter are based on the functions in the *EVIS* (Extreme Values In `S-PLUS`) library written by Alexander McNeil, and the *EVANESCE* (Extreme Value ANalysis Employing Statistical Copula Estimation) library written by Rene Carmona and Julia Morrison and described in Carmona and Morrison (2001) and Carmona (2004). The *EVANESCE* library also contains an extensive set of functions for analyzing and fitting bivariate copulas, which are described in Chapter 19.

5.2 Modeling Maxima and Worst Cases

To motivate the importance of the statistical modeling of extreme losses in finance, consider the following example taken from McNeil (1998). Figure 5.1 shows the daily closing prices and percentage changes in the S&P 500 index over the period January 5, 1960 through October 16, 1987 taken from the `S+FinMetrics` “`timeSeries`” object `sp.raw`

```
> spto87 = getReturns(sp.raw, type="discrete", percentage=T)
> par(mfrow=c(2,1))
> plot(sp.raw, main="Daily Closing Prices")
> plot(spto87, main="Daily Percentage Returns")
```

Before the October crash, the stock market was unusually volatile with several large price drops in the index resulting in large negative returns. Of interest is the characterization of the worst case scenario for a future fall in S&P 500 index utilizing the historical data prior to the crash given in Figure 5.1. To do this, the following two questions will be addressed:

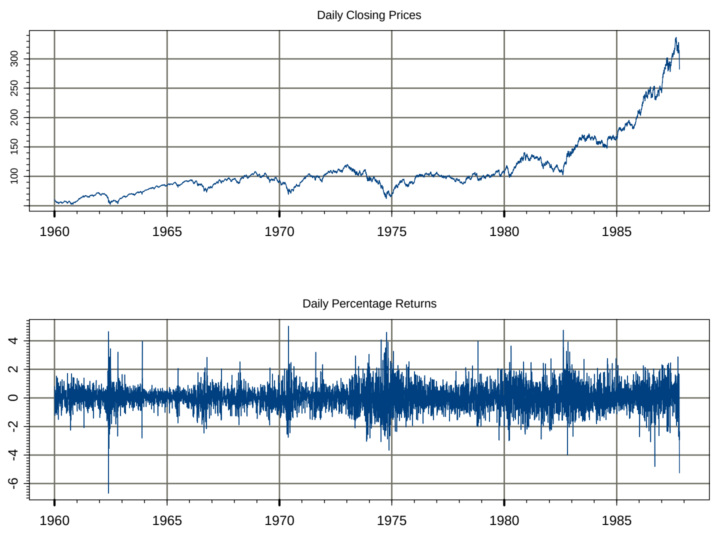


FIGURE 5.1. Daily closing prices and percentage returns on the S&P 500 Index from January, 1960 through October 16, 1987.

- What is the probability that next year's annual maximum negative return exceeds all previous negative returns? In other words, what is the probability that next year's maximum negative return is a new *record*?
- What is the 40-year *return level* of the negative returns? That is, what is the negative return which, on average, should only be exceeded in one year every forty years?

To answer these questions, the distribution of extreme negative returns on S&P 500 index is required. The distribution theory required to analyze maximum values is briefly discussed in the next section.

5.2.1 The Fisher-Tippet Theorem and the Generalized Extreme Value Distribution

Let $X_1, X_2, \ldots$ be iid random variables representing risks or losses with an unknown cumulative distribution function (CDF) $F(x) = \Pr\{X_i \leq x\}$. Examples of the random risks X_i are losses or negative returns on a financial asset or portfolio, operational losses, catastrophic insurance claims, and credit losses. Throughout this chapter, a loss is treated as a positive number and *extreme* events occur when losses take values in the right *tail* of the loss distribution F. Define $M_n = \max\ (X_1, \ldots, X_n)$ as the worst-case loss in a sample of n losses. An important part of *extreme value theory* focuses on the distribution of M_n. From the iid assumption, the CDF of M_n is

$$\Pr\{M_n \leq x\} = \Pr\{X_1 \leq x, \ldots, X_n \leq x\} = \prod_{i=1}^{n} F(x) = F^n(x)$$

Since F^n is assumed to be unknown and the empirical distribution function is often a very poor estimator of $F^n(x)$, an asymptotic approximation to F^n based on the *Fisher-Tippet Theorem* (Fisher and Tippett, 1928) is used to make inferences on M_n. Furthermore, since $F^n(x) \to 0$ or 1 as $n \to \infty$ and x is fixed, the asymptotic approximation is based on the standardized maximum value

$$Z_n = \frac{M_n - \mu_n}{\sigma_n} \tag{5.1}$$

where $\sigma_n > 0$ and μ_n are sequences of real numbers such that σ_n is interpreted as a scale measure and μ_n is interpreted as a location measure. The Fisher-Tippet Theorem states that if the standardized maximum (5.1) converges to some non-degenerate distribution function, it must be a *generalized extreme value* (GEV) distribution of the form

$$H_\xi(z) = \begin{cases} \exp\left\{-(1+\xi z)^{-1/\xi}\right\} & \xi \neq 0,\ 1+\xi z > 0 \\ \exp\left\{-\exp(-z)\right\} & \xi = 0,\ -\infty \leq z \leq \infty \end{cases} \tag{5.2}$$

If (5.1) converges to (5.2), then the CDF F of the underlying data is in the *domain of attraction* of H_ξ. The Fisher-Tippet Theorem is the analog of the *Central Limit Theorem* for extreme values. Whereas the Central Limit Theorem applies to normalized sums of random variables, the Fisher-Tippet Theorem applies to standardized maxima of random variables. The parameter ξ is a *shape* parameter and determines the tail behavior of H_ξ. The parameter $\alpha = 1/\xi$ is called the *tail index* if $\xi > 0$.

The tail behavior of the distribution F of the underlying data determines the shape parameter ξ of the GEV distribution (5.2). If the tail of F declines exponentially, then H_ξ is of the *Gumbel* type and $\xi = 0$. Distributions in the domain of attraction of the Gumbel type are *thin tailed* distributions such as the normal, log-normal, exponential, and gamma. For these distributions, all moments usually exist. If the tail of F declines by a power function, i.e.

$$1 - F(x) = x^{-1/\xi} L(x)$$

for some slowly varying function $L(x)$, then H_ξ is of the *Fréchet* type and $\xi > 0$[2]. Distributions in the domain of attraction of the Fréchet type include *fat tailed* distributions like the Pareto, Cauchy, Student-t, alpha-stable with

[2]A function L on $(0, \infty)$ is slowly varying if $\lim_{x\to\infty} L(tx)/L(x) = 1$ for $t > 0$.

characteristic exponent in $(0, 2)$, as well as various mixture models. Not all moments are finite for these distributions. In fact, it can be shown that $E[X^k] = \infty$ for $k \geq \alpha = 1/\xi$. Last, if the tail of F is finite then H_ξ is of the *Weibull* type and $\xi < 0$. Distributions in the domain of attraction of the Weibull type include distributions with bounded support such as the uniform and beta distributions. All moments exist for these distributions.

The Fisher-Tippet Theorem applies to iid observations. However, the GEV distribution (5.2) may be shown (e.g. Embrechts et. al. (1997)) to be the correct limiting distribution for maxima computed from stationary time series including stationary GARCH-type processes.

The GEV distribution (5.2) characterizes the limiting distribution of the standardized maximum (5.1). It turns out that the GEV distribution (5.2) is invariant to location and scale transformations such that for location and scale parameters μ and $\sigma > 0$

$$H_\xi(z) = H_\xi\left(\frac{x-\mu}{\sigma}\right) = H_{\xi,\mu,\sigma}(x) \tag{5.3}$$

The Fisher-Tippet Theorem may then be interpreted as follows. For large enough n

$$\Pr\{Z_n < z\} = \Pr\left\{\frac{M_n - \mu_n}{\sigma_n} < z\right\} \approx H_\xi(z)$$

Letting $x = \sigma_n z + \mu_n$ then

$$\Pr\{M_n < x\} \approx H_{\xi,\mu,\sigma}\left(\frac{x-\mu_n}{\sigma_n}\right) = H_{\xi,\mu_n,\sigma_n}(x) \tag{5.4}$$

The result (5.4) is used in practice to make inferences about the maximum loss M_n.

Example 24 *Plots of GEV distributions*

The `S+FinMetrics/EVIS` functions `pgev`, `qgev`, `dgev` and `rgev` compute cumulative probability, quantiles, density, and random generation, respectively, from the GEV distribution (5.3) for $\xi \neq 0$ and general values for x, μ and σ. For example, the `S-PLUS` code to compute and plot the GEV CDF H_ξ and the pdf h_ξ for a Fréchet ($\xi = 0.5$), Weibull ($\xi = -0.5$) and Gumbell ($\xi = 0$) is

```
> z.vals = seq(-5, 5, length=200)
> cdf.f = ifelse((z.vals > -2), pgev(z.vals,xi=0.5), 0)
> cdf.w = ifelse((z.vals < 2), pgev(z.vals,xi=-0.5), 1)
> cdf.g = exp(-exp(-z.vals))
> plot(z.vals, cdf.w, type="l", xlab="z", ylab="H(z)")
> lines(z.vals, cdf.f, type="l", lty=2)
> lines(z.vals, cdf.g, type="l", lty=3)
> legend(-5, 1, legend=c("Weibull H(-0.5,0,1)",
```

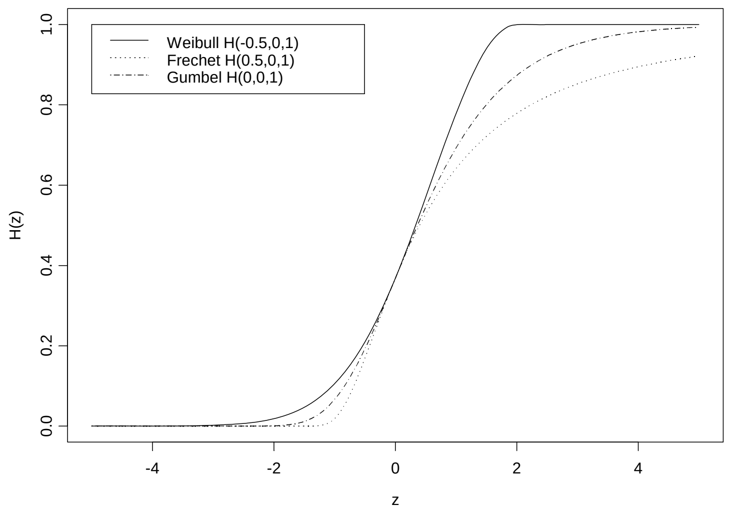


FIGURE 5.2. Generalized extreme value CDFs H_ξ for Fréchet $(\xi = -0.5)$, Weibull $(\xi = 0.5)$ and Gumbell $(\xi = 0)$.

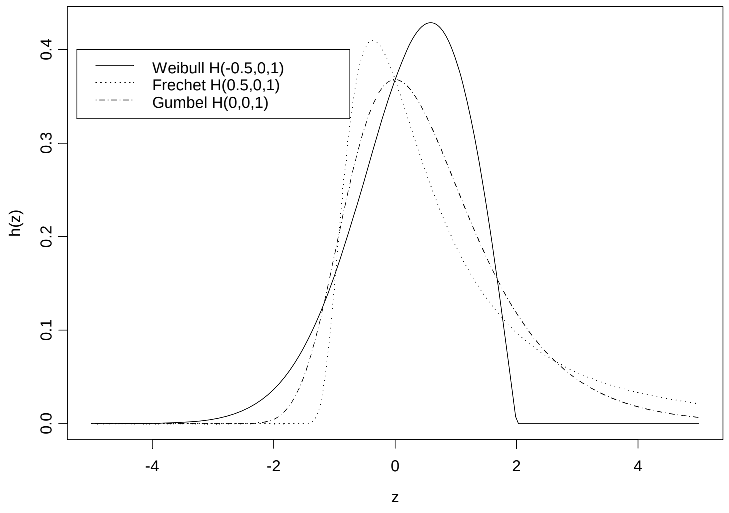


FIGURE 5.3. Generalized extreme value pdfs h_ξ for Fréchet $(\xi = -0.5)$, Weibull $(\xi = 0.5)$ and Gumbell $(\xi = 0)$.

```
+ "Frechet H(0.5,0,1)","Gumbel H(0,0,1)"), lty=1:3)
> # pdfs
> pdf.f = ifelse((z.vals > -2), dgev(z.vals,xi=0.5), 0)
> pdf.w = ifelse((z.vals < 2), dgev(z.vals,xi=-0.5), 0)
> pdf.g = exp(-exp(-z.vals))*exp(-z.vals)
> plot(z.vals, pdf.w, type="l", xlab="z", ylab="h(z)")
> lines(z.vals, pdf.f, type="l", lty=2)
> lines(z.vals, pdf.g, type="l", lty=3)
> legend(-5.25, 0.4, legend=c("Weibull H(-0.5,0,1)",
+ "Frechet H(0.5,0,1)","Gumbel H(0,0,1)"), lty=1:3)
```

The CDF and pdf values are illustrated in Figures 5.2 and 5.3. Notice that the Fréchet is only defined for $z > -2$, and that the Weibull is only defined for $z < 2$.

5.2.2 Estimation of the GEV Distribution

The GEV distribution (5.4) depends on three parameters: the shape parameter ξ and the standardizing constants σ_n and μ_n. These parameters may be estimated using parametric maximum likelihood estimation (MLE). The `S+FinMetrics/EVIS` functions `gev` and `gumbel` fit the GEV distribution (5.2) by MLE to block maxima data. The calculation of the parametric MLE is briefly described below and illustrated with examples.

Parametric Maximum Likelihood Estimator

Let $X_1, \ldots, X_T$ be identically distributed losses from a sample of size T with unknown CDF F and let M_T denote the sample maximum. For inference on M_T using (5.4) the parameters ξ, σ_T and μ_T must be estimated. Since there is only one value of M_T for the entire sample, it is not possible to form a likelihood function for ξ, σ_T and μ_T. However, if interest is on the maximum of X over a large finite subsample or block of size $n < T$, M_n, then a sub-sampling method may be used to form the likelihood function for the parameters ξ, σ_n and μ_n of the GEV distribution for M_n. To do this, the sample is divided into m non-overlapping blocks of essentially equal size $n = T/m$

$$[X_1, \ldots, X_n | X_{n+1}, \ldots, X_{2n} | \ldots | X_{(m-1)n+1}, \ldots, X_{mn}]$$

and $M_n^{(j)}$ is defined as the maximum value of X_i in block $j = 1, \ldots, m$. The likelihood function for the parameters ξ, σ_n and μ_n of the GEV distribution (5.4) is then constructed from the sample of block maxima $\{M_n^{(1)}, \ldots, M_n^{(m)}\}$. It is assumed that the block size n is sufficiently large so that the Fisher-Tippet Theorem holds.

The log likelihood function assuming iid observations from a GEV distribution with $\xi \neq 0$ is

$$\begin{aligned} l(\mu,\sigma,\xi) &= -m\ln(\sigma) - (1+1/\xi)\sum_{i=1}^{m} \ln\left[1+\xi\left(\frac{M_n^{(i)}-\mu}{\sigma}\right)\right] \\ &\quad -\sum_{i=1}^{m}\left[1+\xi\left(\frac{M_n^{(i)}-\mu}{\sigma}\right)\right]^{-1/\xi} \end{aligned}$$

such that

$$1+\xi\left(\frac{M_n^{(i)}-\mu}{\sigma}\right) > 0$$

The log-likelihood for the case $\xi = 0$ (Gumbel family) is

$$\begin{aligned} l(\mu,\sigma) &= -m\ln(\sigma) - \sum_{i=1}^{m}\left(\frac{M_n^{(i)}-\mu}{\sigma}\right) \\ &\quad -\sum_{i=1}^{m}\exp\left[-\left(\frac{M_n^{(i)}-\mu}{\sigma}\right)\right] \end{aligned}$$

Details of the maximum likelihood estimation are discussed in Embrechts et. al. (1997) and Coles (2001). For $\xi > -0.5$ the MLEs for μ, σ and ξ are consistent and asymptotically normally distributed with asymptotic variance given by the inverse of the observed information matrix. The finite sample properties of the MLE will depend on the number of blocks m and the block size n, see McNeil (1998) for an illustration. There is a trade-off between bias and variance. The bias of the MLE is reduced by increasing the block size n, and the variance of the MLE is reduced by increasing the number of blocks m.

Example 25 *MLE of GEV CDF for block maxima from daily S&P 500 returns*

Consider determining the appropriate GEV distribution for the daily negative returns on S&P 500 index discussed at the beginning of this section. A normal qq-plot of the returns computed using

```
> qqPlot(spto87,strip.text="Daily returns on S&P 500",
+ xlab="Quantiles of standard normal",
+ ylab="Quantiles of S&P 500")
```

is shown in Figure 5.4. The returns clearly have fatter tails than the normal distribution which suggests the Fréchet family of GEV distributions with $\xi > 0$ for the block maximum of negative returns.

Before the GEV distribution is fit by MLE, consider first some exploratory data analysis on the annual block maxima of daily negative

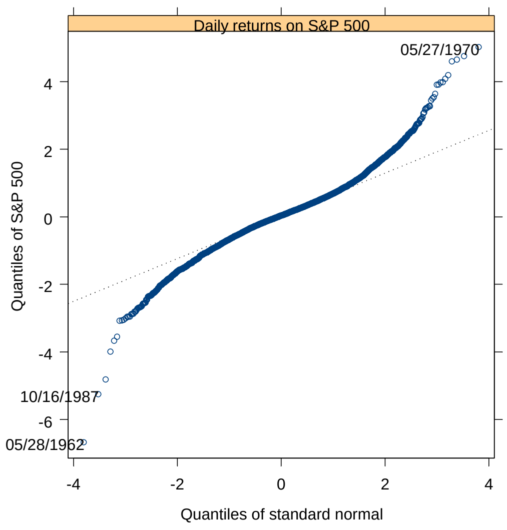


FIGURE 5.4. Normal qq-plot for the daily percentage returns on the S&P 500 index over the period January 5, 1960 through October 16, 1987.

returns. The block maxima may be easily computed using the function `aggregateSeries`:

```
> annualMax.sp500 = aggregateSeries(-spto87, by="years",
+ FUN=max)
```

Figure 5.5 created using

```
> Xn = sort(seriesData(annualMax.sp500))
> par(mfrow=c(2,2))
> plot(annualMax.sp500)
> hist(seriesData(annualMax.sp500),xlab="Annual maximum")
> plot(Xn,-log(-log(ppoints(Xn))),xlab="Annual maximum")
> tmp = records(-spto87)
```

gives several graphical summaries of the annual block maxima. The largest daily negative return in an annual block is 6.68% occurring in 1962. The histogram resembles a Fréchet density (see example above). The qq-plot uses the Gumbel, H_0, as the reference distribution. For this distribution, the quantiles satisfy $H_0^{-1}(p) = -\ln(-\ln(p))$. The downward curve in the plot indicates a GEV distribution with $\xi > 0$. The plot of record development is created with the `S+FinMetrics/EVIS` function `records` and illustrates the developments of records (new maxima) for the daily negative returns along with the expected number of records for iid data, see Embrechts et.

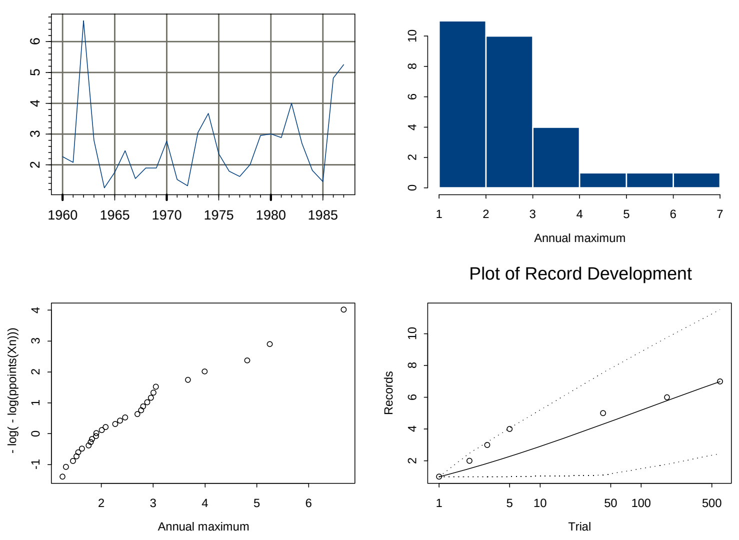


FIGURE 5.5. Annual block maxima, histogram, Gumbel qq-plot and records summary for the daily returns on the S&P 500.

al. (1997) section 6.2.5. Apart from the somewhat large number of records early on, the number of records appears consistent with iid behavior.

The MLEs for the parameters of the GEV distribution with $\xi \neq 0$ using block maxima may be computed using the `S+FinMetrics/EVIS` function `gev`. For example, to compute the MLEs using annual blocks from the daily (negative) returns on S&P 500 index use

```
> gev.fit.year = gev(-spto87, block="year")
> class(gev.fit.year)
[1] "gev"
```

The argument `block` determines the blocking method for the supplied data. An integer value for `block` gives the number of observations in each block. If the data supplied are a "`timeSeries`" then the value of `block` can be also be the character strings "`year`", "`semester`", "`quarter`" or "`month`". If no value for `block` is given then the data are interpreted as block maxima.

The function `gev` returns an sv3 object of class "`gev`" for which there are `print` and `plot` methods. The components of `gev.fit.year` are

```
> names(gev.fit.year)
 [1] "n.all"      "n"          "call"       "block"
 [5] "data"       "par.ests"   "par.ses"    "varcov"
 [9] "converged"  "nllh.final"
```

and a description of these components is given in the online help for `gev.object`. The component `n` gives the number of blocks m:

```
> gev.fit.year$n
[1] 28
```

The block maxima $M_n^{(i)}$ $(i = 1, \ldots, m)$ are in the `data` component. Since the data supplied to `gev` are in a "`timeSeries`", the block maxima in `gev.fit.year$data` are also a "`timeSeries`". The MLEs and asymptotic standard errors for the parameters μ, σ and ξ are:

```
> gev.fit.year
Generalized Extreme Value Distribution Fit --

28  blocks of maxima data
ML estimation converged.
Log-likelihood value:  -38.34


Parameter Estimates, Standard Errors and t-ratios:
        Value Std.Error t value
   xi  0.3344  0.2081    1.6068
sigma  0.6716  0.1308    5.1337
   mu  1.9750  0.1513   13.0549
```

The MLE for ξ is 0.334 with asymptotic standard $\widehat{\text{SE}}(\hat{\xi}) = 0.208$. An asymptotic 95% confidence interval for ξ is $[-0.081, 0.751]$ and indicates considerably uncertainty about the value of ξ.

The fit to the GEV distribution may be evaluated graphically using the `plot` method:

```
> plot(gev.fit.year)

Make a plot selection (or 0 to exit):
1: plot: Scatterplot of Residuals
2: plot: QQplot of Residuals
Selection:
```

Plot options 1 and 2 are illustrated in Figure 5.6. The plots show aspects of the *crude* residuals

$$W_i = \left(1 + \hat{\xi}\frac{M_n^{(i)} - \hat{\mu}}{\hat{\sigma}}\right)^{-1/\xi}$$

which should be iid unit exponentially distributed random variables if the fitted model is correct. The scatter plot of the residuals, with a lowest esti-

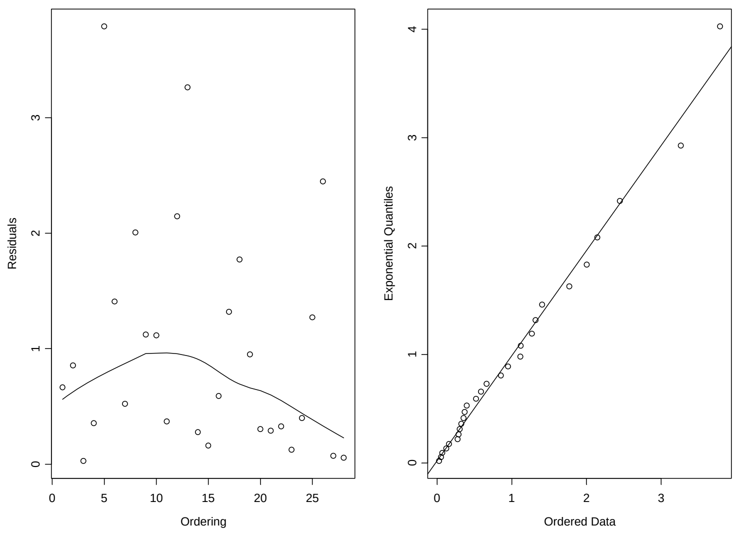


FIGURE 5.6. Residual plots from GEV distribution fit to annual block maxima of daily negative return on the S&P 500 index.

mate of trend, does not reveal any significant unmodeled trend in the data. The qq-plot, using the exponential distribution as the reference distribution, is linear and appears to validate the GEV distribution.

Using the MLEs of the GEV distribution fit to the annual block maxima of the (negative) daily returns on S&P 500 index, the question

- What is the probability that next year's annual maximum negative return exceeds all previous negative returns?

may be answered using (5.4). Since the largest block maxima is 6.68%, this probability is estimated using

$$\Pr\left(M_{260}^{(29)} > \max\left(M_{260}^{(1)}, \ldots, M_{260}^{(28)}\right)\right) = 1 - H_{\hat{\xi},\hat{\mu},\hat{\sigma}}(6.68)$$

Using the S+FinMetrics/EVIS function pgev, the result is

```
> 1- pgev(max(gev.fit.year$data),
+ xi=gev.fit.year$par.ests["xi"],
+ mu=gev.fit.year$par.ests["mu"],
+ sigma=gev.fit.year$par.ests["sigma"])
 0.02677
```

That is, there is a 2.7% chance that a new record maximum daily negative return will be established during the next year.

The above analysis is based on annual block maxima. The GEV distribution fit to quarterly block maxima is obtained using

```
> gev.fit.quarter= gev(-spto87,block="quarter")
> gev.fit.quarter
Generalized Extreme Value Distribution Fit --

112  blocks of maxima data
ML estimation converged.
Log-likelihood value:  -111.9


Parameter Estimates, Standard Errors and t-ratios:
        Value Std.Error t value
   xi  0.1910  0.0695    2.7472
sigma  0.5021  0.0416   12.0701
   mu  1.4013  0.0530   26.4583
```

The MLEs for ξ, μ and σ using quarterly blocks are slightly smaller than the MLEs using annual blocks. Notice, however, that the estimated asymptotic standard errors are much smaller using quarterly block. In particular, an asymptotic 95% confidence interval for ξ is $[0.052, 0.330]$ and contains only positive values for ξ indicating a fat-tailed distribution. An estimate of the probability that next quarter's maximum exceeds all previous maxima is

```
> 1- pgev(max(gev.fit.quarter$data),
+ xi=gev.fit.quarter$par.ests["xi"],
+ mu=gev.fit.quarter$par.ests["mu"],
+ sigma=gev.fit.quarter$par.ests["sigma"])
 0.003138
```

As expected, this probability is smaller than the corresponding probability computed for annual maxima.

5.2.3 Return Level

For $\alpha \in (0,1)$ the $100 \cdot \alpha\%$ quantile of a continuous distribution with distribution function F is the value q_α such that

$$q_\alpha = F^{-1}(\alpha).$$

A useful risk measure for block maxima that is related to a high quantile is the so-called *return level*. The k n-block return level, $R_{n,k}$, is defined to be that level which is exceeded in one out of every k blocks of size n. That is, $R_{n,k}$ is the loss value such that

$$\Pr\{M_n > R_{n,k}\} = 1/k \tag{5.5}$$

The n-block in which the return level is exceeded is called a *stress period.* If the distribution of the maxima M_n in blocks of length n is characterized by (5.4) then $R_{n,k}$ is simply the $1 - 1/k$ quantile of this distribution:

$$R_{n,k} \approx H^{-1}_{\xi,\mu,\sigma}(1 - 1/k) = \mu - \frac{\sigma}{\xi}\left(1 - (-\log(1 - 1/k))^{-\xi}\right) \qquad (5.6)$$

By the invariance property of maximum likelihood estimation, given the MLEs for the parameters ξ, μ and σ, the MLE for $R_{n,k}$ is

$$\hat{R}_{n,k} = \hat{\mu} - \frac{\hat{\sigma}}{\hat{\xi}}\left(1 - (-\log(1 - 1/k))^{-\hat{\xi}}\right)$$

An asymptotically valid confidence interval for $R_{n,k}$ may be computed using the *delta method* (see Greene 2000, p. 118) or from the concentrated/profile log-likelihood function. Given that (5.6) is a highly nonlinear function of σ, μ and ξ, the delta method is not recommended. Details of constructing a confidence interval for $R_{n,k}$ based on the profile log-likelihood are given in chapter three of Coles (2001) and the appendix of McNeil (1998).

The return level probability in (5.5) is based on the GEV distribution $H_{\xi,\mu,\sigma}$ of the maxima M_n. For iid losses X with CDF F, $H_{\xi,\mu,\sigma} \approx F^n$ so that

$$F(R_{n,k}) = \Pr\{X \leq R_{n,k}\} \approx (1 - 1/k)^{1/n} \qquad (5.7)$$

Hence, for iid losses the return level $R_{n,k}$ is approximately the $(1 - 1/k)^{1/n}$ quantile of the loss distribution F.

Example 26 *Return levels for S&P 500 negative returns*

Given the MLEs for the GEV distribution fit to the annual block maxima of the (negative) daily returns on S&P 500 index, the question

- What is the 40-year *return level* of the index returns?

may be answered using (5.6). The `S+FinMetrics/EVIS` function `rlevel.gev` computes (5.6) as well as an asymptotic 95% confidence interval based on the profile likelihood using the information from a "`gev`" object. To compute the 40 year return level for the S&P 500 returns from the "`gev`" object `gev.fit.year` and to create a plot of the 95% confidence interval use

```
> rlevel.year.40 = rlevel.gev(gev.fit.year, k.blocks=40,
+                                type="profile")
> class(rlevel.year.40)
[1] "list"
> names(rlevel.year.40)
[1] "Range"  "rlevel"
> rlevel.year.40$rlevel
[1] 6.833
```

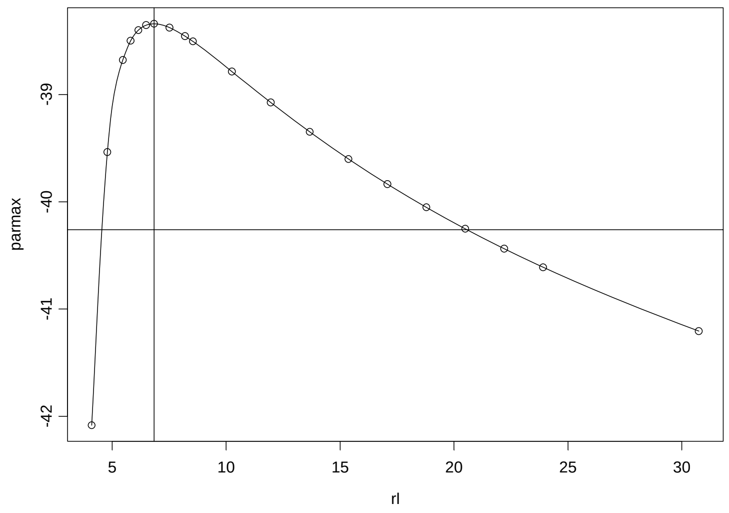


FIGURE 5.7. Asymptotic 95% confidence interval for the 40 year return level based on the profile log-likelihood function.

When `type="profile"`, the function `rlevel.gev` returns a "`list`" object, containing the return level and range information used in the construction of the profile log-likelihood confidence interval, and produces a plot of the profile log-likelihood confidence interval for the return level. The estimate of the 40 year return level is 6.83%. Assuming iid returns and using (5.7), the estimated return level of 6.83% is approximately the 99.99% quantile of the daily return distribution. An asymptotic 95% confidence interval for the true return level is illustrated in Figure 5.7. Notice the asymmetric shape of the asymptotic confidence interval. Although the point estimate of the return level is 6.83%, the upper endpoint of the 95% confidence interval is about 21%. This number may seem large; however, on Monday October 19th 1987 S&P 500 index closed down 20.4%.

By default, the function `rlevel.gev` produces a plot of the asymptotic 95% confidence level. Alternatively, if `rlevel.gev` is called with the optional argument `type="RetLevel"`:

```
> rlevel.year.40 = rlevel.gev(gev.fit.year, k.blocks=40,
+                              type="RetLevel")
> names(rlevel.year.40)
[1] "LowerCB" "rlevel"  "UpperCB"
> rlevel.year.40
$LowerCB:
```

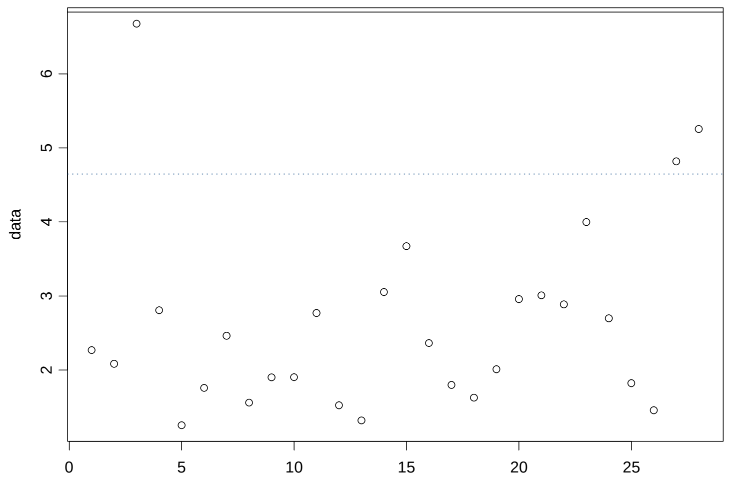


FIGURE 5.8. Estimated 40-year return level with 95% confidence band for the S&P 500 daily negative returns.

```
[1] 4.646

$rlevel:
[1] 6.833

$UpperCB:
[1] 20.5
```

A plot of the estimated return level along with the block maxima, as shown in Figure 5.8, is created, and the components of the returned list are the estimated return level along with the end points of the 95% confidence interval.

The 40 year return level may also be estimated from the GEV distribution fit to quarterly maxima. Since 40 years is 160 quarters, the 40 year return level computed from the "gev" object `gev.fit.quarter` is

```
> rlevel.160.q = rlevel.gev(gev.fit.quarter, k.blocks=160,
+ type="RetLevel")
> rlevel.160.q
$LowerCB:
[1] 4.433

$rlevel:
```

```
[1] 5.699

$UpperCB:
[1] 8.549
```

Here, the estimated return level and asymptotic 95% confidence interval are smaller than the corresponding quantities estimated from annual data.

5.3 Modeling Extremes Over High Thresholds

Modeling only block maxima data is inefficient if other data on extreme values are available. A more efficient alternative approach that utilizes more data is to model the behavior of extreme values above some high threshold. This method is often called *peaks over thresholds* (POT). Another advantage of the POT approach is that common risk measures like *Value-at-Risk* (VaR) and *expected shortfall* (ES) may easily be computed[3].

To illustrate the concepts of VaR and ES, review the daily S&P 500 returns analyzed in the previous section. Suppose the S&P 500 is the only asset in a large portfolio for an investor and that the random variable X with CDF F represents the daily loss on the portfolio. The daily VaR on the portfolio is simply a high quantile of the distribution F of daily losses. For example, the daily 1% VaR on the portfolio is the 99% quantile of X

$$\mathrm{VaR}_{.99} = F^{-1}(0.99).$$

That is, with 1% probability the loss in portfolio value over a day will exceed $\mathrm{VaR}_{.99}$. Often the high quantile $\mathrm{VaR}_{.99}$ is computed assuming $X \sim N(\mu, \sigma^2)$. In this case, the calculation of $\mathrm{VaR}_{.99}$ reduces to the simple formula

$$\mathrm{VaR}_{.99} = \mu + \sigma \cdot q_{.99} \tag{5.8}$$

where $q_{.99}$ is the 99% quantile of the standard normal distribution. The distribution of daily portfolio losses, however, generally has fatter tails than the normal distribution so that (5.8) may severely under-estimate $\mathrm{VaR}_{.99}$. Estimates of VaR based on the POT methodology are much more reliable.

The ES on the portfolio is the average loss given that VaR has been exceeded. For example, the 1% ES is the conditional mean of X given that $X > \mathrm{VaR}_{.99}$

$$\mathrm{ES}_{.99} = E[X|X > \mathrm{VaR}_{.99}]$$

[3]Notice that VaR and ES are based on the distribution of the losses and not on the distribution of the maximum losses. The analysis of block maxima based on the GEV distribution allowed inferences to be made only on the maxima of returns. The POT analysis will allow inferences to be made directly on the distribution of losses.

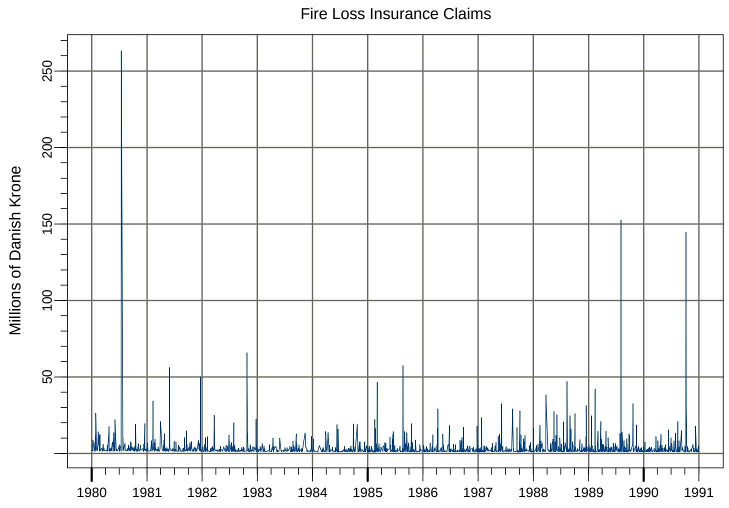


FIGURE 5.9. Large fire loss insurance claims.

If $X \sim N(\mu, \sigma^2)$ then $\text{ES}_{.99}$ may be computed as the mean of a truncated normal random variable:

$$\text{ES}_{.99} = \mu + \sigma \cdot \frac{\phi(z)}{1 - \Phi(z)} \tag{5.9}$$

where $z = (\text{VaR}_{.99} - \mu)/\sigma$, $\phi(z)$ is the standard normal density function and $\Phi(z)$ is the standard normal CDF. Again, if the distribution of losses has fatter tails than the normal, then (5.9) will underestimate $\text{ES}_{.99}$. The POT methodology estimates the distribution of losses over a threshold and produces an estimate of ES as a by-product of the estimation.

For another example, consider the "`timeSeries`" `danish` representing Danish fire loss data in `S+FinMetrics`, which is analyzed in McNeil (1999). The data in `danish` consist of 2167 daily insurance claims for losses exceeding one million Danish Krone from January 3, 1980 through December 31, 1990. The reported loss is an inflation adjusted total loss for the event concerned and includes damages to buildings, damage to contents of buildings as well as loss of profits. Figure 5.9 created using

```
> plot(danish, ain="Fire Loss Insurance Claims",
+ ylab="Millions of Danish Krone")
```

shows the data and reveals several extreme losses. For risk management purposes, insurance companies may be interested in the frequency of occurrence of large claims above some high threshold as well as the average value of the claims that exceed the high threshold. Additionally, they may be interested in daily VaR and ES. The statistical models for extreme values above a high threshold may be used to address these issues.

5.3.1 The Limiting Distribution of Extremes Over High Thresholds and the Generalized Pareto Distribution

As with the analysis of block maxima, let $X_1, X_2, \ldots$ be a sequence of iid random variables representing risks or losses with an unknown CDF F and let $M_n = \max\{X_1, \ldots, X_n\}$. A natural measure of extreme events are values of the X_i that exceed a high threshold u. Define the *excess distribution* above the threshold u as the conditional probability:

$$F_u(y) = \Pr\{X - u \leq y | X > u\} = \frac{F(y+u) - F(u)}{1 - F(u)}, \ y > 0 \qquad (5.10)$$

For the class of distributions F such that the CDF of the standardized value of M_n converges to a GEV distribution (5.2), it can be shown (c.f. Embrechts et. al. (1997)) that for large enough u there exists a positive function $\beta(u)$ such that the excess distribution (5.10) is well approximated by the *generalized Pareto distribution* (GPD)

$$G_{\xi,\beta(u)}(y) = \begin{cases} 1 - \left(1 + (\xi y/\beta(u))^{-1/\xi}\right) \text{ for } \xi \neq 0 \\ 1 - \exp(-y/\beta(u)) \text{ for } \xi = 0 \end{cases}, \ \beta(u) > 0 \qquad (5.11)$$

defined for $y \geq 0$ when $\xi \geq 0$ and $0 \leq y \leq -\beta(u)/\xi$ when $\xi < 0$.

Remarks:

- Operationally, for a sufficiently high threshold u, $F_u(y) \approx G_{\xi,\beta(u)}(y)$ for a wide class of loss distributions F. To implement this result, the threshold value u must be specified and estimates of the unknown parameters ξ and $\beta(u)$ must be obtained.
- There is a close connection between the limiting GEV distribution for block maxima and the limiting GPD for threshold excesses. For a given value of u, the parameters ξ, μ and σ of the GEV distribution determine the parameters ξ and $\beta(u)$. In particular, the shape parameter ξ of the GEV distribution is the same shape parameter ξ in the GPD and is independent of the threshold value u. Consequently, if $\xi < 0$ then F is in the Weibull family and $G_{\xi,\beta(u)}$ is a *Pareto type II* distribution; if $\xi = 0$ then F is in the Gumbell family and $G_{\xi,\beta(u)}$ is an *exponential* distribution; and if $\xi > 0$ then F is in the Fréchet family and $G_{\xi,\beta(u)}$ is a *Pareto* distribution.
- For $\xi > 0$, the most relevant case for risk management purposes, it can be shown that $E[X^k] = \infty$ for $k \geq \alpha = 1/\xi$. For example, if

$\xi = 0.5$ then $E[X^2] = \infty$ and the distribution of losses, X, does not have finite variance. If $\xi = 1$ then $E[X] = \infty$.

- Consider a limiting GPD with shape parameter ξ and scale parameter $\beta(u_0)$ for an excess distribution F_{u_0} with threshold u_0. For an arbitrary threshold $u > u_0$, the excess distribution F_u has a limiting GPD distribution with shape parameter ξ and scale parameter $\beta(u) = \beta(u_0)+\xi(u-u_0)$. Alternatively, for any $y > 0$ the excess distribution F_{u_0+y} has a limiting GPD distribution with shape parameter ξ and scale parameter $\beta(u_0) + \xi y$.

Example 27 *Plots of GPDs*

The `S+FinMetrics/EVIS` functions `pgpd`, `qgpd`, `dgpd` and `rgpd` compute cumulative probability, quantiles, density and random number generation, respectively, from the GPD (5.11) for $\xi \neq 0$ and general values for $\beta(u)$. For example, the `S-PLUS` code to compute and plot the CDFs and pdfs with $\beta(u) = 1$ for a Pareto ($\xi = 0.5$), exponential ($\xi = 0$) and Pareto type II ($\xi = -0.5$) is

```
> par(mfrow=c(1,2))
> y.vals = seq(0,8,length=200)
> cdf.p = pgpd(y.vals, xi=0.5)
> cdf.p2 = ifelse((y.vals < 2), pgpd(y.vals,xi=-0.5), 1)
> cdf.e = 1-exp(-z.vals)
> plot(y.vals, cdf.p, type="l", xlab="y", ylab="G(y)",
+ ylim=c(0,1))
> lines(y.vals, cdf.e, type="l", lty=2)
> lines(y.vals, cdf.p2, type="l", lty=3)
> legend(1,0.2,legend=c("Pareto G(0.5,1)","Exponential G(0,1)",
+ "Pareto II G(-0.5,1)"),lty=1:3)
> # PDFs
> pdf.p = dgpd(y.vals, xi=0.5)
> pdf.p2 = ifelse((y.vals < 2), dgpd(y.vals,xi=-0.5), 0)
> pdf.e = exp(-y.vals)
> plot(y.vals, pdf.p, type="l", xlab="y", ylab="g(y)",
+ ylim=c(0,1))
> lines(y.vals, pdf.e, type="l", lty=2)
> lines(y.vals, pdf.p2, type="l", lty=3)
> legend(2,1,legend=c("Pareto g(0.5,1)","Exponential g(0,1)",
+ "Pareto II g(-0.5,1)"),lty=1:3)
```

The CDFs and pdfs are illustrated in Figure 5.10. Notice that the Pareto type II is only defined for $y < 2$.

Example 28 *qq-plots to determine tail behavior*

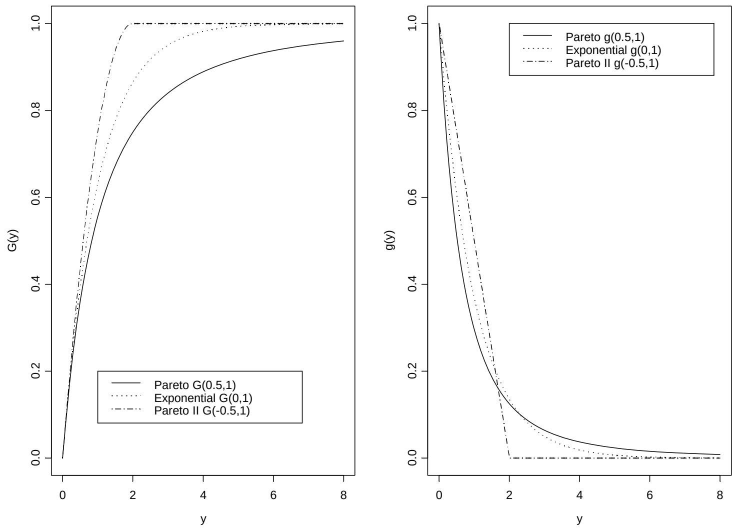


FIGURE 5.10. Generalized Pareto CDFs, $G_{\xi,1}$, and pdfs, $g_{\xi,1}$, for Pareto ($\xi = 0.5$), exponential ($\xi = 0$) and Pareto type II ($\xi = -0.5$).

A simple graphical technique to infer the tail behavior of observed losses is to create a qq-plot using the exponential distribution as a reference distribution. If the excesses over thresholds are from a thin-tailed distribution, then the GPD is exponential with $\xi = 0$ and the qq-plot should be linear. Departures from linearity in the qq-plot then indicate either fat-tailed behavior ($\xi > 0$) or bounded tails ($\xi < 0$). The `S+FinMetrics/EVIS` function `qplot` may be used to create a qq-plot using a GPD as a reference distribution. For example, to create qq-plots with the exponential distribution as the reference distribution for the S&P 500 negative returns over the threshold $u = 1$ and the Danish fire loss data over the threshold $u = 10$, use

```
> par(mfrow=c(1,2))
> qplot(-spto87, threshold=1, main="S&P 500 negative returns")
> qplot(danish, threshold=10, main="Danish fire losses")
```

Figure 5.11 shows these qq-plots. There is a slight departure from linearity for the negative S&P 500 returns and a rather large departure from linearity for the Danish fire losses.

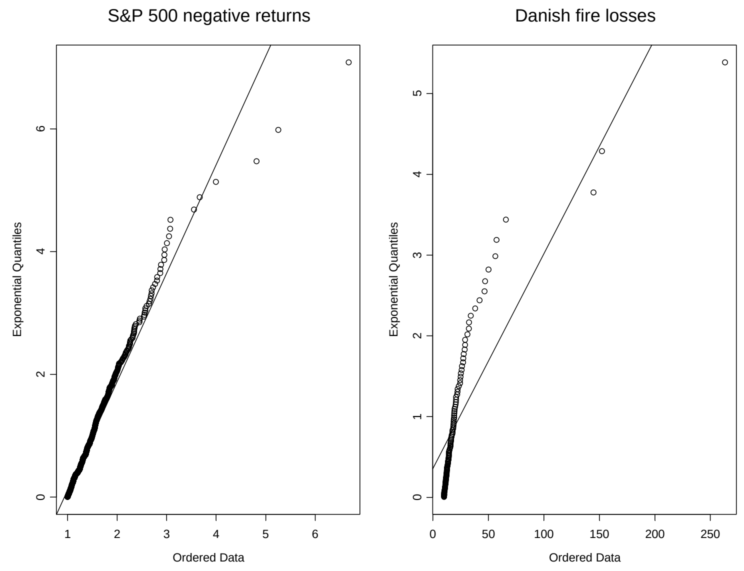


FIGURE 5.11. QQ-plots with exponential reference distribution for the S&P 500 negative returns over the threshold $u = 1$ and the Danish fire losses over the threshold $u = 10$.

Mean Excess Function

Suppose the threshold excess $X - u_0$ follows a GPD with parameters $\xi < 1$ and $\beta(u_0)$. Then the *mean excess* over the threshold u_0 is

$$E[X - u_0|X > u_0] = \frac{\beta(u_0)}{1-\xi}. \tag{5.12}$$

For any $u > u_0$, define the *mean excess function* $e(u)$ as

$$e(u) = E[X - u|X > u] = \frac{\beta(u_0) + \xi(u - u_0)}{1-\xi}. \tag{5.13}$$

Alternatively, for any $y > 0$

$$e(u_0 + y) = E[X - (u_0 + y)|X > u_0 + y] = \frac{\beta(u_0) + \xi y}{1-\xi}. \tag{5.14}$$

Notice that for a given value of ξ, the mean excess function is a linear function of $y = u - u_0$. This result motivates a simple graphical way to infer the appropriate threshold value u_0 for the GPD. Define the *empirical mean excess function*

$$e_n(u) = \frac{1}{n_u}\sum_{i=1}^{n_u}(x_{(i)} - u) \tag{5.15}$$

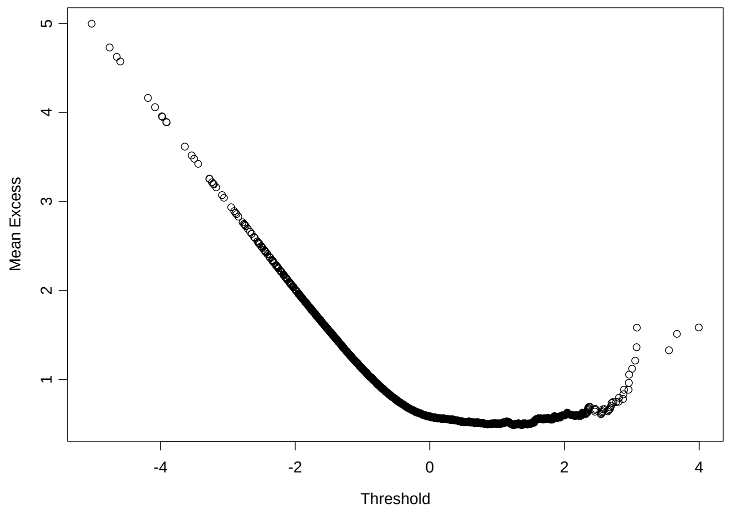


FIGURE 5.12. Mean excess plot for the S&P 500 negative returns.

where $x_{(i)}$ $(i = 1, \ldots, n_u)$ are the values of x_i such that $x_i > u$. The *mean excess plot* is a plot of $e_n(u)$ against u and should be linear in u for $u > u_0$. An upward sloping plot indicates heavy-tailed behavior. In particular, a straight line with positive slope above u_0 is a sign of Pareto behavior in tail. A downward trend shows thin-tailed behavior, whereas a line with zero slope shows an exponential tail.

Example 29 *Mean excess plots for S&P 500 and fire loss data*

The `S+FinMetrics/EVIS` function `meplot` computes the empirical mean excess function (5.15) and creates the mean excess plot. The mean excess functions and mean excess plots for the S&P 500 negative returns and the Danish fire losses are computed using

```
> me.sp500 = meplot(-spto87)
> me.dainsh = meplot(danish)
> class(me.sp500)
[1] "data.frame"
> colIds(me.sp500)
[1] "threshold" "me"
```

The function `meplot` returns a data frame containing the thresholds u and the mean excesses $e_n(u)$ and produces a mean excess plot. The mean excess plots for the S&P 500 and Danish data are illustrated in Figures

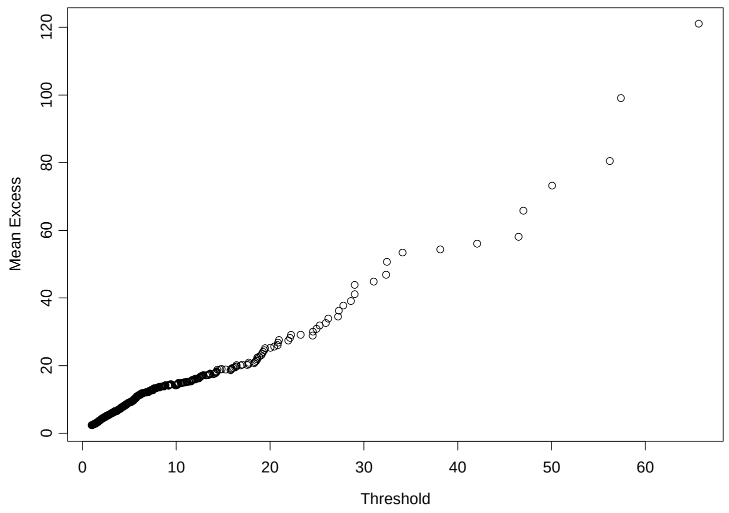


FIGURE 5.13. Mean excess plot for the Danish fire loss data.

5.12 and 5.13. The mean excess plot for the S&P 500 negative returns is linear in u with positive slope for $u > 1$ indicating Pareto tail behavior. The plot for the fire loss data is upward sloping and linear for almost all values of u. However, there is a slight kink at $u = 10$.

5.3.2 Estimating the GPD by Maximum Likelihood

Let $x_1, \ldots, x_n$ be iid sample of losses with unknown CDF F. For a given high threshold u, extreme values are those x_i values for which $x_i - u > 0$. Denote these values $x_{(1)}, \ldots, x_{(k)}$ and define the threshold excesses as $y_i = x_{(i)} - u$ for $i = 1, \ldots, k$. The results of the previous section imply that if u is large enough then $\{y_1, \ldots, y_k\}$ may be thought of as a random sample from a GPD with unknown parameters ξ and $\beta(u)$. For $\xi \neq 0$, the log-likelihood function based on (5.11) is

$$l(\xi, \beta(u)) = -k \ln(\beta(u)) - (1 + 1/\xi) \sum_{i=1}^{k} \ln(1 + \xi y_i/\beta(u))$$

provided $y_i \geq 0$ when $\xi > 0$ and $0 \leq y_i \leq -\beta(u)/\xi$ when $\xi < 0$. For $\xi = 0$ the log-likelihood function is

$$l(\beta(u)) = -k \ln(\beta(u)) - \beta(u)^{-1} \sum_{i=1}^{k} y_i.$$

5.3.3 Estimating the Tails of the Loss Distribution

For a sufficiently high threshold u, $F_u(y) \approx G_{\xi,\beta(u)}(y)$. Using this result in (5.10) and setting $x = u + y$, an approximation to the tails of the loss distribution $F(x)$ for $x > u$ is given by

$$F(x) = (1 - F(u))G_{\xi,\beta(u)}(y) + F(u) \tag{5.16}$$

The CDF value $F(u)$ may be estimated non-parametrically using the empirical CDF

$$\hat{F}(u) = \frac{(n-k)}{n} \tag{5.17}$$

where k denotes the number of exceedences over the threshold u. Combining the parametric representation (5.11) with the non-parametric estimate (5.17) gives the resulting estimate of (5.16)

$$\hat{F}(x) = 1 - \frac{k}{n}\left(1 + \hat{\xi} \cdot \frac{x-u}{\hat{\beta}(u)}\right) \tag{5.18}$$

where $\hat{\xi}$ and $\hat{\beta}(u)$ denote the MLEs of ξ and $\beta(u)$, respectively.

Example 30 *Estimating the GPD for the S&P 500 negative returns*

Maximum likelihood estimation of the parameters ξ and $\beta(u)$ of the GPD (5.11) may be computed using the `S+FinMetrics/EVIS` function `gpd`. In order to compute the MLE, a threshold value u must be specified. The threshold should be large enough so that the GPD approximation is valid but low enough so that a sufficient number of observations k are available for a precise fit.

To illustrate, consider fitting GPD to the negative returns on the S&P 500 index. The `S+FinMetrics/EVIS` function `gpd` may be used to compute the MLEs for the GPD (5.11) for a given threshold u. The mean excess plot for the S&P 500 returns in Figure 5.12 suggests a value of $u = 1$ may be appropriate for the GPD approximation to be valid. The MLE using $u = 1$ is computed using

```
> gpd.sp500.1 = gpd(-spto87, threshold=1)
> class(gpd.sp500.1)
[1] "gpd"
```

The function `gpd` returns an object of class "`gpd`" for which there are `print` and `plot` methods. The components of a "`gpd`" object are numerous, and are described in the online help for `gpd.object`.

The MLEs for ξ and $\beta(1)$ and asymptotic standard errors are

```
> gpd.sp500.1
Generalized Pareto Distribution Fit --
```

```
Total of  6985  observations

Upper Tail Estimated with ml --
Upper Threshold at  1  or  8.518 % of the data
ML estimation converged.
Log-likelihood value:  -183.6


Parameter Estimates, Standard Errors and t-ratios:
       Value Std.Error t value
  xi  0.0677  0.0397    1.7033
beta  0.4681  0.0267   17.5376
```

Notice that $\hat{\xi} = 0.068$ is fairly close to zero and indicates that the return distribution is not so heavy-tailed. Also, the GPD estimate of ξ is quite a bit smaller than the GEV estimate $\hat{\xi} = 0.334$ based on annual data, but it is very close to the GEV estimate $\hat{\xi} = 0.069$ based on quarterly data.

Diagnostic plots of the GDP fit are created using the `plot` method

```
> plot(gpd.sp500.1)

Make a plot selection (or 0 to exit):
1: plot: Excess Distribution
2: plot: Tail of Underlying Distribution
3: plot: Scatterplot of Residuals
4: plot: QQplot of Residuals
Selection:
```

The four plot options are depicted in Figure 5.14. The first plot option shows the GPD estimate of the excess distribution, and the second plot option shows the tail estimate (5.18). The GPD appears to fit the distribution of threshold excesses fairly well. Note, the `S+FinMetrics/EVIS` function `tailplot` may be used to compute plot option 2 directly.

The `S+FinMetrics/EVIS` function `shape` can be used to create a plot showing how the MLE of the shape parameter ξ varies with the selected threshold u:

```
> shape(-spto87, end=600)
```

The optional argument `end=600` specifies the maximum number of threshold exceedences to consider. The resulting plot is shown in Figure 5.15. The estimates of ξ are fairly stable and close to zero for threshold values less than 1.2, and increase slightly for threshold values between 1.2 and 2.

The `S+FinMetrics/EVANESCE` function `shape.plot` may also be used to produce a plot similar to 5.15. For example

```
> shape.plot(-spto87, from = 0.9, to = 0.98)
```

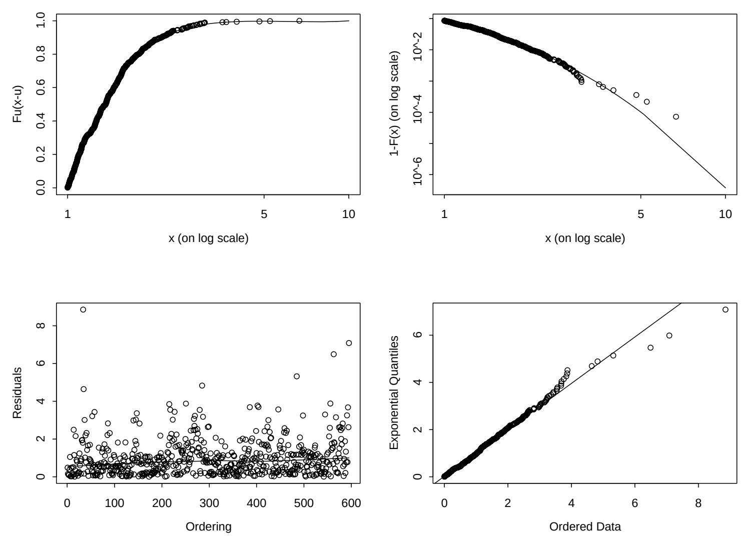


FIGURE 5.14. Diagnostic plots for GPD fit to daily negative returns on S&P 500 index.

estimates ξ for threshold values starting at 90th percentile and ending at the 98th percentile of the data.

Example 31 *Estimating the GPD for lower and upper tails of S&P 500 returns*

Sometimes it is desirable to estimate the parameters ξ and $\beta(u)$ of the GPD (5.11) separately for the lower and upper tails. This may be done using the `S+FinMetrics/EVANESCE` function `gpd.tail`. In order to compute the MLE, threshold values u_{lower} and u_{upper} must be specified. The previous analysis found $u_{\text{lower}} = -1$. A guess for the upper threshold may be obtained from the mean excess plot

```
> me.sp500 = meplot(spto87)
```

illustrated in Figure 5.16. For $u_{\text{upper}} = 1$, the plot appears linear with a positive slope indicating Pareto tail behavior.

The two-tailed MLEs may then be computed using

```
> gpd.sp500.2tail = gpd.tail(spto87, upper = 1, lower = -1,
+                            plot = T)
> class(gpd.sp500.2tail)
[1] "gpd"
> gpd.sp500.2tail
```

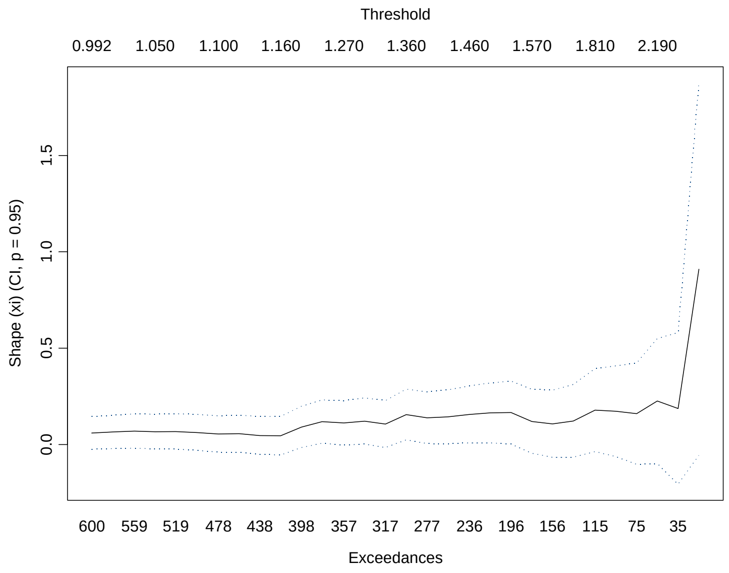


FIGURE 5.15. Estimates of shape parameter ξ for S&P 500 negative returns as a function of the threshold value u.

```
Generalized Pareto Distribution Fit --

Total of  6985  observations

Upper Tail Estimated with ml --
Upper Threshold at  1  or  8.533 % of the data
ML estimation converged.
Log-likelihood value:  -277.7


Parameter Estimates, Standard Errors and t-ratios:
      Value Std.Error t value
  xi 0.0415     NA        NA
beta 0.5624     NA        NA

Lower Tail Estimated with ml --
Lower Threshold at  -1  or  8.518 % of the data
ML estimation converged.
Log-likelihood value:  -183.6


Parameter Estimates, Standard Errors and t-ratios:
```

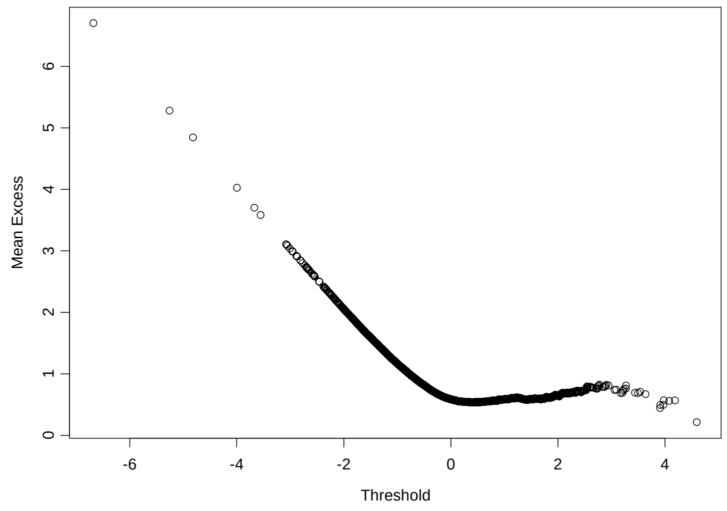


FIGURE 5.16.

```
       Value Std.Error t value
  xi 0.0677     NA        NA
beta 0.4681     NA        NA
```

The lower tail estimates are the same as in the previous example[4]. The upper tail estimates are similar. The optional argument `plot=T`, produces qq-plots of excesses over the specified thresholds versus GPD quantiles using the estimated lower and upper tail shape parameters. The linear nature of these plots, given in Figure 5.17, supports the assumption that the lower and upper excesses have GPD distributions.

Example 32 *Estimating the GPD for the Danish fire loss data*

The mean excess plot in Figure 5.13 suggests a threshold value of $u = 10$. The MLEs of the GPD parameters for the Danish fire loss data using a high threshold of 10 million Krone are computed using

```
> gpd.danish.10 = gpd(danish, threshold=10)
> gpd.danish.10
Generalized Pareto Distribution Fit --
```

[4]Currently, the function `gpd.tail` does not compute the estimated covariance matrix for the estimated parameters. To get standard errors, use the function `gpd` on both tails separately.

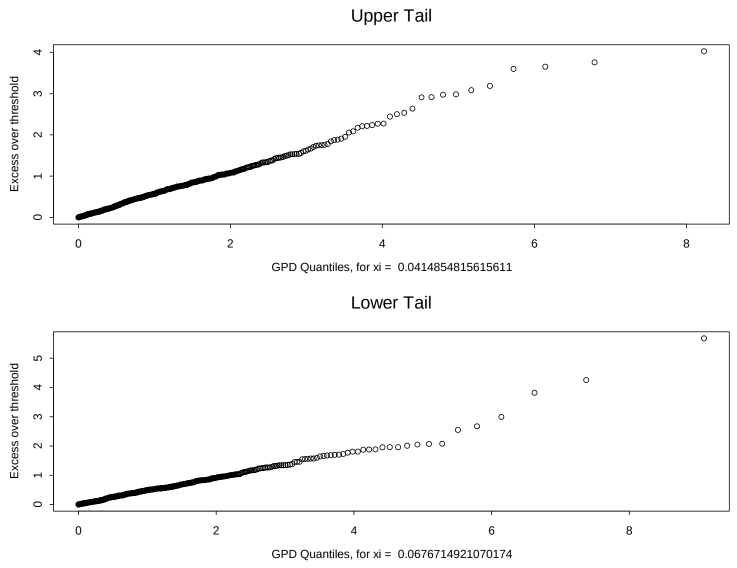


FIGURE 5.17.

```
Total of  2167  observations

Upper Tail Estimated with ml --
Upper Threshold at  10  or  5.03 % of the data
ML estimation converged.
Log-likelihood value:  -374.9


Parameter Estimates, Standard Errors and t-ratios:
      Value Std.Error t value
  xi 0.4970 0.1363    3.6467
beta 6.9755 1.1135    6.2645
```

The estimate of ξ shows heavy tails and suggests that the variance may not be finite. The diagnostic plots in Figure 5.18, created using

```
> par(mfrow=c(1,2))
> tailplot(gpd.danish.10)
> shape(danish)
```

show that the GPD fits the data well and that the estimates of ξ are fairly stable for a wide range of threshold values.

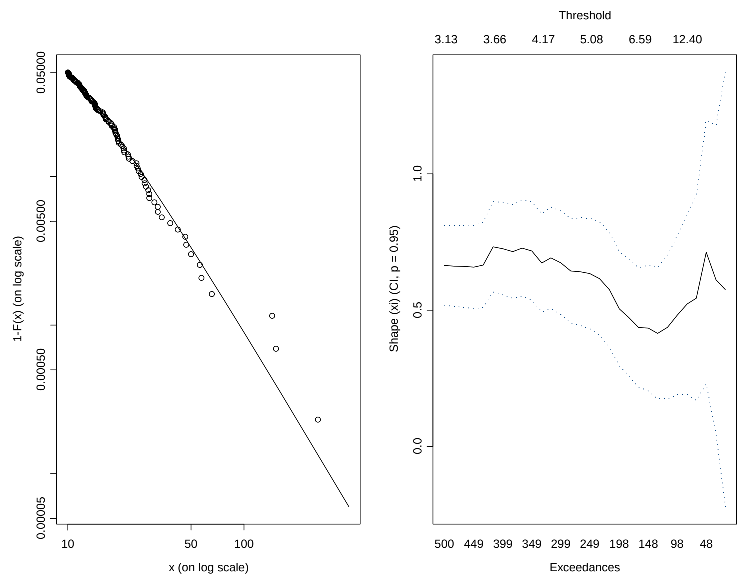


FIGURE 5.18. Diagnostic plots from GPD fit to Danish fire loss data.

5.3.4 Risk Measures

As mentioned in the introduction to this section, two common risk measures are *Value-at-Risk* (VaR) and *expected shortfall* (ES). VaR is a high quantile of the loss distribution. That is, for $0.95 \leq q < 1$, say, VaR_q is the qth quantile of the distribution F

$$\text{VaR}_q = F^{-1}(q) \tag{5.19}$$

where F^{-1} is the inverse of F. For a given probability $q > F(u)$, an estimate of (5.19) based on inverting the tail estimation formula (5.18) is

$$\widehat{\text{VaR}}_q = u + \frac{\hat{\beta}(u)}{\hat{\xi}}\left(\left(\frac{n}{k}(1-q)\right)^{-\hat{\xi}} - 1\right) \tag{5.20}$$

Expected shortfall is the expected loss size, given that VaR_q is exceeded

$$\text{ES}_q = E[X|X > \text{VaR}_q] \tag{5.21}$$

The measure ES_q is related to VaR_q via

$$\text{ES}_q = \text{VaR}_q + E[X - \text{VaR}_q|X > \text{VaR}_q]. \tag{5.22}$$

where the second term in (5.22) is simply the mean of the excess distribution $F_{\text{VaR}_q}(y)$ over the threshold VaR_q. By the translation property of the GPD distribution, the GPD approximation to $F_{\mathrm{VaR}_q}(y)$ has shape parameter ξ and scale parameter $\beta(u) + \xi(\mathrm{VaR}_q - u)$. Consequently, using (5.13)

$$E[X - \mathrm{VaR}_q | X > \mathrm{VaR}_q] = \frac{\beta(u) + \xi(\mathrm{VaR}_q - u)}{1 - \xi} \tag{5.23}$$

provided $\xi < 1$. Combining (5.23) with (5.20) and substituting into (5.22) gives the GPD approximation to ES_q

$$\widehat{\mathrm{ES}}_q = \frac{\widehat{\mathrm{VaR}}_q}{1 - \hat{\xi}} + \frac{\hat{\beta}(u) - \hat{\xi}u}{1 - \hat{\xi}}. \tag{5.24}$$

Example 33 *Computing VaR and ES for negative S&P 500 returns*

The S+FinMetrics/EVIS function riskmeasures computes estimates of VaR_q and ES_q based on the GPD approximations (5.20) and (5.24), respectively, using the information from a "gpd" object. For example, the VaR_q and ES_q estimates for the negative S&P 500 negative returns for $q = 0.95, 0.99$ are computed using

```
> riskmeasures(gpd.sp500.1, p = c(0.95,0.99))
        p quantile  sfall
[1,] 0.95   1.2539 1.7744
[2,] 0.99   2.0790 2.6594
```

That is, with 5% probability the daily return could be as low as -1.254% and, given that the return is less than 1.254%, the average return value is -1.774%. Similarly, with 1% probability the daily return could be as low as -2.079% with an average return of -2.659% given that the return is less than -2.079%.

It is instructive to compare these results to those based on the assumption of normally distributed returns. Using the formulas (5.8) and (5.9), estimates of VaR_q and ES_q for $q = 0.95, 0.99$ may be computed using the following function

```
riskmeasures.normal <- function(data,p=c(0.95,0.99)) {
     mu = colMeans(data)
     sd = colStdevs(data)
     q = mu + sd*qnorm(p)
     sq = (q - mu)/sd
     sf = mu + sd*dnorm(sq)/(1 - pnorm(sq))
     cbind(p, quantile = q, sfall = sf)
}
> riskmeasures.normal(-spto87)
        p quantile    sfall
[1,] 0.95 1.299051 1.635526
[2,] 0.99 1.847814 2.120681
```

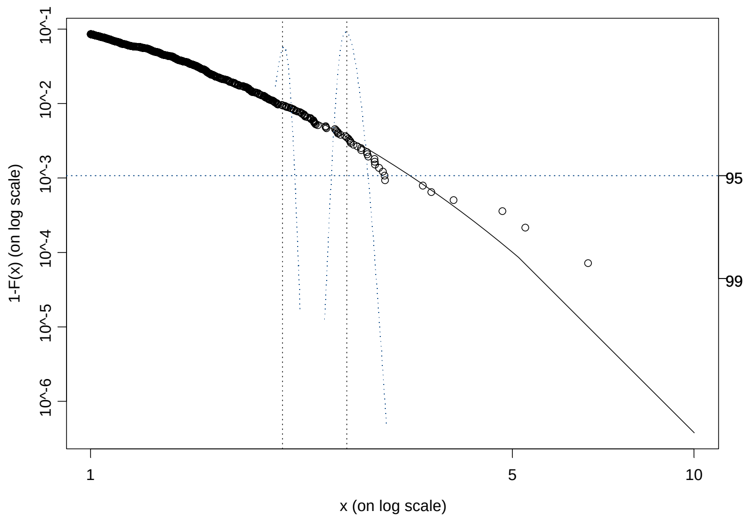


FIGURE 5.19.

The estimates of VaR_q and ES_q based on the normal distribution are fairly close to the estimates based on the GPD for $q = 0.95$. For $q = 0.99$, VaR_q and ES_q based on the normal distribution are a bit smaller than the values based on the GPD.

Estimates and asymptotically valid confidence intervals for VaR_q and ES_q may be computed using the `S+FinMetrics/EVIS` function `gpd.q` and `gpd.sfall`, respectively. Wald-type confidence intervals based on the delta method or likelihood ratio-type confidence intervals based on the profile log-likelihood function may be computed, and these confidence intervals may be visualized on a plot with the tail estimate (5.18). First create plot of the excess distribution using the `S+FinMetrics/EVIS` function `tailplot`

```
> tailplot(gpd.sp500.1)
```

After the plot has been created, the asymptotic confidence intervals for VaR_q and ES_q may be added using

```
> gpd.q(0.99,plot=T)
> gpd.sfall(0.99,plot=T)
```

The combined plots are illustrated in Figure 5.19.

Notice the slightly asymmetric confidence interval for $\text{ES}_{.99}$. This result is due to the uncertainty created by only a few observations in the extreme tails of the distribution.

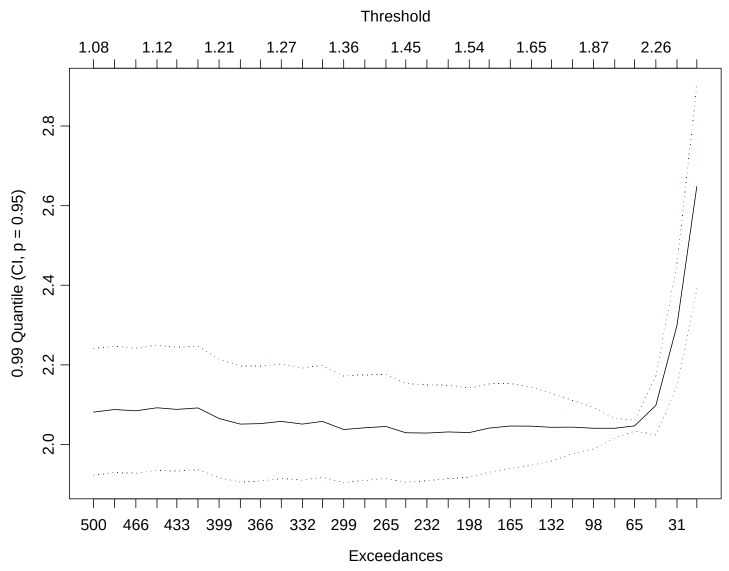


FIGURE 5.20.

The sensitivity of the VaR_q estimates to changes in the threshold u may be investigated using the `S+FinMetrics/EVIS` function `quant`. For example, to see how the $\text{VaR}_{.99}$ estimates vary with u, use

```
> quant(-spto87,p=0.99)
```

which produces the graph in Figure 5.20.

The $\text{VaR}_{.99}$ estimates are stable for $u < 2$.

5.4 Hill's Non-parametric Estimator of Tail Index

The shape parameter ξ, or equivalently, the tail index $a = 1/\xi$, of the GEV and GPD distributions (5.2) and (5.11) may be estimated non-parametrically in a number of ways. A popular method due to Hill (1975) applies to the case where $\xi > 0$ ($\alpha > 0$) so that the data is generated by some fat-tailed distribution in the domain of attraction of a Fréchet type GEV. To describe the Hill estimator, consider a sample of losses $X_1, \ldots, X_T$ and define the order statistics as

$$X_{(1)} \geq X_{(2)} \geq \cdots \geq X_{(T)}$$

For a positive integer k, the *Hill estimator* of ξ is defined as

$$\hat{\xi}^{\text{Hill}}(k) = \frac{1}{k} \sum_{j=1}^{k} \left(\log X_{(j)} - \log X_{(k)} \right) \tag{5.25}$$

and the Hill estimator of α is

$$\hat{\alpha}^{\text{Hill}}(k) = 1/\hat{\xi}^{\text{Hill}}(k) \tag{5.26}$$

The Hill estimators of ξ and α depend on the integer k. Notice that k in (5.26) plays the same role as k in (5.17) for the analysis of the GPD. It can be shown that if F is in the domain of attraction of a GEV distribution, then $\hat{\xi}^{\text{Hill}}(k)$ converges in probability to ξ as $k \to \infty$ and $\frac{k}{n} \to 0$, and that $\hat{\xi}^{\text{Hill}}(k)$ is asymptotically normally distributed with asymptotic variance

$$\text{avar}(\hat{\xi}^{\text{Hill}}(k)) = \frac{\xi^2}{k}$$

By the delta method, $\hat{\alpha}^{\text{Hill}}(k)$ is asymptotically normally distributed with asymptotic variance

$$\text{avar}(\hat{\alpha}^{\text{Hill}}(k)) = \frac{\alpha^2}{k}$$

In practice, the Hill estimators $\hat{\xi}^{\text{Hill}}(k)$ or $\hat{\alpha}^{\text{Hill}}(k)$ are often plotted against k to find the value of k such that the estimator appears stable.

5.4.1 Hill Tail and Quantile Estimation

Suppose that the loss distribution F is such that $1 - F(x) = x^{-\alpha}L(x)$ with $\alpha = 1/\xi > 0$, where $L(x)$ is a slowly varying function. Let $x > X_{(k+1)}$ where $X_{(k+1)}$ is a high order statistic. Then the Hill estimator of $F(x)$ is given by

$$\hat{F}^{\text{Hill}}(x) = 1 - \frac{k}{n}\left(\frac{x}{X_{(k+1)}}\right)^{-\hat{\alpha}^{\text{Hill}}(k)}, \; x > X_{(k+1)} \tag{5.27}$$

Inverting the Hill tail estimator (5.27) gives the Hill quantile estimator

$$\hat{x}^{\text{Hill}}_{q,k} = X_{(k+1)} - X_{(k+1)}\left(\left(\frac{n}{k}(1-q)\right)^{-\hat{\xi}^{\text{Hill}}(k)} - 1\right) \tag{5.28}$$

where $q > 1 - k/n$. The Hill quantile estimator (5.28) is very similar to the ML GPD quantile estimator (5.20) with $u = X_{(k+1)}$.

Example 34 *Nonparametric estimation of ξ for Danish fire loss data*

The Hill estimates of α, ξ and the quantile $x_{q,k}$ may be computed and plotted using the `S+FinMetrics/EVIS` function `hill`. The arguments expected by `hill` are

```
> args(hill)
function(data, option = "alpha", start = 15, end = NA,
p = NA, ci = 0.95, plot = T, reverse = F,
auto.scale = T, labels = T, ...)
```

where `data` is a univariate numeric vector or “`timeSeries`”, `option` determines if α (“`alpha`”), ξ (“`xi`”) or $x_{q,k}$ (“`quantile`”) is to be computed, `start` and `end` specify the starting and ending number of order statistics to use in computing the estimates, `p` specifies the probability required when `option=\quantile"`, `ci` determines the probability for asymptotic confidence bands, and `plot` determines if a plot is to be created. To illustrate the use of `hill`, consider the computation of (5.25) for the Danish fire loss data using all of the order statistics less than $X_{(15)}$

```
> hill.danish = hill(danish, option="xi")
> class(hill.danish)
[1] "data.frame"
> names(hill.danish)
[1] "xi"        "orderStat" "threshold"
```

The function `hill` returns a data frame with components `xi` containing the estimates of ξ, `orderStat` containing the order statistic labels k, and `threshold` containing the order statistic or threshold values $X_{(k)}$. Since the default option `plot=T` is used, `hill` also produces the plot shown in Figure 5.21. For $k > 120$ ($X_{(k)} < 9$), $\hat{\xi}^{\text{Hill}}(k)$ is fairly stable around 0.7. The GPD estimate of ξ with threshold $u = 10$ is 0.497. The Hill estimates for threshold values near 10 are

```
> idx = (hill.danish$threshold >= 9.8 &
+ hill.danish$threshold <= 10.2)
> hill.danish[idx,]
         xi orderStat threshold
2059 0.6183       109     9.883
2060 0.6180       108    10.011
2061 0.6173       107    10.072
2062 0.6191       106    10.137
2063 0.6243       105    10.178
2064 0.6285       104    10.185
```

The 99% quantile estimates (5.28) for $15 \leq k \leq 500$ are computed using

```
> hill.danish.q = hill(danish, option="quantile", p=0.99,
+ end=500)
```

and are illustrated in Figure 5.22.

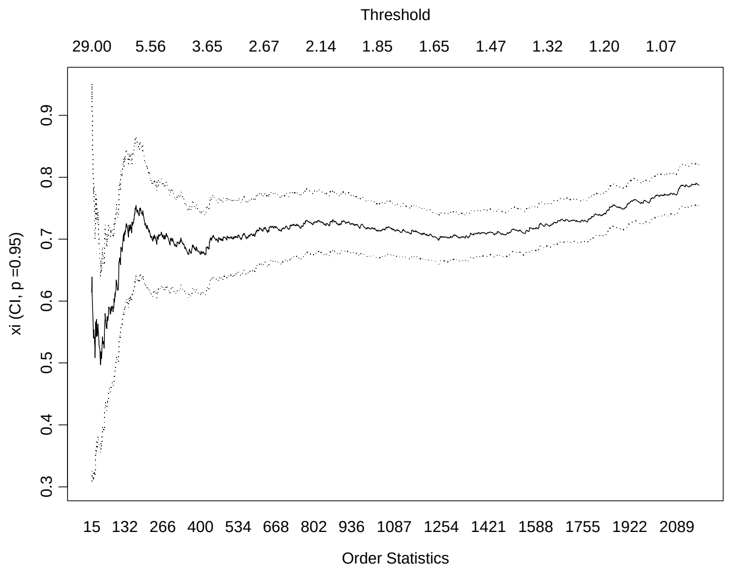


FIGURE 5.21. Hill estimates of ξ for the Danish fire loss data.

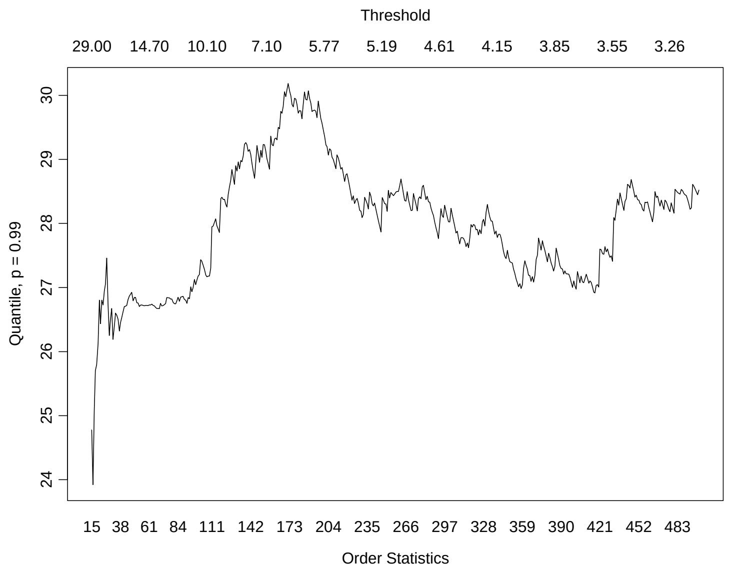


FIGURE 5.22. Hill estimates of 1% quantile of Danish fire loss data.

5.5 References

Alexander, C. (2001). *Market Models: A Guide to Financial Data Analysis.* John Wiley & Sons, Chichester, UK.

Carmona, R. (2004). *Statistical Analysis of Financial Data in S-PLUS.* Springer-Verlag, New York.

Carmona, R. and J. Morrisson (2001). "Heavy Tails and Copulas with Evanesce," ORFE Tech. Report, Princeton University.

Coles, S (2001). *An Introduction to Statistical Modeling of Extreme Values.* Springer-Verlag, London.

Danielsson, J. and C.G. De Vries (1997). "Tail Index and Quantile Estimation with Very High Frequency Data," *Journal of Empirical Finance*, 4, 241-257.

Diebold, F.X., T. Schuermann, and J.D. Stroughair (1997), "Pitfalls and Opportunities in the Use of Extreme Value Theory in Risk Management," *Advances in Computational Management Science* 2, 3-12.

Embrechts, P. C. Kloppelberg, and T. Mikosch (1997). *Modelling Extremal Events.* Springer-Verlag, Berlin.

Embrechts, P., A. McNeil, and D. Straumann (2000). "Correlation and Dependence in Risk Management: Properties and Pitfalls," in M. Dempster and H. K. Moffatt (eds.), *Risk management: value at risk and beyond.* Cambridge University Press, Cambridge.

Fisher, R. and L. Tippett (1928). "Limiting Forms of the Frequency Distribution of the Largest or Smallest Member of a Sample," *Proceedings of the Cambridge Philosophical Society* 24, 180-190.

Greene, W. (2000). *Econometric Analysis*, *Fourth Edition.* Prentice Hall, Upper Saddle River.

Hill, B.M. (1975). "A Simple General Approach to Inference about the Tail of a Distribution," *The Annals of Statistics*, 3, 1163-1174.

Jorian, P. (2001). *Value at Risk, Second Edition.* McGraw-Hill, New York.

Longin, F.M. (2000). "From Value-at-Risk to Stress Testing: The Extreme Value Approach," *Journal of Banking an Finance* 24, 1097-1130.

McNeil, A.J. (1998). “On Extremes and Crashes,” *RISK*, January, page 99.

McNeil, A.J. (1998). “Calculating Quantile Risk Measures for Financial Returns using Extreme Value Theory,” ETH E-Collection `http://e-collection.ethbib.ethz.ch/show?type=bericht&nr=85`.

McNeil, A.J. (1999). “Extreme Value Theory for Risk Managers,” in *Internal Modelling and CAD II,* RISK Books, 93-113.

McNeil, A.J. and R. Frey (2000). “Estimation of Tail-Related Risk Measures for Heteroskedastic Financial Time Series: An Extreme Value Approach,” *Journal of Empirical Finance* 7, 271-300.

McNeil A.J. and T. Saladin (2000). “Developing Scenarios for Future Extreme Losses Using the POT Method,” in P. Embrechts (ed.) *Extremes and Integrated Risk Management* , RISK books, London.

Morrison, J. E. (2001). *Extreme Value Statistics with Apllications in Hydrology and Financial Engineering*, Ph.D. thesis, Princeton University.

RiskMetrics, 1995. “RiskMetrics Technical Document,” J.P. Morgan, 3rd ed.

Tsay, R.S. (2001). *Analysis of Financial Time Series*, John Wiley & Sons, New York.

6
Time Series Regression Modeling

6.1 Introduction

Time series regression techniques are widely used in the analysis of financial data and for estimating and testing models for asset prices and returns like the capital asset pricing model and the arbitrage pricing model. They are used to uncover and exploit predictive relationships between financial variables. For example, the predictability of asset returns using valuation ratios like dividend/price, earnings/price and book/market is usually established using time series regression techniques, and the resulting regression models are used to forecast future returns. Time series regression techniques are also used for testing the informational efficiency of financial markets. Market efficiency often implies that certain financial variables should not be predictable based on observable information, and time series regression techniques may be used to verify efficiency implications.

Regression modeling with financial time series requires some care because the time series properties of the data can influence the properties of standard regression estimates and inference methods. In general, standard regression techniques are appropriate for the analysis of $I(0)$/stationary data. For example, asset returns are often treated as stationary and ergodic, and standard regression techniques are then used to estimate models involving asset returns. For nonstationary trending data like asset prices, however, standard regression techniques may or may not be appropriate depending on the nature of the trend. This chapter discusses regression modeling techniques appropriate for $I(0)$/stationary and introduces and illustrates the use of various `S+FinMetrics` functions designed for time series regression analysis.

The rest of the chapter is organized as follows: Section 6.2 gives an overview of the linear time series regression model and covers estimation, goodness of fit, inference and residual diagnostics. Section 6.3 introduces the `S+FinMetrics` function `OLS` that extends the `S-PLUS` linear model function `lm` to handle general time series regression and illustrates the use of `OLS` through examples. Section 6.4 reviews dynamic regression models involving distributed lags of the dependent and explanatory variables and gives examples of how `OLS` may be used analyze these models. Section 6.5 discusses heteroskedasticity and autocorrelation consistent coefficient covariance matrices and their use in constructing robust standard errors for estimated regression coefficients. Section 6.6 ends the chapter with a discussion of recursive regression techniques for assessing the parameter stability of time series regression models.

In this chapter, the technical details of time series regression are kept to a minimum. Excellent treatments of time series regression models from an econometric perspective are given in Hamilton (1994) and Hayashi (2000). Many applications of time series regression to financial data can be found in Mills (1999).

6.2 Time Series Regression Model

Consider the *linear time series regression model*

$$y_t = \beta_0 + \beta_1 x_{1t} + \cdots + \beta_k x_{kt} + \varepsilon_t = \mathbf{x}_t'\boldsymbol{\beta} + \varepsilon_t,\ t = 1, \ldots, T \tag{6.1}$$

where $\mathbf{x}_t = (1, x_{1t}, \ldots, x_{kt})'$ is a $(k+1) \times 1$ vector of explanatory variables, $\boldsymbol{\beta} = (\beta_0, \beta_1, \ldots, \beta_k)'$ is a $(k+1)\times 1$ vector of coefficients, and ε_t is a random error term. In matrix form the model is expressed as

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon} \tag{6.2}$$

where $\mathbf{y}$ and $\boldsymbol{\varepsilon}$ are $(T \times 1)$ vectors and $\mathbf{X}$ is a $(T \times (k+1))$ matrix.

The standard assumptions of the time series regression model are (e.g., Hayashi 2000, Chaps. 1 and 2):

- The linear model (6.1) is correctly specified.
- $\{y_t, \mathbf{x}_t\}$ is jointly stationary and ergodic.
- The regressors $\mathbf{x}_t$ are *predetermined*: $E[x_{is}\varepsilon_t] = 0$ for all $s \leq t$ and $i = 1, \ldots, k$.
- $E[\mathbf{x}_t\mathbf{x}_t'] = \boldsymbol{\Sigma}_{XX}$ is of full rank $k + 1$.

- $\{\mathbf{x}_t\varepsilon_t\}$ is an uncorrelated process with finite $(k+1)\times(k+1)$ covariance matrix $E[\varepsilon_t^2\mathbf{x}_t\mathbf{x}_t'] = \mathbf{S} = \sigma^2\mathbf{\Sigma}_{XX}$.

The second assumption rules out trending regressors, the third rules out endogenous regressors but allows lagged dependent variables, the fourth avoids redundant regressors or exact multicolinearity, and the fifth implies that the error term is a serially uncorrelated process with constant unconditional variance σ^2. In the time series regression model, the regressors $\mathbf{x}_t$ are random and the error term ε_t is not assumed to be normally distributed.

6.2.1 *Least Squares Estimation*

Ordinary least squares (OLS) estimation is based on minimizing the sum of squared residuals

$$\mathrm{SSR}(\boldsymbol{\beta}) = \sum_{t=1}^{T}(y_t - \mathbf{x}_t'\boldsymbol{\beta})^2 = \sum_{t=1}^{T}\varepsilon_t^2$$

and produces the fitted model

$$y_t = \mathbf{x}_t'\hat{\boldsymbol{\beta}} + \hat{\varepsilon}_t,\ t = 1, \ldots, T$$

where $\hat{\boldsymbol{\beta}} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{y}$ and $\hat{\varepsilon}_t = y_t - \hat{y}_t = y_t - \mathbf{x}_t'\hat{\boldsymbol{\beta}}$. The error variance is estimated as $\hat{\sigma}^2 = \hat{\boldsymbol{\varepsilon}}'\hat{\boldsymbol{\varepsilon}}/(T-k-1)$.

Under the assumptions described above, the OLS estimates $\hat{\boldsymbol{\beta}}$ are consistent and asymptotically normally distributed. A consistent estimate of the asymptotic variance of $\hat{\boldsymbol{\beta}}$, $\mathrm{avar}(\hat{\boldsymbol{\beta}})$, is given by[1]

$$\widehat{\mathrm{avar}}(\hat{\boldsymbol{\beta}}) = \hat{\sigma}^2(\mathbf{X}'\mathbf{X})^{-1} \tag{6.3}$$

Estimated standard errors for $\hat{\beta}_i$ $(i = 0, ..., k)$, denoted $\widehat{\mathrm{SE}}(\hat{\beta}_i)$, are given by the square root of the diagonal elements of (6.3).

6.2.2 *Goodness of Fit*

Goodness of fit is summarized by the R^2 of the regression

$$R^2 = 1 - \frac{\hat{\boldsymbol{\varepsilon}}'\hat{\boldsymbol{\varepsilon}}}{(\mathbf{y} - \bar{y}\mathbf{1})'(\mathbf{y} - \bar{y}\mathbf{1})}$$

[1]The following convention is used throughout this book. A consistent and asymptotically normal estimator $\hat{\boldsymbol{\beta}}$ satisfies $\sqrt{T}(\hat{\boldsymbol{\beta}} - \boldsymbol{\beta}) \xrightarrow{d} N(0, \mathbf{V})$ where $\xrightarrow{d}$ denotes convergence in distribution. Call $\mathbf{V}$ the asymptotic variance of $\sqrt{T}(\hat{\boldsymbol{\beta}} - \boldsymbol{\beta})$ and $T^{-1}\mathbf{V}$ the asymptotic variance of $\hat{\boldsymbol{\beta}}$. Use the notation $\hat{\boldsymbol{\beta}} \overset{A}{\sim} N(\boldsymbol{\beta}, T^{-1}\mathbf{V})$ to denote the asymptotic approximating distribution of $\hat{\boldsymbol{\beta}}$ and $\widehat{avar}(\hat{\boldsymbol{\beta}})$ to denote the asymptotic variance $T^{-1}\mathbf{V}$.

where $\bar{y}$ is the sample mean of y_t and $\mathbf{1}$ is a $(T \times 1)$ vector of 1's. R^2 measures the percentage of the variability of y_t that is explained by the regressors, $\mathbf{x}_t$. The usual R^2 has the undesirable feature of never decreasing as more variables are added to the regression, even if the extra variables are irrelevant. To remedy this, the R^2 statistic may be adjusted for degrees of freedom giving

$$R_a^2 = 1 - \frac{\hat{\boldsymbol{\varepsilon}}'\hat{\boldsymbol{\varepsilon}}/(T-k)}{(\mathbf{y}-\bar{y}\mathbf{1})'(\mathbf{y}-\bar{y}\mathbf{1})/(T-1)} = \frac{\hat{\sigma}^2}{\widehat{\text{var}}(y_t)}$$

The *adjusted* R^2, R_a^2, may decrease with the addition of variables with low explanatory power. If fact, it can be shown (e.g., Greene 2000, p. 240) that R_a^2 will fall (rise) when a variable is deleted from the regression if the absolute value of the t-statistic associated with this variable is greater (less) than 1.

6.2.3 Hypothesis Testing

The simple null hypothesis

$$H_0 : \beta_i = \beta_i^0$$

is tested using the *t-ratio*

$$t = \frac{\hat{\beta}_i - \beta_i^0}{\widehat{\text{SE}}(\hat{\beta}_i)} \tag{6.4}$$

which is asymptotically distributed $N(0,1)$ under the null. With the additional assumption of *iid* Gaussian errors and regressors independent of the errors for all t, $\hat{\boldsymbol{\beta}}$ is normally distributed in finite samples and the t-ratio is distributed Student-t with $T-k-1$ degrees of freedom.

Linear hypotheses of the form

$$H_0 : \mathbf{R}\boldsymbol{\beta} = \mathbf{r} \tag{6.5}$$

where $\mathbf{R}$ is a fixed $q \times (k+1)$ matrix of rank q and $\mathbf{r}$ is a fixed $q \times 1$ vector are tested using the *Wald statistic*

$$\text{Wald} = (\mathbf{R}\hat{\boldsymbol{\beta}} - \mathbf{r})' \left[\mathbf{R}\widehat{\text{avar}}(\hat{\boldsymbol{\beta}})\mathbf{R}'\right]^{-1} (\mathbf{R}\hat{\boldsymbol{\beta}} - \mathbf{r}) \tag{6.6}$$

Under the null, the Wald statistic is asymptotically distributed $\chi^2(q)$. Under the additional assumption of *iid* Gaussian errors and regressors independent of the errors for all t, Wald$/q$ is distributed $F(q, T-k-1)$ in finite samples.

The statistical significance of all of the regressors excluding the constant is captured by the F-statistic

$$F = \frac{R^2/k}{(1-R^2)/(T-k-1)}$$

which is distributed $F(k, T-k-1)$ under the null hypothesis that all slope coefficients are zero and the errors are *iid* Gaussian.

6.2.4 Residual Diagnostics

In the time series regression models, several residual diagnostic statistics are usually reported along with the regression results. These diagnostics are used to evaluate the validity of some of the underlying assumptions of the model and to serve as warning flags for possible misspecification. The most common diagnostic statistics are based on tests for normality and serial correlation in the residuals of (6.1).

The most common diagnostic for serial correlation based on the estimated residuals $\hat{\varepsilon}_t$ is the *Durbin-Watson statistic*

$$\mathrm{DW} = \frac{\sum_{t=2}^{T}(\hat{\varepsilon}_t - \hat{\varepsilon}_{t-1})^2}{\sum_{t=1}^{T}\hat{\varepsilon}_t^2}.$$

It is easy to show that $\mathrm{DW} \approx 2(1-\hat{\rho})$, where $\hat{\rho}$ is the estimated correlation between and $\hat{\varepsilon}_t$ and $\hat{\varepsilon}_{t-1}$. Hence, values of DW range between 0 and 4. Values of DW around 2 indicate no serial correlation in the errors, values less than 2 suggest positive serial correlation, and values greater than 2 suggest negative serial correlation[2]. Another common diagnostic for serial correlation is the Ljung-Box modified Q statistic discussed in Chapter 3.

Although error terms in the time series regression model are not assumed to be normally distributed, severe departures from normality may cast doubt on the validity of the asymptotic approximations utilized for statistical inference especially if the sample size is small. Therefore, another diagnostic statistic commonly reported is the Jarque-Bera test for normality discussed in Chapter 3.

6.3 Time Series Regression Using the `S+FinMetrics` Function `OLS`

Ordinary least squares estimation of the time series regression model (6.1) in `S-PLUS` is carried out with the `S+FinMetrics` function `OLS`. `OLS` extends the `S-PLUS` linear model function `lm` to handle time series regression in a

[2]The DW statistic is an optimal test only for the special case that ε_t in (1) follows an AR(1) process and that the regressors $\mathbf{x}_t$ are fixed. Critical values for the bounding distribution of DW in this special case are provided in most econometrics textbooks. However, in practice there is often little reason to believe that ε_t follows an AR(1) process and the regressors are rarely fixed and so the DW critical values are of little practical use.

more natural way. The arguments expected by OLS are similar to those for lm:

```
> args(OLS)
function(formula, data, weights, subset, na.rm = F, method
= "qr", contrasts = NULL, start = NULL, end = NULL,...)
```

The main arguments are formula, which is an S-PLUS formula with the response variable(s) on the left hand side of the ~ character and the response variables separated by + on the right hand side[3], and data, which is "timeSeries" or data frame in which to interpret the variables named in the formula and subset arguments. The other arguments will be explained and their use will be illustrated in the examples to follow.

The function OLS produces an object of class "OLS" for which there are print, summary, plot and predict methods as well as extractor functions coefficients (coef), residuals (resid), fitted.values (fitted), vcov and IC. The extractor functions coef, resid and fitted are common to many S-PLUS model objects. Note that if "timeSeries" objects are used in the regression then the extracted residuals and fitted values are also "timeSeries" objects. The extractor functions vcov, which extracts $\widehat{\text{avar}}(\hat{\boldsymbol{\beta}})$, and IC, which extracts information criteria, are specific to S+FinMetrics model objects and work similarly to the extractor functions vcov and AIC from the MASS library.

There are several important differences between lm and OLS. First, the argument formula is modified to accept lagged values of the dependent variable through the use of AR terms and lagged values of regressors through the use of the S+FinMetrics functions tslag and pdl. Second, subset regression for "timeSeries" data is simplified through the use of the start and end options. Third, summary output includes time series diagnostic measures and standard econometric residual diagnostic tests may be computed from OLS objects. Fourth, heteroskedasticity consistent as well as heteroskedasticity and autocorrelation consistent coefficient covariance matrices may be computed from OLS objects.

The use of OLS for time series regression with financial data is illustrated with the following examples

Example 35 *Estimating and testing the capital asset pricing model*

The famous *Capital Asset Pricing Model* (CAPM) due to Sharpe, Litner and Mosen is usually estimated using the *excess return single index model*

$$R_{it} - r_{ft} = \alpha_i + \beta_i(R_{Mt} - r_{ft}) + \varepsilon_{it},\ i = 1, \ldots, N; t = 1, \ldots, T \qquad (6.7)$$

where R_{it} is the return on asset i ($i = 1, \ldots, N$) between time periods $t-1$ and t, R_{Mt} is the return on a *market index* portfolio between time

[3]See Chapter 1 for details on specifying formulas in S-PLUS.

periods $t-1$ and t, r_{ft} denotes the rate of return between times $t-1$ and t on a risk-free asset, and ε_{it} is a normally distributed random error such that $\varepsilon_{it} \sim GWN(0, \sigma_i^2)$. The market index portfolio is usually some well diversified portfolio like the S&P 500 index, the Wilshire 5000 index or the CRSP[4] equally or value weighted index. In practice, r_{ft} is taken as the T-bill rate to match the investment horizon associated with R_{it}. The CAPM is an equilibrium model for asset returns and, if R_{Mt} is the value-weighted portfolio of all publicly traded assets, it imposes the relationship

$$E[R_{it}] - r_{ft} = \beta_i (E[R_{Mt}] - r_{ft}).$$

In other words, the above states that the *risk premium* on asset i is equal to its beta, β_i, times the risk premium on the market portfolio. Hence, β_i is the appropriate risk measure for asset i. In the excess returns single index model, the CAPM imposes the testable restriction that $\alpha_i = 0$ for all assets.

The intuition behind the CAPM is as follows. The market index R_{Mt} captures "macro" or market-wide systematic risk factors that affect all returns in one way or another. This type of risk, also called *covariance risk, systematic risk* and *market risk*, cannot be eliminated in a well diversified portfolio. The beta of an asset captures the magnitude of this nondiversifiable risk. The random error term ε_{it} represents random "news" that arrives between time $t-1$ and t that captures "micro" or firm-specific risk factors that affect an individual asset's return that are not related to macro events. For example, ε_{it} may capture the news effects of new product discoveries or the death of a CEO. This type of risk is often called *firm specific risk*, *idiosyncratic risk*, *residual risk* or *non-market risk*. This type of risk can be eliminated in a well diversified portfolio. The CAPM says that in market equilibrium the risk premium on any asset is directly related to the magnitude of its nondiversifiable risk (beta). Diversifiable risk is not priced; i.e., diversifiable risk does not command a risk premium because it can be eliminated by holding a well diversified portfolio.

In the CAPM, the independence between R_{Mt} and ε_{it} allows the unconditional variability of an asset's return R_{it} to be decomposed into the variability due to the market index, $\beta_i^2 \sigma_M^2$, plus the variability of the firm specific component, σ_i^2. The proportion of the variance R_{it} explained by the variability in the market index is the usual regression R^2 statistic. Accordingly, $1-R^2$ is then the proportion of the variability of R_{it} that is due to firm specific factors. One can think of R^2 as measuring the proportion of risk in asset i that cannot be diversified away when forming a portfolio and $1-R^2$ as the proportion of risk that can be diversified away.

[4]CRSP refers to the Center for Research in Security Prices at the University of Chicago.

Estimating the CAPM Using the S+FinMetrics Function OLS

Consider the estimation of the CAPM regression (6.7) for Microsoft using monthly data over the ten year period January 1990 through January 2000. The S&P 500 index is used for the market proxy, and the 30 day T-bill rate is used for the risk-free rate. The S+FinMetrics "timeSeries" singleIndex.dat contains the monthly price data for Microsoft, and the S&P 500 index and the "timeSeries" rf.30day contains the monthly 30 day T-bill rate. The excess return data are created using

```
> colIds(singleIndex.dat)
[1] "MSFT"  "SP500"
> colIds(rf.30day)
[1] "RF"
> ret.ts = getReturns(singleIndex.dat, type="continuous")
> excessRet.ts = seriesMerge(ret.ts,log(1+rf.30day))
> excessRet.ts[,"MSFT"] = excessRet.ts[,"MSFT"] -
+ excessRet.ts[,"RF"]
> excessRet.ts[,"SP500"] = excessRet.ts[,"SP500"] -
+ excessRet.ts[,"RF"]
> excessRet.ts = excessRet.ts[,1:2]
```

Time plots and a scatterplot of the excess returns created by

```
> par(mfrow=c(2,1))
> plot(excessRet.ts, plot.args=list(lty=c(1,3)),
+ main="Monthly excess returns on Microsoft and S&P 500 Index")
> legend(0, -0.2, legend=c("MSFT","S&P 500"), lty=c(1,3))
> plot(seriesData(excessRet.ts[,"SP500"]),
+ seriesData(excessRet.ts[,"MSFT"]),
+ main="Scatterplot of Returns",
+ xlab="SP500", ylab="MSFT")
```

are given in Figure 6.1. The returns on Microsoft and the S&P 500 index appear stationary and ergodic and tend to move in the same direction over time with the returns on Microsoft being more volatile than the returns on the S & P 500 index. The estimate of the CAPM regression for Microsoft using OLS is:

```
> ols.fit = OLS(MSFT~SP500, data=excessRet.ts)
> class(ols.fit)
[1] "OLS"
```

OLS produces an object of class "OLS" with the following components

```
> names(ols.fit)
 [1] "R"         "coef"      "df.resid"  "fitted"
 [5] "residuals" "assign"    "contrasts" "ar.order"
 [9] "terms"     "call"
```

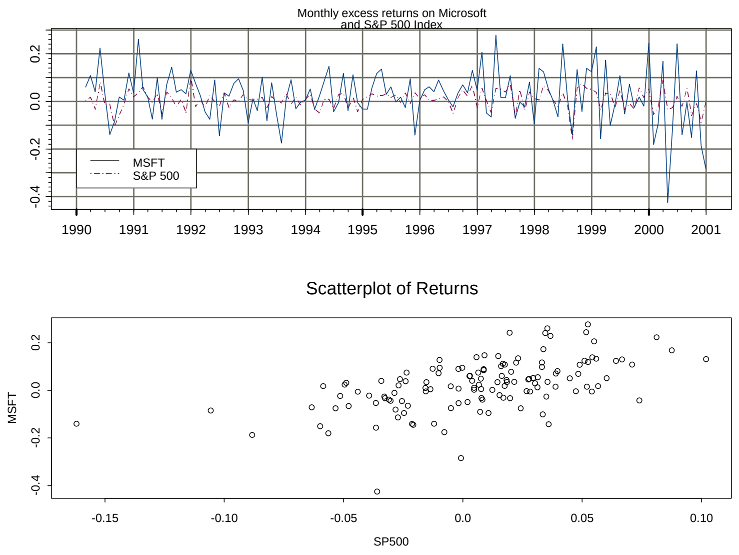


FIGURE 6.1. Monthly excess returns on Microsoft and the S&P 500 index.

The results of the OLS fit are displayed using the generic print and summary methods. The print method produces minimal output:

```
> ols.fit
Call:
OLS(formula = MSFT ~SP500, data = excessRet.ts)

Coefficients:
 (Intercept)  SP500
 0.0128      1.5259

Degrees of freedom: 131 total; 129 residual
Time period: from Feb 1990 to Dec 2000
Residual standard error: 0.09027
```

Notice that since the object specified in data is a "timeSeries", the start and end dates of the estimation sample are printed. The summary method produces the standard econometric output:

```
> summary(ols.fit)
Call:
OLS(formula = MSFT ~SP500, data = excessRet.ts)

Residuals:
     Min      1Q  Median      3Q     Max
 -0.3835 -0.0566  0.0023  0.0604  0.1991

Coefficients:
             Value Std. Error t value Pr(>|t|)
(Intercept) 0.0128 0.0080     1.6025  0.1115
      SP500 1.5259 0.1998     7.6354  0.0000

Regression Diagnostics:

         R-Squared 0.3113
Adjusted R-Squared 0.3059
Durbin-Watson Stat 2.1171

Residual Diagnostics:
                Stat P-Value
Jarque-Bera 41.6842  0.0000
  Ljung-Box 11.9213  0.9417

Residual standard error: 0.09027 on 129 degrees of freedom
Time period: from Feb 1990 to Dec 2000
F-statistic: 58.3 on 1 and 129 degrees of freedom, the
p-value is 4.433e-012
```

The estimated value for β for Microsoft is 1.526 with an estimated standard error $\widehat{\mathrm{SE}}(\hat{\beta}) = 0.200$. An approximate 95% confidence interval for β is $\hat{\beta} \pm 2\cdot\, \widehat{\mathrm{SE}}(\hat{\beta}) = [1.126, 1.926]$, and so Microsoft is judged to be riskier than the S&P 500 index. The estimated value of α is 0.013 with an estimated standard error of $\widehat{\mathrm{SE}}(\hat{\alpha}) = 0.008$. An approximate 95% confidence interval for α is $\hat{\alpha} \pm 2\cdot\, \widehat{\mathrm{SE}}(\hat{\alpha}) = [-0.003, 0.029]$. Since $\alpha = 0$ is in the confidence interval the CAPM restriction hold for Microsoft. The percentage of nondiversifiable (market specific) risk is $R^2 = 0.31$ and the percentage of diversifiable (firm specific) risk is $1 - R^2 = 0.69$. The estimated magnitude of diversifiable risk is $\hat{\sigma} = 0.090$ or 9% per month. Notice that the Jarque-Bera statistic indicates that the residuals from the CAPM regression are not normally distributed. The DW and Ljung-Box statistics, however, indicate that the residuals are serially uncorrelated (at least at the first lag).

The extractor functions for an "`OLS`" object are used to extract the vectors of estimated coefficients $\hat{\boldsymbol{\beta}}$, fitted values $\hat{\mathbf{y}}$, residuals $\hat{\boldsymbol{\varepsilon}}$ and asymptotic variance matrix $\widehat{\mathrm{avar}}(\hat{\boldsymbol{\beta}})$:

```
> coef(ols.fit)
 (Intercept) SP500
     0.01281 1.526
```

```
> fitted(ols.fit)[1:3]
 Positions        1
 Feb 1990   0.01711
 Mar 1990   0.03965
 Apr 1990  -0.03927

> resid(ols.fit)[1:3]
 Positions       1
 Feb 1990  0.04258
 Mar 1990  0.06868
 Apr 1990  0.07870

> vcov(ols.fit)
           (Intercept)      SP500
(Intercept)  0.00006393 -0.0002618
      SP500 -0.00026181  0.0399383
```

Notice that to use the extractor functions `residuals` and `fitted.values` one only has to type `resid` and `fitted`. Since the data used for estimation is a "`timeSeries`" object, the extracted residuals and fitted values are also "`timeSeries`" objects.

To illustrate the use of the extractor functions, the t-statistics for testing the null hypothesis that the intercept is zero and the slope is unity are

```
> (coef(ols.fit)-c(0,1))/sqrt(diag(vcov(ols.fit)))
 (Intercept) SP500
       1.603 2.631
```

and summary statistics for the residuals using the `S+FinMetrics` function `summaryStats` are

```
> summaryStats(residuals(ols.fit))

Sample Quantiles:
     min       1Q   median      3Q    max
 -0.3835 -0.05661 0.002342 0.06037 0.1991

Sample Moments:
        mean     std skewness kurtosis
 -7.204e-018 0.08993  -0.7712    5.293

Number of Observations:  131
```

Testing Linear Restrictions

The CAPM regression (6.7) in matrix form is (6.2) with $\mathbf{x}_t = (1, R_{Mt} - r_{ft})'$ and $\boldsymbol{\beta} = (\alpha, \beta)'$. Consider testing the joint null hypothesis $H_0 : \alpha = 0$ and

$\beta = 1$. This hypothesis imposes two linear restrictions on the parameter vector $\boldsymbol{\beta} = (\alpha, \beta)'$ that may be written in the form (6.5) with

$$\mathbf{R} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \ \mathbf{r} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

The Wald statistic (6.6) may be computed directly as

```
> Rmat = diag(2)
> rvec = c(0,1)
> bhat = coef(ols.fit)
> avarRbhat = Rmat%*%vcov(ols.fit)%*%t(Rmat)
> wald.stat =
+ t(Rmat%*%bhat-rvec)%*%solve(avarRbhat)%*%(Rmat%*%bhat-rvec)
> as.numeric(wald.stat)
[1] 11.17
> p.value = 1 - pchisq(wald.stat,2)
> p.value
[1] 0.003745
```

The small p-value suggests that null $H_0 : \alpha = 0$ and $\beta = 1$ should be rejected at any reasonable significance level. The F-statistic version of the Wald statistic based on normal errors is

```
> F.stat = wald.stat/2
> p.value = 1 - pf(F.stat,2,ols.fit$df.resid)
> p.value
[1] 0.004708
```

and also suggests rejection of the null hypothesis.

The F-statistic version of the Wald statistic for general linear restrictions of the form (6.5) may be conveniently computed using the `S+FinMetrics` function `waldTest`. For example,

```
> waldTest(ols.fit,Intercept==0,SP500==1)

Wald Test of Coefficients:

 Null Hypothesis: constraints are true
  Test Statistic: 5.587
Dist. under Null: F with ( 2 , 129 ) degrees of freedom
         P-value: 0.004708
```

produces the F-statistic version of the Wald test for the null hypothesis $H_0 : \alpha = 0$ and $\beta = 1$. Notice how the restrictions under the null being tested are reflected in the call to `waldTest`. More complicated linear restrictions like $H_0 : \alpha + 2\beta = 2$ are also easily handled

```
> waldTest(ols.fit,Intercept-2*SP500==2)
```

```
Wald Test of Coefficients:

 Null Hypothesis: constraints are true
  Test Statistic: 157.8
Dist. under Null: F with ( 1 , 129 ) degrees of freedom
         P-value: 0
```

Likelihood ratio (LR) statistics for testing linear hypotheses may also be computed with relative ease since the OLS estimates are the maximum likelihood estimates assuming the errors have a normal distribution. The log-likelihood value of the OLS fit assuming normal errors may be extracted using the S+FinMetrics function IC. For example, the log-likelihood for the unrestricted CAPM fit is

```
> IC(ols.fit, "loglike")
[1] 130.2
```

Consider testing the CAPM restriction $H_0 : \alpha = 0$ using a LR statistic. The restricted OLS fit, imposing $\alpha = 0$, is computed using

```
> ols.fit2 = OLS(MSFT~SP500-1,data=excessRet.ts)
```

The LR statistic is then computed as

```
> LR = -2*(IC(ols.fit2,"loglike")-IC(ols.fit,"loglike"))
> LR
[1] 2.571
> 1 - pchisq(LR,1)
[1] 0.1089
```

Given the p-value of 0.109, the CAPM restriction is not rejected at the 10% significance level.

Graphical Diagnostics

Graphical summaries of the OLS fit are produced with the generic plot function. By default, plot produces a menu of plot choices:

```
> plot(ols.fit)
Make a plot selection (or 0 to exit):

1: plot: all
2: plot: response vs fitted values
3: plot: response and fitted values
4: plot: normal QQ-plot of residuals
5: plot: residuals
6: plot: standardized residuals
7: plot: residual histogram
8: plot: residual ACF
```

Plot Function	**Description**
`xygPlot`	Trellis xyplot with grid and strip.text options
`rvfPplot`	Trellis response vs. fitted plot with grid and strip.text options
`rafPlot`	Trellis plot of response and fitted values
`histPlot`	Trellis density estimate with strip.text options
`qqPlot`	Trellis QQ-plot with grid and strip.text options
`residPlot`	Trellis plot of residuals
`acfPlot`	Trellis ACF plot

TABLE 6.1. `S+FinMetrics` utility Trellis plotting functions

```
9: plot: residual PACF
10: plot: residual^2 ACF
11: plot: residual^2 PACF
Selection:
```

The plot choices are different from those available for "`lm`" objects and focus on time series diagnostics. All plots are generated using Trellis graphics[5]. Table 6.1 summarizes the utility plot functions used to create the various OLS plots. See the help files for more information about the plot functions. Figures 6.2 and 6.3 illustrate plot choices 3 (response and fitted) and 8 (residual ACF). From the response and fitted plot, it is clear that the return on the S&P 500 index is a weak predictor of return on Microsoft. The residual ACF plot indicates that the residuals do not appear to be autocorrelated, which supports the results from the residual diagnostics reported using `summary`.

Individual plots can be created directly, bypassing the plot menu, using the `which.plot` option of `plot.OLS`. For example, the following command creates a normal qq-plot of the residuals:

```
> plot(ols.fit,which.plot=3)
```

Notice that number used for the qq-plot specified by `which.plot` is one less than the value specified in the menu. The qq-plot may also be created by calling the Trellis utility plot function `qqPlot` directly:

```
> qqPlot(resid(ols.fit), strip.text="ols.fit",
+ xlab="Quantile of Standard Normal",
+ ylab="Residuals",main="Normal QQ Plot")
```

Residual Diagnostics

The residual diagnostics reported by `summary` may be computed directly from an "`OLS`" object. The `normalTest` and `autocorTest` functions in

[5]Unfortunately, the Trellis plots cannot be easily combined into multipanel plots.

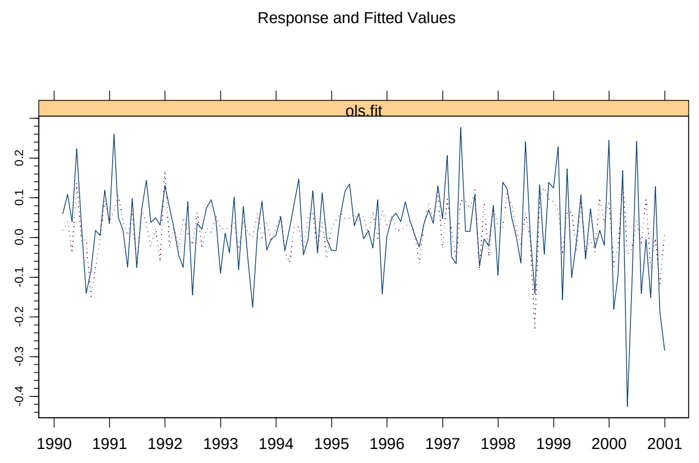


FIGURE 6.2. Response and fitted values from the OLS fit to the CAPM regression for Microsoft.

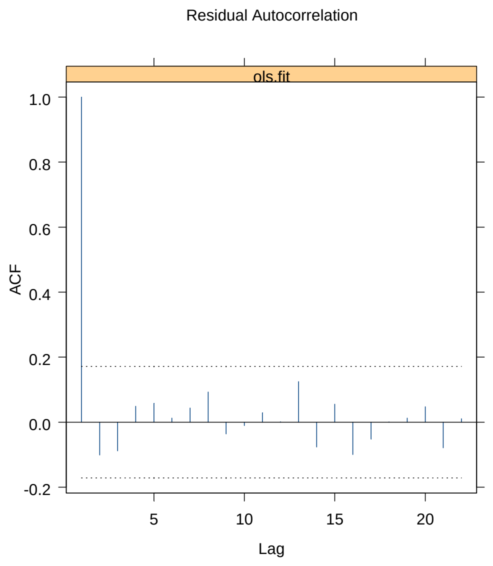


FIGURE 6.3. Residual ACF plot from the OLS fit to the CAPM regression for Microsoft.

S+FinMetrics may be used to compute test statistics for normality and autocorrelation from the residuals of an OLS fit. For example, to compute the Jarque-Bera normality test and the Ljung-Box test from the residuals of the CAPM regression use

```
> normalTest(ols.fit,method="jb")

Test for Normality: Jarque-Bera

Null Hypothesis: data is normally distributed

Test Statistics:

Test Stat 41.68
  p.value  0.00

Dist. under Null: chi-square with 2 degrees of freedom
   Total Observ.: 131

> autocorTest(ols.fit,method="lb")

Test for Autocorrelation: Ljung-Box

Null Hypothesis: no autocorrelation

Test Statistics:

Test Stat 11.9213
  p.value  0.9417

Dist. under Null: chi-square with 21 degrees of freedom
   Total Observ.: 131
```

The Durbin-Watson statistic may be recovered using the S-PLUS function durbinWatson as follows

```
> durbinWatson(residuals(ols.fit))
Durbin-Watson Statistic: 2.117063
Number of observations: 131
```

The Breusch-Godfrey LM test for residual autocorrelation may be computed using autocorTest by specifying method=\lm". For example, to compute the LM statistic for the null of no autocorrelation against the alternative of serial correlation up to lag two use

```
> autocorTest(ols.fit,method="lm",lag.n=2)
```

```
Test for Autocorrelation: Breusch-Godfrey LM

Null Hypothesis: no autocorrelation

Test Statistics:

Test Stat 2.9768
  p.value 0.2257

Dist. under Null: chi-square with 2 degrees of freedom
   Total Observ.: 131
```

Consistent with the Ljung-Box test, the LM test indicates no residual autocorrelation.

Subset Regression

The estimated β for Microsoft uses all of the data over the 11 year period from January 1990 to December 2000. It is generally thought that β does not stay constant over such a long time period. To estimate β using only the most recent five years of data the `start` option of `OLS` may be utilized

```
> OLS(MSFT~SP500, data=excessRet.ts,
+ start="Jan 1996", in.format="%m %Y")

Call:
OLS(formula = MSFT ~SP500, data = excessRet.ts, start =
"Jan 1996", in.format = "%m %Y")

Coefficients:
 (Intercept)  SP500
 0.0035      1.7828

Degrees of freedom: 60 total; 58 residual
Time period: from Jan 1996 to Dec 2000
Residual standard error: 0.1053
```

Notice that date string passed to `start` uses the same display format as the "`timeDate`" objects in the `positions` slot of `excessRet.ts`, and that this format is specified directly using `in.format="%m %Y"`. Estimation over general sub-periods follows by specifying both the start date and the end date of the sub-period in the call to `OLS`.

Regression estimates may be computed over general subsets by using the optional argument `subset` to specify which observations should be used in the fit. Subsets can be specified using a logical vector (which is replicated to have length equal to the number of observations), a numeric vector indicating the observation numbers to be included, or a character vector of the observation names that should be included. For example, to estimate the CAPM only for the observations for which the excess return on the S&P 500 is positive, use

```
> OLS(MSFT~SP500, data=excessRet.ts, subset=(SP500>=0))

Call:
OLS(formula = MSFT ~SP500, data = excessRet.ts, subset = (
SP500 >= 0))

Coefficients:
 (Intercept)  SP500
 0.0231      1.3685

Degrees of freedom: 80 total; 78 residual
Residual standard error: 0.08341
```

Regression with Dummy Variables

In the analysis of asset returns, it is often noticed that excess returns are higher in January than in any other month. To investigate this claim, a dummy variable is created which equals 1 if the month is January and 0 otherwise:

```
> is.Jan = (months(positions(excessRet.ts))=="Jan")
> Jan.dum = timeSeries(pos=positions(excessRet.ts),
+ data=as.integer(is.Jan))
```

Next, the January dummy variable is added to the time series of excess returns:

```
> newdat.ts = seriesMerge(excessRet.ts,Jan.dum)
> colIds(newdat.ts)[3] = "Jan.dum"
```

The CAPM regression allowing for a different intercept in January is

```
> summary(OLS(MSFT~SP500+Jan.dum, data=newdat.ts))

Call:
OLS(formula = MSFT ~SP500 + Jan.dum, data = newdat.ts)

Residuals:
     Min      1Q  Median      3Q     Max
 -0.3804 -0.0532  0.0065  0.0604  0.2032

Coefficients:
             Value Std. Error t value Pr(>|t|)
(Intercept) 0.0090 0.0082     1.0953  0.2755
```

```
      SP500 1.5085 0.1986     7.5972  0.0000
    Jan.dum 0.0513 0.0295     1.7371  0.0848

Regression Diagnostics:

         R-Squared 0.3271
Adjusted R-Squared 0.3166
Durbin-Watson Stat 2.0814

Residual Diagnostics:
               Stat P-Value
Jarque-Bera 43.4357  0.0000
  Ljung-Box 12.1376  0.9358

Residual standard error: 0.08958 on 128 degrees of freedom
Time period: from Feb 1990 to Dec 2000
F-statistic: 31.11 on 2 and 128 degrees of freedom, the
p-value is 9.725e-012
```

The coefficient on the January dummy is positive and significant at the 9% level indicating that excess returns are slightly higher in January than in other months. To allow for a different intercept and slope in the regression, use

```
> summary(OLS(MSFT~SP500*Jan.dum, data=newdat.ts))

Call:
OLS(formula = MSFT ~SP500 * Jan.dum, data = tmp1.ts)

Residuals:
     Min      1Q  Median      3Q     Max
 -0.3836 -0.0513  0.0047  0.0586  0.2043

Coefficients:
               Value Std. Error t value Pr(>|t|)
  (Intercept) 0.0095 0.0082     1.1607  0.2479
        SP500 1.4307 0.2017     7.0917  0.0000
      Jan.dum 0.0297 0.0317     0.9361  0.3510
SP500:Jan.dum 1.6424 0.9275     1.7707  0.0790

Regression Diagnostics:

         R-Squared 0.3433
Adjusted R-Squared 0.3278
Durbin-Watson Stat 2.0722
```

```
Residual Diagnostics:
               Stat P-Value
Jarque-Bera 51.4890  0.0000
  Ljung-Box 12.7332  0.9177

Residual standard error: 0.08884 on 127 degrees of freedom
Time period: from Feb 1990 to Dec 2000
F-statistic: 22.13 on 3 and 127 degrees of freedom, the
p-value is 1.355e-011
```

Notice that the formula uses the short-hand notation `A*B = A+B+A:B`. Interestingly, when both the slope and intercept are allowed to be different in January only the slope is significantly higher.

Predictions

Predictions or forecasts from an OLS fit may be computed using the generic `predict` function. For example, consider computing forecasts of the excess return on Microsoft conditional on specified values for the S&P 500 excess return based on the CAPM fit. The excess returns on the S&P 500 for the conditioinal forecasts are

```
> sp500.new = data.frame(c(-0.2,0,2))
> colIds(sp500.new) = "SP500"
```

These new data values must be in a data frame with the same name as the variable containing the S&P 500 data in `excessRet.ts`. The forecasts are computed using

```
> ols.pred = predict(ols.fit,n.predict=3,newdata=sp500.new)
> class(ols.pred)
[1] "forecast"
> ols.pred

Predicted Values:

[1] -0.2924  0.0128  3.0646
```

The result of `predict` is an object of class "`forecast`" for which there are `print`, `summary` and `plot` methods. The `print` method shows just the forecasts. The `summary` method shows the forecasts and forecast standard errors (ignoring parameter estimation error)

```
> summary(ols.pred)

Predicted Values with Standard Errors:

             prediction std.err
```

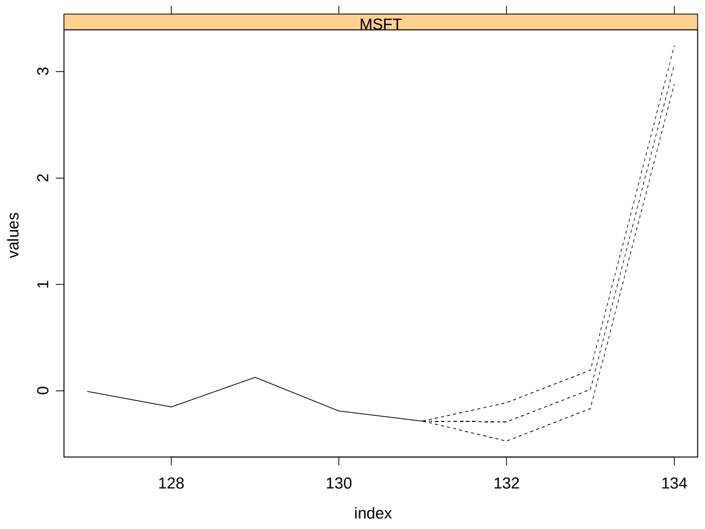


FIGURE 6.4. Conditional forecasts for the excess returns on Microsoft from the CAPM regression.

```
1-step-ahead -0.2924     0.0903
2-step-ahead  0.0128     0.0903
3-step-ahead  3.0646     0.0903
```

To view the forecasts and standard error band along with the historical data use

```
> plot(ols.pred, xold=excessRet.ts[,1], n.old=5, width=2)
```

The argument `xold` contains the historical data for the response variable, `n.old` determines how many historical observations to plot along with the forecasts and `width` specifies the multiplier for the forecast standard errors in the construction of the error bands. The created plot is illustrated in Figure 6.4.

6.4 Dynamic Regression

Often the time series regression model (6.1) contains lagged variables as regressors to capture dynamic effects. The general dynamic time series regression model contains lagged values of the response variable y_t and lagged values of the exogenous stationary regressors $x_{1t}, \ldots, x_{kt}$:

$$y_t = \alpha + \sum_{j=1}^{p} \phi_j y_{t-j} + \sum_{j=0}^{q_1} \beta_{1j} x_{1t-j} + \cdots + \sum_{j=0}^{q_k} \beta_{kj} x_{kt-j} + \varepsilon_t \tag{6.8}$$

where the error term ε_t is assumed to be $WN(0, \sigma^2)$. The model (6.8) is called an *autoregressive distributed lag* (ADL) model and generalizes an AR(p) by including exogenous stationary regressors.

The main issues associated with the analysis and interpretation of the ADL model (6.8) can be illustrated using the following simple ADL model with a single exogenous variable x_t:

$$y_t = \alpha + \phi y_{t-1} + \beta_0 x_t + \beta_1 x_{t-1} + \varepsilon_t \tag{6.9}$$

Since x_t is assumed to be stationary, and $\varepsilon_t \sim WN(0, \sigma^2)$, y_t behaves like an AR(1) process

$$y_t = \alpha + \phi y_{t-1} + w_t$$

where $w_t = \beta_0 x_t + \beta_1 x_{t-1} + \varepsilon_t$ is a composite error term. Therefore, the ADL model (6.9) is stationary provided $|\phi| < 1$. Given that y_t is stationary it has an infinite order moving average representation (impulse response function) in terms of the composite errors w_t

$$\begin{aligned} y_t &= \mu + \sum_{j=0}^{\infty} \psi_j w_{t-j} \\ &= \mu + w_t + \psi_1 w_{t-1} + \psi_2 w_{t-2} + \cdots \end{aligned}$$

where $\mu = 1/(1 - \phi)$ and $\psi_j = \phi^j$. Substituting $w_t = \beta_0 x_t + \beta_1 x_{t-1} + \varepsilon_t$ and $\psi_j = \phi^j$ into the above moving average representation gives

$$\begin{aligned} y_t &= \mu + (\beta_0 x_t + \beta_1 x_{t-1} + \varepsilon_t) + \phi(\beta_0 x_{t-1} + \beta_1 x_{t-2} + \varepsilon_{t-1}) \\ &\quad + \phi^2(\beta_0 x_{t-2} + \beta_1 x_{t-3} + \varepsilon_{t-2}) + \cdots \\ &= \mu + \beta_0 x_t + (\beta_1 + \phi\beta_0) x_{t-1} + \phi(\beta_1 + \phi\beta_0) x_{t-2} + \cdots \qquad (6.10) \\ &\quad + \phi^{j-1}(\beta_1 + \phi\beta_0) x_{t-j} + \cdots + \varepsilon_t + \phi\varepsilon_{t-1} + \phi^2 \varepsilon_{t-2} + \cdots \end{aligned}$$

Using (6.10), the interpretation of the coefficients in (6.9) becomes clearer. For example, the *immediate impact multiplier* is the impact of a change in x_t on y_t

$$\frac{\partial y_t}{\partial x_t} = \beta_0$$

The first lag multiplier is the impact of a change in x_{t-1} on y_t

$$\frac{\partial y_t}{\partial x_{t-1}} = \beta_1 + \phi\beta_0$$

which incorporates a feedback effect $\phi\beta_0$ due to the lagged response variable in (6.9). The second lag multiplier is

$$\frac{\partial y_t}{\partial x_{t-2}} = \phi\left(\beta_1 + \phi\beta_0\right)$$

and is smaller in absolute value than the first lag multiplier since $|\phi| < 1$. In general, the kth lag multiplier is

$$\frac{\partial y_t}{\partial x_{t-k}} = \phi^{k-1}\left(\beta_1 + \phi\beta_0\right)$$

Notice that as $k \to \infty$, $\frac{\partial y_t}{\partial x_{t-k}} \to 0$ so that eventually the effect of a change in x_t on y_t dies out. The *long-run effect* of a change in x_t on y_t is defined as the cumulative sum of all the lag impact multipliers

$$\begin{aligned}\text{long-run effect} &= \frac{\partial y_t}{\partial x_t} + \frac{\partial y_t}{\partial x_{t-1}} + \frac{\partial y_t}{\partial x_{t-2}} + \cdots \\ &= \sum_{k=0}^{\infty} \phi^k(\beta_0 + \beta_1) \\ &= \frac{\beta_0 + \beta_1}{1-\phi}\end{aligned}$$

The parameter ϕ on y_{t-1} determines the *speed of adjustment* between the immediate impact of x_t on y_t and the long-run impact. If $\phi = 0$, then the long-run impact is reached in one time period since $\frac{\partial y_t}{\partial x_{t-k}} = 0$ for $k > 1$. In contrast, if $\phi \approx 1$, then the long-run impact takes many periods. A parameter often reported is the *half-life* of the adjustment; that is, the lag at which one half of the adjustment to the long-run impact has been reached. In the simple ADL (6.9), it can be shown that the half-life is equal to $\ln(2)/\ln(\phi)$.

For the general ADL model (6.8), stationarity of y_t requires that all x_{it} be stationary and that the roots of the characteristic polynomial $\phi(z) = 1 - \phi_1 z - \cdots - \phi_p z^p = 0$ have modulus greater than one. The k immediate impact multipliers are the coefficients $\beta_{10}, \ldots, \beta_{k0}$ and the k long-run multipliers are

$$\frac{\sum_{j=0}^{q_2} \beta_{1j}}{1 - \phi_1 - \cdots - \phi_p}, \ldots, \frac{\sum_{j=0}^{q_k} \beta_{kj}}{1 - \phi_1 - \cdots - \phi_p}$$

The speed of adjustment to the long-run impacts is determined by the sum of the coefficients on the lagged responses $\phi_1 + \cdots + \phi_p$.

Example 36 *Estimating a simple dynamic CAPM regression for Microsoft*

Consider estimating a simple dynamic version of the CAPM regression

$$R_{it} - r_{ft} = \alpha + \phi(R_{it-1} - r_{ft-1}) + \beta_0(R_{Mt} - r_{ft}) + \beta_1(R_{Mt-1} - r_{ft-1}) + \varepsilon_{it}$$

using the monthly excess return data for Microsoft and the S&P 500 index. The "short-run beta" for Microsoft is β_0 and the "long-run beta" is $(\beta_0 + \beta_1)/(1-\phi)$. The dynamic CAPM estimated using OLS is

```
> adl.fit = OLS(MSFT~SP500+ar(1)+tslag(SP500),
+ data=excessRet.ts)
```

In the regression formula, the lagged dependent variable (`MSFT`) is specified using the `ar(1)` term, and the lagged explanatory variable (`SP500`) is created using the `S+FinMetrics` function `tslag`. The dynamic regression results are

```
> summary(adl.fit)

Call:
OLS(formula = MSFT ~SP500 + ar(1) + tslag(SP500), data =
excessRet.ts)

Residuals:
     Min      1Q  Median      3Q     Max
 -0.3659 -0.0514  0.0059  0.0577  0.1957

Coefficients:
               Value Std. Error t value Pr(>|t|)
 (Intercept)  0.0156  0.0083     1.8850  0.0617
       SP500  1.5021  0.2017     7.4474  0.0000
tslag(SP500) -0.0308  0.2453    -0.1257  0.9001
        lag1 -0.1107  0.0921    -1.2021  0.2316

Regression Diagnostics:

         R-Squared 0.3248
Adjusted R-Squared 0.3087
Durbin-Watson Stat 1.9132

Residual Diagnostics:
               Stat P-Value
Jarque-Bera 41.5581  0.0000
  Ljung-Box 10.5031  0.9716

Residual standard error: 0.0904 on 126 degrees of freedom
Time period: from Mar 1990 to Dec 2000
F-statistic: 20.2 on 3 and 126 degrees of freedom, the
p-value is 9.384e-011
```

The least squares estimates of dynamic CAPM parameters are $\hat{\alpha} = 0.016$, $\hat{\phi} = -0.111$, $\hat{\beta}_0 = 1.502$ and $\hat{\beta}_1 = -0.031$. The estimated "short-run beta" for Microsoft is 1.502 and the estimated "long-run beta" is[6]

```
> bhat = coef(adl.fit)
> lr.beta = (bhat[2]+bhat[3])/(1-bhat[4])
> lr.beta
 SP500
 1.325
```

Notice that the "long-run beta" is smaller than the "short-run beta". However, since the standard errors on the dynamic terms $\hat{\phi}$ and $\hat{\beta}_1$ are large relative to the estimated values, the data do not support the dynamic CAPM model.

6.4.1 Distributed Lags and Polynomial Distributed Lags

A special case of the general ADL model (6.8) is the distributed lag model

$$y_t = \alpha + \sum_{j=0}^{q} \beta_j x_{t-j} + \varepsilon_t \tag{6.11}$$

For simplicity, the model is shown with one exogenous variable x. The extension to multiple exogenous variables is straightforward. Given the results of the previous section, β_j is interpreted as the jth lag multiplier, and the long-run impact on y of a change in x is $\sum_{j=1}^{q} \beta_j$.

Determining the Lag Length

In many applications, the lag length q needs to be long to adequately capture the dynamics in the data. To determine the lag length, all models with $q \leq q_{\max}$ are fit and the preferred model minimizes some model selection criterion like the Akaike (AIC) or Schwarz (BIC). For the distributed lag model, the AIC and BIC have the form

$$\begin{aligned} \text{AIC}(q) &= \ln(\tilde{\sigma}^2(q)) + \frac{2}{T}q \\ \text{BIC}(q) &= \ln(\tilde{\sigma}^2(q)) + \frac{\ln T}{T}q \end{aligned}$$

where $\tilde{\sigma}^2(q)$ is the least squares estimate of σ^2 without a degrees of freedom correction. For objects of class "`OLS`", `S+FinMetrics` provides the extractor function `IC` to compute the AIC or BIC information criteria.

[6]Since the "long-run beta" is a nonlinear function of the least squares estimates, estimated standard errors for the "long-run beta" may be computed using the so-called "delta method". See Greene (2000) page 118.

If the exogenous variable x_t is highly persistent, then lagged values $x_t, x_{t-1}, \ldots, x_{t-q}$ may be highly correlated and problems associated with near multicollinearity may occur in (6.11)[7]. In this case, the S+FinMetrics function collinearTest may be used to diagnose the extent of near multicollinearity in the data. The function collinearTest computes either the *condition number* for $\mathbf{X}'\mathbf{X}$ or the *variance inflation statistics* associated with each variable.

Example 37 *Distributed lag model for U.S. real GDP growth*

The S+FinMetrics "timeSeries" policy.dat contains monthly data on U.S. real GDP and the Federal Funds rate. Consider estimating a distributed lag model with $q = 12$ for the growth rate in real GDP using the Federal Funds rate as an exogenous variable over the period January 1990 to March 1998:

```
> dl.fit = OLS(diff(log(GDP))~FFR+tslag(FFR,1:12),data=
+ policy.dat, start="Jan 1990",in.format="%m %Y",na.rm=T)
```

The AIC and BIC information criteria may be extracted using

```
> IC(dl.fit,type="AIC")
[1] -1271
> IC(dl.fit,type="BIC")
[1] -1235
```

The model may be re-fit with different values of q and the preferred model is the one which produces the smallest value of the chosen information criterion.

The condition number and variance inflation statistics from the least square fit are

```
> collinearTest(dl.fit, method="cn")
[1] 311.2
> collinearTest(dl.fit, method="vif")
   FFR tslag(FFR, 1:12)lag1 tslag(FFR, 1:12)lag2
 111.8                278.7                  293

 tslag(FFR, 1:12)lag3 tslag(FFR, 1:12)lag4
                304.9                331.8

 tslag(FFR, 1:12)lag5 tslag(FFR, 1:12)lag6
                344.9                370.5

 tslag(FFR, 1:12)lag7 tslag(FFR, 1:12)lag8
```

[7]See Greene (2000) pages 255-259 for a discussion of the problems associated with near multicollinearity.

```
                   369                    390

 tslag(FFR, 1:12)lag9 tslag(FFR, 1:12)lag10
                389.5                     410

 tslag(FFR, 1:12)lag11 tslag(FFR, 1:12)lag12
                 424.5                  162.6
```

The large condition number and variance inflation statistics indicate that high correlation among the regressors is a potential problem.

6.4.2 Polynomial Distributed Lag Models

The unrestricted distributed lag model (6.11) may produce unsatisfactory results due to high correlation among the lagged variables. If the sample size is small and the lag length q is large then these problems are exacerbated. In these cases, one may want to restrict the behavior of the lag coefficients β_j in (6.11). One popular way to do this is to use the polynomial distributed lag (PDL) model[8]. The PDL model specifies that β_j follows a polynomial

$$\beta_j = \alpha_0 + \alpha_1 j + \alpha_2 j^2 + \cdots + \alpha_d j^d \tag{6.12}$$

for $j = 1, \ldots, q > d$. Usually, the order of the polynomial, d, is small. Whereas the general distributed lag model (6.11) has q lag parameters the PDL model has only $d + 1$ lag parameters. To see this more explicitly, the distributed lag model with p lags under the restriction (6.12) may be re-written as the linear regression with d variables

$$y_t = \alpha + \alpha_0 z_{0t} + \alpha_1 z_{1t} + \cdots + \alpha_d z_{dt} + \varepsilon_t \tag{6.13}$$

where

$$z_{jt} = \sum_{i=1}^{q} i^j x_{t-i} \tag{6.14}$$

Example 38 *PDL model for U.S. real GDP growth*

To estimate a PDL model for U.S. GDP growth using the Federal Funds rate with $d = 2$ and $q = 12$ use

```
> pdl.fit = OLS(diff(log(GDP))~pdl(FFR,d=2,q=12),
+ data=policy.dat, start="Jan 1990",
+ in.format="%m %Y", na.rm=T)
> pdl.fit
```

[8]The PDL model is also known as the Almon lag model.

```
Call:
OLS(formula = diff(log(GDP)) ~pdl(FFR, d = 2, q = 12),
data = policy.dat, na.rm = T, start = "Jan 1990",
in.format = "%m %Y")

Coefficients:
 (Intercept) pdl(FFR, d = 2, q = 12)FFR.PDL0
  0.0006     -0.0070

 pdl(FFR, d = 2, q = 12)FFR.PDL1
  0.0031

 pdl(FFR, d = 2, q = 12)FFR.PDL2
 -0.0002

Degrees of freedom: 97 total; 93 residual
dropped 1 cases due to missing observations.
Time period: from Feb 1990 to Feb 1998
Residual standard error: 0.0003371
```

The `S+FinMetrics` function `pdl` used in the formula compute the regressors (6.14) for the PDL regression (6.13).

6.5 Heteroskedasticity and Autocorrelation Consistent Covariance Matrix Estimation

In the time series regression model, the efficiency of the least squares estimates and the validity of the usual formulas for the estimated coefficient standard errors and test statistics rely on validity of the underlying assumptions of the model outlined in the beginning of Section 6.2. In empirical applications using financial time series, it is often the case that the error terms ε_t have non constant variance (heteroskedasticity) as well as autocorrelation. As long as the regressors $\mathbf{x}_t$ are uncorrelated with the errors ε_t the least squares estimates of $\boldsymbol{\beta}$ will generally still be consistent and asymptotically normally distributed. However, they will not be efficient and the usual formula (6.3) for computing $\widehat{\text{avar}}(\hat{\boldsymbol{\beta}})$ will not be correct. As a result, any inference procedures based on (6.3) will also be incorrect. If the form of heteroskedasticity and autocorrelation is known, then efficient estimates may be computed using a *generalized least squares* procedure[9]. If the form of heteroskedasticity and autocorrelation is not known, it is possi-

[9]The `S-PLUS` function `gls` may be used to compute generalized least squares estimates using a variety of models for heteroskedasticity and autocorrelation.

ble to estimate $\widehat{\text{avar}}(\hat{\boldsymbol{\beta}})$ consistently so that valid standard errors and test statistics may be obtained. This section describes the construction of heteroskedasticity and autocorrelation consistent estimates of $\widehat{\text{avar}}(\hat{\boldsymbol{\beta}})$. First, the heteroskedasticity consistent estimate of $\widehat{\text{avar}}(\hat{\boldsymbol{\beta}})$ due to Eicker (1967) and White (1980) is discussed and then the heteroskedasticity and autocorrelation consistent estimate of $\widehat{\text{avar}}(\hat{\boldsymbol{\beta}})$ due to Newey and West (1987) is covered.

6.5.1 The Eicker-White Heteroskedasticity Consistent (HC) Covariance Matrix Estimate

A usual assumption of the time series regression model is that the errors ε_t are *conditionally homoskedastic*; i.e., $E[\varepsilon_t^2|\mathbf{X}] = \sigma^2 > 0$. In many situations it may be more appropriate to assume that the variance of ε_t is a function of $\mathbf{x}_t$ so that ε_t is *conditionally heteroskedastic*: $E[\varepsilon_t^2|\mathbf{x}_t] = \sigma^2 f(\mathbf{x}_t) > 0$. Formally, suppose the assumptions of the time series regression model hold but that $E[\varepsilon_t^2\mathbf{x}_t\mathbf{x}_t'] = \mathbf{S} \neq \boldsymbol{\sigma}^2\Sigma_{XX}$. This latter assumption allows the regression errors to be *conditionally heteroskedastic* and dependent on $\mathbf{x}_t$; i.e., $E[\varepsilon_t^2|\mathbf{x}_t] = \sigma^2 f(\mathbf{x}_t)$. In this case, it can be shown that the asymptotic variance matrix of the OLS estimate, $\hat{\boldsymbol{\beta}}$, is

$$\text{avar}(\hat{\boldsymbol{\beta}}) = T^{-1}\boldsymbol{\Sigma}_{XX}^{-1}\mathbf{S}\boldsymbol{\Sigma}_{XX}^{-1}. \tag{6.15}$$

The above *generalized* OLS asymptotic variance matrix will not be equal to the usual OLS asymptotic matrix $\sigma^2\boldsymbol{\Sigma}_{XX}^{-1}$, and the usual estimate $\widehat{\text{avar}}(\hat{\boldsymbol{\beta}}) = \hat{\sigma}^2(\mathbf{X}'\mathbf{X})^{-1}$ will not be correct. Hence, in the presence of heteroskedasticity the usual OLS t-statistics, standard errors, Wald statistics cannot be trusted.

If the values of $f(\mathbf{x}_t)$ are known, then the *generalized* or *weighted least squares* (GLS) estimator

$$\hat{\boldsymbol{\beta}}_{\text{GLS}} = (\mathbf{X}'\mathbf{V}(\mathbf{X})^{-1}\mathbf{X})^{-1}\mathbf{X}'\mathbf{V}(\mathbf{X})\mathbf{y},$$

where $\mathbf{V}(\mathbf{X})$ is a $(T \times T)$ diagonal matrix with $f(\mathbf{x}_t)$ along the diagonal, is efficient.

In most circumstances $f(\mathbf{x}_t)$ is not known so that the efficient GLS estimator cannot be computed. If the OLS estimator is to be used, then a consistent estimate for the generalized OLS covariance matrix is needed for proper inference. A heteroskedasticity consistent (HC) estimate of avar$(\hat{\boldsymbol{\beta}})$ due to Eicker (1967) and White (1980) is

$$\widehat{\text{avar}}_{\text{HC}}(\hat{\boldsymbol{\beta}}) = (\mathbf{X}'\mathbf{X})^{-1}\hat{\mathbf{S}}_{\text{HC}}(\mathbf{X}'\mathbf{X})^{-1} \tag{6.16}$$

where

$$\hat{\mathbf{S}}_{\text{HC}} = \frac{1}{T-k}\sum_{t=1}^{T}\hat{\varepsilon}_t^2\mathbf{x}_t\mathbf{x}_t' \tag{6.17}$$

and $\hat{\varepsilon}_t$ is the OLS residual at time t.

The square root of the diagonal elements of $\widehat{\text{avar}}_{\text{HC}}(\hat{\boldsymbol{\beta}})$ gives the Eicker-White *heteroskedasticity consistent standard errors* (HCSEs) for the least squares estimates of β_i. These are denoted $\widehat{\text{SE}}_{\text{HC}}(\hat{\beta}_i)$. Heteroskedasticity robust t-statistics and Wald statistics are computed in the usual way using (6.4) and (6.6) but with $\widehat{\text{avar}}_{\text{HC}}(\hat{\boldsymbol{\beta}})$ and $\widehat{\text{SE}}_{\text{HC}}(\hat{\beta}_i)$ replacing $\widehat{\text{avar}}(\hat{\boldsymbol{\beta}})$ and $\widehat{\text{SE}}(\hat{\beta}_i)$, respectively.

Example 39 *Heteroskedasticity robust inference for the CAPM*

Once a model has been fit using OLS, the HC estimate (6.16) may be extracted using `vcov` and $\widehat{\text{SE}}_{\text{HC}}(\hat{\beta}_i)$ may be computed using `summary` by specifying the optional argument `correction="white"` as follows

```
> ols.fit = OLS(MSFT~SP500, data=excessRet.ts)
> avar.HC = vcov(ols.fit, correction="white")
> summary(ols.fit, correction="white")

Call:
OLS(formula = MSFT ~SP500, data = excessRet.ts)

Residuals:
     Min      1Q  Median      3Q     Max
 -0.3835 -0.0566  0.0023  0.0604  0.1991

Coefficients:
             Value Std. Error t value Pr(>|t|)
(Intercept) 0.0128 0.0080     1.5937  0.1134
      SP500 1.5259 0.1920     7.9463  0.0000

Regression Diagnostics:

         R-Squared 0.3113
Adjusted R-Squared 0.3059
Durbin-Watson Stat 2.1171

Residual Diagnostics:
               Stat P-Value
Jarque-Bera 41.6842  0.0000
  Ljung-Box 11.9213  0.9417

Residual standard error: 0.09027 on 129 degrees of freedom
Time period: from Feb 1990 to Dec 2000
F-statistic: 58.3 on 1 and 129 degrees of freedom, the
p-value is 4.433e-012
```

Here, the HCSE values $\widehat{\text{SE}}_{\text{HC}}(\hat{\beta}_i)$ are almost identical to the usual OLS values $\widehat{\text{SE}}(\hat{\beta}_i)$ which suggests that the errors are not heteroskedastic.

6.5.2 Testing for Heteroskedasticity

If the error terms in the time series regression model are heteroskedastic, then the OLS estimates are consistent but not efficient and the usual formula (6.3) for computing $\widehat{\text{avar}}(\hat{\boldsymbol{\beta}})$ is incorrect. As shown in the previous section, $\widehat{\text{avar}}_{\text{HC}}(\hat{\boldsymbol{\beta}})$ given by (6.16) provides a consistent estimate of the generalized asymptotic variance (6.15). If the errors are not heteroskedastic, however, (6.15) is still consistent, but the usual formula (6.3) will generally give smaller standard errors and more powerful tests. Therefore, it is of interest to test for the presence of heteroskedasticity. If the errors are heteroskedastic and depend on some function of exogenous variables, then tests for heteroskedasticity may help determine which variables affect the error variance and how they might affect the variance. Finally, if the time series regression model is misspecified, e.g. some important variables have been omitted or the parameters are not constant over time, then often the errors will appear to be heteroskedastic. Hence, the presence of heteroskedasticity may also signal inadequacy of the estimated model. In this section, two common tests for heteroskedasticity are introduced. The first is Breusch and Pagan's (1979) LM test for heteroskedasticity caused by specified exogenous variables and the second is White's (1980) general test for unspecified heteroskedasticity.

Breusch-Pagan Test for Specific Heteroskedasticity

Suppose it is suspected that the variance of ε_t in (6.1) is functionally related to some known $(p \times 1)$ vector of exogenous variables $\mathbf{z}_t$, whose first element is unity, via the relation

$$E[\varepsilon_t|\mathbf{x}_t] = f(\mathbf{z}_t'\boldsymbol{\alpha})$$

where $f(\cdot)$ is an unknown positive function. Let $\hat{\varepsilon}_t$ denote the least squares residual from (6.1), and consider the *auxiliary regression*

$$\frac{\hat{\varepsilon}_t^2}{\bar{\sigma}^2} = \mathbf{z}_t'\boldsymbol{\alpha} + \text{ error} \tag{6.18}$$

where $\bar{\sigma}^2 = T^{-1}\sum_{t=1}^{T}\hat{\varepsilon}_t^2$. Since the first element of $\mathbf{z}_t$ is unity, the null hypothesis of homoskedasticity, $E[\varepsilon_t^2|\mathbf{x}_t] = \sigma^2$, implies that all of the elements of $\boldsymbol{\alpha}$ except the first are equal to zero. Under the homoskedasticity null, Breusch and Pagan (1979) showed that the test statistic

$$\frac{1}{2}\text{RSS}_{\text{aux}} \overset{A}{\sim} \chi^2(p-1)$$

where $\mathrm{RSS}_{\mathrm{aux}}$ is the residual sum of squares from the auxiliary regression (6.18).

The Breusch-Pagan LM test is based on the assumption that the error terms are normally distributed. Koenker and Basset (1982) suggested a modification of the Breusch-Pagan LM test that is robust to non-normal errors and generally has more power than the Breusch-Pagan test when the errors are non-normal.

White's Test for General Heteroskedasticity

Suppose ε_t is generally heteroskedastic such that $E[\varepsilon_t^2 \mathbf{x}_t \mathbf{x}_t'] = \mathbf{S}$, where $\mathbf{S}$ is a $(k \times k)$ matrix. Recall, if ε_t is homoskedastic then $\mathbf{S} = \sigma^2 \mathbf{\Sigma}_{XX}$. Now, under general heteroskedasticity $\hat{\mathbf{S}}_{\mathrm{HC}}$ in (6.17) is a consistent estimate of $\mathbf{S}$ and $\hat{\sigma}^2(\mathbf{X}'\mathbf{X})^{-1}$ is a consistent estimate of $\sigma^2 \mathbf{\Sigma}_{XX}$ and $\mathbf{S} \neq \sigma^2 \mathbf{\Sigma}_{XX}$. However, under the null hypothesis of homoskedasticity, the difference between $\hat{\mathbf{S}}_{\mathrm{HC}}$ and $\hat{\sigma}^2(\mathbf{X}'\mathbf{X})^{-1}$ should go to zero as the sample size gets larger. White (1980) utilized this result to develop a very simple test for general heteroskedasticity. To describe this test, let $\boldsymbol{\psi}_t$ denote the $(m \times 1)$ vector of unique and nonconstant elements of the $(k \times k)$ matrix $\mathbf{x}_t \mathbf{x}_t'$. Let $\hat{\varepsilon}_t$ denote the least squares residual from (6.1) and form the auxiliary regression

$$\hat{\varepsilon}_t = \boldsymbol{\psi}_t' \boldsymbol{\gamma} + \text{ error} \tag{6.19}$$

Under the null hypothesis of homoskedasticity, White (1980) showed that

$$T \cdot R^2_{\mathrm{aux}} \sim \chi^2(m)$$

where R^2_{aux} is the R^2 from the auxiliary regression (6.19).

Testing for Heteroskedasticity Using the `S+FinMetrics` Function `heteroTest`

Once a model has been fit using `OLS` (or `lm`), the Breusch-Pagan, Koenker-Basset and White tests for heteroskedasticity may be computed using the `S+FinMetrics` function `heteroTest`. For example, consider the simple CAPM regression for Microsoft

```
> ols.fit = OLS(MSFT~SP500, data=excessRet.ts)
```

To apply the Breusch-Pagan LM test, a set of variables $\mathbf{z}_t$ for which $\mathrm{var}(\varepsilon_t)$ is related must be identified. For illustrative purposes let $\mathbf{z}_t = (R_{Mt} - r_{ft}, (R_{Mt} - r_{ft})^2)'$. The LM may then be computed using

```
> z1 = as.matrix(seriesData(excessRet.ts[,"SP500"]))
> zmat = cbind(z1,z1^2)
> heteroTest(ols.fit, method="lm", regressors=zmat)

Test for Heteroskedasticity: Breusch-Pagan LM Test
```

```
 Null Hypothesis: data is homoskedastic
  Test Statistic: 0.152
Dist. under Null: chi-square with 2 degrees of freedom
         P-value: 0.9268

Coefficients:
 Intercept   SP500 SP500^2
   1.041    -1.320 -20.407

Degrees of freedom: 131 total; 128 residual
Residual standard error: 2.095
```

Notice that the regressors specified for the LM test must be in the form of a matrix. The high p-value of the test clearly signals that the null of homoskedasticity should not be rejected against the alternative that var(ε_t) depends on $\mathbf{z}_t$. To compute the Koenker-Basset robust LM test, set the optional argument `robust=T` in the call to `heteroTest`.

The application of White's test for heteroskedasticity is more straightforward since var(ε_t) is assumed to be functionally related to the variables used in the OLS fit:

```
> heteroTest(ols.fit, method="white")

Test for Heteroskedasticity: White General Test

 Null Hypothesis: data is homoskedastic
  Test Statistic: 0.152
Dist. under Null: chi-square with 2 degrees of freedom
         P-value: 0.9268

Coefficients:
 Intercept   SP500 SP500^2
  0.0084   -0.0106 -0.1638

Degrees of freedom: 131 total; 128 residual
Residual standard error: 0.01681
```

Notice that in this particular example the LM test and White's test are identical.

6.5.3 The Newey-West Heteroskedasticity and Autocorrelation Consistent (HAC) Covariance Matrix Estimate

In some applications of time series regression, ε_t in (6.1) may be both conditionally heteroskedastic and serially correlated. In this case, the error covariance matrix $E[\boldsymbol{\varepsilon}\boldsymbol{\varepsilon}'|\mathbf{X}]$ is non-diagonal. Under certain assumptions about the nature of the error heteroskedasticity and serial correlation a consistent estimate of the generalized OLS covariance matrix can be computed. The most popular heteroskedasticity and autocorrelation consistent (HAC) covariance matrix estimate, due to Newey and West (1987), has the form

$$\widehat{\text{avar}}_{\text{HAC}}(\hat{\boldsymbol{\beta}}) = (\mathbf{X}'\mathbf{X})^{-1}\hat{\mathbf{S}}_{\text{HAC}}(\mathbf{X}'\mathbf{X})^{-1} \tag{6.20}$$

where

$$\hat{\mathbf{S}}_{\text{HAC}} = \sum_{t=1}^{T}\hat{\varepsilon}_t^2\mathbf{x}_t\mathbf{x}_t' + \sum_{l=1}^{q} w_l \sum_{t=l+1}^{T}(\mathbf{x}_t\hat{\varepsilon}_t\hat{\varepsilon}_{t-l}\mathbf{x}_{t-l}' + \mathbf{x}_{t-l}\hat{\varepsilon}_{t-l}\hat{\varepsilon}_t\mathbf{x}_t') \tag{6.21}$$

is a nonparametric long-run variance estimate, and w_l is the *Bartlett weight function*

$$w_l = 1 - \frac{l}{q+1}$$

The Bartlett weight function, w_l, depends on a truncation parameter q that must grow with the sample size in order for the estimate to be consistent. Newey and West suggested choosing q to be the integer part of $4(T/100)^{2/9}$. In some cases, a *rectangular* weight function

$$w_l = \begin{cases} 1, & \text{for } l \leq q \\ 0, & \text{for } l > q \end{cases}$$

is used if it is known that autocovariances in (6.21) cut off at lag q. The square root of the diagonal elements of the HAC estimate (6.21) gives the *heteroskedasticity and autocorrelation consistent standard errors* (HAC-SEs) for the least squares estimates of β_i. These are denoted $\widehat{\text{SE}}_{\text{HAC}}(\hat{\beta}_i)$. Heteroskedasticity robust t-statistics and Wald statistics are computed in the usual way using (6.4) and (6.6) but with $\widehat{\text{avar}}_{\text{HAC}}(\hat{\boldsymbol{\beta}})$ and $\widehat{\text{SE}}_{\text{HAC}}(\hat{\beta}_i)$ replacing $\widehat{\text{avar}}(\hat{\boldsymbol{\beta}})$ and $\widehat{\text{SE}}(\hat{\beta}_i)$, respectively.

Example 40 *Long horizon regressions of stock returns on dividend-price ratio*

There has been much interest recently in whether long-term stock returns are predictable by valuation ratios like dividend-to-price and earnings-to-price. See Chapter 7 in Campbell, Lo, and MacKinlay (1997), and Chapter 20 in Cochrane (2001) for reviews of this literature. Predictability is investigated by regressing future multiperiod stock returns on current values of valuation ratios. To illustrate, let r_t denote the continuously compounded real annual total return on an asset in year t and and let $d_t - p_t$ denote the log dividend price ratio. The typical long-horizon regression has the form

$$r_{t+1} + \cdots + r_{t+K} = \alpha_K + \beta_K(d_t - p_t) + \varepsilon_{t+K},\ t = 1, \ldots, T \tag{6.22}$$

where $r_{t+1} + \cdots + r_{t+K}$ is the continuously compounded future K-year real total return. The dividend-price ratio predicts future returns if $\beta_K \neq 0$ at some horizon. Since the sampling frequency of the data is annual and the return horizon of the dependent variable is K years the dependent variable and error term in (6.22) will behave like an MA$(K-1)$ process. This serial correlation invalidates the usual formula for computing the estimated standard error of the least squares estimate of β_K. The HACSE, $\widehat{\mathrm{SE}}_{\mathrm{HAC}}(\hat{\beta}_K)$, however, will provide a consistent estimate.

The long-horizon regression (6.22) with $K = 10$ years is estimated using the annual stock price and dividend data on the S&P 500 composite index in the `S+FinMetrics` “`timeSeries`” object `shiller.annual`. The relevant data are constructed as

```
> colIds(shiller.annual)
[1] "price"         "dividend"      "earnings"
[4] "cpi"           "real.price"    "real.dividend"
[7] "real.earnings" "pe.10"
> # compute log of real data
> ln.p = log(shiller.annual[,"real.price"])
> colIds(ln.p) = "ln.p"
> ln.d = log(shiller.annual[,"real.dividend"])
> colIds(ln.d) = "ln.d"
> ln.dpratio = ln.d - ln.p
> colIds(ln.dpratio) = "ln.dpratio"
> # compute cc real total returns
> ln.r = diff(ln.p) + log(1+exp(ln.dpratio[-1,]))
> colIds(ln.r) = "ln.r"
> # create 10-year cc total returns
> ln.r.10 = aggregateSeries(ln.r,moving=10,FUN=sum)
> colIds(ln.r.10) = "ln.r.10"
> stockdiv.ts = seriesMerge(ln.p,ln.d,ln.dpratio,
+ ln.r,ln.r.10,pos="union")
```

The continuously compounded real total return is computed as

$$\begin{aligned} r_t &= \ln\left(\frac{P_t + D_t - P_{t-1}}{P_{t-1}}\right) = \\ &\quad \ln(P_t/P_{t-1}) + \ln(1 + \exp(\ln(D_t) - \ln(P_t))) \end{aligned}$$

where P_t is the real price and D_t is the real dividend. Notice how the `S-PLUS` function `aggregateSeries` is used to compute the 10 year continuously compounded real total returns.

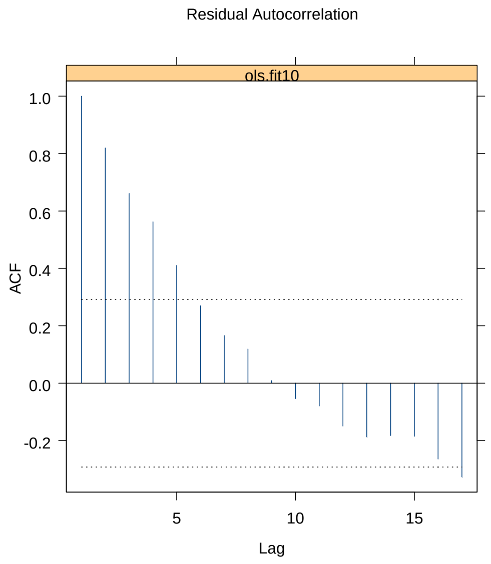


FIGURE 6.5. Residual ACF from regression of ten year real returns on dividend-price ratio.

The long-horizon regression (6.22) using10 year real total return returns over the postwar period 1947 - 1995 is computed using

```
> ols.fit10 = OLS(ln.r.10~tslag(ln.dpratio),data=stockdiv.ts,
+ start="1947", end="1995", in.format="%Y", na.rm=T)
```

Figure 6.5 shows the residual ACF. There is clearly serial correlation in the residuals. The HACSEs are computed using `summary` with the optional argument `correction="nw"`. By default, the Newey-West HAC covariance matrix is computed using a Bartlett kernel with automatic lag truncation $q = 4(T/100)^{2/9}$. In the present context, the serial correlation is known to be of the form of an MA(9) process. Therefore, it is more appropriate to compute the Newey-West HAC covariance using a rectangular weight function with $q = 9$ which is accomplished by specifying `bandwidth=9` and `window="rectangular"` in the call to `summary`:

```
> summary(ols.fit10, correction="nw", bandwidth=9,
+ window="rectangular")

Call:
OLS(formula = ln.r.10 ~tslag(ln.dpratio), data =
stockdiv.ts, na.rm = T, start = "1947", end =
"1995", in.format = "%Y")
```

```
Residuals:
     Min      1Q  Median      3Q     Max
 -0.6564 -0.2952  0.0030  0.1799  0.9997

Coefficients:
                   Value Std. Error t value Pr(>|t|)
      (Intercept) 5.7414 0.9633     5.9600  0.0000
tslag(ln.dpratio) 1.5604 0.3273     4.7668  0.0000

Regression Diagnostics:

         R-Squared 0.5012
Adjusted R-Squared 0.4896
Durbin-Watson Stat 0.2554

Residual Diagnostics:
                Stat  P-Value
Jarque-Bera   1.7104   0.4252
  Ljung-Box 105.9256   0.0000

Residual standard error: 0.4116 on 43 degrees of freedom
Time period: from 1947 to 1991
F-statistic: 43.21 on 1 and 43 degrees of freedom, the
p-value is 5.359e-008
```

Notice the low DW statistic and the large value of the Ljung-Box statistic indicating serial correlation in the residuals. The regression results with the corrected standard errors indicate that future 10 year real total returns are highly predictable and positively related to the current dividend-price ratio. The predictability coefficient is $\hat{\beta}_{10} = 1.560$ with $\widehat{\mathrm{SE}}_{\mathrm{HAC}}(\hat{\beta}_{10}) = 0.416$ and $R^2 = 0.501$.

6.6 Recursive Least Squares Estimation

The time series regression model (6.1) assumes that the parameters of the model, $\boldsymbol{\beta}$, are constant over the estimation sample. A simple and intuitive way to investigate parameter constancy is to compute recursive estimates of $\boldsymbol{\beta}$; that is, to estimate the model

$$y_t = \boldsymbol{\beta}_t'\mathbf{x}_t + \varepsilon_t \tag{6.23}$$

by least squares recursively for $t = k+1, \ldots, T$ giving $T-k$ *recursive least squares* (RLS) estimates $(\hat{\boldsymbol{\beta}}_{k+1}, \ldots, \hat{\boldsymbol{\beta}}_T)$. If $\boldsymbol{\beta}$ is really constant then the recursive estimates $\hat{\boldsymbol{\beta}}_t$ should quickly settle down near a common value. If some of the elements in $\boldsymbol{\beta}$ are not constant then the corresponding RLS estimates should show instability. Hence, a simple graphical technique for uncovering parameter instability is to plot the RLS estimates $\hat{\beta}_{it}$ $(i = 0, \ldots, k)$ and look for instability in the plots.

An alternative approach to investigate parameter instability is to compute estimates of the model's parameters over a fixed rolling window of a given length. Such rolling analysis is discussed in Chapter 9.

6.6.1 CUSUM and CUSUMSQ Tests for Parameter Stability

Brown, Durbin and Evans (1976) utilize the RLS estimates of (6.23) and propose two simple tests for parameter instability. These tests, know as the *CUSUM* and *CUSUMSQ* tests, are based on the standardized 1-step ahead recursive residuals

$$\hat{w}_t = \frac{\hat{\varepsilon}_t}{\hat{f}_t} = \frac{y_t - \hat{\boldsymbol{\beta}}'_{t-1}\mathbf{x}_t}{\hat{f}_t}$$

where $\hat{f}_t^2$ is an estimate of the recursive error variance

$$f_t^2 = \sigma^2\left[1 + \mathbf{x}_t'(\mathbf{X}_{t-1}'\mathbf{X}_{t-1})^{-1}\mathbf{x}_t\right]$$

and $\mathbf{X}_t$ is the $(t \times k)$ matrix of observations on $\mathbf{x}_s$ using data from $s = 1, \ldots, t$.

The CUSUM test is based on the cumulated sum of the standardized recursive residuals

$$\text{CUSUM}_t = \sum_{j=k+1}^{t} \frac{\hat{w}_j}{\hat{\sigma}_w}$$

where $\hat{\sigma}_w$ is the sample standard deviation of $\hat{w}_j$. Under the null hypothesis that $\boldsymbol{\beta}$ in (6.1) is constant, CUSUM_t has mean zero and variance that is proportional to $t-k-1$. Brown, Durbin and Evans (1976) show that approximate 95% confidence bands for CUSUM_t are given by the two lines which connect the points $(k, \pm 0.948\sqrt{T-k-1})$ and $(T, \pm 0.948 \cdot 3\sqrt{T-k-1})$. If CUSUM_t wanders outside of these bands, then the null of parameter stability may be rejected.

The CUSUMSQ test is based on the cumulative sum of the *squared* standardized recursive residuals and is given by

$$\text{CUSUMSQ}_t = \frac{\sum_{j=k+1}^{t} \hat{w}_j^2}{\sum_{j=k+1}^{T} \hat{w}_j^2}.$$

The distribution of CUSUMSQ_t under the null of parameter stability is given in Brown, Durbin and Evans (1976) where it is shown that 95% confidence bands for CUSUMSQ_t have the form $c \pm t/\sqrt{T-k-1}$. As with the

CUSUM_t statistic, if CUSUMSQ_t wanders outside the confidence bands, then the null of parameter stability may be rejected.

6.6.2 *Computing Recursive Least Squares Estimates Using the* `S+FinMetrics` *Function* `RLS`

Efficient RLS estimation of the time series regression model (6.1) may be performed using the `S+FinMetrics` function `RLS`. The calling syntax of `RLS` is exactly the same as that of `OLS` so that any model that may be estimated using `OLS` may also be estimated using `RLS`. For example, to compute the RLS estimates of the CAPM regression for Microsoft use

```
> rls.fit = RLS(MSFT~SP500, data=excessRet.ts)
> class(rls.fit)
[1] "RLS"
```

`RLS` produces an object of class "`RLS`" for which there are `coef`, `plot`, `print` and `residuals` methods. The `print` method give a basic description of the RLS fit

```
> rls.fit

Call:
RLS(formula = MSFT ~SP500, data = excessRet.ts)
Time period: from Feb 1990 to Dec 2000

Coefficients:
          (Intercept)  SP500
     mean 0.0284      1.2975
std. dev. 0.0121      0.1531

Recursive Residuals Summary:
    mean std. dev.
 -0.0157  0.0893
```

The recursive intercept and slope estimates do not seem to vary too much. The `plot` method allows one to see the recursive coefficients, CUSUM and CUSUMSQ residuals

```
> plot(rls.fit)

Make a plot selection (or 0 to exit):

1: plot: All
2: plot: Coefficient Estimates
3: plot: CUSUM of Residuals
4: plot: CUSUM of Squared Residuals
```

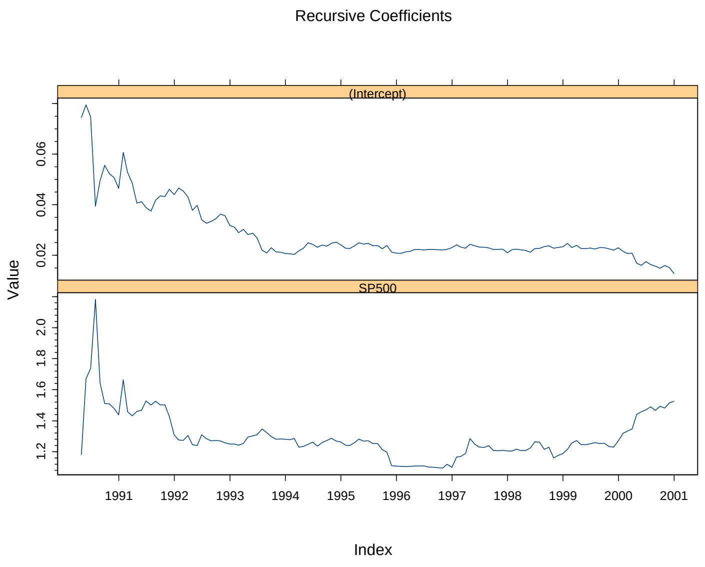


FIGURE 6.6. RLS coefficient estimates from the CAPM regression for Microsoft.

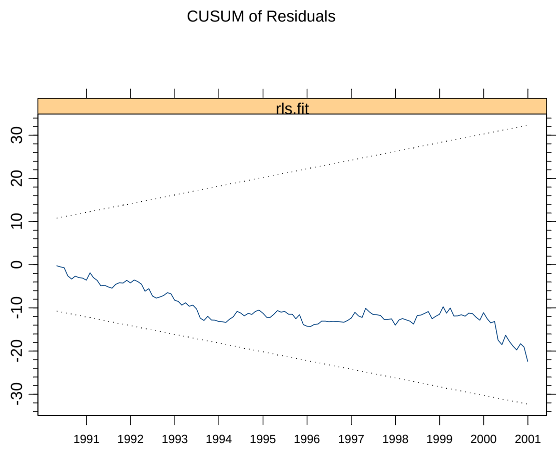


FIGURE 6.7. CUSUM of residuals from the CAPM regression for Microsoft.

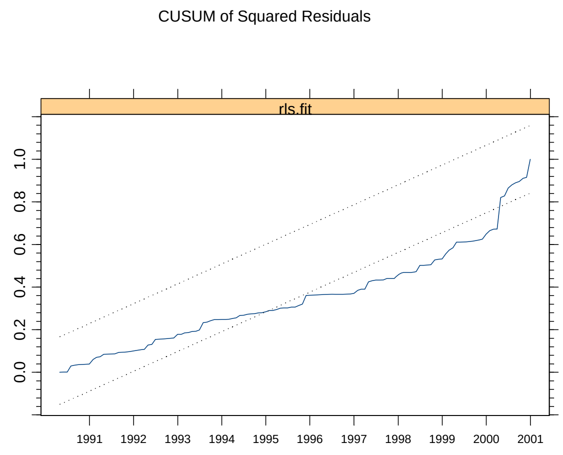


FIGURE 6.8. CUSUMSQ of residuals from CAPM regression for Microsoft.

```
Selection:
```

Figures 6.6, 6.7 and 6.8 show the plots from options 2, 3 and 4. The RLS estimates of α and β settle down in the middle of the sample but then the estimates of α decrease and the estimates of β increase. The CUSUM$_t$ statistics stay within the 95% confidence bands but the CUSUMSQ$_t$ statistics wander outside the bands. Hence, there is some evidence for instability in the CAPM coefficients.

6.7 References

Breusch, T. and A. Pagan (1979). "A Simple Test for Heteroscedasticity and Random Coefficient Variation," *Econometrica*, 47, 1287-1294.

Brown, R., J. Durbin and J. Evans (1976). "Techniques for Testing the Constancy of Regression Relationships over Time," *Journal of the Royal Statistical Society, Series B*, 37, 149-172.

Campbell, J. A. Lo, C. MacKinlay (1997). *The Econometrics of Financial Markets*. Princeton University Press, Princeton, NJ.

Cochrane, J. (2001). *Asset Pricing.* Princeton University Press, Princeton, NJ.

Eicker, F. (1967). "Limit Theorems for Regression with Unequal and Dependent Errors," in L. LeCam and J. Neyman (eds.), *Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability.* University of California Press, Berkeley.

Greene, W. (2000). *Econometric Analysis, Fifth Edition.* Prentice Hall, New Jersey.

Hamilton, J.D. (1994). *Time Series Analysis.* Princeton University Press, Princeton, NJ.

Hayashi, F. (2000). *Econometrics.* Princeton University Press, Princeton, NJ.

Jarque, C.M. and A.K. Bera (1981). "Efficients Tests for Normality, Homoskedasticity and Serial Dependence of Regression Residuals," *Economics Letters*, 6, 255-259.

Koenker, R. and G. Bassett (1982). "Robust Tests for Heteroscedasticity Based on Regression Quantiles," *Econometrica*, 50, 43-61.

Mills, T.C. (1999). *The Econometrics Modelling of Financial Time Series, Second Edition.* Cambridge University Press, Cambridge.

Newey, W.K. and K.D. West (1987). "A Simple Positive Semidefinite Heteroskedasticity and Autocorrelation Consistent Covariance Matrix," *Econometrica*, 55, 703-708.

White, H. (1980). "A Heteroskedasticity Consistent Covariance Matrix Estimator and a Direct Test for Heteroskedasticity," *Econometrica*, 48, 817-838.

7
Univariate GARCH Modeling

7.1 Introduction

Previous chapters have concentrated on modeling and predicting the conditional mean, or the first order moment, of a univariate time series, and are rarely concerned with the conditional variance, or the second order moment, of a time series. However, it is well known that in financial markets large changes tend to be followed by large changes, and small changes tend to be followed by small changes. In other words, the financial markets are sometimes more volatile, and sometimes less active.

The volatile behavior in financial markets is usually referred to as the "volatility". Volatility has become a very important concept in different areas in financial theory and practice, such as risk management, portfolio selection, derivative pricing, etc. In statistical terms, volatility is usually measured by variance, or standard deviation. This chapter introduces the class of univariate *generalized autoregressive conditional heteroskedasticity* (GARCH) models developed by Engle (1982), Bollerslev (1986), Nelson (1991), and others, which are capable of modeling time varying volatility and capturing many of the stylized facts of the volatility behavior usually observed in financial time series. It will show how to formulate, estimate, evaluate and predict from various types of GARCH models, such as EGARCH, TGARCH, PGARCH, etc.

The outline of the chapter follows. Section 7.2 shows how to test for ARCH effects in a time series, then section 22.16 introduces the basic GARCH model and its properties. GARCH model estimation and diagnos-

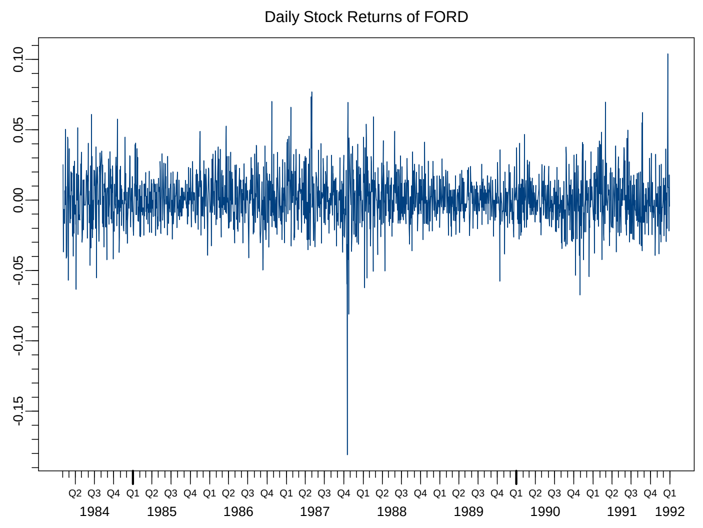


FIGURE 7.1. Daily Ford stock returns: `ford.s`.

tics using the `S+FinMetrics` family of GARCH functions are illustrated in section 7.4. Section 7.5 extends the basic GARCH model to accommodate some well-known stylized facts of financial time series. Prediction and simulation from various GARCH models are treated at the end of the chapter.

The statistical properties of GARCH models are nicely summarized in Hamilton (1994), Tsay (2001) and the review papers by Bera and Higgins (1986), Bolerslev, Engle and Nelson (1994) and Diebold and Lopez (1996). Bollerslev, Chou and Kroner (1992) give a comprehensive survey of GARCH modeling in finance. Alexander (2001) provides many examples of the use of GARCH models in finance, and Engle (2001) and Engle and Patton (2001) discuss the usefulness of volatility modeling.

7.2 The Basic ARCH Model

Figure 7.1 plots a daily time series of Ford stock returns as contained in the “`timeSeries`” object `ford.s` in `S+FinMetrics`:

```
> class(ford.s)
[1] "timeSeries"
> plot(ford.s, reference.grid=F)
```

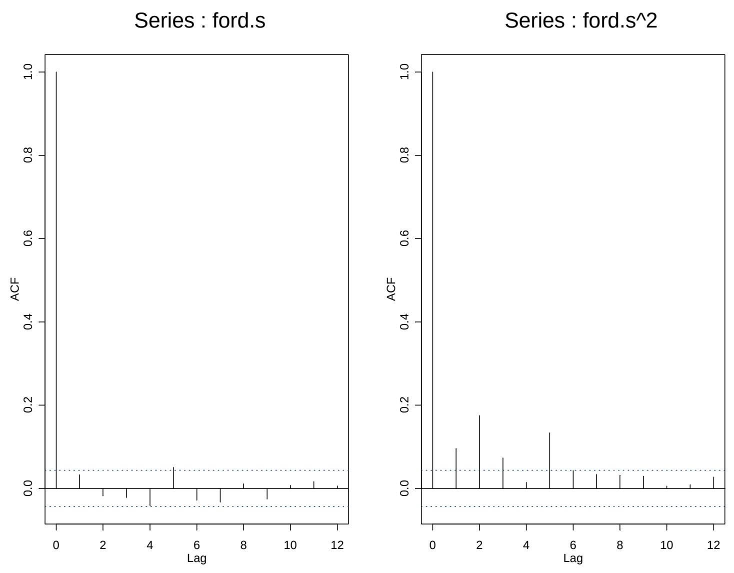


FIGURE 7.2. ACF of `ford.s` and `ford.s^2`.

Although there is little serial correlation in the time series `ford.s` itself, it seems that both large changes and small changes are clustered together, which is typical of many high frequency macroeconomic and financial time series. To confirm this conjecture, use the `S-PLUS` function `acf` to look at the autocorrelation plot of Ford returns and its squared returns:

```
> par(mfrow=c(1,2))
> tmp = acf(ford.s, lag=12)
> tmp = acf(ford.s^2, lag=12)
> par(mfrow=c(1,1))
```

The plots are shown in Figure 7.2. Obviously there is no autocorrelation in the return series itself, while the squared returns exhibit significant autocorrelation at least up to lag 5. Since the squared returns measure the second order moment of the original time series, this result indicates that the variance of `ford.s` conditional on its past history may change over time, or equivalently, the time series `ford.s` may exhibit *time varying conditional heteroskedasticity* or *volatility clustering*.

The serial correlation in squared returns, or conditional heteroskedasticity, can be modeled using a simple autoregressive (AR) process for squared residuals. For example, let y_t denote a stationary time series such as financial returns, then y_t can be expressed as its mean plus a white noise if there is no significant autocorrelation in y_t itself:

$$y_t = c + \epsilon_t \tag{7.1}$$

where c is the mean of y_t, and ϵ_t is iid with mean zero. To allow for volatility clustering or conditional heteroskedasticity, assume that $Var_{t-1}(\epsilon_t) = \sigma_t^2$ with $Var_{t-1}(\cdot)$ denoting the variance conditional on information at time $t-1$, and

$$\sigma_t^2 = a_0 + a_1\epsilon_{t-1}^2 + \cdots + a_p\epsilon_{t-p}^2. \tag{7.2}$$

since ϵ_t has a zero mean, $Var_{t-1}(\epsilon) = E_{t-1}(\epsilon_t^2) = \sigma_t^2$, the above equation can be rewritten as:

$$\epsilon_t^2 = a_0 + a_1\epsilon_{t-1}^2 + \cdots + a_p\epsilon_{t-p}^2 + u_t \tag{7.3}$$

where $u_t = \epsilon_t^2 - E_{t-1}(\epsilon_t^2)$ is a zero mean white noise process. The above equation represents an AR(p) process for ϵ_t^2, and the model in (7.1) and (7.2) is known as the **auto**r**egressive **c**onditional **h**eteroskedasticity (ARCH) model of Engle (1982), which is usually referred to as the ARCH(p) model.

An alternative formulation of the ARCH model is

$$\begin{aligned} y_t &= c + \epsilon_t \\ \epsilon_t &= z_t\sigma_t \\ \sigma_t^2 &= a_0 + a_1\epsilon_{t-1}^2 + \cdots + a_p\epsilon_{t-p}^2 \end{aligned}$$

where z_t is an iid random variable with a specified distribution. In the basic ARCH model z_t is assumed to be iid standard normal. The above representation is convenient for deriving properties of the model as well as for specifying the likelihood function for estimation.

Exercise 41 *Simulating an ARCH(p) model*

The S+FinMetrics function simulate.garch may be used to simulate observations from a variety of time-varying conditional heteroskedasticity models. For example, to simulate 250 observations from the ARCH(p) model (7.1)-(7.2) with $c = 0$, $p = 1$, $a_0 = 0.01$ and $a_1 = 0.8$ use

```
> sim.arch1 = simulate.garch(model=list(a.value=0.01,arch=0.8),
+                            n=250, rseed=196)
> names(sim.arch1)
[1] "et"      "sigma.t"
```

The component et contains the ARCH errors ϵ_t and the component sigma.t contains the conditional standard deviations σ_t. These components are illustrated in Figure 7.3 created using

```
> par(mfrow=c(2,1))
> tsplot(sim.arch1$et,main="Simulated ARCH(1) errors",
```

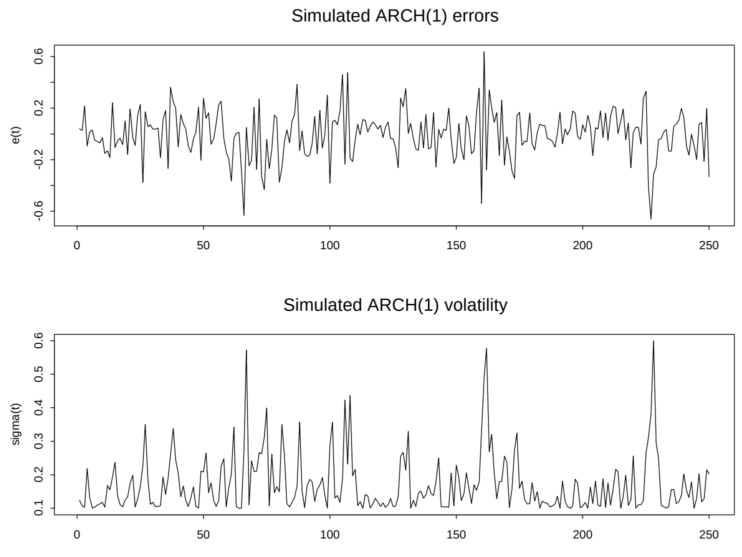


FIGURE 7.3. Simulated values of ε_t and σ_t from ARCH(1) process.

```
+ ylab="e(t)")
> tsplot(sim.arch1$sigma.t,
+ main="Simulated ARCH(1) volatility",ylab="sigma(t)")
```

Some summary statistics for the simulated data are

```
> summaryStats(sim.arch1$et)

Sample Quantiles:
     min      1Q median     3Q    max
 -0.6606 -0.1135 0.0112 0.1095 0.6357

Sample Moments:
      mean    std skewness kurtosis
 -0.003408 0.1846  -0.2515    4.041

Number of Observations:  250
```

Notice the somewhat high kurtosis value (relative to the kurtosis value of 3 for a normal distribution). Finally, Figure 7.4 shows the sample ACFs for ϵ_t and σ_t^2. Both series exhibit almost identical serial correlation properties.

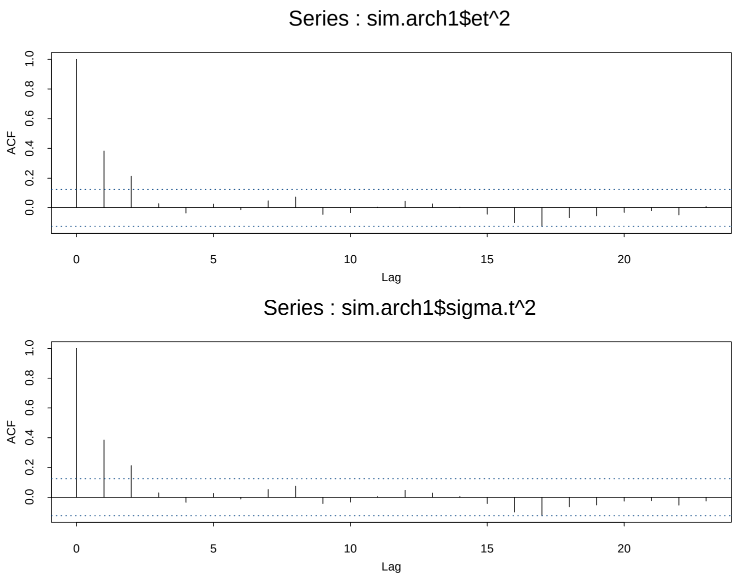


FIGURE 7.4. Sample ACFs for ε_t^2 and σ_t^2 from simulated ARCH(1) process.

7.2.1 Testing for ARCH Effects

Before estimating a full ARCH model for a financial time series, it is usually good practice to test for the presence of ARCH effects in the residuals. If there are no ARCH effects in the residuals, then the ARCH model is unnecessary and misspecified.

Since an ARCH model can be written as an AR model in terms of squared residuals as in (7.3), a simple Lagrange Multiplier (LM) test for ARCH effects can be constructed based on the auxiliary regression (7.3). Under the null hypothesis that there are no ARCH effects: $a_1 = a_2 = \cdots = a_p = 0$, the test statistic

$$\mathrm{LM} = T \cdot R^2 \overset{A}{\sim} \chi^2(p)$$

where T is the sample size and R^2 is computed from the regression (7.3) using estimated residuals.[1]

The S+FinMetrics function archTest can be used to carry out the above test for ARCH effects. For example, to test for the presence of ARCH effects in ford.s, use the following command:

```
> archTest(ford.s, lag.n=12)

Test for ARCH Effects: LM Test
```

[1]We refer to Engle (1982) for details.

```
Null Hypothesis: no ARCH effects

Test Statistics:
              FORD
Test Stat 112.6884
  p.value   0.0000

Dist. under Null: chi-square with 12 degrees of freedom
   Total Observ.: 2000
```

In this case, the p-value is essentially zero, which is smaller than the conventional 5% level, so reject the null hypothesis that there are no ARCH effects. Note that `archTest` function takes a time series and an optional argument `lag.n` specifying the order of the ARCH effects. Since `S-PLUS` allows lazy evaluation, `lag` instead of `lag.n` could have been supplied as the optional argument.

7.3 The GARCH Model and Its Properties

If the LM test for ARCH effects is significant for a time series, one could proceed to estimate an ARCH model and obtain estimates of the time varying volatility σ_t based on past history. However, in practice it is often found that a large number of lags p, and thus a large number of parameters, is required to obtain a good model fit. A more parsimonious model proposed by Bollerslev (1986) replaces the AR model in (7.2) with the following formulation:

$$\sigma_t^2 = a_0 + \sum_{i=1}^{p} a_i \epsilon_{t-i}^2 + \sum_{j=1}^{q} b_j \sigma_{t-j}^2 \tag{7.4}$$

where the coefficients a_i $(i = 0, \cdots, p)$ and b_j $(j = 1, \cdots, q)$ are all assumed to be positive to ensure that the conditional variance σ_t^2 is always positive.[2] The model in (7.4) together with (7.1) is known as the generalized ARCH or GARCH(p, q) model. When $q = 0$, the GARCH model reduces to the ARCH model.

Under the GARCH(p, q) model, the conditional variance of ϵ_t, σ_t^2, depends on the squared residuals in the previous p periods, and the conditional variance in the previous q periods. Usually a GARCH(1,1) model with only three parameters in the conditional variance equation is adequate to obtain a good model fit for financial time series.

[2]Positive coefficients are sufficient but not necessary conditions for the positivity of conditional variance. We refer to Nelson and Cao (1992) for the general conditions.

7.3.1 ARMA Representation of GARCH Model

Just as an ARCH model can be expressed as an AR model of squared residuals, a GARCH model can be expressed as an ARMA model of squared residuals. Consider the GARCH(1,1) model:

$$\sigma_t^2 = a_0 + a_1\epsilon_{t-1}^2 + b_1\sigma_{t-1}^2. \tag{7.5}$$

Since $E_{t-1}(\epsilon_t^2) = \sigma_t^2$, the above equation can be rewritten as:

$$\epsilon_t^2 = a_0 + (a_1 + b_1)\epsilon_{t-1}^2 + u_t - b_1 u_{t-1} \tag{7.6}$$

which is an ARMA(1,1) model with $u_t = \epsilon_t^2 - E_{t-1}(\epsilon_t^2)$ being the white noise disturbance term.

Given the ARMA representation of the GARCH model, many properties of the GARCH model follow easily from those of the corresponding ARMA process for ϵ_t^2. For example, for the GARCH(1,1) model to be stationary, requires that $a_1 + b_1 < 1$ as in (7.6). Assuming the stationarity of GARCH(1,1) model, the unconditional variance of ϵ_t can be shown to be $Var(\epsilon_t) = E(\epsilon_t^2) = a_0/(1 - a_1 - b_1)$, because from (7.6):

$$E(\epsilon_t^2) = a_0 + (a_1 + b_1)E(\epsilon_{t-1}^2)$$

and thus

$$E(\epsilon_t^2) = a_0 + (a_1 + b_1)E(\epsilon_t^2)$$

based on the assumption that ϵ_t^2 is stationary.

For the general GARCH(p, q) model (7.4), the squared residuals ϵ_t behave like an ARMA$(\max(p, q), q)$ process. Covariance stationarity requires $\sum_{i=1}^p a_i + \sum_{j=1}^q b_i < 1$ and the unconditional variance of ϵ_t is

$$Var(\epsilon_t) = \frac{a_0}{1 - \left(\sum_{i=1}^p a_i + \sum_{j=1}^q b_i\right)}. \tag{7.7}$$

7.3.2 GARCH Model and Stylized Facts

In practice, researchers have uncovered many so-called "stylized facts" about the volatility of financial time series; Bollerslev, Engle and Nelson (1994) gave a complete account of these facts. Using the ARMA representation of GARCH models shows that the GARCH model is capable of explaining many of those stylized facts. This section will focus on three important ones: volatility clustering, fat tails, and volatility mean reversion. Other stylized facts are illustrated and explained in later sections.

Volatility Clustering

Consider the GARCH$(1, 1)$ model in (7.5). Usually the GARCH coefficient b_1 is found to be around 0.9 for many weekly or daily financial time series.

Given this value of b_1, it is obvious that large values of σ_{t-1}^2 will be followed by large values of σ_t^2, and small values of σ_{t-1}^2 will be followed by small values of σ_t^2. The same reasoning can be obtained from the ARMA representation in (7.6), where large/small changes in ϵ_{t-1}^2 will be followed by large/small changes in ϵ_t^2.

Fat Tails

It is well known that the distribution of many high frequency financial time series usually have fatter tails than a normal distribution. That is, large changes are more often to occur than a normal distribution would imply. Bollerslev (1986) gave the condition for the existence of the fourth order moment of a GARCH$(1,1)$ process. Assuming the fourth order moment exists, Bollerslev (1986) showed that the kurtosis implied by a GARCH$(1,1)$ process is greater than 3, the kurtosis of a normal distribution. He and Teräsvirta (1999a, 1999b) extended these results to general GARCH(p,q) models. Thus a GARCH model can replicate the fat tails usually observed in financial time series.

Volatility Mean Reversion

Although financial markets may experience excessive volatility from time to time, it appears that volatility will eventually settle down to a long run level. The previous subsection showed that the long run variance of ϵ_t for the stationary GARCH$(1,1)$ model is $a_0/(1-a_1-b_1)$. In this case, the volatility is always pulled toward this long run level by rewriting the ARMA representation in (7.6) as follows:

$$(\epsilon_t^2 - \frac{a_0}{1-a_1-b_1}) = (a_1+b_1)(\epsilon_{t-1}^2 - \frac{a_0}{1-a_1-b_1}) + u_t - b_1 u_{t-1}.$$

If the above equation is iterated k times, one can show that

$$(\epsilon_{t+k}^2 - \frac{a_0}{1-a_1-b_1}) = (a_1+b_1)^k(\epsilon_t^2 - \frac{a_0}{1-a_1-b_1}) + \eta_{t+k}$$

where η_t is a moving average process. Since $a_1+b_1<1$ for a stationary GARCH$(1,1)$ model, $(a_1+b_1)^k \to 0$ as $k\to\infty$. Although at time t there may be a large deviation between ϵ_t^2 and the long run variance, $\epsilon_{t+k}^2 - a_0/(1-a_1-b_1)$ will approach zero "on average" as k gets large, i.e., the volatility "mean reverts" to its long run level $a_0/(1-a_1-b_1)$. In contrast, if $a_1+b_1>1$ and the GARCH model is non-stationary, the volatility will eventually explode to infinity as $k\to\infty$. Similar arguments can be easily constructed for a GARCH(p,q) model.

7.4 GARCH Modeling Using S+FinMetrics

7.4.1 GARCH Model Estimation

This section illustrates how to estimate a GARCH model using functions in S+FinMetrics. Recall, the general GARCH(p, q) model has the form

$$y_t = c + \epsilon_t \tag{7.8}$$

$$\sigma_t^2 = a_0 + \sum_{i=1}^{p} a_i \epsilon_{t-i}^2 + \sum_{j=1}^{q} b_j \sigma_{t-j}^2 \tag{7.9}$$

for $t = 1, \cdots, T$, where $\sigma_t^2 = Var_{t-1}(\epsilon_t)$. Assuming that ϵ_t follows normal or Gaussian distribution conditional on past history, the prediction error decomposition of the log-likelihood function of the GARCH model conditional on initial values is:

$$\log L = -\frac{T}{2}\log(2\pi) - \frac{1}{2}\sum_{t=1}^{T}\log\sigma_t^2 - \frac{1}{2}\sum_{t=1}^{T}\frac{\epsilon_t^2}{\sigma_t^2}. \tag{7.10}$$

The unknown model parameters c, a_i $(i = 0, \cdots, p)$ and b_j $(j = 1, \cdots, q)$ can be estimated using conditional maximum likelihood estimation (MLE). Details of the maximization are given in Hamilton (1994). Once the MLE estimates of the parameters are found, estimates of the time varying volatility σ_t $(t = 1, \ldots, T)$ are also obtained as a side product.

For a univariate time series, S+FinMetrics provides the garch function for GARCH model estimation. For example, to fit a simple GARCH(1,1) model as in (7.8) and (7.9) to the "timeSeries" object ford.s, use the command:

```
> ford.mod11 = garch(ford.s~1, ~garch(1,1))
Iteration   0  Step Size =  1.00000  Likelihood =  2.62618
Iteration   0  Step Size =  2.00000  Likelihood =  2.61237
Iteration   1  Step Size =  1.00000  Likelihood =  2.62720
Iteration   1  Step Size =  2.00000  Likelihood =  2.62769
Iteration   1  Step Size =  4.00000  Likelihood =  2.59047
Iteration   2  Step Size =  1.00000  Likelihood =  2.62785
Iteration   2  Step Size =  2.00000  Likelihood =  2.62795
Iteration   2  Step Size =  4.00000  Likelihood =  2.62793

Convergence R-Square = 4.630129e-05 is less than tolerance
= 0.0001
Convergence reached.
```

In the above example, the garch function takes two arguments: the first argument is an S-PLUS formula which specifies the conditional mean equation (7.8), while the second argument is also an S-PLUS formula which specifies the conditional variance equation (7.9). The specification of the conditional mean formula is the same as usual S-PLUS formulas.[3] For the conditional variance formula, nothing needs to be specified on the left hand side, and the garch(1,1) term on the right hand side denotes the GARCH$(1,1)$ model. By default, the progress of the estimation is printed on screen. Those messages can be suppressed by setting the optional argument trace=F in the call to the garch function.

The object returned by garch function is of class "garch". Typing the name of the object at the command line invokes its print method:

```
> class(ford.mod11)
[1] "garch"
> ford.mod11

Call:
garch(formula.mean = ford.s ~ 1, formula.var =  ~ garch(1, 1))

Mean Equation: ford.s ~ 1

Conditional Variance Equation:  ~ garch(1, 1)

Coefficients:

       C 7.708e-04
       A 6.534e-06
 ARCH(1) 7.454e-02
GARCH(1) 9.102e-01
```

The print method for a "garch" object shows the formulas for the conditional mean equation and conditional variance equation, together with the estimated model coefficients. Note that in the output C corresponds to the constant c in the conditional mean equation (7.8), A, ARCH(1) and GARCH(1) correspond to a_0, a_1 and b_1 in the conditional variance equation (7.9), respectively. Notice that the estimated GARCH(1) parameter is close to one and the ARCH(1) parameter is close to zero. The sum of these parameters is 0.985 which indicates a covariance stationary model with a high degree of persistence in the conditional variance. Use the S-PLUS function names to extract the component names for a "garch" object. For example:

```
> names(ford.mod11)
 [1] "residuals" "sigma.t"    "df.residual" "coef" "model"
 [6] "cond.dist" "likelihood" "opt.index"   "cov"  "prediction"
[11] "call"      "asymp.sd"   "series"
```

[3]Chapter 1 provides a review of the usage of S-PLUS formulas and modeling functions.

It should be clear what most of the components are and the on-line help file for the `garch` function provides details for these components. Of particular interest is the component `asymp.sd`, which gives an estimate of the unconditional standard deviation of the GARCH residuals provided the GARCH model is stationary. That is,

```
> ford.mod11$asymp.sd
[1] 0.02068
```

is an estimate of the square root of $a_0/(1 - \alpha_1 - b_1)$.

For most components that a user is interested in, `S+FinMetrics` provides methods for generic functions such as `coef`, `residuals`, and `vcov` for extracting those components. For example, the estimated coefficients can be extracted by calling the generic `coef` function:

```
> coef(ford.mod11)

       C 7.708418e-04
       A 6.534363e-06
 ARCH(1) 7.454134e-02
GARCH(1) 9.101842e-01
```

Similarly, call the generic `vcov` function to obtain the covariance matrix of the estimated coefficients:

```
> vcov(ford.mod11)
                   C            A      ARCH(1)     GARCH(1)
       C  1.41574e-07 -1.21204e-13 -3.56991e-07  2.21310e-07
       A -1.21204e-13  3.04607e-12  2.55328e-09 -1.24396e-08
 ARCH(1) -3.56991e-07  2.55328e-09  2.87505e-05 -3.43277e-05
GARCH(1)  2.21310e-07 -1.24396e-08 -3.43277e-05  7.67660e-05
```

By default, the `vcov` method for "`garch`" objects uses the covariance matrix based on the outer product of gradients. However, for maximum likelihood estimation, there are three different ways of computing the covariance matrix of model parameters which are asymptotically equivalent if the underlying error distribution is Gaussian: one based on the outer product of gradients, one based on the numerical Hessian matrix, and one based on the asymptotic formula for *quasi-maximum likelihood estimation* (QMLE). These different covariance matrices can be obtained by setting the optional argument `method` to `"op"`, `"hessian"`, or `"qmle"`, respectively. For example, to obtain the covariance matrix of `ford.mod11` parameters based on QMLE formula, use the following command:

```
> vcov(ford.mod11, method="qmle")
                   C            A      ARCH(1)     GARCH(1)
       C  1.26671e-07 -7.54398e-11  5.67606e-07 -7.71183e-08
       A -7.54398e-11  2.69841e-11  1.37576e-07 -2.00363e-07
```

```
 ARCH(1)  5.67606e-07  1.37576e-07  1.28016e-03 -1.46718e-03
GARCH(1) -7.71183e-08 -2.00363e-07 -1.46718e-03  1.84173e-03
```

This covariance matrix is sometimes referred to as the *robust* covariance matrix, because it is robust to possible misspecification of the error distribution, or the *sandwich* estimate, because of the form of the asymptotic formula (see Bollerslev and Wooldrige, 1992 or Davidson and MacKinnon, 1993).

The `residuals` method for a "`garch`" object takes an optional argument `standardize`, which can be used to obtain estimates of the standardized residuals ϵ_t/σ_t. For example:

```
> residuals(ford.mod11, standardize=T)
```

returns the standardized residuals of the fitted GARCH model `ford.mod11`. `S+FinMetrics` also provides another function `sigma.t` for extracting the fitted volatility series σ_t. Note that if the original data is a "`timeSeries`" object, the calendar information of the original data is also retained in the residual and volatility series.

7.4.2 GARCH Model Diagnostics

The previous subsection showed how to estimate a GARCH model using the `S+FinMetrics` function `garch` and how to extract various components of the fitted model. To assess the model fit, `S+FinMetrics` provides method functions for two generic functions: `summary` and `plot`, one for statistical summary information and the other for visual diagnostics of the model fit.

For example, to obtain a more detailed summary of `ford.mod11`, call the generic `summary` function:

```
> summary(ford.mod11)

Call:
garch(formula.mean = ford.s ~ 1, formula.var =  ~ garch(1, 1))

Mean Equation: ford.s ~ 1

Conditional Variance Equation:  ~ garch(1, 1)

Conditional Distribution:  gaussian

--------------------------------------------------------------

Estimated Coefficients:
--------------------------------------------------------------
             Value Std.Error t value  Pr(>|t|)
       C 7.708e-04 3.763e-04   2.049 2.031e-02
```

```
      A 6.534e-06 1.745e-06   3.744 9.313e-05
 ARCH(1) 7.454e-02 5.362e-03  13.902 0.000e+00
GARCH(1) 9.102e-01 8.762e-03 103.883 0.000e+00

--------------------------------------------------------------

AIC(4) = -10503.79
BIC(4) = -10481.39

Normality Test:
--------------------------------------------------------------
 Jarque-Bera P-value Shapiro-Wilk P-value
       364.2       0       0.9915  0.9777

Ljung-Box test for standardized residuals:
--------------------------------------------------------------
 Statistic P-value Chi^2-d.f.
     14.82  0.2516         12

Ljung-Box test for squared standardized residuals:
--------------------------------------------------------------
 Statistic P-value Chi^2-d.f.
     14.04  0.2984         12

Lagrange multiplier test:
--------------------------------------------------------------
 Lag 1  Lag 2  Lag 3   Lag 4   Lag 5   Lag 6 Lag 7   Lag 8
 2.135 -1.085 -2.149 -0.1347 -0.9144 -0.2228 0.708 -0.2314

   Lag 9 Lag 10  Lag 11  Lag 12      C
 -0.6905 -1.131 -0.3081 -0.1018 0.9825

  TR^2 P-value F-stat P-value
 14.77  0.2545  1.352  0.2989
```

By default, the `summary` method shows the standard errors and p-values for the t-statistics for testing that the true coefficients are zero, together with various tests on the standardized residuals $\hat{\epsilon}_t/\hat{\sigma}_t$ for assessing the model fit. The standard errors and p-values are computed using the default covariance estimate. To use robust or numerical Hessian based standard errors to compute the p-values, the `summary` method takes an optional argument `method` just like the `vcov` method does.

The various tests returned by the `summary` method can also be performed separately by using standard `S+FinMetrics` functions. For example, if the model is successful at modeling the serial correlation structure in the conditional mean and conditional variance, then there should be no autocorrelation left in the standardized residuals and squared standardized residuals. This can be checked by using the S+FinMetrics function autocorTest:

```
> autocorTest(residuals(ford.mod11, standardize=T), lag=12)

Test for Autocorrelation: Ljung-Box

Null Hypothesis: no autocorrelation

Test Statistics:

Test Stat 14.8161
  p.value  0.2516

Dist. under Null: chi-square with 12 degrees of freedom
   Total Observ.: 2000

> autocorTest(residuals(ford.mod11, standardize=T)^2, lag=12)

Test for Autocorrelation: Ljung-Box

Null Hypothesis: no autocorrelation

Test Statistics:

Test Stat 14.0361
  p.value  0.2984

Dist. under Null: chi-square with 12 degrees of freedom
   Total Observ.: 2000
```

In both cases, the tests are the same as those returned by the summary method, and the null hypothesis that there is no autocorrelation left cannot be rejected because the *p*-values in both cases are greater than the conventional 5% level. Note that lag was chosen to be 12 to match the results returned by the summary method.

Similarly, one can also apply the ARCH test on the standardized residuals to see if there are any ARCH effects left. For example, call archTest on the standardized residuals of ford.mod11 as follows:

```
> archTest(residuals(ford.mod11, standardize=T), lag=12)

Test for ARCH Effects: LM Test

Null Hypothesis: no ARCH effects
```

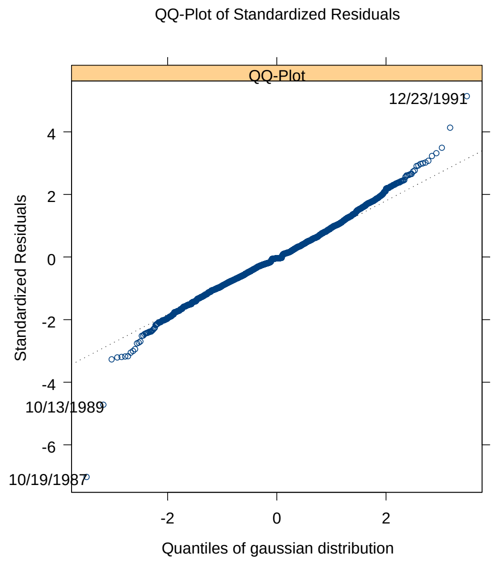


FIGURE 7.5. Normal qq-plot of standardized residuals: `ford.mod11`.

```
Test Statistics:

Test Stat 14.7664
  p.value  0.2545

Dist. under Null: chi-square with 12 degrees of freedom
   Total Observ.: 2000
```

Again, the results match the Lagrange Multiplier test as returned by the `summary` method.

The basic garch model assumes a normal distribution for the errors ϵ_t. If the model is correctly specified then the estimated standardized residuals ϵ_t/σ_t should behave like a standard normal random variable. To evaluate the normality assumption, the `summary` method reports both the Jarque-Bera test and the Shapiro-Wilks test for the standardized residuals, which again can be performed separately using the `S+FinMetrics` function `normalTest`. However, in the above example, the Jarque-Bera test and the Shapiro-Wilks test lead to opposite conclusions, with one p-value close to zero and the other close to one.

To get a more decisive conclusion regarding the normality assumption, resort to the qq-plot by calling the generic `plot` function on a "`garch`" object:

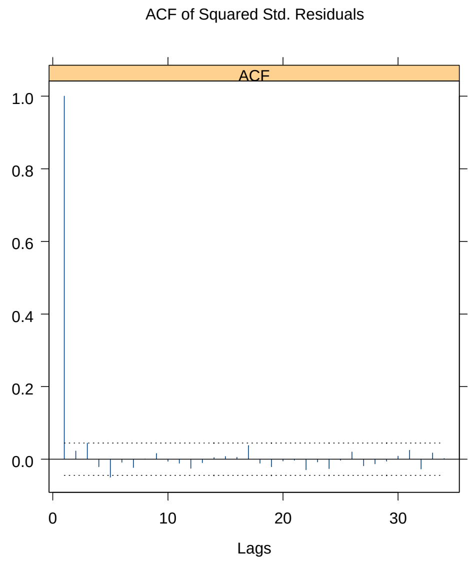


FIGURE 7.6. ACF of squared standardized residuals: `ford.mod11`.

```
> plot(ford.mod11)

Make a plot selection (or 0 to exit):

1: plot: All
2: plot: Series and Conditional SD
3: plot: Series with 2 Conditional SD Superimposed
4: plot: ACF of the Observations
5: plot: ACF of Squared Observations
6: plot: Cross Correlation between Squared Series and Series
7: plot: Residuals
8: plot: Conditional Standard Deviations
9: plot: Standardized Residuals
10: plot: ACF of Standardized Residuals
11: plot: ACF of Squared Standardized Residuals
12: plot: Cross Correlation between Squared Std.Res and Std.
13: plot: QQ-Plot of Standardized Residuals
Selection:
```

By selecting `13`, the qq-plot of standardized residuals can be obtained as shown in Figure 7.5 In this case, there is significant deviation in both tails from the normal qq-line, and thus it seems that the normality assumption for the residuals may not be appropriate. Section 7.5.6 will show how to

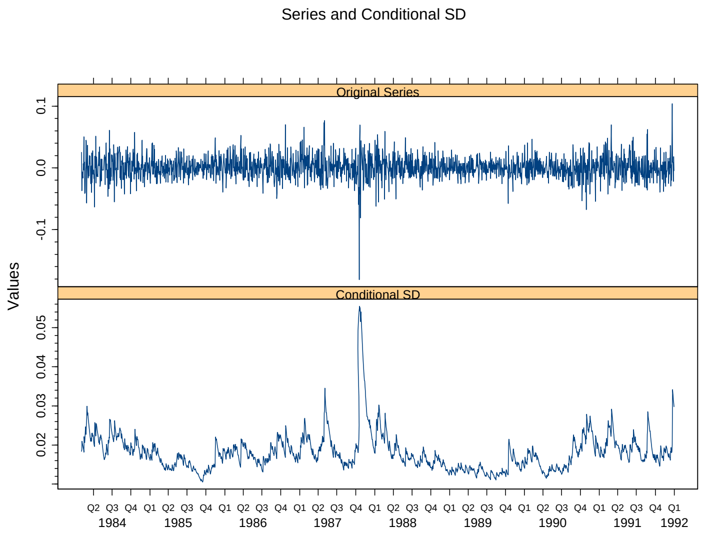


FIGURE 7.7. Daily Ford stock returns and conditional volatility.

use alternative distributions.Other plots can also be chosen to visualize the model fit. For example, choosing `11` generates the ACF plot of squared standardized residuals as shown in Figure 7.6, which shows that there is little autocorrelation left in the squared standardized residuals. Choosing `2` plots the original return series and the fitted volatility series as shown in Figure 7.7.

7.5 GARCH Model Extensions

In many cases, the basic GARCH model (7.4) provides a reasonably good model for analyzing financial time series and estimating conditional volatility. However, there are some aspects of the model which can be improved so that it can better capture the characteristics and dynamics of a particular time series. For example, the previous section showed that the normality assumption may not be appropriate for the time series `ford.s`. This section introduces several extensions to the basic GARCH model that make GARCH modeling more flexible and shows how to estimate those models using the `S+FinMetrics` `garch` function.

7.5.1 Asymmetric Leverage Effects and News Impact

In the basic GARCH model (7.9), since only squared residuals ϵ_{t-i}^2 enter the equation, the signs of the residuals or shocks have no effects on conditional volatility. However, a stylized fact of financial volatility is that bad news (negative shocks) tends to have a larger impact on volatility than good news (positive shocks). Black (1976) attributed this effect to the fact that bad news tends to drive down the stock price, thus increasing the leverage (i.e., the debt-equity ratio) of the stock and causing the stock to be more volatile. Based on this conjecture, the asymmetric news impact is usually referred to as the *leverage effect*. All the GARCH variants implemented in `S+FinMetrics` are capable of incorporating leverage effects. This subsection focuses on the EGARCH, TGARCH and PGARCH models.

EGARCH Model

Nelson (1991) proposed the following *exponential* GARCH (EGARCH) model to allow for leverage effects:

$$h_t = a_0 + \sum_{i=1}^{p} a_i \frac{|\epsilon_{t-i}| + \gamma_i \epsilon_{t-i}}{\sigma_{t-i}} + \sum_{j=1}^{q} b_j h_{t-j} \tag{7.11}$$

where $h_t = \log \sigma_t^2$ or $\sigma_t^2 = e^{h_t}$. Note that when ϵ_{t-i} is positive or there is "good news", the total effect of ϵ_{t-i} is $(1+\gamma_i)|\epsilon_{t-i}|$; in contrast, when ϵ_{t-i} is negative or there is "bad news", the total effect of ϵ_{t-i} is $(1-\gamma_i)|\epsilon_{t-i}|$. Bad news can have a larger impact on volatility, and the value of γ_i would be expected to be negative.

The `garch` function can be used to fit an EGARCH model by specifying `~egarch(p,q)` as the conditional variance formula. For example, to fit an EGARCH(1, 1) model with leverage effects using the daily Hewlett-Packard stock returns contained in the `S+FinMetrics` "`timeSeries`" object `hp.s`, use the following command:

```
> hp.egarch = garch(hp.s~1, ~egarch(1,1), leverage=T, trace=F)
> hp.egarch

Call:
garch(formula.mean = hp.s ~ 1, formula.var =  ~ egarch(1, 1),
      leverage = T, trace = F)

Mean Equation: hp.s ~ 1

Conditional Variance Equation:  ~ egarch(1, 1)

Coefficients:
       C  0.000313
       A -1.037907
 ARCH(1)  0.227878
GARCH(1)  0.886652
  LEV(1) -0.133998
```

Note that the optional argument `trace=F` is set to suppress the iteration messages, and set `leverage=T` to impose leverage effects. In the output, the estimated γ_1 coefficient for the leverage effect is denoted by `LEV(1)`and is negative in this case. The t-statistic for testing $\gamma_1 = 0$ is

```
> coef(hp.egarch)[5]/sqrt(vcov(hp.egarch)[5,5])
[1] -2.159
```

Another advantage of the EGARCH model over the basic GARCH model is that the conditional variance σ_t^2 is guaranteed to be positive regardless of the values of the coefficients in (7.11), because the logarithm of σ_t^2 instead of σ_t^2 itself is modeled.

TGARCH Model

Another GARCH variant that is capable of modeling leverage effects is the *threshold* GARCH (TGARCH) model,[4] which has the following form:

$$\sigma_t^2 = a_0 + \sum_{i=1}^{p} a_i \epsilon_{t-i}^2 + \sum_{i=1}^{p} \gamma_i S_{t-i} \epsilon_{t-i}^2 + \sum_{j=1}^{q} b_j \sigma_{t-j}^2 \tag{7.12}$$

where

$$S_{t-i} = \begin{cases} 1 & \text{if} \quad \epsilon_{t-i} < 0 \\ 0 & \text{if} \quad \epsilon_{t-i} \geq 0 \end{cases}$$

That is, depending on whether ϵ_{t-i} is above or below the threshold value of zero, ϵ_{t-i}^2 has different effects on the conditional variance σ_t^2: when ϵ_{t-i} is positive, the total effects are given by $a_i \epsilon_{t-i}^2$; when ϵ_{t-i} is negative, the total effects are given by $(a_i + \gamma_i)\epsilon_{t-i}^2$. So one would expect γ_i to be positive for bad news to have larger impacts. This model is also known as the GJR model because Glosten, Jagannathan and Runkle (1993) proposed essentially the same model.

Use the **garch** function to estimate a TGARCH model by specifying `~tgarch(p,q)` as the conditional variance formula. For example, to fit a TGARCH instead of an EGARCH model to `hp.s`, use the following command:

```
> hp.tgarch = garch(hp.s~1, ~tgarch(1,1), trace=F)
> hp.tgarch
```

[4]The original TGARCH model proposed by Zakoian (1994) models σ_t instead of σ_t^2.

```
Call:
garch(formula.mean = hp.s ~ 1, formula.var =  ~ tgarch(1, 1),
      trace = F)

Mean Equation: hp.s ~ 1

Conditional Variance Equation:  ~ tgarch(1, 1)

Coefficients:

       C 3.946e-04
       A 3.999e-05
 ARCH(1) 6.780e-02
GARCH(1) 8.369e-01
GAMMA(1) 3.306e-02
```

Note that when using the TGARCH model, the leverage effects are automatically imposed, so it is not necessary to set `leverage=T`. Also, the coefficient γ_1 for leverage effects is denoted by `GAMMA(1)` in the output to distinguish it from the EGARCH-type formulation of leverage effects. The estimated value of γ_1 is positive, indicating the presence of leverage effects, and is statistically different from zero at the 5% significance level since its t-statistic is greater than 2:

```
> coef(hp.tgarch)[5]/sqrt(vcov(hp.tgarch)[5,5])
[1] 2.5825
```

PGARCH Model

The basic GARCH model in `S+FinMetrics` is also extended to allow for leverage effects. This is made possible by treating the basic GARCH model as a special case of the *power* GARCH (PGARCH) model proposed by Ding, Granger and Engle (1993):

$$\sigma_t^d = a_0 + \sum_{i=1}^{p} a_i(|\epsilon_{t-i}| + \gamma_i \epsilon_{t-i})^d + \sum_{j=1}^{q} b_j \sigma_{t-j}^d \tag{7.13}$$

where d is a positive exponent, and γ_i denotes the coefficient of leverage effects. Note that when $d = 2$, (7.13) reduces to the basic GARCH model with leverage effects. Ding, Granger and Engle (1993) showed that the PGARCH model also includes many other GARCH variants as special cases.

To estimate a basic GARCH$(1,1)$ model with leverage effects, specify `~garch(1,1)` as the conditional variance formula and set the optional argument `leverage=T`:

```
> hp.garch = garch(hp.s~1, ~garch(1,1), leverage=T, trace=F)
> hp.garch

Call:
garch(formula.mean = hp.s ~ 1, formula.var =  ~ garch(1, 1),
      leverage = T, trace = F)

Mean Equation: hp.s ~ 1

Conditional Variance Equation:  ~ garch(1, 1)

Coefficients:

       C  4.536e-04
       A  3.823e-05
 ARCH(1)  7.671e-02
GARCH(1)  8.455e-01
  LEV(1) -1.084e-01
```

The estimated value of γ_1 is negative and its t-statistic

```
> coef(hp.garch)[5]/sqrt(vcov(hp.garch)[5,5])
[1] -2.2987
```

is less than 2 so one can reject the null of no leverage effects. If `~pgarch(p,q)` instead of `~garch(p,q)` is used as the conditional variance formula, the `garch` function will estimate the PGARCH model (7.13) where the exponent d is also estimated by MLE.

One can fix the exponent d in PGARCH model at a value other than two. For example, a popular choice is to set $d = 1$ so that the GARCH model is robust to outliers. To fit such a model, simply use `~pgarch(p,q,d)` as the conditional variance formula:

```
> hp.pgarch = garch(hp.s~1,~pgarch(1,1,1),leverage=T,trace=F)
> hp.pgarch

Call:
garch(formula.mean = hp.s~1, formula.var = ~pgarch(1, 1, 1),
      leverage = T, trace = F)

Mean Equation: hp.s ~ 1

Conditional Variance Equation:  ~ pgarch(1, 1, 1)

Coefficients:

       C  0.0003312
       A  0.0015569
```

GARCH(p,q)	$\bar{\sigma}^2 = a_0/[1 - \sum_{i=1}^{p} a_i(1+\gamma_i^2) - \sum_{j=1}^{q} b_j]$
TGARCH(p,q)	$\bar{\sigma}^2 = a_0/[1 - \sum_{i=1}^{p}(a_i + \gamma_i/2) - \sum_{j=1}^{q} b_j]$
PGARCH$(p,q,1)$	$\bar{\sigma}^2 = a_0^2/[1 - \sum_{i=1}^{p} a_i\sqrt{2/\pi} - \sum_{j=1}^{q} b_j]^2$
EGARCH(p,q)	$\bar{\sigma}^2 = \exp\{(a_0 + \sum_{i=1}^{p} a_i\sqrt{2/\pi})/(1 - \sum_{j=1}^{q} b_j)\}$

TABLE 7.1. Unconditional variance of GARCH processes

```
 ARCH(1)  0.0892505
GARCH(1)  0.8612378
  LEV(1) -0.1499219

> coef(hp.pgarch)[5]/sqrt(vcov(hp.pgarch)[5,5])
[1] -2.2121
```

News Impact Curve

The above subsections have shown that GARCH, EGARCH, TGARCH and PGARCH models are all capable of modeling leverage effects. The choice of a particular model can be made by using a model selection criterion such as the Bayesian information criterion (BIC). Alternatively, Engle and Ng (1993) proposed that the *news impact curve* could also be used to compare these different models. Here is the definition of the news impact curve following Engle and Ng (1993):

The news impact curve is the functional relationship between conditional variance at time t and the shock term (error term) at time $t-1$, holding constant the information dated $t-2$ and earlier, and with all lagged conditional variance evaluated at the level of the unconditional variance.

To facilitate the comparison of news impact curves of different GARCH models, Table 7.1 summarizes the unconditional variance, $\bar{\sigma}^2$, of various GARCH models and Table 7.2 summarizes the news impact curves for models with $p = 1$ and $q = 1$.

For example, to compare the news impact curves implied by `hp.tgarch`, `hp.pgarch` and `hp.garch`, plot the corresponding news impact curves using the following commands:

```
> a0 = hp.tgarch$coef[2]
> a1 = hp.tgarch$coef[3]
> b1 = hp.tgarch$coef[4]
> g1 = hp.tgarch$coef[5]
> A = a0 + b1 * hp.tgarch$asymp.sd^2
```

GARCH(1,1)	$\sigma_t^2 = A + a_1(\|\epsilon_{t-1}\| + \gamma_1\epsilon_{t-1})^2$ $A = a_0 + b_1\bar{\sigma}^2$
TGARCH(1,1)	$\sigma_t^2 = A + (a_1 + \gamma_1 S_{t-1})\epsilon_{t-1}^2$ $A = a_0 + b_1\bar{\sigma}^2$
PGARCH(1,1,1)	$\sigma_t^2 = A + 2\sqrt{A}a_1(\|\epsilon_{t-1}\| + \gamma_1\epsilon_{t-1})$ $+a_1^2(\|\epsilon_{t-1}\| + \gamma_1\epsilon_{t-1})^2$, $A = (a_0 + b_1\bar{\sigma})^2$
EGARCH(1,1)	$\sigma_t^2 = A\exp\{a_1(\|\epsilon_{t-1}\| + \gamma_1\epsilon_{t-1})/\bar{\sigma}\}$ $A = \bar{\sigma}^{2b_1}\exp\{a_0\}$

TABLE 7.2. News impact curves of GARCH processes

```
> epsilon = seq(-0.21, 0.14, length=100)
> sigma2.t.TGARCH = A + (a1+g1*(epsilon < 0))*(epsilon^2)

> a0 = hp.pgarch$coef[2]
> a1 = hp.pgarch$coef[3]
> b1 = hp.pgarch$coef[4]
> g1 = hp.pgarch$coef[5]
> A = (a0 + b1 * hp.pgarch$asymp.sd)^2

> error = abs(epsilon) + g1*epsilon
> sigma2.t.PGARCH = A + 2*sqrt(A)*a1*error + (a1*error)^2

> a0 = hp.garch$coef[2]
> a1 = hp.garch$coef[3]
> b1 = hp.garch$coef[4]
> g1 = hp.garch$coef[5]
> A = a0 + b1 * hp.garch$asymp.sd^2

> error = abs(epsilon) + g1*epsilon
> sigma2.t.GARCH = A + a1*(error^2)

> matplot(cbind(epsilon, epsilon, epsilon), cbind(
  sigma2.t.TGARCH, sigma2.t.PGARCH, sigma2.t.GARCH), type="l")

> key(-0.05, 0.0045, lines=list(type="l", lty=1:3), text=
  list(c("TGARCH", "PGARCH", "GARCH")), border=1, adj=1)
```

In this plot, the range of ϵ_t is determined by the residuals from the fitted models. The resulting plot is shown in Figure 7.8. This plot shows that the news impact curves are all asymmetric because leverage effects are allowed

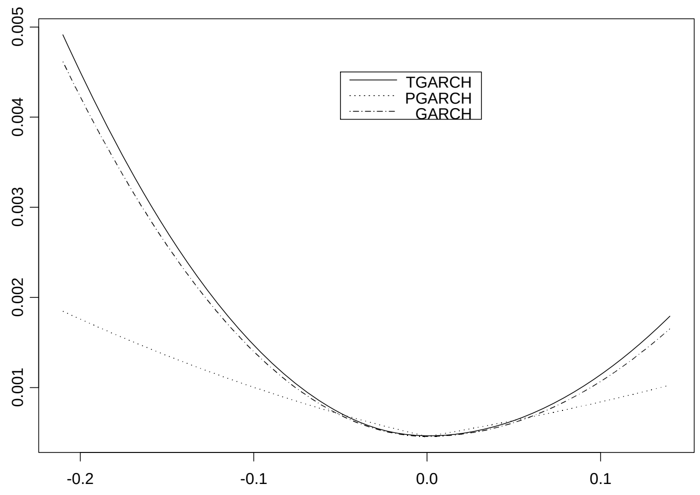


FIGURE 7.8. Camparison of news impact curves.

in all three models, and negative shocks or bad news have larger impacts on volatility. The TGARCH model suggests larger impacts of shocks on volatility than the GARCH model with leverage effects, regardless of the size of the shock. Moreover, since the PGARCH model with $d = 1$ is more robust to extreme shocks, impacts of small shocks implied by the PGARCH model are larger compared to those from GARCH and TGARCH models, whereas impacts of large shocks implied by the PGARCH model are smaller compared to those from GARCH and TGARCH models.

7.5.2 Two Components Model

Section 7.3.2 illustrated that the GARCH model can be used to model mean reversion in conditional volatility; that is, the conditional volatility will always "mean revert" to its long run level if the GARCH model is stationary. Recall the mean reverting form of the basic GARCH(1, 1) model:

$$(\epsilon_t^2 - \bar{\sigma}^2) = (a_1 + b_1)(\epsilon_{t-1}^2 - \bar{\sigma}^2) + u_t - b_1 u_{t-1}$$

where $\bar{\sigma}^2 = a_0/(1 - a_1 - b_1)$ is the unconditional long run level of volatility. As previous examples have shown, the mean reverting rate $a_1 + b_1$ implied by most fitted models is usually very close to 1. For example, the `ford.mod11` object fitted in Section 7.4,has the following mean reverting rate:

```
> ford.mod11$coef[3] + ford.mod11$coef[4]
[1] 0.9847255
```

which is almost one. The half life of a volatility shock implied by this mean reverting rate is:[5]

```
> log(0.5)/log(ford.mod11$coef[3] + ford.mod11$coef[4])
[1] 45.03192
```

which amounts to more than two calendar months. So the fitted GARCH model implies that the conditional volatility is very persistent.

Engle and Lee (1999) suggested that the high persistence in volatility may be due to a time-varying long run volatility level. In particular, they suggested decomposing conditional variance into two components:

$$\sigma_t^2 = q_t + s_t \tag{7.14}$$

where q_t is a highly persistent long run component, and s_t is a transitory short run component.

`S+FinMetrics` supports a wide range of two component models by extending all the previously discussed GARCH variants to incorporate the two components form (7.14). The general form of the two components model is based on a modified version of Ding and Granger (1996):

$$\sigma_t^d = q_t^d + s_t^d \tag{7.15}$$
$$q_t^d = \alpha_1|\epsilon_{t-1}|^d + \beta_1 q_{t-1}^d \tag{7.16}$$
$$s_t^d = a_0 + \alpha_2|\epsilon_{t-1}|^d + \beta_2 s_{t-1}^d. \tag{7.17}$$

That is, the long run component q_t follows a highly persistent PGARCH$(1,1)$ model, and the transitory component s_t follows another PGARCH$(1,1)$ model. By expressing the above two PGARCH models using lag operator notation

$$\begin{aligned} q_t^d &= (1-\beta_1 L)^{-1}\alpha_1|\epsilon_{t-1}|^d \\ s_t^d &= a_0 + (1-\beta_2 L)^{-1}\alpha_2|\epsilon_{t-1}|^d \end{aligned}$$

and then substituting them into (7.15), it can be shown that the reduced form of the two components model is:

$$\begin{aligned} \sigma_t^d = a_0 + (\alpha_1+\alpha_2)|\epsilon_{t-1}|^d - (\alpha_1\beta_2 + \alpha_2\beta_1)|\epsilon_{t-2}|^d \\ + (\beta_1+\beta_2)\sigma_{t-1}^d - \beta_1\beta_2\sigma_{t-2}^d \end{aligned}$$

which is in the form of a constrained PGARCH$(2,2)$ model. However, the two components model is not fully equivalent to the PGARCH$(2,2)$

[5]See Chapter 2 for the definition of half life.

model because not all PGARCH$(2,2)$ models have the component structure. In fact, since the two components model is a constrained version of the PGARCH$(2,2)$ model, the estimation of a two components model is often numerically more stable than the estimation of an unconstrained PGARCH$(2,2)$ model.

Although the PGARCH$(1,1)$ model is used here as the component for the two components model, `S+FinMetrics` actually allows any valid GARCH variant as the component, and leverage effects are also allowed correspondingly. For example, to fit a two components model using a GARCH component, EGARCH component, or PGARCH component, simply use the conditional variance formulas `~garch.2comp`, `~egarch.2comp`, `~pgarch.2comp(d)`, respectively. Since a two components model reduces to a GARCH$(2,2)$ model of the corresponding type, the orders of the ARCH and GARCH terms need not be given in the formula specification. The only exception is the PGARCH two components model, which can explicitly specify the exponent d for the underlying PGARCH model. For example, to estimate a two components PGARCH model with $d = 2$ using the daily Ford stock returns `ford.s`, use the following command:

```
> ford.2comp = garch(ford.s~1, ~pgarch.2comp(2))
> summary(ford.2comp)

Call:
garch(formula.mean = ford.s ~ 1, formula.var =
~ pgarch.2comp(2))

Mean Equation: ford.s ~ 1

Conditional Variance Equation:  ~ pgarch.2comp(2)

Conditional Distribution:  gaussian

--------------------------------------------------------------

Estimated Coefficients:
--------------------------------------------------------------
             Value Std.Error t value  Pr(>|t|)
       C 6.870e-04 3.795e-04   1.810 3.519e-02
       A 1.398e-06 5.877e-07   2.379 8.716e-03
ALPHA(1) 2.055e-02 6.228e-03   3.300 4.925e-04
ALPHA(2) 1.422e-01 2.532e-02   5.617 1.110e-08
 BETA(1) 9.664e-01 8.637e-03 111.883 0.000e+00
 BETA(2) 3.464e-01 1.091e-01   3.175 7.617e-04
```

The coefficients for the two components, α_1, β_1, α_2 and β_2, are identified by `ALPHA(1)`, `BETA(1)`, `ALPHA(2)` and `BETA(2)` in the output. As expected, the long run component associated with α_1 and β_1 is very persistent, whereas the second component associated with α_2 and β_2 is not persistent at all. Also, all the coefficients are highly significant.

In the above example, fixing $d = 2$ for the two components PGARCH model can be easily verified that the model is equivalent to a two components GARCH model. If the exponent d is not specified in the formula, it will be estimated by MLE. In addition, setting `leverage=T` when fitting a two components model, the coefficients for leverage effects will also be estimated, and the form of leverage effects is same as in (7.11) and (7.13). However, for the two components PGARCH model, `S+FinMetrics` also allows leverage effects to take the form as in the TGARCH model (7.12). The resulting model can be estimated by using `~two.comp(i,d)` as the conditional variance formula, with $i = 2$ corresponding to the leverage effects as in (7.12), and $i = 1$ corresponding to the leverage effects as in (7.13). For example, the following model is essentially the two components TGARCH model:

```
> garch(ford.s~1, ~two.comp(2,2), leverage=T, trace=F)

Call:
garch(formula.mean = ford.s ~ 1, formula.var =
~ two.comp(2, 2), leverage = T, trace = F)

Mean Equation: ford.s ~ 1

Conditional Variance Equation:  ~ two.comp(2, 2)

Coefficients:

       C  5.371e-04
       A  1.368e-06
ALPHA(1)  1.263e-02
ALPHA(2)  1.154e-01
 BETA(1)  9.674e-01
 BETA(2)  2.998e-01
  LEV(1)  8.893e-02
  LEV(2) -5.235e-02
```

7.5.3 GARCH-in-the-Mean Model

In financial investment, high risk is often expected to lead to high returns. Although modern capital asset pricing theory does not imply such a simple relationship, it does suggest there are some interactions between expected returns and risk as measured by volatility. Engle, Lilien and Robins (1987) proposed to extend the basic GARCH model so that the conditional volatil-

$g(\sigma_t)$	Formula name
σ	`sd.in.mean`
σ^2	`var.in.mean`
$\ln(\sigma^2)$	`logvar.in.mean`

TABLE 7.3. Possible functions for $g(\sigma_t)$

ity can generate a *risk premium* which is part of the expected returns. This extended GARCH model is often referred to as *GARCH-in-the-mean* (GARCH-M) model.

The GARCH-M model extends the conditional mean equation (7.8) as follows:

$$y_t = c + \alpha g(\sigma_t) + \epsilon_t \tag{7.18}$$

where $g(\cdot)$ can be an arbitrary function of volatility σ_t. The `garch` function allows the GARCH-M specification in the conditional mean equation together with any valid conditional variance specification. However, the function $g(\sigma_t)$ must be one of the functions listed in Table 7.3, where the corresponding formula specifications are also given.

For example, to fit a GARCH-M model with $g(\sigma_t) = \sigma_t^2$ to Hewlett-Packard stock returns using a PGARCH$(1,1,1)$ model with leverage effects, use the following command:

```
> hp.gmean = garch(hp.s~var.in.mean,~pgarch(1,1,1),leverage=T)
Iteration   0  Step Size =  1.00000  Likelihood =  2.40572
Iteration   0  Step Size =  2.00000  Likelihood =  2.40607
Iteration   0  Step Size =  4.00000  Likelihood =  2.38124
Iteration   1  Step Size =  1.00000  Likelihood =  2.40646
Iteration   1  Step Size =  2.00000  Likelihood =  2.40658
Iteration   1  Step Size =  4.00000  Likelihood =  2.40611
Iteration   2  Step Size =  1.00000  Likelihood =  2.40667
Iteration   2  Step Size =  2.00000  Likelihood =  2.40669
Iteration   2  Step Size =  4.00000  Likelihood =  2.40653

Convergence R-Square = 7.855063e-05 is less than tolerance
= 0.0001
Convergence reached.
> summary(hp.gmean)

Call:
garch(formula.mean = hp.s ~ var.in.mean, formula.var =
      ~ pgarch(1, 1, 1), leverage = T)

Mean Equation: hp.s ~ var.in.mean

Conditional Variance Equation:  ~ pgarch(1, 1, 1)
Conditional Distribution:  gaussian

--------------------------------------------------------------

Estimated Coefficients:
--------------------------------------------------------------
                Value Std.Error t value  Pr(>|t|)
           C -0.001712 0.0013654  -1.254 1.050e-01
ARCH-IN-MEAN  4.373179 2.8699425   1.524 6.386e-02
           A  0.001648 0.0003027   5.444 2.920e-08
     ARCH(1)  0.093854 0.0096380   9.738 0.000e+00
    GARCH(1)  0.853787 0.0176007  48.509 0.000e+00
      LEV(1) -0.161515 0.0648241  -2.492 6.399e-03
```

The coefficient α in (7.18) is identified by `ARCH-IN-MEAN` in the output. In this case, the risk premium is positive which implies that high risk (volatility) leads to high expected returns. However, the p-value for the t-statistic is slightly larger than the conventional 5% level.

7.5.4 ARMA Terms and Exogenous Variables in Conditional Mean Equation

So far the conditional mean equation has been restricted to a constant when considering GARCH models, except for the GARCH-M model where volatility was allowed to enter the mean equation as an explanatory variable. The `garch` function in `S+FinMetrics` allows ARMA terms as well as exogenous explanatory variables in the conditional mean equation. The most general form for the conditional mean equation is

$$y_t = c + \sum_{i=1}^{r} \phi_i y_{t-i} + \sum_{j=1}^{s} \theta_j \epsilon_{t-j} + \sum_{l=1}^{L} \boldsymbol{\beta}_l' \mathbf{x}_{t-l} + \epsilon_t \tag{7.19}$$

where $\mathbf{x}_t$ is a $k \times 1$ vector of weakly exogenous variables, and $\boldsymbol{\beta}_l$ is the $k \times 1$ vector of coefficients. Note that distributed lags of the exogenous variables in $\mathbf{x}_t$ are also allowed. To include AR(r), MA(s), or ARMA(r, s) terms in the conditional mean, simply add `ar(r)`, `ma(s)`, or `arma(r,s)` to the conditional mean formula.

Example 42 *Single factor model with GARCH errors*

From the Capital Asset Pricing Model (CAPM), stock returns should be correlated with the returns on a market index, and the regression coefficient is usually referred to as the "market beta". `S+FinMetrics` comes with a "`timeSeries`" object `nyse.s` which represents daily returns on a value weighted New York Stock Exchange index and covers the same time period

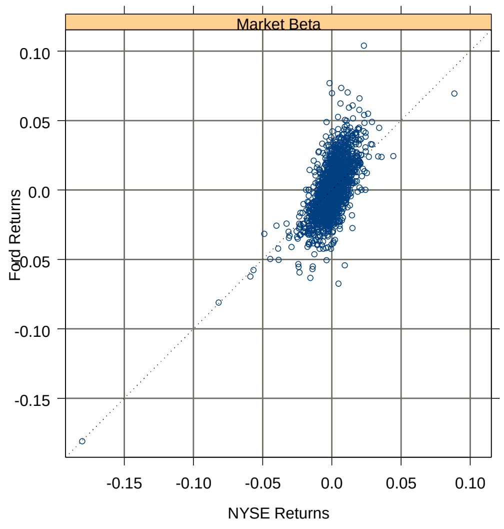


FIGURE 7.9. Daily Ford returns versus NYSE returns.

as `ford.s`. Use the `S+FinMetrics` function `rvfPlot` to generate a Trellis scatter plot of `ford.s` versus `nyse.s`:

```
> rvfPlot(ford.s, nyse.s, strip.text="Market Beta",
          id.n=0, hgrid=T, vgrid=T,
          xlab="NYSE Returns", ylab="Ford Returns")
```

The plot is shown in Figure 7.9, from which a clear linear relationship can be seen. To estimate the market beta for daily Ford returns allowing for a GARCH$(1, 1)$ error, use the following command:

```
> ford.beta = garch(ford.s~ma(1)+seriesData(nyse.s),
  ~garch(1,1), trace=F)
> summary(ford.beta)

Call:
garch(formula.mean = ford.s ~ ma(1) + seriesData(nyse.s),
      formula.var = ~ garch(1, 1), trace = F)

Mean Equation: ford.s ~ ma(1) + seriesData(nyse.s)

Conditional Variance Equation:  ~ garch(1, 1)

Conditional Distribution:  gaussian
```

```
--------------------------------------------------------------

Estimated Coefficients:
--------------------------------------------------------------
                        Value Std.Error  t value  Pr(>|t|)
                   C 8.257e-05 3.063e-04   0.2695 3.938e-01
              MA(1) 4.448e-02 2.186e-02   2.0348 2.100e-02
seriesData(nyse.s) 1.234e+00 2.226e-02  55.4418 0.000e+00
                   A 1.406e-06 5.027e-07   2.7971 2.603e-03
            ARCH(1) 3.699e-02 4.803e-03   7.7019 1.044e-14
           GARCH(1) 9.566e-01 6.025e-03 158.7691 0.000e+00
```

Note that an MA(1) term has also been added in the mean equation to allow for first order serial correlation in the daily returns caused by the possible bid-ask bounce often observed in daily stock prices. The above summary shows that both the MA(1) coefficient and market beta are highly significant. The estimated volatility is shown in Figure 7.10, which is obtained by choosing choice `8` from the `plot` method. Compare this with the estimated volatility without using `nyse.s` as shown in Figure 7.7: the estimated volatility has the same pattern, but the magnitude of volatility has significantly decreased. Since the market effects are taken into consideration here by using `nyse.s` as an explanatory variable, the resulting volatility may be interpreted as the "idiosyncratic" volatility, while the volatility in Figure 7.7 includes both the idiosyncratic component and the systematic market component.

7.5.5 Exogenous Explanatory Variables in the Conditional Variance Equation

Adding explanatory variables into the conditional variance formula may have impacts on conditional volatility.[6] To illustrate, it is widely believed that trading volume affects the volatility. The `S+FinMetrics` object `dell.s` contains a time series of daily stock returns of Dell Computer Corporation, and `dell.v` contains daily trading volume of Dell stocks spanning the same time period. In the next example, use the percentage change in trading volume to forecast volatility.

Example 43 *Trading volume and volatility*

First, use the `S+FinMetrics` function `getReturns` to compute rates of changes in trading volume. Then look at the scatter plot of log absolute returns versus the changes in trading volume:

[6]To guarantee that the conditional variance is always positive, one has to make sure that exogenous variables are positive unless an EGARCH type model is selected.

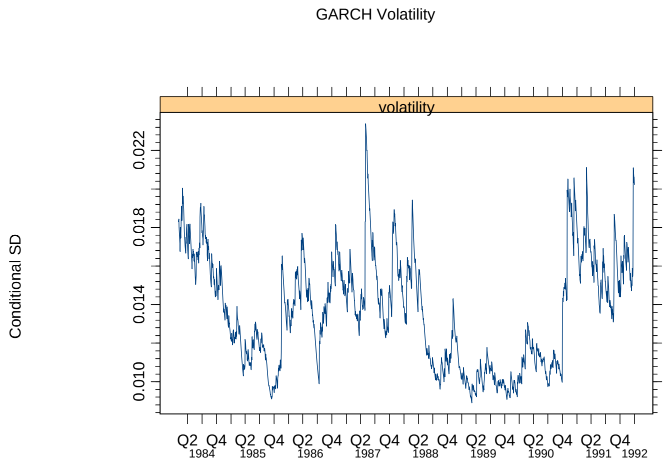


FIGURE 7.10. Idiosyncratic volatility of daily Ford returns.

```
> log.abs.ret = log(abs(dell.s-mean(dell.s)))[-1]
> d.volume = getReturns(dell.v)
> rvfPlot(log.abs.ret, d.volume, strip="Scatter Plot",
          id.n=0, hgrid=T, vgrid=T,
          xlab="% Volume", ylab="Volatility")
```

The resulting plot is shown in Figure 7.11. There seems to exist a fairly linear relationship between the changes in volume and the volatility as measured by the log absolute returns. Based on this observation, use the changes in volume as an explanatory variable in the EGARCH variance equation:

```
> dell.mod = garch(dell.s~1,~egarch(1,1)+seriesData(d.volume),
  series.start=2)
> summary(dell.mod)

Call:
garch(formula.mean = dell.s ~ 1, formula.var = ~ egarch(1, 1)
      + seriesData(d.volume), series.start = 2)

Mean Equation: dell.s ~ 1

Conditional Variance Equation:  ~ egarch(1, 1) +
      seriesData(d.volume)
```

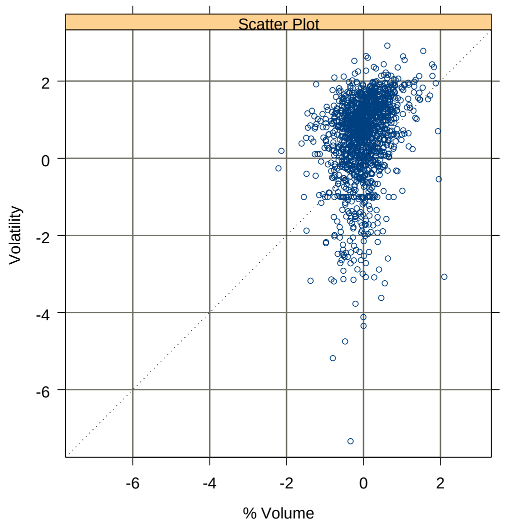


FIGURE 7.11. Log absolute returns versus changes in volume: Dell.

```
Conditional Distribution:  gaussian

-------------------------------------------------------------

Estimated Coefficients:
-------------------------------------------------------------
                        Value Std.Error  t value  Pr(>|t|)
                   C  0.15678   0.06539   2.3977 8.321e-03
                   A -0.02078   0.03927  -0.5293 2.984e-01
             ARCH(1)  0.14882   0.03721   3.9992 3.364e-05
            GARCH(1)  0.95140   0.01695  56.1226 0.000e+00
seriesData(d.volume)  1.39898   0.08431  16.5928 0.000e+00
```

The optional argument `series.start=2` is used because the “`timeSeries`” `d.volume` has one less observation than the “`timeSeries`” `dell.s`. From the summary output, the coefficient on changes in volume is estimated to be 1.4 and is highly significant with a p-value essentially equal to zero. The estimated model implies a 1% change in trading volume causes about a 1.4% change in conditional variance.

7.5.6 Non-Gaussian Error Distributions

In all the examples illustrated so far, a normal error distribution has been exclusively used. However, given the well known fat tails in financial time series, it may be more desirable to use a distribution which has fatter tails than the normal distribution. The `garch` function in `S+FinMetrics` allows three fat-tailed error distributions for fitting GARCH models: the *Student's t distribution*; the *double exponential distribution*; and the *generalized error distribution.*

Student's t Distribution

If a random variable u_t has a Student's t distribution with ν degrees of freedom and a scale parameter s_t, the probability density function (PDF) of u_t is given by:

$$f(u_t) = \frac{\Gamma[(\nu+1)/2]}{(\pi\nu)^{1/2}\Gamma(\nu/2)} \frac{s_t^{-1/2}}{[1+u_t^2/(s_t\nu)]^{(\nu+1)/2}}$$

where $\Gamma(\cdot)$ is the gamma function. The variance of u_t is given by:

$$Var(u_t) = \frac{s_t\nu}{\nu-2},\ v > 2.$$

If the error term ϵ_t in a GARCH model follows a Student's t distribution with ν degrees of freedom and $Var_{t-1}(\epsilon_t) = \sigma_t^2$, the scale parameter s_t should be chosen to be

$$s_t = \frac{\sigma_t^2(\nu-2)}{\nu}.$$

Thus the log-likelihood function of a GARCH model with Student's t distributed errors can be easily constructed based on the above PDF.

Generalized Error Distribution

Nelson (1991) proposed to use the *generalized error distribution* (GED) to capture the fat tails usually observed in the distribution of financial time series. If a random variable u_t has a GED with mean zero and unit variance, the PDF of u_t is given by:

$$f(u_t) = \frac{\nu \exp[-(1/2)|u_t/\lambda|^\nu]}{\lambda \cdot 2^{(\nu+1)/\nu}\Gamma(1/\nu)}$$

where

$$\lambda = \left[\frac{2^{-2/\nu}\Gamma(1/\nu)}{\Gamma(3/\nu)}\right]^{1/2}$$

and ν is a positive parameter governing the thickness of the tail behavior of the distribution. When $\nu = 2$ the above PDF reduces to the standard normal PDF; when $\nu < 2$, the density has thicker tails than the normal density; when $\nu > 2$, the density has thinner tails than the normal density.

When the tail thickness parameter $\nu = 1$, the PDF of GED reduces to the PDF of *double exponential distribution*:

$$f(u_t) = \frac{1}{\sqrt{2}} e^{-\sqrt{2}|u_t|}.$$

Based on the above PDF, the log-likelihood function of GARCH models with GED or double exponential distributed errors can be easily constructed. Refer to Hamilton (1994) for an example.

GARCH Estimation with Non-Gaussian Error Distributions

To estimate a GARCH model with the above three non-Gaussian error distributions using the `garch` function, simply set the optional argument `cond.dist` to `"t"` for the Student's t distribution, `"ged"` for the GED distribution, and `"double.exp"` for the double exponential distribution, respectively.

For example, to estimate a basic GARCH$(1, 1)$ model with Student's t distribution using daily Ford stock returns `ford.s`, use the command:

```
> ford.mod11.t = garch(ford.s~1, ~garch(1,1), cond.dist="t")
Iteration   0  Step Size =  1.00000  Likelihood =  2.64592
Iteration   0  Step Size =  2.00000  Likelihood = -1.00000e+10
Iteration   1  Step Size =  1.00000  Likelihood =  2.64788
Iteration   1  Step Size =  2.00000  Likelihood =  2.64367
Iteration   2  Step Size =  1.00000  Likelihood =  2.64808
Iteration   2  Step Size =  2.00000  Likelihood =  2.64797

Convergence R-Square = 4.712394e-05 is less than tolerance
= 0.0001
Convergence reached.
```

The distribution information is saved in the `cond.dist` component of the returned object:

```
> ford.mod11.t$cond.dist
$cond.dist:
[1] "t"

$dist.par:
[1] 7.793236

$dist.est:
[1] T
```

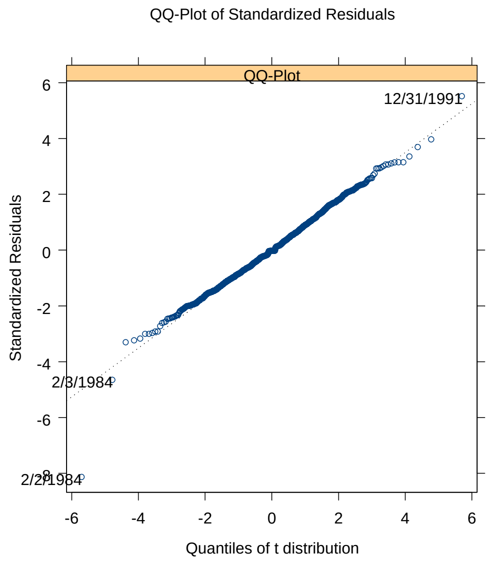


FIGURE 7.12. Student-t QQ-plot of standardized residuals: `ford.mod11.t`.

where the `dist.par` component contains the estimated degree of freedom ν for Student's t distribution. Calling the generic `summary` function on the returned object will also show the standard error of the estimate of ν.

To assess the goodness-of-fit of `ford.mod11.t`, generate the qq-plot based on the estimated Student's t distribution by calling the `plot` function on `ford.mod11.t`, which is shown in Figure 7.12. Compare this with Figure 7.5 and the Student's t error distribution provides a much better fit than the normal error distribution.

When using Student's t or GED distributions, the distribution parameter ν is estimated as part of the MLE procedure by default. One can also choose to fix ν at a certain value during the estimation. For example, to fix $\nu = 1$ for GED distribution, use the command:

```
> ford.mod11.dexp = garch(ford.s~1, ~garch(1,1),
+ cond.dist="ged", dist.par=1, dist.est=F)
```

where the optional argument `dist.par` is used to set the value, and `dist.est` is used to exclude the distribution parameter for MLE. It can be easily verified that this is equivalent to setting `cond.dist="double.exp"`.

7.6 GARCH Model Selection and Comparison

The previous sections have illustrated the variety of GARCH models available in S+FinMetrics. Obviously selecting the best model for a particular data set can be a daunting task. Model diagnostics based on standardized residuals and news impact curves for leverage effects can be used to compare the effectiveness of different aspects of GARCH models. In addition, since GARCH models can be treated as ARMA models for squared residuals, traditional model selection criteria such as Akaike information criterion (AIC) and Bayesian information criterion (BIC) can also be used for selecting models.

To facilitate the selection and comparison of different GARCH models, S+FinMetrics provides the function compare.mgarch to compare the fits of different "garch" objects.[7] For example, to compare the GARCH(1,1) fits of the "garch" objects ford.mod11 and ford.mod11.t, one fitted with the Gaussian distribution and the other with the Student's t distribution, use the following command:

```
> ford.compare = compare.mgarch(ford.mod11, ford.mod11.t)
> oldClass(ford.compare)
[1] "compare.garch"  "compare.mgarch"
> ford.compare
           ford.mod11 ford.mod11.t
       AIC     -10504       -10582
       BIC     -10481       -10554
Likelihood       5256         5296
```

The returned object ford.compare is an S version 3 object with class "compare.garch", which inherits from the class "compare.mgarch". The print method for this class of objects shows the AIC, BIC, and log-likelihood values of the fitted models. Since the BIC of ford.mod11.t is much smaller than that of ford.mod11, Student's t distribution seems to provide a much better fit than the normal distribution.

S+FinMetrics also provides a method for the generic plot function for objects inheriting from class "compare.mgarch". To see the arguments of the plot method, use the args function as usual:

```
> args(plot.compare.mgarch)
function(x, qq = F, hgrid = F, vgrid = F, lag.max = NULL,
        ci = 2, ...)
> plot(ford.compare)
```

[7]This is originally designed as a method function for the generic compare function for an S version 3 object. However, for S-PLUS 6 which is based on S version 4, the generic function compare does not work correctly when more than two objects are compared. So we suggest calling compare.mgarch directly.

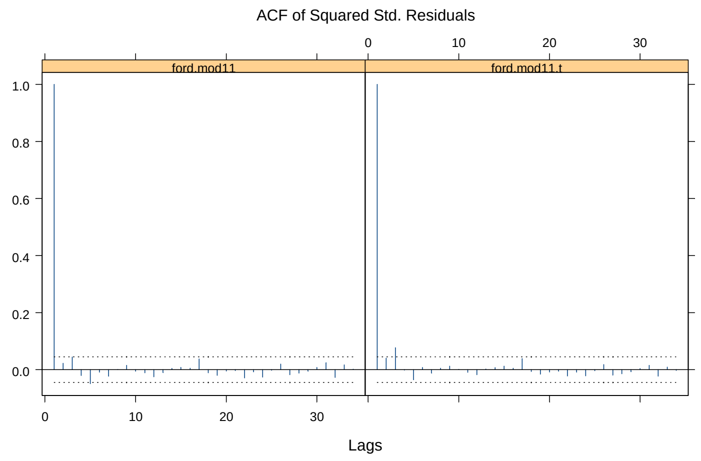


FIGURE 7.13. Comparison of ACF of squared std. residuals.

The resulting plot is shown Figure 7.13. By default, the `plot` method compares the ACF of squared standardized residuals from the fitted models. This plot demonstrates that both models are successful at modeling conditional volatility. If the optional argument is set at `qq=T`, then a comparison of qq-plots is generated:

```
> plot(ford.compare, qq=T, hgrid=T, vgrid=T)
```

which is shown in Figure 7.14. Note that since `ford.mod11` is fitted using the normal distribution, the qq-plot is based on normal assumption. In contrast, since `ford.mod11.t` is fitted using Student's t distribution, the qq-plot is based on a Student's t distribution with degrees of freedom taken from the `cond.dist` component of the object.

7.6.1 Constrained GARCH Estimation

For a GARCH model, some model parameters can also be fixed at certain values to evaluate the fit of a particular model. Section 13.7 in Chapter 13 provides some examples.

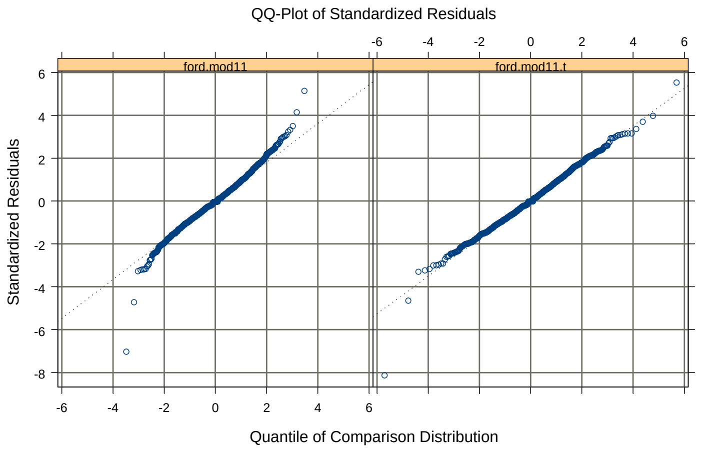


FIGURE 7.14. Comparison of QQ-plot of std. residuals.

7.7 GARCH Model Prediction

An important task of modeling conditional volatility is to generate accurate forecasts for both the future value of a financial time series as well as its conditional volatility. Since the conditional mean of the general GARCH model (7.19) assumes a traditional ARMA form, forecasts of future values of the underlying time series can be obtained following the traditional approach for ARMA prediction. However, by also allowing for a time varying conditional variance, GARCH models can generate accurate forecasts of future volatility, especially over short horizons. This section illustrates how to forecast volatility using GARCH models.

For simplicity, consider the basic GARCH$(1,1)$ model:

$$\sigma_t^2 = a_0 + a_1\epsilon_{t-1}^2 + b_1\sigma_{t-1}^2$$

which is estimated over the time period $t = 1, 2, \cdots, T$. To obtain $E_T[\sigma_{T+k}^2]$, the forecasts of future volatility σ_{T+k}^2, for $k > 0$, given information at time T. The above equation can easily obtain:

$$\begin{aligned} E_T[\sigma_{T+1}^2] &= a_0 + a_1 E_T[\epsilon_T^2] + b_1 E_T[\sigma_T^2] \\ &= a_0 + a_1\epsilon_T^2 + b_1\sigma_T^2 \end{aligned}$$

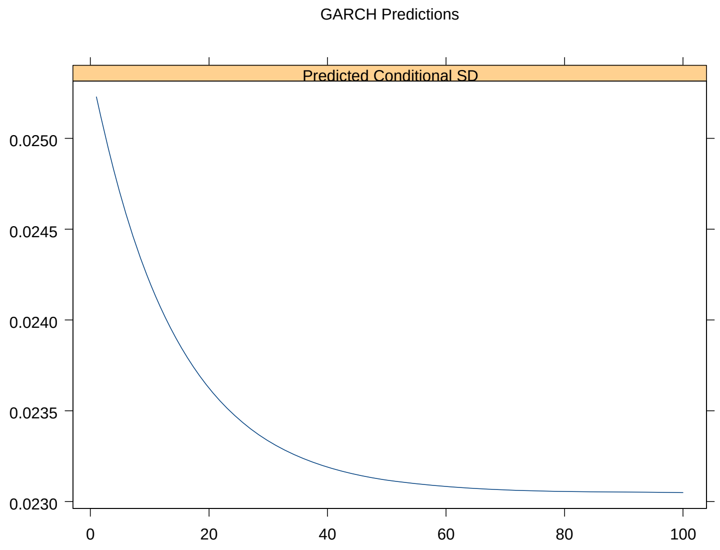


FIGURE 7.15. PGARCH forecasts of future volatility: `hp.pgarch`.

since it already has ϵ_T^2 and σ_T^2 after the estimation.[8] Now for $T+2$

$$\begin{aligned} E_T[\sigma_{T+2}^2] &= a_0 + a_1 E_T[\epsilon_{T+1}^2] + b_1 E_T[\sigma_{T+1}^2] \\ &= a_0 + (a_1 + b_1) E_T[\sigma_{T+1}^2]. \end{aligned}$$

since $E_T[\varepsilon_{T+1}^2] = E_T[\sigma_{T+1}^2]$. The above derivation can be iterated to give the conditional volatility forecasting equation

$$E_T[\sigma_{T+k}^2] = a_0 \sum_{i=1}^{k-2} (a_1 + b_1)^i + (a_1 + b_1)^{k-1} E_T[\sigma_{T+1}^2] \tag{7.20}$$

for $k \geq 2$. Notice that as $k \to \infty$, the volatility forecast in (7.20) approaches the unconditional variance $a_0/(1-a_1-b_1)$ if the GARCH process is stationary (i.e., if $\alpha_1 + b_1 < 1$).

The forecasting algorithm (7.20) produces forecasts for the conditional variance σ_{T+k}^2. The forecast for the conditional volatility, σ_{T+k}, is defined as the square root of the forecast for σ_{T+k}^2.

The `predict` method for "`garch`" objects in `S+FinMetrics` implements the forecasting procedure as described above for all the supported GARCH variants, allowing for leverage effects and the use of exogenous variables

[8]We are a little bit loose with notations here because ϵ_T and σ_T^2 are actually the fitted values instead of the unobserved "true" values.

in both the conditional mean and the conditional variance. The forecasts can be easily obtained by calling the generic `predict` function on a fitted model object with the desired number of forecasting periods. For example, consider the PGARCH object `hp.pgarch` in Section 7.5.1. To obtain 10-step-ahead forecasts, simply use the command:

```
> hp.pgarch.pred = predict(hp.pgarch,10)
> class(hp.pgarch.pred)
[1] "predict.garch"
> names(hp.pgarch.pred)
[1] "series.pred" "sigma.pred"  "asymp.sd"
> hp.pgarch.pred
$series.pred:
 [1] 0.0003312 0.0003312 0.0003312 0.0003312 0.0003312
 [6] 0.0003312 0.0003312 0.0003312 0.0003312 0.0003312

$sigma.pred:
 [1] 0.02523 0.02508 0.02494 0.02482 0.02470 0.02458 0.02448
 [8] 0.02438 0.02429 0.02421

$asymp.sd:
[1] 0.02305

attr(, "class"):
[1] "predict.garch"
```

The returned object `hp.pgarch.pred` is of class "`predict.garch`" for which there is only a `plot` method. Since the conditional mean was restricted to a constant in `hp.pgarch`, the forecasts of the future values contained in the component `series.pred` are simply the estimate of the mean. The component `sigma.pred` contains the forecasts of σ_t, and the component `asymp.sd` contains the estimate of the unconditional standard deviation if the estimated model is stationary. If a very large number of steps lie ahead, the forecasted volatility should approach the unconditional level. This can be easily verified for `hp.pgarch` as follows:

```
> plot(predict(hp.pgarch, 100))
```

where the `plot` method for the returned object can be directly invoked and the resulting plot is shown in Figure 7.15. Note that a plot of the forecasted series values can also be obtained. See the on-line help file for `plot.predict.garch` for details.

The forecasted volatility can be used together with forecasted series values to generate confidence intervals of the forecasted series values. In many cases, the forecasted volatility is of central interest, and confidence intervals for the forecasted volatility can be obtained as well. However, analytic formulas for confidence intervals of forecasted volatility are only known for some special cases (see Baillie and Bollerslev, 1992). The next section will show how a simulation-based method can be used to obtain confidence intervals for forecasted volatility from any of the GARCH variants available in `S+FinMetrics`.

7.8 GARCH Model Simulation

`S+FinMetrics` provides a method for the generic `S-PLUS` function `simulate` for objects of class "`garch`". This function, `simulate.garch`, allows observations as well as volatility to be simulated from a user-specified GARCH model or from the model contained in a fitted "`garch`" object. This section illustrates the use of `simulate` to create confidence intervals for volatility forecasts from a fitted GARCH model.

Example 44 *Simulation-based GARCH forecasts*

To obtain volatility forecasts from a fitted GARCH model, simply simulate from the last observation of the fitted model. This process can be repeated many times to obtain an "ensemble" of volatility forecasts. For example, suppose 100-step-ahead volatility forecasts need to be generated from `hp.pgarch`, take the residual term and fitted volatility of the last observation:[9]

```
> sigma.start = as.numeric(hp.pgarch$sigma.t[2000])
> eps.start = as.numeric(hp.pgarch$residuals[2000])
> eps.start = matrix(eps.start, 1, 1000)
> error = rbind(eps.start, matrix(rnorm(100*1000), 100))
```

Note that the first row of `error` contains the pre-sample values of ϵ_t to start off the simulation for each of the 1000 replications, whereas the rest of `error` are simply random variates with zero mean and unit variance which will be updated by the simulation procedure to result in GARCH errors. Now use these values to obtain the simulations as follows:

```
> set.seed(10)
> hp.sim.pred = simulate(hp.pgarch, n=100, n.rep=1000,
      sigma.start=sigma.start, etat=error)$sigma.t
```

The argument `n` specifies the desire to simulate 100 steps ahead, and `n.rep` specifies wanting to repeat this 1000 times. The simulation procedure returns both the simulated GARCH errors and volatility. Only take the simulated volatility contained in the `sigma.t` component; thus `hp.sim.pred` is a 100×1000 matrix with each column representing each simulation path.

[9]If the order of the fitted GARCH model is $m = \max(p, q)$, then m last observations are required.

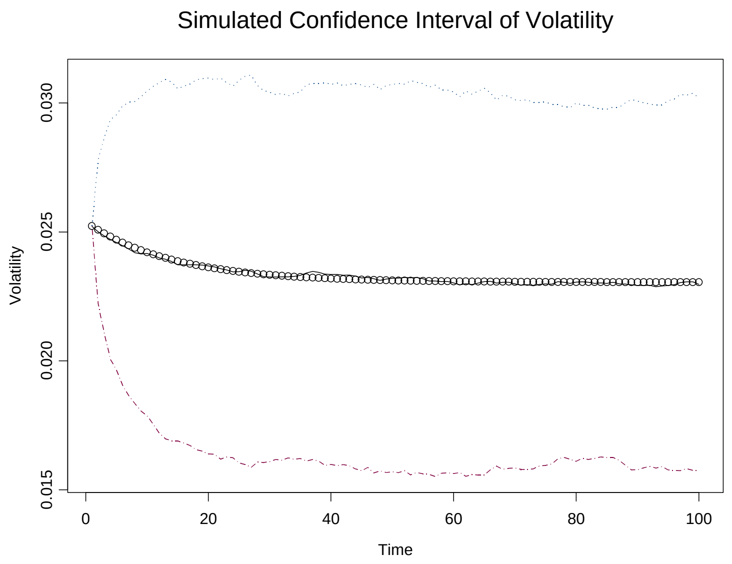


FIGURE 7.16. Simulation-based volatility forecasts: `hp.pgarch`.

The simulation-based forecasts are simply the average of the 1000 simulation paths. 95% confidence intervals for the forecasts may be computed in two ways. They can be computed using the usual formula based on normally distributed forecasts; that is, mean forecast $\pm 2\cdot$ standard deviation of forecasts. Alternatively, the 95% confidence interval may be constructed from the 2.5% and 97.5% quantiles of the simulated forecasts. Use the following code to compute the forecasts and plot the 95% confidence interval based on the normal formula:

```
> vol.mean = rowMeans(hp.sim.pred)
> vol.stdev = rowStdevs(hp.sim.pred)
> vol.cf = cbind(vol.mean+2*vol.stdev, vol.mean-2*vol.stdev)
> tsplot(cbind(vol.mean, vol.cf))
> points(predict(hp.pgarch, 100)$sigma.pred, pch=1)
> title(main="Simulated Confidence Interval of Volatility",
    xlab="Time", ylab="Volatility")
```

The resulting plot is shown in Figure 7.16. Note that analytic forecasts are also added as points in the plot for comparison. The simulation-based forecasts agree with the analytic ones produced by the `predict` method.

In the above example, the "standardized" errors were generated by random sampling from the standard normal distribution. In practice, it may be desirable to generate standardized errors by bootstrapping from standardized residuals.

Formula	Model
`~garch(p,q)`	GARCH(p, q) model
`~egarch(p,q)`	EGARCH(p, q) model
`~tgarch(p,q)`	TGARCH(p, q) model
`~pgarch(p,q)`	PGARCH(p, q) model with free exponent d
`~pgarch(p,q,d)`	PGARCH(p, q) model with fixed exponent d
`~garch.2comp`	GARCH TWO.COMP model
`~egarch.2comp`	EGARCH TWO.COMP model
`~pgarch.2comp`	PGARCH TWO.COMP model with free exponent d
`~pgarch.2comp(d)`	PGARCH TWO.COMP model with fixed exponent d
`~two.comp(i)`	PGARCH TWO.COMP model with choice of leverage effects
`~two.comp(i,d)`	PGARCH TWO.COMP model with choice of leverage effects and exponent d

TABLE 7.4. GARCH Formula Specifications

7.9 Conclusion

This chapter illustrated how to estimate and forecast from various GARCH models. The range of GARCH models supported by `S+FinMetrics` is very broad. Table 7.4 summarizes all the conditional variance formulas supported by the `garch` function.

7.10 References

Alexander, C. (2001). *Market Models: A Guide to Financial Data Analysis*, John Wiley & Sons, Chichester, UK.

Bera, A.K. and M.L. Higgins (1995). "On ARCH Models: Properties, Estimation and Testing," *Journal of Economic Surveys*, 7, 305-362.

Black, F. (1976). "Studies in Stock Price Volatility Changes," *Proceedings of the 1976 Business Meeting of the Business and Economics Statistics Section*, American Statistical Association, 177-181.

Bollerslev, T. (1986). "Generalized Autoregressive Conditional Heteroskedasticity," *Journal of Econometrics*, 31, 307-327.

Bollerslev, T., R.Y. Chu and K.F. Kroner (1994). "ARCH Modeling in Finance: A Selective Review of the Theory and Empirical Evidence," *Journal of Econometrics*, 52, 5-59.

Bollerslev, T., Engle, R. F., and Nelson, D. B. (1994). "ARCH Models," in R. F. Engle and D. L. McFadden (eds.), *Handbook of Econometrics*, Vol. 4, Elsevier Science B. V., Amsterdam.

Bollerslev, T., and Wooldrige, J. M. (1992). "Quasi-maximum Likelihood Estimation and Inference in Dynamic Models with Time-varying Covariances," *Econometric Reviews*, 11, 143-172.

Davidson, R., MacKinnon, J. G. (1993). *Estimation and Inference in Econometrics*, Oxford University Press, Oxford.

Diebold, F.X. and J.A. Lopez (1996). "Modeling Volatility Dynamics," in K. Hoover (ed.) *Macroeconomics: Developments, Tensions and Prospects*. Kluwer, Boston.

Ding, Z., and Granger, C. W. J. (1996). "Modeling Volatility Persistence of Speculative Returns: A New Approach," *Journal of Econometrics*, 73, 185-215.

Ding, Z., Granger, C. W. J., and Engle, R. F. (1993). "A Long Memory Property of Stock Market Returns and a New Model," *Journal of Empirical Finance*, 1, 83-106.

Engle, R. F. (1982). "Autoregressive Conditional Heteroskedasticity with Estimates of the Variance of United Kingdom Inflation," *Econometrica*, 50(4), 987-1007.

Engle, R. F., and Lee, G. J. (1999). "A Long-Run and Short-Run Component Model of Stock Return Volatility," in R. F. Engle and H. White (eds.), *Cointegration, Causality, and Forecasting*. Oxford University Press, Oxford.

Engle, R. F., Lilien, D. M., and Robins, R. P. (1987). "Estimating Time Varying Risk Premia in the Term-Structure: the ARCH-M Model," *Econometrica*, 55(2), 391-407.

Engle, R. F., and Ng, V. (1993). "Measuring and Testing the Impact of News on Volatility," *Journal of Finance*, 48(5), 1749-1778.

Engle, R.F. (2001). "GARCH 101: The Use of ARCH/GARCH Models in Applied Economics," *Journal of Economic Perspectives*, 15, 157-168.

Engle, R.F. and A.J. Patton (2001). "What Good is a Volatility Model?" *QuantitativeFinance*, 237-245.

GLOSTEN, L. R., JAGANNATHAN, R., AND RUNKLE, D. E. (1993). "On the Relation Between the Expected Value and the Volatility of the Nominal Excess Return on Stocks," *Journal of Finance*, 48(5), 1779-1801.

HAMILTON, J. D. (1994). *Time Series Analysis.* Princeton University Press, Princeton, NJ.

HE, C., TERÄSVIRTA, T. (1999a). "Properties of Moments of a Family of GARCH Processes," *Journal of Econometrics*, 92, 173-192.

HE, C., TERÄSVIRTA, T. (1999b). "Fourth Moment Structure of the GARCH(p, q) Process," *Econometric Theory*, 15, 824-846.

NELSON, D. B. (1991). "Conditional Heteroskedasticity in Asset Returns: a New Approach," *Econometrica*, 59(2), 347-370.

NELSON, D. B., AND CAO, C. Q. (1992). "Inequality Constraints in the Univariate GARCH Model," *Journal of Business and Economic Statistics*, 10(2), 229-235.

TSAY, R.S. (2001). *Analysis of Financial Time Series*, John Wiley & Sons, New York.

ZAKOIAN, M. (1994), "Threshold Heteroscedastic Models," *Journal of Economic Dynamics and Control*, 18, 931-955.

8 Long Memory Time Series Modeling

8.1 Introduction

Earlier chapters have demonstrated that many macroeconomic and financial time series like nominal and real interest rates, real exchange rates, exchange rate forward premiums, interest rate differentials and volatility measures are very persistent, i.e., that an unexpected shock to the underlying variable has long lasting effects. Persistence can occur in the first or higher order moments of a time series. The persistence in the first moment, or levels, of a time series can be confirmed by applying either unit root tests or stationarity tests to the levels, while the persistence in the volatility of the time series is usually exemplified by a highly persistent fitted GARCH model. Although traditional stationary ARMA processes often cannot capture the high degree of persistence in financial time series, the class of non-stationary unit root or $I(1)$ processes have some unappealing properties for financial economists. In the last twenty years, more applications have evolved using long memory processes, which lie halfway between traditional stationary $I(0)$ processes and the non-stationary $I(1)$ processes. There is substantial evidence that long memory processes can provide a good description of many highly persistent financial time series.

This chapter will cover the concept of long memory time series. Section 8.3 will explain various tests for long memory, or long range dependence, in a time series and show how to perform these tests using functions in `S+FinMetrics` module. In Section 8.4 will illustrate how to estimate the long memory parameter using R/S statistic and two periodogram-based method. Section 8.5 will extend the traditional ARIMA processes to fractional ARIMA (FARIMA) processes, which can be used to model the long range dependence and short run dynamics simultaneously. The semiparametric fractional autoregressive (SEMIFAR) process recently proposed by Beran and his coauthors will also be introduced. Section 8.6 will extend GARCH models to fractionally integrated GARCH models to allow for long memory in conditional volatility. Finally, section 8.7 will consider the prediction from long memory models such as FARIMA and FIGARCH/FIEGARCH models. Beran (1994) gives an exhaustive treatment of statistical aspects of modeling with long memory processes, while Baillie (1996) provides a comprehensive survey of econometric analysis of long memory processes and applications in economics and finance.

8.2 Long Memory Time Series

To illustrate the long memory property in financial time series, consider the daily returns on the S&P500 index from January 4, 1928 to August 30, 1991 contained in the `S+FinMetrics` "`timeSeries`" object `sp500`. Since daily returns usually have a mean very close to zero, the absolute return is sometimes used as a measure of volatility. The sample autocorrelation function of the daily absolute returns can be plotted using the following commands:

```
> tmp = acf(abs(sp500), lag=200)
> sp500.ar = ar(abs(sp500))
> sp500.ar$order
[1] 37
> tmp.mod = list(ar=as.vector(sp500.ar$ar), sigma2=1, d=0)
> ar.acf = acf.FARIMA(tmp.mod, lag.max=200)
> lines(ar.acf$lags, ar.acf$acf/ar.acf$acf[1])
```

and the plot is shown in Figure 8.1. The autocorrelation of absolute returns is highly persistent and remains very significant at lag 200. In the above code fragment, the `S-PLUS` function `ar` is used to select the best fitting AR process using AIC, which turns out to be an AR(37) model. The `S+FinMetrics` function `acf.FARIMA` compares the theoretical autocorrelation function implied by the AR(37) process with the sample autocorrelation function. The following comments apply to this example:

1. Traditional stationary ARMA processes have short memory in the sense that the autocorrelation function decays exponentially. In the above example, the theoretical autocorrelation closely matches the sample autocorrelation at small lags. However, for large lags, the sample autocorrelation decays much more slowly than the theoretical autocorrelation.

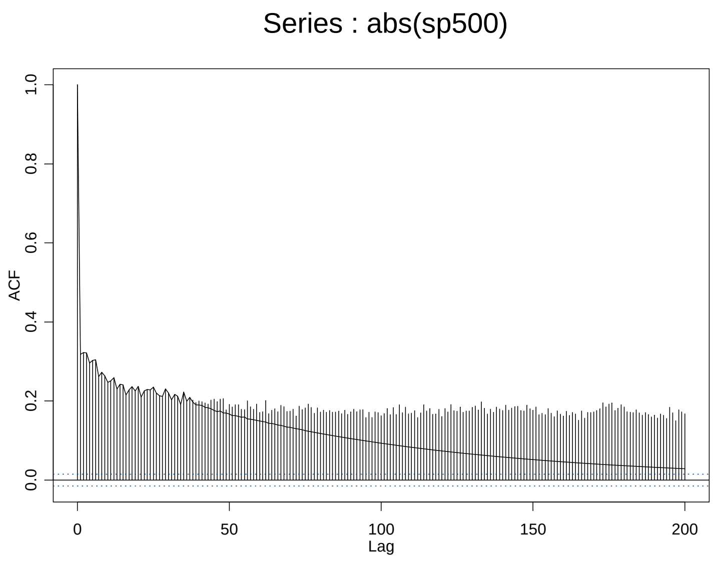


FIGURE 8.1. ACF of daily absolute returns of S&P500 index.

2. When the sample autocorrelation decays very slowly, traditional stationary ARMA processes usually result in an excessive number of parameters. In the above example, 37 autoregressive coefficients were found necessary to capture the dependence in the data.

Based on the above observations, a stationary process y_t has *long memory*, or *long range dependence*, if its autocorrelation function behaves like

$$\rho(k) \to C_\rho k^{-\alpha} \text{ as } k \to \infty \tag{8.1}$$

where C_ρ is a positive constant, and α is a real number between 0 and 1. Thus the autocorrelation function of a long memory process decays slowly at a *hyperbolic* rate. In fact, it decays so slowly that the autocorrelations are not summable:

$$\sum_{k=-\infty}^{\infty} \rho(k) = \infty.$$

For a stationary process, the autocorrelation function contains the same information as its spectral density. In particular, the spectral density is defined as:

$$f(\omega) = \frac{1}{2\pi} \sum_{k=-\infty}^{\infty} \rho(k) e^{ik\omega}$$

where ω is the Fourier frequency (c.f. Hamilton, 1994). From (8.1) it can be shown that

$$f(\omega) \to C_f \omega^{\alpha-1} \text{ as } \omega \to 0 \tag{8.2}$$

where C_f is a positive constant. So for a long memory process, its spectral density tends to infinity at zero frequency. Instead of using α, in practice use

$$H = 1 - \alpha/2 \in (0.5, 1), \tag{8.3}$$

which is known as the *Hurst coefficient* (see Hurst, 1951) to measure the long memory in y_t. The larger H is, the longer memory the stationary process has.

Based on the scaling property in (8.1) and the frequency domain property in (8.2), Granger and Joyeux (1980) and Hosking (1981) independently showed that a long memory process y_t can also be modeled parametrically by extending an integrated process to a *fractionally integrated process*. In particular, allow for fractional integration in a time series y_t as follows:

$$(1-L)^d (y_t - \mu) = u_t \tag{8.4}$$

where L denotes the lag operator, d is the fractional integration or fractional difference parameter, μ is the expectation of y_t, and u_t is a stationary short-memory disturbance with zero mean.

In practice, when a time series is highly persistent or appears to be non-stationary, let $d = 1$ and difference the time series once to achieve stationarity. However, for some highly persistent economic and financial time series, it appears that an integer difference may be too much, which is indicated by the fact that the spectral density vanishes at zero frequency for the differenced time series. To allow for long memory and avoid taking an integer difference of y_t, allow d to be fractional. The *fractional difference filter* is defined as follows, for any real $d > -1$:

$$(1-L)^d = \sum_{k=0}^{\infty} \begin{pmatrix} d \\ k \end{pmatrix} (-1)^k L^k \tag{8.5}$$

with binomial coefficients:

$$\begin{pmatrix} d \\ k \end{pmatrix} = \frac{d!}{k!(d-k)!} = \frac{\Gamma(d+1)}{\Gamma(k+1)\Gamma(d-k+1)}.$$

Notice that the fractional difference filter can be equivalently treated as an infinite order autoregressive filter.[1] It can be shown that when $|d| > 1/2$, y_t is non-stationary; when $0 < d < 1/2$, y_t is stationary and has long

[1]The `S+FinMetrics` function `FARIMA.d2ar` can be used to compute the autoregressive representation of the fractional difference filter.

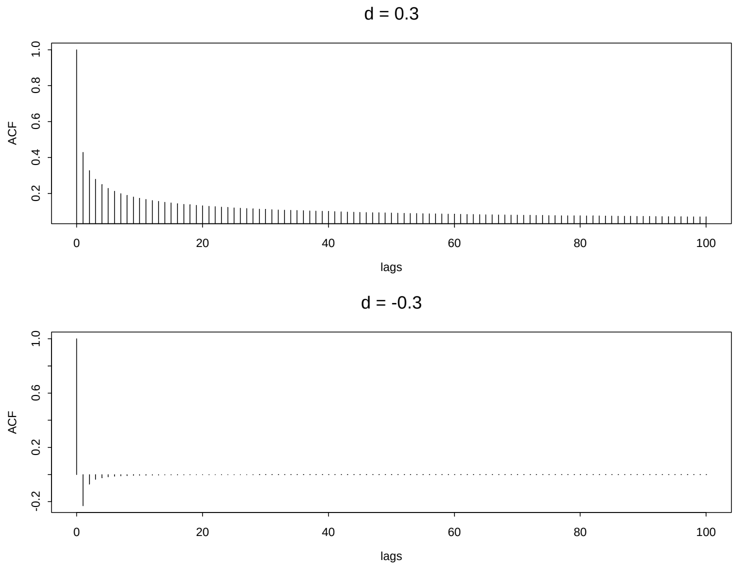


FIGURE 8.2. Autocorrelation of fractional integrated process.

memory; when $-1/2 < d < 0$, y_t is stationary and has short memory, and is sometimes referred to as *anti-persistent*.

When a fractionally integrated series y_t has long memory, it can also be shown that

$$d = H - 1/2, \tag{8.6}$$

and thus d and H can be used interchangeably as the measure of long memory. Hosking (1981) showed that the scaling property in (8.1) and the frequency domain property in (8.2) are satisfied when $0 < d < 1/2$.

Example 45 *Theoretical ACF of fractionally integrated processes*

In this example,use the S+FinMetrics function acf.FARIMA to plot the theoretical autocorrelation function of a fractionally integrated process with a standard normal disturbance u_t, for $d = 0.3$ and $d = -0.3$, respectively:

```
> d.pos = acf.FARIMA(list(d=0.3, sigma2=1), 100)
> d.pos$acf = d.pos$acf / d.pos$acf[1]
> d.neg = acf.FARIMA(list(d=-0.3, sigma2=1), 100)
> d.neg$acf = d.neg$acf / d.neg$acf[1]

> par(mfrow=c(2,1))
> plot(d.pos$lags, d.pos$acf, type="h", main="d = 0.3",
+      xlab="lags", ylab="ACF")
```

```
> plot(d.neg$lags, d.neg$acf, type="h", main="d = -0.3",
+      xlab="lags", ylab="ACF")
> par(mfrow=c(1,1))
```

and the plot is shown in Figure 8.2. Notice that the signs of the ACF coefficients are determined by the sign of d.

8.3 Statistical Tests for Long Memory

Given the scaling property of the autocorrelation function, the frequency domain property and the fractionally integrated process representation of a long memory time series, various tests have been proposed to determine the existence of long memory in a time series. This section introduces the R/S statistic and GPH test. However, before getting into the details of those test statistics, it is important to note that the definition of long memory does not dictate the general behavior of the autocorrelation function or its spectral density. Instead, they only specify the asymptotic behavior when $k \to \infty$ or $\omega \to 0$. What this means is that for a long memory process, it is not necessary for the autocorrelation to remain significant at large lags as in the previous `sp500` example, as long as the autocorrelation function decays slowly. Beran (1994) gives an example to illustrate this property.

8.3.1 R/S Statistic

The best-known test for long memory or long range dependence is probably the *rescaled range*, or *range over standard deviation*, or simply R/S statistic, which was originally proposed by Hurst (1951), and later refined by Mandelbrot and his coauthors. The R/S statistic is the range of partial sums of deviations of a time series from its mean, rescaled by its standard deviation. Specifically, consider a time series y_t, for $t = 1, \cdots, T$. The R/S statistic is defined as:

$$Q_T = \frac{1}{s_T}\left[\max_{1\le k\le T}\sum_{j=1}^{k}(y_j-\bar{y}) - \min_{1\le k\le T}\sum_{j=1}^{k}(y_j-\bar{y})\right] \tag{8.7}$$

where $\bar{y} = 1/T\sum_{i=1}^{T} y_i$ and $s_T = \sqrt{1/T\sum_{i=1}^{T}(y_i-\bar{y})^2}$. If y_t's are i.i.d. normal random variables, then

$$\frac{1}{\sqrt{T}}Q_T \Rightarrow V$$

where $\Rightarrow$ denotes weak convergence and V is the range of a Brownian bridge on the unit interval. Lo (1991) gives selected quantiles of V.

Lo (1991) pointed out that the R/S statistic is not robust to short range dependence. In particular, if y_t is autocorrelated (has short memory) then the limiting distribution of $Q_T/\sqrt{T}$ is V scaled by the square root of the long run variance of y_t. To allow for short range dependence in y_t, Lo (1991) modified the R/S statistic as follows:

$$\tilde{Q}_T = \frac{1}{\hat{\sigma}_T(q)} \left[\max_{1 \le k \le T} \sum_{j=1}^{k} (y_j - \bar{y}) - \min_{1 \le k \le T} \sum_{j=1}^{k} (y_j - \bar{y}) \right] \tag{8.8}$$

where the sample standard deviation is replaced by the square root of the Newey-West estimate of the long run variance with bandwidth q.[2] Lo (1991) showed that if there is short memory but no long memory in y_t, $\tilde{Q}_T$ also converges to V, the range of a Brownian bridge.

The `S+FinMetrics` function `rosTest` can be used to test for long memory in a time series using the R/S statistic (8.7) or the modified R/S statistic (8.8). For example, to test for long memory in the absolute returns of S&P500 index, use the following command:

```
> rosTest(abs(sp500))

Test for Long Memory: Modified R/S Test

Null Hypothesis: no long-term dependence

Test Statistics:

 7.8823**

 * : significant at 5% level
** : significant at 1% level

 Total Observ.: 17054
    Bandwidth : 14
```

By default, Lo's modified R/S statistic is computed and the bandwidth q for obtaining the long run variance is chosen to be $[4(T/100)^{1/4}]$, where T is the sample size, and $[\cdot]$ denotes integer part of. In the above example, the modified R/S statistic is significant at 1% level of significance. A different bandwidth can be used by setting the optional argument `bandwidth`. If `bandwidth` is set to zero, then classical R/S statistic is returned:

```
> rosTest(abs(sp500), bandwidth=0)
```

[2]See Chapter 2 for the definition and estimation of long run variance and the online help file for the `S+FinMetrics` function `asymp.var`.

```
Test for Long Memory: R/S Test

Null Hypothesis: no long-term dependence

Test Statistics:

 17.821**

 * : significant at 5% level
** : significant at 1% level

 Total Observ.: 17054
```

which is also significant at 1% level of significance in this case.

8.3.2 GPH Test

Based on the fractionally integrated process representation of a long memory time series as in (8.4), Geweke and Porter-Hudak (1983) proposed a semi-nonparametric approach to testing for long memory. In particular, the *spectral density* of the fractionally integrated process y_t is given by:

$$f(\omega) = [4\sin^2(\frac{\omega}{2})]^{-d} f_u(\omega) \tag{8.9}$$

where ω is the Fourier frequency, and $f_u(\omega)$ is the spectral density corresponding to u_t. Note that the fractional difference parameter d can be estimated by the following regression:

$$\ln f(\omega_j) = \beta - d\ln[4\sin^2(\frac{\omega_j}{2})] + e_j, \tag{8.10}$$

for $j = 1, 2, \cdots, n_f(T)$. Geweke and Porter-Hudak (1993) showed that using a *periodogram* estimate of $f(\omega_j)$, the least squares estimate $\hat{d}$ using the above regression is normally distributed in large samples if $n_f(T) = T^\alpha$ with $0 < \alpha < 1$:

$$\hat{d} \sim N(d, \frac{\pi^2}{6\sum_{j=1}^{n_f}(U_j - \bar{U})^2})$$

where

$$U_j = \ln[4\sin^2(\frac{\omega_j}{2})]$$

and $\bar{U}$ is the sample mean of U_j, $j = 1, \cdots, n_f$. Under the null hypothesis of no long memory ($d = 0$), the t-statistic

$$t_{d=0} = \hat{d} \cdot \left(\frac{\pi^2}{6\sum_{j=1}^{n_f}(U_j - \bar{U})^2}\right)^{-1/2} \tag{8.11}$$

has a limiting standard normal distribution.

The `S+FinMetrics` function `gphTest` can be used to estimate d from (8.10) and compute the test statistic (8.11), which is usually referred to as the *GPH test*. The arguments taken by `gphTest` are:

```
> args(gphTest)
function(x, spans = 1, taper = 0.1, pad = 0, detrend = F,
        demean = T, alpha = 0.5, na.rm = F)
```

The optional arguments `spans`, `taper`, `pad`, `detrend` and `demean` are actually passed to the `S-PLUS` function `spec.pgram` to obtain a periodogram estimate.[3] The optional argument `alpha` is used to choose the number of frequencies $n_f(T)$. By default, $n_f(T) = T^{\alpha}$ with $\alpha = 0.5$. To illustrate the use of `gphTest`, consider estimating d and testing for long memory in the S&P 500 index absolute returns:

```
> gph.sp500 = gphTest(abs(sp500),taper=0)
> class(gph.sp500)
[1] "gphTest"
> names(gph.sp500)
[1] "d"       "n"       "na"      "n.freq"  "std.err"
```

The result of `gphTest` is an object of class "`gphTest`" for which there is only a `print` method:

```
> gph.sp500

Test for Long Memory: GPH Test

Null Hypothesis: d = 0

Test Statistics:

   d 0.4573
stat 7.608**

 * : significant at 5% level
** : significant at 1% level

 Total Observ.: 17054
Number of Freq: 130
```

The estimated value of d from (8.10) is $\hat{d} = 0.457$, which suggests long memory, and the gph test statistic (8.11) is 7.608. Hence, the null of no long memory is rejected at the 1% significance level. The estimate of d is

[3]See *S-PLUS Guide to Statistics* for an introduction to the estimation of periodogram and the usage of `spec.pgram`.

close to the nonstationary range. In fact, a 95% confidence interval for d based on the asymptotic standard error

```
> gph.sp500$std.err
[1] 0.06011
```

is $[0.337, 0.578]$ and contains $d > 0.5$.

8.4 Estimation of Long Memory Parameter

The previous section introduced the R/S statistic and the log-periodogram regression to test for long memory in a time series. Cheung (1993) conducted a Monte Carlo comparison of these tests. Obtaining an estimate of the long memory parameter H or d is also of interest. The GPH test produces an estimate of d automatically. This section will show that the R/S statistic can also be used to obtain an estimate of the Hurst coefficient H. It will also introduce two periodogram-based methods for estimating the long memory parameter: the periodogram method and Whittle's method. In addition, the fractional difference parameter d can also be estimated by using a general FARIMA(p, d, q) model, which will be introduced in the next section. Taqqu, Teverovsky and Willinger (1995) and Taqqu and Teverovsky (1998) compared the performance of many different estimators of the long memory parameter, including the above mentioned methods.

8.4.1 R/S Analysis

Section 8.3.1 mentioned that when there is no long memory in a stationary time series, the R/S statistic converges to a random variable at rate $T^{1/2}$. However, when the stationary process y_t has long memory, Mandelbrot (1975) showed that the R/S statistic converges to a random variable at rate T^H, where H is the Hurst coefficient. Based on this result, the log-log plot of the R/S statistic versus the sample size used should scatter around a straight line with slope 1/2 for a short memory time series. In contrast, for a long memory time series, the log-log plot should scatter around a straight line with slope equal to $H > 1/2$, provided the sample size is large enough.

To use the above method to estimate the long memory parameter H, first compute the R/S statistic using k_1 consecutive observations in the sample, where k_1 should be large enough. Then increase the number of observations by a factor of f; that is, compute the R/S statistic using $k_i = fk_{i-1}$ consecutive observations for $i = 2, \cdots, s$. Note that to obtain the R/S statistic with k_i consecutive observations, one can actually divide the sample into $[T/k_i]$ blocks and obtain $[T/k_i]$ different values, where $[\cdot]$ denotes the integer part of a real number. Obviously, the larger k_i is, the

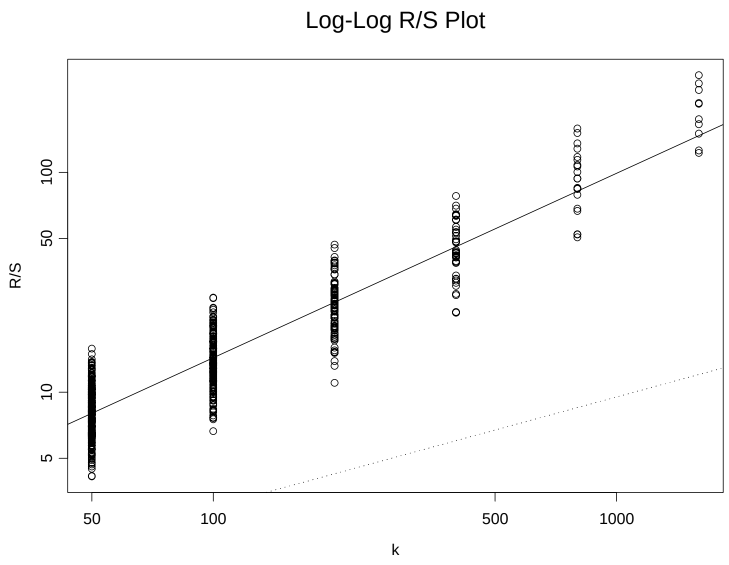


FIGURE 8.3. R/S estimate of long memory parameter.

smaller $[T/k_i]$ is. A line fit of all those R/S statistics versus k_i, $i = 1, \cdots, s$, on the log-log scale yields an estimate of the Hurst coefficient H.

The `S+FinMetrics` function `d.ros` implements the above procedure for estimating H. The arguments taken by `d.ros` are:

```
> args(d.ros)
function(x, minK = 4, k.ratio = 2, minNumPoints = 3,
        method = "ls", output = "d", plot = F, ...)
```

where `minK` specifies the value for k_1, `k.ratio` specifies the ratio factor f, and `minNumPoints` specifies the minimum requirement for $[T/k_s]$. For example, if `minNumPoints=3`, s must be such that one can divide T observations into three blocks with at least k_s observations in each block. The optional argument `output` specifies the type of output: if `output="H"`, then the Hurst coefficient is returned; if `output="d"`, then the fractional difference parameter d is returned. For example, to estimate the Hurst coefficient for absolute returns of S&P500 index using R/S statistic, use the following command:

```
> d.ros(abs(sp500), minK=50, k.ratio=2, minNumPoints=10,
+ output="H", plot=T)
[1] 0.8393
```

By setting `plot=T`, the log-log plot of R/S statistics versus k_i is generated, as shown in Figure 8.3: the solid line represents the fitted line, and the dotted line represents the case for no long memory. In this case, the solid line is far away from the dotted line, which is substantial evidence for long memory. The estimate of d using (8.6) is 0.3393.

The weakness of the above procedure is that for a particular sample, it is not clear what value of k_1 is "large enough". In addition, for large values of k_i, few values of the R/S statistic can be calculated unless the sample size is very large. To mitigate the latter problem, set the optional argument `method="l1"` when calling `d.ros`, which will direct the procedure to use the $L1$ method or least absolute deviation (LAD) method, for the line fit, and thus result in a robust estimate of the long memory parameter. For the S&P 500 absolute returns, the results using the $L1$ method are essentially the same as using the least squares method:

```
> d.ros(abs(sp500), minK=50, k.ratio=2, minNumPoints=10,
+ output="H", method="l1", plot=F)
[1] 0.8395
```

8.4.2 Periodogram Method

Section 8.3 demonstrates that for a long memory process, its spectral density approaches $C_f \omega^{1-2H}$ when the frequency ω approaches zero. Since the spectral density can be estimated by a periodogram, the log-log plot of periodogram versus the frequency should scatter around a straight line with slope $1 - 2H$ for frequencies close to zero. This method can also be used to obtain an estimate of the long memory parameter H, and it is usually referred to as the *periodogram method*.

The `S+FinMetrics` function `d.pgram` implements a procedure to estimate the long memory parameter using the periodogram method, which calls the `S-PLUS` function `spec.pgram` to obtain an estimate of periodogram. The arguments taken by `d.pgram` are:

```
> args(d.pgram)
function(x, spans = 1, taper = 0.1, pad = 0, detrend = F,
     demean = T, method = "ls", output = "d",
     lower.percentage = 0.1, minNumFreq = 10, plot = F, ...)
```

Similar to the `gphTest` function, the optional arguments `spans`, `taper`, `pad`, `detrend` and `demean` are passed to `spec.pgram` to obtain the periodogram estimate. The optional argument `lower.percentage=0.1` specifies that only the lower 10% of the frequencies are used to estimate H. For example, to estimate the long memory parameter H of `abs(sp500)` with no tapering,use the following command:

```
> d.pgram(abs(sp500), taper=0, output="H", plot=F)
[1] 0.8741311
```

The implied estimate of d is then 0.3741.

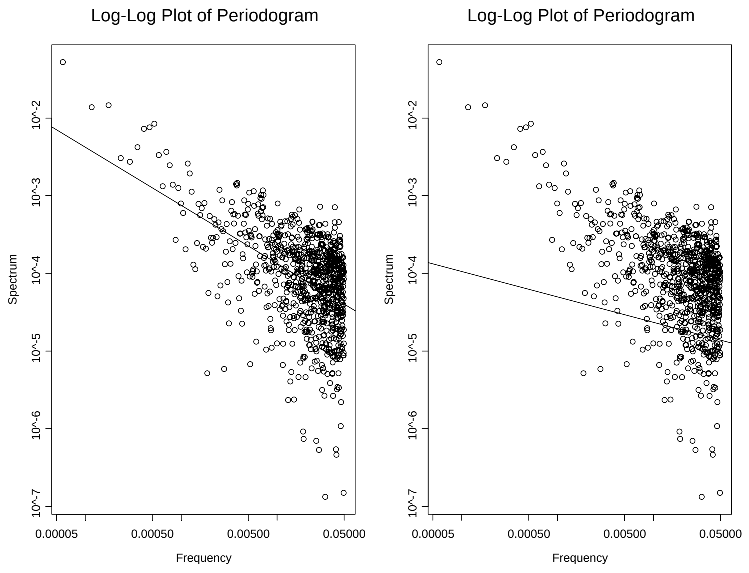


FIGURE 8.4. Periodogram estimates of long memory parameter using least squares and LAD.

Just like with the R/S estimate of the long memory parameter, it can be difficult to choose the value for `lower.percentage`. To obtain a more robust line fit, set the optional argument `method="l1"` when calling `d.pgram`, to use $L1$ method or LAD method instead of the default least squares fit. For example, to compare the least squares and $L1$ fits for `abs(sp500)`use

```
> par(mfrow=c(1,2))
> H.ls = d.pgram(abs(sp500),taper=0, output="d",plot=T)
> H.l1 = d.pgram(abs(sp500),taper=0, output="d",method="l1",
+ plot=T)
> H.ls
[1] 0.3741
> H.l1
[1] 0.1637
```

8.4.3 Whittle's Method

Whittle's method for estimating d is based on a frequency domain maximum likelihood estimation of a fractionally integrated process (8.4). It can be shown that the unknown parameters in (8.4) can be estimated by minimizing a discretized version of

$$Q(\theta) = \int_{-\pi}^{\pi} \frac{I(\omega)}{f(\theta;\omega)} d\omega$$

where θ is the vector of unknown parameters including the fractional difference parameter d, $I(\omega)$ is the periodogram of y_t, and $f(\theta, \omega)$ is the theoretical spectral density of y_t. Refer to Beran (1994) for the derivation of Whittle's method.

To use Whittle's method to estimate the fractional difference parameter d, use the S+FinMetrics function d.whittle. The syntax of d.whittle is similar to but more simple than that of d.pgram:

```
> args(d.whittle)
function(x, spans = 1, taper = 0.1, pad = 0, detrend = F,
        demean = T, output = "d")
```

where again the arguments spans, taper, pad, detrend and demean are passed to the S-PLUS function spec.pgram to obtain the periodogram. For example, to estimate the fractional difference parameter d of abs(sp500) with no tapering, use the command:

```
> d.whittle(abs(sp500), taper=0)
[1] 0.2145822
```

A caveat to using d.whittle is that although the Whittle's method is defined for a general fractionally integrated process y_t in (8.4), it is implemented assuming that u_t is a standard normal disturbance and thus y_t follows a FARIMA$(0, d, 0)$ process.

8.5 Estimation of FARIMA and SEMIFAR Models

Previous sections illustrated how to test for long memory and estimate the long memory parameter H or d. This section introduces the more flexible fractional ARIMA models, which are capable of modeling both the long memory and short run dynamics in a stationary time series. It will also introduce a semiparametric model for long memory, which allows a semiparametric estimation of a trend component.

Many empirical studies have found that there is strong evidence for long memory in financial volatility series, for example, see Lobato and Savin (1998) and Ray and Tsay (2000). Indeed, Andersen, Bollerslev, Diebold and Labys (1999) suggested to use FARIMA models to forecast daily volatility based on logarithmic realized volatility. This section will focus on modeling a volatility series for the examples.

8.5.1 Fractional ARIMA Models

The traditional approach to modeling an $I(0)$ time series y_t is to use the ARIMA model:

$$\phi(L)(1-L)^d(y_t-\mu)=\theta(L)\epsilon_t \tag{8.12}$$

where $\phi(L)$ and $\theta(L)$ are lag polynomials

$$\phi(L)=1-\sum_{i=1}^{p}\phi_i L^i$$
$$\theta(L)=1-\sum_{j=1}^{q}\theta_j L^j$$

with roots outside the unit circle, and ϵ_t is assumed to be an *iid* normal random variable with zero mean and variance σ^2. This is usually referred to as the ARIMA(p,d,q) model. By allowing d to be a real number instead of a positive integer, the ARIMA model becomes the **a**utoregressive **f**ractionally **i**ntegrated **m**oving **a**verage (ARFIMA) model, or simply, fractional ARIMA (FARIMA) model[4].

For a stationary FARIMA model with $-1/2<d<1/2$, Sowell (1992) described how to compute the exact maximum likelihood estimate (MLE). The `S-PLUS` function `arima.fracdiff` implements a very fast procedure based on the approximate MLE proposed by Haslett and Raftery (1989), and refer the reader to the *S-PLUS Guide to Statistics* for a discussion of this procedure.

However, for many economic and financial time series, the data usually seem to lie on the borderline separating stationarity from non-stationarity. As a result, one usually needs to decide whether or not to difference the original time series before estimating a stationary FARIMA model, and the inference of unknown FARIMA model parameters ignores this aspect of uncertainty in d. Beran (1995) extended the estimation of FARIMA models for any $d>-1/2$ by considering the following variation the FARIMA model:

$$\phi(L)(1-L)^\delta[(1-L)^m y_t-\mu]=\theta(L)\epsilon_t \tag{8.13}$$

where $-1/2<\delta<1/2$, and $\phi(L)$ and $\theta(L)$ are defined as above. The integer m is the number of times that y_t must be differenced to achieve stationarity, and thus the difference parameter is given by $d=\delta+m$. In the following discussions and in the `S+FinMetrics` module, restrict m to be either 0 or 1, which is usually sufficient for modeling economic and financial time series. Note that when $m=0$, μ is the expectation of y_t; in contrast, when $m=1$, μ is the slope of the linear trend component in y_t.

[4]The `S+FinMetrics` module actually provides a convenience function `FAR` for estimating a FARIMA$(p,d,0)$ model.

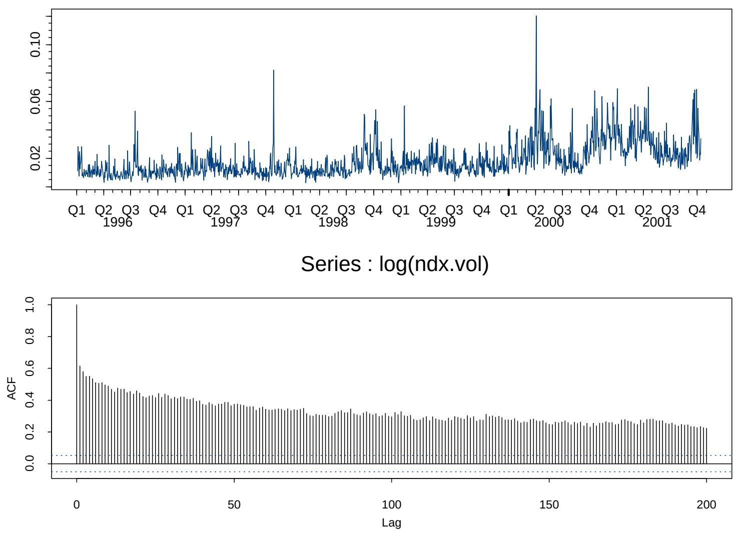


FIGURE 8.5. Garman-Klass volatility of daily NASDAQ-100 returns.

The S+FinMetrics function FARIMA implements a procedure (based on arima.fracdiff) to estimate the FARIMA model (8.13), and the standard errors of unknown parameters are computed using the asymptotic distribution derived by Beran (1995), which takes into account that m is also determined by data rather than by *a prior* decision.

To illustrate the usage of the FARIMA function, consider modeling the volatility of daily NASDAQ-100 index returns. In recent years, intra-day security prices have been employed to compute daily realized volatility, for example, see Andersen, Bollerslev, Diebold and Labys (2001a, 2001b). Since intra-day security prices can be hard to obtain, compute daily volatility based on the daily opening, highest, lowest, and closing prices, as proposed by Garman and Klass (1980) and implemented by the S+FinMetrics function TA.garmanKlass.

Example 46 *Long memory modeling of NASDAQ-100 index volatility*

The S+FinMetrics "timeSeries" ndx.dat contains the daily opening, highest, lowest and closing prices of NASDAQ-100 index from January 2, 1996 to October 12, 2001. First compute the volatility series using the Garman-Klass estimator and visualize its sample ACF:

```
> ndx.vol = TA.garmanKlass(ndx.dat[,"Open"], ndx.dat[,"High"],
+           ndx.dat[,"Low"], ndx.dat[,"Close"])
> par(mfrow=c(2,1))
> plot(ndx.vol, reference.grid=F)
> tmp = acf(log(ndx.vol), lag=200)
> par(mfrow=c(1,1))
```

The volatility series `ndx.vol` and the sample ACF of logarithmic volatility are shown in Figure 8.5. The ACF decays very slowly and remains highly significant at lag 200, which indicates that the series may exhibit long memory.

First estimate a FARIMA$(0, d, 0)$ model for logarithmic volatility as follows:

```
> ndx.d = FARIMA(log(ndx.vol), p=0, q=0)
> class(ndx.d)
[1] "FARIMA"
> names(ndx.d)
 [1] "call"      "model"     "m"         "delta"
 [5] "n.used"    "BIC"       "loglike"   "residuals"
 [9] "fitted"    "x.name"    "cov"       "CI"
```

The result of `FARIMA` is an object of class "`FARIMA`", for which there are `print`, `summary`, `plot` and `predict` methods as well as extractor functions `coef`, `fitted`, `residuals` and `vcov`. The `summary` method gives

```
> summary(ndx.d)

Call:
FARIMA(x = log(ndx.vol), p = 0, q = 0)

Coefficients:
    Value Std. Error t value Pr(>|t|)
d  0.3534  0.0205    17.1964  0.0000

Information Criteria:
 log-likelihood     BIC
  -732.3         1471.9

Residual scale estimate: 0.4001

                   total residual
Degree of freedom:  1455     1453
Time period: from 01/04/1996 to 10/12/2001
```

The estimated model appears stationary and has long memory since $0 < \hat{d} < 1/2$. Notice that m is estimated to be zero:

```
> ndx.d$m
[1] 0
```

To allow for long memory and short memory at the same time, use a FARIMA(p, d, q) model with $p \neq 0$ or $q \neq 0$. However, in practice, it is usually difficult to choose the appropriate value for p or q. The `FARIMA` function can choose the best fitting FARIMA model based on finding values of $p \leq p_{\max}$ and $q \leq q_{\max}$ which minimize the Bayesian Information Criterion (BIC). For example, to estimate all the FARIMA models with $0 \leq p \leq 2$ and $0 \leq q \leq 2$, use the optional arguments `p.range` and `q.range` as follows:

```
> ndx.bic = FARIMA(log(ndx.vol), p.range=c(0,2),
+ q.range=c(0,2), mmax=0)
p = 0  q = 0
p = 0  q = 1
p = 0  q = 2
p = 1  q = 0
p = 1  q = 1
p = 1  q = 2
p = 2  q = 0
p = 2  q = 1
p = 2  q = 2
```

In the above example, set `mmax=0` to restrict m to be zero because the previous FARIMA$(0, d, 0)$ model fit suggests that the data may be stationary. A summary of the fitted model is

```
> summary(ndx.bic)

Call:
FARIMA(x = log(ndx.vol), p.range = c(0, 2), q.range = c(0, 2),
       mmax = 0)

Coefficients:
        Value Std. Error t value Pr(>|t|)
    d  0.4504  0.0287    15.6716  0.0000
MA(1)  0.2001  0.0359     5.5687  0.0000

Information Criteria:
 log-likelihood        BIC
  -717.9342      1450.4325

Residual scale estimate: 0.3963

                   total residual
Degree of freedom:  1454     1451
Time period: from 01/05/1996 to 10/12/2001
```

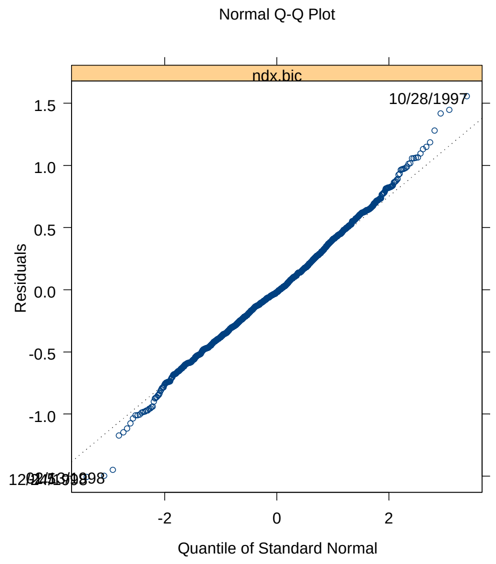


FIGURE 8.6. FARIMA residual QQ-plot of `log(ndx.vol)`.

```
BIC of all models estimated:
          q=0      q=1      q=2
p=0 1466.898 1450.432 1451.055
p=1 1457.319 1462.694 1455.590
p=2 1464.800 1457.243 1464.238
```

The BIC values for all the models considered are shown in the output. The model minimizing the BIC is a FARIMA$(0, d, 1)$ model. The estimates of d and the moving average coefficient are very significant, but the 95% Wald-type confidence interval of d includes $1/2$ and thus the non-stationary case.[5]

Further diagnostics of the model fit can be obtained by using the `plot` method:

```
> plot(ndx.bic)

Make a plot selection (or 0 to exit):

1: plot: all
2: plot: response vs fitted values
```

[5]Currently the standard error of the mean parameter is not available because `arima.fracdiff` concentrates out the mean and thus does not compute its standard error.

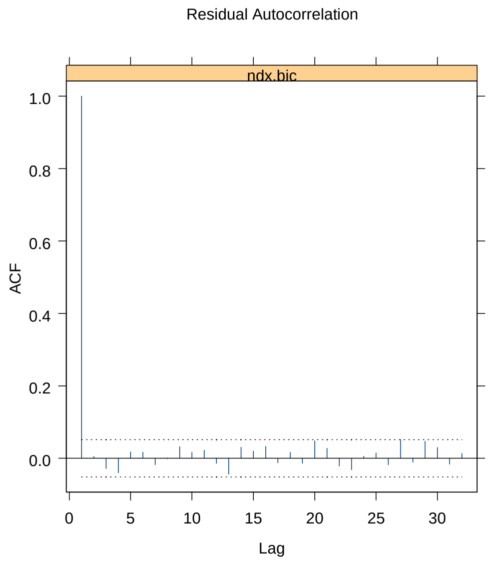


FIGURE 8.7. FARIMA residual ACF of `log(ndx.vol)`.

```
3: plot: response and fitted values
4: plot: normal QQ-plot of residuals
5: plot: residuals
6: plot: standardized residuals
7: plot: residual histogram
8: plot: residual ACF
9: plot: residual PACF
10: plot: residual^2 ACF
11: plot: residual^2 PACF
Selection:
```

For example, if 4 is chosen at the prompt the normal qq-plot of the model residuals ϵ_t will be shown as in Figure 8.6. It seems that the normality assumption agrees well with the data. If `8` is chosen at the prompt, the ACF of model residuals will be shown as in Figure 8.7, and the FARIMA model is very successful at capturing the long memory in logarithmic volatility.

In the above example, m can also be allowed to be estimated:

```
> ndx.bic2 = FARIMA(log(ndx.vol),p.range=c(0,2),
+ q.range=c(0,2), mmax=1)
p = 0  q = 0
...
p = 2  q = 2
```

```
> ndx.bic2$m
[1] 1

> summary(ndx.bic2)

Call:
FARIMA(x = log(ndx.vol), p.range = c(0, 2), q.range =
c(0, 2), mmax = 1)

Coefficients:
        Value Std. Error t value Pr(>|t|)
    d  0.5161  0.1056     4.8864  0.0000
AR(1)  1.1387  0.3753     3.0340  0.0025
AR(2) -0.1561  0.3724    -0.4193  0.6751
MA(1)  1.4364  0.4416     3.2528  0.0012
MA(2) -0.4309  0.7574    -0.5689  0.5695

Information Criteria:
 log-likelihood     BIC
  -696.3         1429.0

Residual scale estimate: 0.3903

                   total residual
Degree of freedom:  1453     1447
Time period: from 01/08/1996 to 10/12/2001

BIC of all models estimated:
     q=0  q=1  q=2
p=0 1467 1450 1451
p=1 1457 1459 1456
p=2 1456 1454 1429
```

Here the best fitting model is a FARIMA$(2, 0.52, 2)$ model. However, the values of the AR and MA coefficients indicate an explosive model. The problem appears to be near canceling roots in the AR and MA polynomials. If the model is re-fitted with $p = q = 1$, the results make more sense:

```
> ndx.bic2 = FARIMA(log(ndx.vol), p=1, q=1, mmax=1)
> summary(ndx.bic2)

Call:
FARIMA(x = log(ndx.vol), p = 1, q = 1, mmax = 1)

Coefficients:
        Value Std. Error t value Pr(>|t|)
```

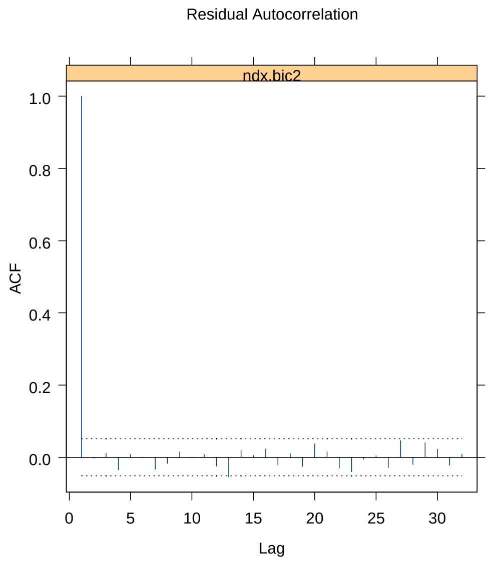


FIGURE 8.8. Residual ACF from FARIMA$(1, 0.51, 1)$ model fit to `log(ndx.vol)`.

```
    d  0.5051  0.0436    11.5965  0.0000
AR(1)  0.2376  0.0687     3.4597  0.0006
MA(1)  0.4946  0.0367    13.4894  0.0000

Information Criteria:
 log-likelihood     BIC
  -712.7         1447.3

Residual scale estimate: 0.3948

                   total residual
Degree of freedom:  1454     1450
Time period: from 01/05/1996 to 10/12/2001
```

Figure 8.8 gives the residual ACF from the above model. The long memory behavior has been well captured by the model. However, the fitted model has the undesirable property of being non-stationary.

8.5.2 SEMIFAR Model

The previous subsection demonstrated that for logarithmic volatility of NASDAQ-100 index returns, the FARIMA model chosen by BIC suggests that the underlying series may be non-stationary. In addition, from the time series plot in Figure 8.5, the volatility has become much larger since the middle of 2000. To allow for a possible deterministic trend in a time series, in addition to a stochastic trend, long memory and short memory components, Beran, Feng and Ocker (1998), Beran and Ocker (1999), and Beran and Ocker (2001) proposed the **semi**parametric **f**ractional **a**uto**r**egressive (SEMIFAR) model. The SEMIFAR model is based on the following extension to the FARIMA$(p, d, 0)$ model (8.12):

$$\phi(L)(1-L)^{\delta}[(1-L)^{m}y_t - g(i_t)] = \epsilon_t \tag{8.14}$$

for $t = 1, \cdots, T$. The above equation is very similar to (8.13), except that the constant term μ is now replaced by $g(i_t)$, a smooth trend function on $[0, 1]$, with $i_t = t/T$. By using a nonparametric kernel estimate of $g(i_t)$, the S+FinMetrics function SEMIFAR implements a procedure to estimate the SEMIFAR model, and it uses BIC to choose the short memory autoregressive order p. Refer to Beran, Feng and Ocker (1998) for a detailed description of the algorithm.

Example 47 *Estimation of SEMIFAR model for NASDAQ-100 index volatility*

To obtain a SEMIFAR model of logarithmic volatility of NASDAQ-100 index returns, use the following command:

```
> ndx.semi = SEMIFAR(log(ndx.vol), p.range=c(0,2), trace=F)
> class(ndx.semi)
[1] "SEMIFAR"
```

Note that the optional argument `trace=F` is used to suppress the messages printed by the procedure. The result of `SEMIFAR` is an object of class "`SEMIFAR`" for which there are `print`, `summary`, `plot` and `predict` methods as well as extractor functions `coef`, `residuals` and `vcov`. The components of `ndx.semi` are

```
> names(ndx.semi)
 [1] "model"     "m"         "delta"     "BIC"
 [5] "loglike"   "trend"     "g.CI"      "bandwidth"
 [9] "Cf"        "nu"        "residuals" "cov"
[13] "CI"        "call"
```

The basic fit is

```
> ndx.semi

Call:
SEMIFAR(x = log(ndx.vol), p.range = c(0, 2), trace = F)

Difference:
0: estimates based on original series.
```

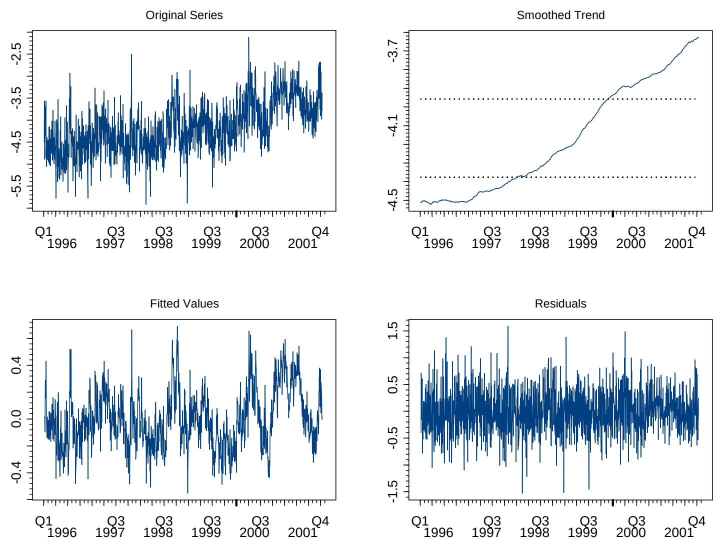


FIGURE 8.9. SEMIFAR decomposition of `log(ndx.vol)`.

```
FAR coefficients:
       d
 0.2928

Residual scale estimate: 0.3946

                   total residual
Degree of freedom:  1453     1452
Time period: from 01/08/1996 to 10/12/2001
```

From the above output, after accounting for a smooth nonparametric trend component $g(i_t)$, the logarithmic volatility appears to be stationary and has long memory.

The estimated trend component can be visualized by calling the `plot` method of fitted model object:

```
> plot(ndx.semi)

Make a plot selection (or 0 to exit):

1: plot: all
2: plot: trend, fitted values, and residuals
3: plot: normal QQ-plot of residuals
```

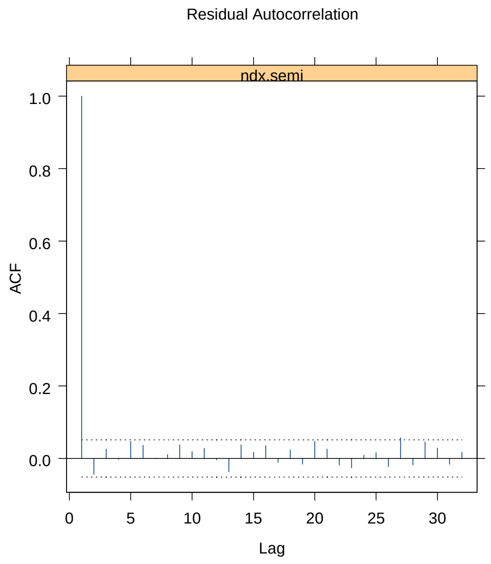


FIGURE 8.10. SEMIFAR residual ACF of `log(ndx.vol)`.

```
4: plot: standardized residuals
5: plot: residual histogram
6: plot: residual ACF
7: plot: residual PACF
8: plot: residual^2 ACF
9: plot: residual^2 PACF
Selection:
```

If `2` is selected at the prompt, a plot as in Figure 8.9 will be shown, which indicates the original time series, the estimated smooth trend component, the fitted values and model residuals. The smooth trend component is also plotted with a confidence band. If the trend falls outside the confidence band, it indicates that the trend component is significant. In this case, the trend in logarithmic volatility appears to be very significant, at least for the time period investigated. The model fit can also be checked by choosing `6` at the prompt, which will generate the ACF plot of residuals, as shown in Figure 8.10. Again, the SEMIFAR model seems to be very successful at modeling the long memory in the original time series.

Prediction from SEMIFAR models will be discussed in section 8.7.

8.6 Long Memory GARCH Models

8.6.1 FIGARCH and FIEGARCH Models

The previous section showed that the FARIMA or SEMIFAR model can be used directly to model the long memory behavior observed in the volatility of financial asset returns, given that a time series representing the volatility exists. However, sometimes a reliable estimate of volatility may be hard to obtain, or the user may want to model the dynamics of the asset returns together with its volatility. In those situations, the GARCH class models provide viable alternatives for volatility modeling.

Section 7.5.2 of Chapter 7 has illustrated that two components GARCH models can be used to capture the high persistence in volatility by allowing a highly persistent long run component and a short run transitory component in volatility. This subsection shows how GARCH models can be extended to allow directly for long memory and high persistence in volatility.

FIGARCH Model

Section 22.16 of Chapter 7 shows that a basic GARCH$(1,1)$ model can be written as an ARMA$(1,1)$ model in terms of squared residuals. In the same spirit, for the GARCH(p,q) model:

$$\sigma_t^2 = a + \sum_{i=1}^{p} a_i \epsilon_{t-i}^2 + \sum_{j=1}^{q} b_j \sigma_{t-j}^2$$

easily shows that it can be rewritten as follows:

$$\phi(L)\epsilon_t^2 = a + b(L)u_t \tag{8.15}$$

where

$$\begin{aligned} u_t &= \epsilon_t^2 - \sigma_t^2 \\ \phi(L) &= 1 - \phi_1 L - \phi_2 L^2 - \cdots - \phi_m L^m \\ b(L) &= 1 - b_1 L - b_2 L^2 - \cdots - b_q L^q \end{aligned}$$

with $m = \max(p,q)$ and $\phi_i = a_i + b_i$. Obviously equation (8.15) represents an ARMA(m,q) process in terms of squared residuals ϵ_t^2 with u_t being a MDS disturbance term.

The high persistence in GARCH models suggests that the polynomial $\phi(z) = 0$ may have a unit root, in which case the GARCH model becomes the integrated GARCH (IGARCH) model. See Nelson (1990) for which the unconditional variance does not exist. To allow for high persistence and long memory in the conditional variance while avoiding the complications of IGARCH models, extend the ARMA(m,q) process in (8.15) to a

FARIMA(m, d, q) process as follows:

$$\phi(L)(1-L)^d\epsilon_t^2 = a + b(L)u_t \tag{8.16}$$

where all the roots of $\phi(z) = 0$ and $b(z) = 0$ lie outside the unit circle. When $d = 0$, this reduces to the usual GARCH model; when $d = 1$, this becomes the IGARCH model; when $0 < d < 1$, the *fractionally differenced squared residuals*, $(1-L)^d\epsilon_t^2$, follow a stationary ARMA(m, q) process. The above FARIMA process for ε_t^2 can be rewritten in terms of the conditional variance σ_t^2:

$$b(L)\sigma_t^2 = a + [b(L) - \phi(L)(1-L)^d]\epsilon_t^2. \tag{8.17}$$

Baillie, Bollerslev and Mikkelsen (1996) refered to the above model as the fractionally integrated GARCH, or FIGARCH(m, d, q) model. When $0 < d < 1$, the coefficients in $\phi(L)$ and $b(L)$ capture the short run dynamics of volatility, while the fractional difference parameter d models the long run characteristics of volatility.

FIEGARCH

The FIGARCH model directly extends the ARMA representation of squared residuals, which results from the GARCH model, to a fractionally integrated model. However, to guarantee that a general FIGARCH model is stationary and the conditional variance σ_t^2 is always positive, usually complicated and intractable restrictions have to be imposed on the model coefficients. For example, see Baillie, Bollerslev and Mikkelsen (1996) or Bollerslev and Mikkelsen (1996) for a discussion.

Noting that an EGARCH model can be represented as an ARMA process in terms of the logarithm of conditional variance and thus always guarantees that the conditional variance is positive, Bollerslev and Mikkelsen (1996) proposed the following fractionally integrated EGARCH (FIEGARCH) model:

$$\phi(L)(1-L)^d \ln\sigma_t^2 = a + \sum_{j=1}^{q}(b_j|x_{t-j}| + \gamma_j x_{t-j}) \tag{8.18}$$

where $\phi(L)$ is defined as earlier for the FIGARCH model, $\gamma_j \neq 0$ allows the existence of leverage effects, and x_t is the standardized residual:

$$x_t = \frac{\epsilon_t}{\sigma_t} \tag{8.19}$$

Bollerslev and Mikkelsen (1996) showed that the FIEGARCH model is stationary if $0 < d < 1$.

8.6.2 Estimation of Long Memory GARCH Models

Given the iterative formulations of conditional variance as in (8.17) and (8.18), the FIGARCH and FIEGARCH model coefficients can be estimated using maximum likelihood estimation (MLE), if the residuals follow a conditional normal distribution. The S+FinMetrics function fgarch can be used to estimate the long memory FIGARCH or FIEGARCH model.

The syntax of fgarch is very similar to that of the garch function, except that ~figarch(m,q) is used as the FIGARCH conditional variance formula and ~fiegarch(m,q) as the FIEGARCH conditional variance formula. For example, to fit a FIGARCH$(1, d, 1)$ model to daily stock returns of Dell Computer contained in the S+FinMetrics "timeSeries" object dell.s, simply use the following command:

```
> dell.figarch = fgarch(dell.s~1, ~figarch(1,1))
Initializing model parameters.
Iteration No.   1: log-likelihood=-3282.303431
...
Iteration No.  10: log-likelihood=-3279.508705
Convergence in gradient.
> oldClass(dell.figarch)
[1] "fgarch" "garch"
```

The returned object is of class "fgarch", which inherits the "garch" class. Consequently, most of the method functions for a "garch" object (e.g. print, summary, plot, predict, coef, residuals, sigma.t, vcov)also work for a "fgarch" object. One exception is that currently there is no simulate method for "fgarch" objects. For example, the print method gives

```
> dell.figarch

Call:
fgarch(formula.mean = dell.s~1, formula.var = ~figarch(1, 1))

Mean Equation: dell.s ~1

Conditional Variance Equation:  ~figarch(1, 1)

Coefficients:

       C 0.4422
       A 0.6488
GARCH(1) 0.6316
 ARCH(1) 0.4481
fraction 0.2946
```

The estimate of d is 0.295, which indicates the existence of long memory. However, the sum ARCH(1) and GARCH(1) is greater than one which indicates a nonstationary model.

If the FIEGARCH model instead of FIGARCH model is desired, the optional argument `leverage` can be used to allow for leverage effects. For example,

```
> dell.fiegarch = fgarch(dell.s~1, ~fiegarch(1,1), leverage=T)
Initializing model parameters.
Iteration No.   1: log-likelihood=-3286.169656
...
Iteration No.  20: log-likelihood=-3274.244677
Convergence in gradient.

> summary(dell.fiegarch)

Call:
fgarch(formula.mean = dell.s~1, formula.var = ~fiegarch( 1, 1),
leverage = T)

Mean Equation: dell.s ~1

Conditional Variance Equation:  ~fiegarch(1, 1)

--------------------------------------------------------------

Estimated Coefficients:
--------------------------------------------------------------
            Value Std.Error t value   Pr(>|t|)
       C  0.39494   0.08981   4.397 5.946e-006
       A -0.06895   0.04237  -1.627 5.195e-002
GARCH(1)  0.65118   0.17820   3.654 1.343e-004
 ARCH(1)  0.15431   0.04578   3.370 3.867e-004
  LEV(1) -0.09436   0.02691  -3.507 2.346e-004
fraction  0.34737   0.11408   3.045 1.188e-003

--------------------------------------------------------------

AIC(6) = 6560.5
BIC(6) = 6591.3

Normality Test:
--------------------------------------------------------------
 Jarque-Bera  P-value Shapiro-Wilk P-value
       13.22 0.001348       0.9888  0.7888

Ljung-Box test for standardized residuals:
--------------------------------------------------------------
```

```
 Statistic P-value Chi^2-d.f.
     13.13  0.3597         12

Ljung-Box test for squared standardized residuals:
--------------------------------------------------------------
 Statistic P-value Chi^2-d.f.
     14.51  0.2696         12

Lagrange multiplier test:
--------------------------------------------------------------
  Lag 1  Lag 2  Lag 3  Lag 4  Lag 5  Lag 6    Lag 7  Lag 8
 -0.925 0.6083 -1.581 0.2593 0.3943 0.6991 -0.03191 0.3339

 Lag 9  Lag 10 Lag 11 Lag 12      C
 1.959 -0.8794  2.422 0.1089 0.8896

 TR^2 P-value F-stat P-value
 15.1  0.2362  1.389  0.2797
```

In the above output, `C` corresponds to the constant term in the conditional mean equation, `A` corresponds to the constant term a, `GARCH(1)` corresponds to b_1, `ARCH(1)` corresponds to ϕ_1, `LEV(1)` corresponds to γ_1 and `fraction` corresponds to the fractional difference parameter d in the conditional variance equation (8.18). Notice that the leverage term is negative and significant, and the sum of ARCH(1) and GARCH(1) is now less than one. It appears that the FIEGARCH model fits the data better than the FIGARCH model.

Just like for "`garch`" objects, the generic `plot` function can be used visually to diagnose the model fit. Use `compare.mgarch` to compare multiple model fits. For example, consider comparing the above two FIGARCH, FIEGARCH with short memory GARCH and EGARCH models:

```
> dell.garch = garch(dell.s~1, ~garch(1,1), trace=F)
> dell.egarch = garch(dell.s~1, ~egarch(1,1),
+ leverage=T, trace=F)
> dell.comp = compare.mgarch(dell.garch,dell.egarch,
+ dell.figarch,dell.fiegarch)
> dell.comp
           dell.garch dell.egarch dell.figarch dell.fiegarch
       AIC       6564        6559         6569          6560
       BIC       6585        6584         6595          6591
Likelihood      -3278       -3274        -3280         -3274
```

Here, the EGARCH and FIEGARCH models seem to provide better fits than the GARCH and FIGARCH models. The qq-plots of standardized residuals for the four models can be compared using:

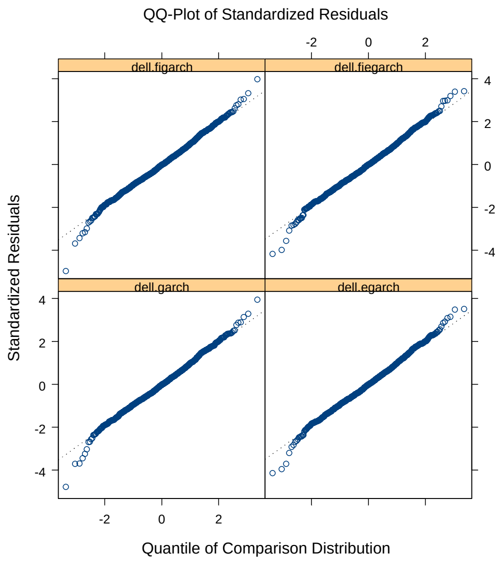


FIGURE 8.11. qq-plot of standardized residuals from long memory GARCH models.

```
> plot(dell.comp, qq=T)
```

and the plot is shown in Figure 8.11, where the FIEGARCH model seems to provide a slightly better fit to the outliers in both tails.

8.6.3 Custom Estimation of Long Memory GARCH Models

ARMA Terms and Exogenous Variables

Just like with the `garch` function, the `fgarch` function also allows ARMA terms and exogenous variables in the conditional mean equation, as well as the conditional variance equation.

Example 48 *Trading volume and volatility (extended)*

The previous subsection shows that the fitted FIEGARCH model object `dell.fiegarch` suggests that there may be long memory in the volatility of Dell stocks. In Section 7.5 of Chapter 7, the changes in trading volume were used to explain the volatility of Dell stocks. If there is a 1% change in trading volume, it will cause about 1.4% change in conditional variance using an EGARCH model for volatility. In this example, the analysis using the FIEGARCH model instead of the EGARCH model is done again.

```
> dell.mod2 = fgarch(dell.s~1, ~fiegarch(1,1) +
+ seriesData(d.volume), series.start=2)
> summary(dell.mod2)

Call:
fgarch(formula.mean = dell.s ~1, formula.var = ~fiegarch(1,1)
       + seriesData(d.volume), series.start = 2)

Mean Equation: dell.s ~ 1

Conditional Variance Equation:  ~ fiegarch(1, 1) +
       seriesData(d.volume)

--------------------------------------------------------------

Estimated Coefficients:
--------------------------------------------------------------
                        Value Std.Error  t value   Pr(>|t|)
                   C  0.14514   0.06245   2.3242 1.014e-002
                   A -0.13640   0.03117  -4.3761 6.542e-006
            GARCH(1)  0.04123   0.10703   0.3852 3.501e-001
             ARCH(1)  0.16600   0.03809   4.3583 7.091e-006
seriesData(d.volume)  1.49123   0.07814  19.0849 0.000e+000
            fraction  0.80947   0.07523  10.7596 0.000e+000

...
```

First, compare the above output with `dell.fiegarch`, the FIEGARCH model fitted in the previous subsection. After controlling for the effects of trading volume, the GARCH coefficient has decreased significantly and become insignificant, while the fractional difference parameter has increased from 0.34 to 0.8. Second, compare this with the EGARCH model `dell.mod` in Chapter 7: after allowing for long memory, the GARCH coefficient decreased from 0.95 to 0.04, while the effects of trading volume remain almost the same.

Control of Model Estimation

For a "`fgarch`" object, all the model specific information is contained in the `model` component of the object. For example, view the model information of the fitted `dell.figarch` object as follows:

```
> dell.figarch$model

Mean Equation: dell.s ~ 1

Conditional Variance Equation:  ~ figarch(1, 1)
```

```
                 Values
constant in mean 0.4422
 constant in var 0.6488
        GARCH(1) 0.6316
         ARCH(1) 0.4481
        fraction 0.2946
```

This model object can be edited to provide starting values for re-estimating the same model with the same or a different time series.[6] For example, to use this set of values as starting values for a FIGARCH model of the time series `hp.s`, use the following command:

```
> hp.figarch = fgarch(series=hp.s*100,model=dell.figarch$model)
Iteration No.   1: log-likelihood=-4419.644144
...
Iteration No.  10: log-likelihood=-4390.179116
Convergence in gradient.
> hp.figarch

Call:
fgarch(series = hp.s * 100, model = dell.figarch$model)

Mean Equation: dell.s ~ 1

Conditional Variance Equation:  ~ figarch(1, 1)

Coefficients:

       C 0.05776
       A 0.55897
GARCH(1) 0.49103
 ARCH(1) 0.40210
fraction 0.22533
```

Unlike the `garch` and `mgarch` functions which use the BHHH algorithm for MLE, the FIGARCH/FIEGARCH models are estimated using the BFGS algorithm (for example, see Press, Teukolsky, Vetterling, and Flannery, 1992 for details). Since daily financial returns are very small numbers, the algorithm can become badly scaled and may fail to converge. That is why in the above example the percentage returns are used to improve the convergence.

[6]However, unlike "`garch`" and "`mgarch`" objects, currently the coefficients cannot be fixed at certain values during the estimation of long memory GARCH models. See Section 13.7 in Chapter 13 for discussions related to "`garch`" and "`mgarch`" objects.

Other aspects of the BFGS algorithm can be controlled by passing the optional argument `control` to the `fgarch` function, where `control` must be set to an object returned by the `fgarch.control` function. For example, to change the convergence tolerance of gradient zeroing from the default value of `1e-5` to `1e-6` when fitting a FIGARCH model to `dell.s`, use the following command:

```
> fgarch(dell.s~1, ~figarch(1,1), control=
+ fgarch.control(tolg=1e-6))
```

The on-line help file for `fgarch.control` provides more details for the arguments accepted by the `fgarch.control` function.

Finally, introducing the FIGARCH/FIEGARCH models illustrated that both models are essentially an ARMA model fitted to the fractionally differenced squared residuals or fractionally differenced logarithmic conditional variance. The fractional difference operator is defined in (8.5), which involves an infinite order autoregressive filter. In practice, a very large number is usually chosen to approximate the fractional difference operator. Following Bollerslev and Mikkelsen (1996), the `fgarch` function sets the order to be 1000 by default. To change this number to another value, pass the optional argument `lag` to `fgarch.control`. For example, the command

```
> fgarch(dell.s~1, ~figarch(1,1), control=
+ fgarch.control(lag=500))
```

estimates a FIGARCH model using only 500 lags to approximate the fractional difference operator.

8.7 Prediction from Long Memory Models

`S+FinMetrics` long memory modeling functions such as `FARIMA`, `SEMIFAR` and `fgarch` all return objects for which there are corresponding `predict` methods. Therefore, predictions from those fitted model objects can be readily generated. This section gives an overview of how to predict from a long memory process. In particular, the truncation method and the best linear predictor will be introduced, see Bhansali and Kokoszka (2001). How to predict from fitted model objects in `S+FinMetrics` module will be illustrated.

8.7.1 Prediction from FARIMA/SEMIFAR Models

To illustrate prediction from long memory processes, consider the FARIMA model in (8.12), which can be rewritten as:

$$\frac{\phi(L)(1-L)^d}{\theta(L)}(y_t - \mu) = \epsilon_t$$

The lag polynomial on the left hand side of the above equation can be expressed as an infinite order polynomial so that a FARIMA(p, d, q) model can be equivalently expressed as an AR(∞) model. Once the parameters of the FARIMA(p, d, q) model are known, one can solve for the parameters of the equivalent AR(∞) model. In practice, however, forecasting from the AR(∞) representation usually truncates the AR(∞) model to an AR(p) model with a very large value of p. This method is usually referred to as the *truncation method*.

In the truncation method, the AR(p) coefficients are the first p coefficients of the AR(∞) representation of the FARIMA(p, d, q) model. However, for any stationary process, choose to use p lagged values to predict future values:

$$\hat{y}_{T+1} = \psi_1 y_T + \cdots + \psi_p y_{T-p+1}$$

where ψ_i for $i = 1, \cdots, p$ are chosen to yield the best linear predictor of y_{T+1} in terms of $y_T, \cdots, y_{T-p+1}$ for any T. Note that although both the above method and the truncation method use an AR(p) model for prediction, the AR(p) coefficients in the truncation method do not necessarily correspond to best linear prediction coefficients ψ_i. Brockwell and Davis (1991) showed that the best linear prediction coefficients can be recursively computed using the Durbin-Levinson algorithm given the autocovariance function of the stationary process.[7]

The `predict` method for "`FARIMA`" objects in `S+FinMetrics` implements the Durbin-Levinson algorithm to compute the forecasts. The arguments taken by `predict.FARIMA` are:

```
> args(predict.FARIMA)
function(x, n.predict = 1, ar.approx = 50, kapprox = 100000,
         series = NULL)
```

where `n.predict` indicates the number of steps to predict ahead, `ar.approx` gives the order p of the AR representation used for prediction, `kapprox` is passed to `acf.FARIMA` to obtain the theoretical autocovariance function of the FARIMA model, and `series` can be used to pass the original time series used to fit the model. For example, to predict 100 steps ahead using an AR(100) representation from the fitted model object `ndx.bic`, use the following command:

```
> ndx.pred1 = predict(ndx.bic, n.predict=100, ar.approx=100)
> class(ndx.pred1)
```

[7]Although exact expressions of the autocovariance functions for FARIMA(p, d, q) models have been given by Sowell (1992), the derivation assumes that all the roots of the AR polynomial are distinct. The `S+FinMetrics` function `acf.FARIMA` implements a numerical quadrature procedure based on fast Fourier transform to approximate the autocovariance function of the FARIMA models, as proposed by Bhansali and Kokoszka (2001).

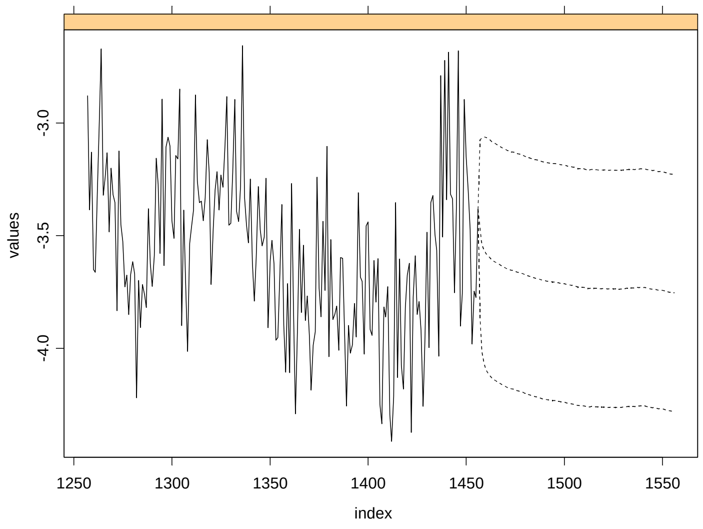


FIGURE 8.12. Predictions from a FARIMA model.

```
[1] "forecast"
```

The returned object has class "`forecast`" and has components

```
> names(ndx.pred1)
[1] "values"  "std.err" "coef"
```

where the `values` contains the predicted values, `std.err` contains the standard errors of the predictions, and `coef` contains the best linear prediction coefficients ψ_i $(i = 1, \ldots, p)$. The predictions and standard errors can be seen by calling the `summary` function on a "`forecast`" object. For example:

```
> summary(ndx.pred1)

Predicted Values with Standard Errors:

              prediction std.err
 1-step-ahead -3.4713     0.3965
 2-step-ahead -3.5407     0.4732
 3-step-ahead -3.5638     0.5023
 4-step-ahead -3.5792     0.5148
 5-step-ahead -3.5883     0.5204
...
```

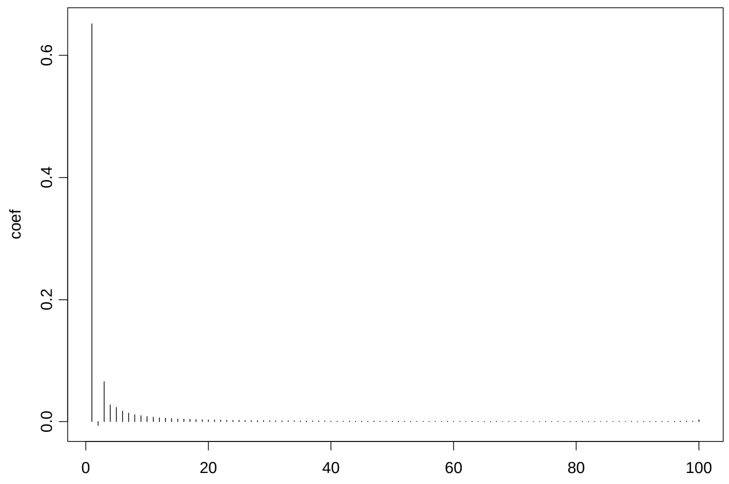


FIGURE 8.13. Best linear prediction coefficients.

A "`forecast`" object can be plotted together with the original data to visualize the predictions. For example, since `ndx.bic` was fitted using `log(ndx.vol)`, the predictions can be visualized as follows:

```
> plot(ndx.pred1, log(ndx.vol), n.old=200)
```

where the optional argument `n.old` specifies the number of observations in the original data to be used in the plot. The plot is shown Figure 8.12. Also, the best linear prediction coefficients can also be visualized to see the effects of using more lags for prediction. For example:

```
> plot(ndx.pred1$coef, type="h", ylab="coef")
```

generates the coefficient plot shown in Figure 8.13. Adding lags beyond 30 should not change the predictions very much.

In `S+FinMetrics`, predictions from SEMIFAR models are computed in a similar fashion to predictions from FARIMA models, except that there is a choice to use a constant extrapolation or linear extrapolation for the trend component[8]

```
> args(predict.SEMIFAR)
function(x, n.predict = 1, ar.approx = 50, kapprox = 100000,
```

[8]We refer to Beran and Ocker (1999) for the details of predicting from a SEMIFAR model.

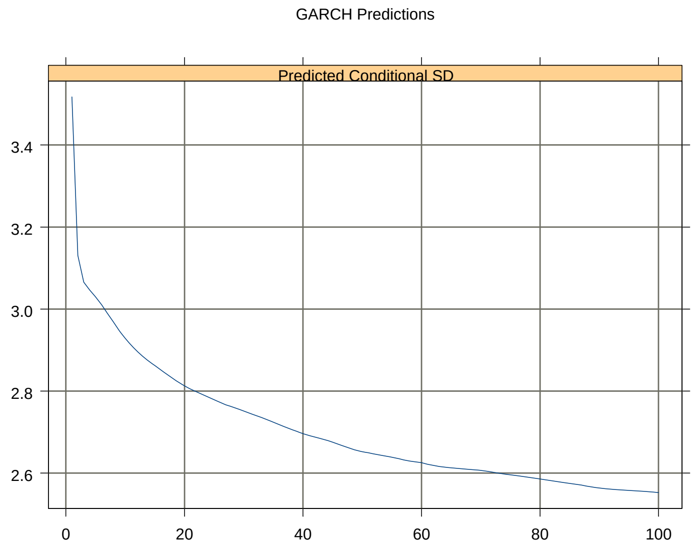


FIGURE 8.14. Predictions from FIGARCH model.

```
trend = "constant", series = NULL)
```

For example, to produce 100 steps ahead forecasts from the fitted model object `ndx.semi` using constant extrapolation, use the following command:

```
> ndx.pred2 = predict(ndx.semi,n.predict=100,trend="constant")
```

The returned object is also a “`forecast`” object, so the predictions can be visualized together with the original data

```
> plot(ndx.pred2, ndx.vol, n.old=200)
```

8.7.2 Prediction from FIGARCH/FIEGARCH Models

Predictions from the `S+FinMetrics` long memory GARCH models are computed using the truncation method because the user needs to generate forecasts for both the level and the volatility of the series at the same time. The arguments taken by the `predict` method are:

```
> args(predict.fgarch)
function(object, n.predict = 1, n.lag = 1000)
NULL
```

where `n.predict` specifies the number of periods to predict ahead, and `n.lag` specifies the order p of the AR(p) representation used in the truncation method. For example, to use an AR(100) representation to predict 100 steps ahead from the fitted model object `dell.figarch`, use the following command:

```
> dell.pred3 = predict(dell.figarch, n.predict=100, n.lag=100)
> oldClass(dell.pred3)
[1] "predict.fgarch" "predict.garch"
```

The returned object is of class "`predict.fgarch`", which inherits from the class "`predict.garch`". So just like for a "`predict.garch`" object, use the generic `plot` function to visualize the volatility forecast:

```
> plot(dell.pred3, hgrid=T, vgrid=T)
```

and the plot is shown in Figure 8.14. The volatility predictions approach the long run level in a slowly decaying fashion for the long memory GARCH model[9].

8.8 References

Andersen, T., Bollerslev, T., Diebold, F. X., and Labys, P. (1999): "(Understanding, Optimizing, Using and Forecasting) Realized Volatility and Correlation," Manuscript, Northwestern University, Duke University and University of Pennsylvania.

Andersen, T., Bollerslev, T., Diebold, F. X., and Labys, P. (2001a): "The Distribution of Realized Exchange Rate Volatility," *Journal of the American Statistical Association*, 96, 42-55.

Andersen, T., Bollerslev, T., Diebold, F. X., and Labys, P. (2001b): "The Distribution of Realized Stock Return Volatility," *Journal of Financial Economics*, 61, 43-76.

Baillie, R. T. (1996). "Long Memory Processes and Fractional Integration in Econometrics," *Journal of Econometrics*, 73, 5-59.

Baillie, R. T., Bollerslev, T., and Mikkelsen, H. O. (1996). "Fractionally Integrated Generalized Autoregressive Conditional Heteroskedasticity," *Journal of Econometrics*, 74, 3-30.

Beran, J. (1994). *Statistics for Long Memory Processes*, Chapman and Hall, New York.

Beran, J. (1995). "Maximum Likelihood Estimation of the Differencing Parameter for Invertible Short and Long Memory ARIMA Models," *Journal of Royal Statistical Society Series B*, 57(4), 659-672.

[9]Currently, standard errors are not available for the volatility predictions.

Beran, J., Feng, Y., and Ocker, D. (1999). "SEMIFAR Models," Technical Report, 3/1999, SFB 475 University of Dortmund.

Beran, J., and Ocker, D. (1999). "SEMIFAR Forecasts, with Applications to Foreign Exchange Rates," *Journal of Statistical Planning and Inference*, 80, 137-153.

Beran, J., and Ocker, D. (2001). "Volatility of Stock Market Indices - An Analysis Based on SEMIFAR Models," *Journal of Business and Economic Statistics*, 19(1), 103-116.

Bhansali, R. J., and Kokoszka, P. S. (2001). "Computation of the Forecast Coefficients for Multistep Prediction of Long-range Dependent Time Series," *International Journal of Forecasting*, 18(2), 181-206.

Bollerslev, T., and Mikkelsen, H. O. (1996). "Modeling and Pricing Long Memory in Stock Market Volatility," *Journal of Econometrics*, 73, 151-184.

Brockwell, P. J., and Davis, R. A. (1991). *Time Series: Theory and Methods*, Springer-Verlag, New York.

Cheung, Y.W. (1993). "Tests for Fractional Integration: A Monte Carlo Investigation," *Journal of Time Series Analysis*, 14, 331-345.

Garman, M. B., and Klass, M. J. (1980). "On the Estimation of Security Price Volatility from Historical Data," *Journal of Business*, 53, 67-78.

Geweke, J., and Porter-Hudak, S. (1983). "The Estimation and Application of Long Memory Time Series Models," *Journal of Time Series Analysis*, 4, 221-237.

Granger, C. W. J., and Joyeux, R. (1980). "An Introduction to Long-Memory Time Series Models and Fractional Differencing," *Journal of Time Series Analysis*, 1, 15-29.

Hamilton, J. D. (1994). *Time Series Analysis.* Princeton University Press, Princeton, NJ.

Haslett, J., and Raftery, A. E. (1989). "Space-time Modelling with Long-Memory Dependence: Assessing Ireland's Wind Power Resource," *Journal of Royal Statistical Society Series C*, 38, 1-21.

Hosking, J. R. M. (1981). "Fractional Differencing," *Biometrika*, 68, 165-176.

HURST, H. E. (1951). "Long Term Storage Capacity of Reservoirs," *Transactions of the American Society of Civil Engineers*, 116, 770-799.

LO, A. W. (1991). "Long Term Memory in Stock Market Prices," *Econometrica*, 59, 1279-1313.

LOBATO, I. N., AND SAVIN, N. E. (1998). "Real and Spurious Long-Memory Properties of Stock-Market Data," *Journal of Business and Economic Statistics*, 16 (3), 261-268.

MANDELBROT, B. B. (1975). "Limit Theorems on the Self-Normalized Range for Weakly and Strongly Dependent Processes," *Zeitschrift für Wahrscheinlichkeitstheorie und verwandte Gebiete*, 31, 271-285.

PRESS, W. H., TEUKOLSKY, S. A., VETTERLING, W. T., AND FLANNERY, B. P. (1992). *Numerical Recipes in C: The Art of Scientific Computing*, Cambridge University Press, Cambridge.

RAY, B. K., AND TSAY, R. S. (2000). "Long-Range Dependence in Daily Stock Volatilities," *Journal of Business and Economic Statistics*, 18, 254-262.

SOWELL, F. (1992). "Maximum Likelihood Estimation of Stationary Univariate Fractionally Integrated Time Series Models," *Journal of Econometrics*, 53, 165-188.

TAQQU, M. S., AND TEVEROVSKY, V. (1998). "On Estimating the Intensity of Long-Range Dependence in Finite and Infinite Variance Time Series", in R. J. Adler, R. E. Feldman and M. S. Taqqu (eds.), *A Practical Guide to Heavy Tails: Statistical Techniques and Applications.* Birkhaüser, Boston.

TAQQU, M. S., TEVEROVSKY, V., WILLINGER, W. (1995). "Estimators for Long Range Dependence: An Empirical Study," *Fractals*, 3(4), 785-798.

9
Rolling Analysis of Time Series

9.1 Introduction

A rolling analysis of a time series model is often used to assess the model's stability over time. When analyzing financial time series data using a statistical model, a key assumption is that the parameters of the model are constant over time. However, the economic environment often changes considerably, and it may not be reasonable to assume that a model's parameters are constant. A common technique to assess the constancy of a model's parameters is to compute parameter estimates over a rolling window of a fixed size through the sample. If the parameters are truly constant over the entire sample, then the estimates over the rolling windows should not be too different. If the parameters change at some point during the sample, then the rolling estimates should capture this instability.

Rolling analysis is commonly used to backtest a statistical model on historical data to evaluate stability and predictive accuracy. Backtesting generally works in the following way. The historical data is initially split into an estimation sample and a prediction sample. The model is then fit using the estimation sample and h-step ahead predictions are made for the prediction sample. Since the data for which the predictions are made are observed h-step ahead prediction errors can be formed. The estimation sample is then rolled ahead a given increment and the estimation and prediction exercise is repeated until it is not possible to make any more h-step predictions. The statistical properties of the collection of h-step ahead prediction errors are then summarized and used to evaluate the adequacy of the statistical model.

Moving average methods are common in rolling analysis, and these methods lie at the heart of the technical analysis of financial time series. Moving averages typically use either equal weights for the observations or exponentially declining weights. One way to think of these simple moving average models is that they are a "poor man's" time varying parameter model. Sometimes simple moving average models are not adequate, however, and a general time varying parameter model is required. In these cases, the state space models discussed in Chapter 14 should be used.

This chapter describes various types of rolling analysis of financial time series using `S-PLUS`. Section 9.2 covers rolling descriptive statistics for univariate and bivariate time series with an emphasis on moving average techniques, and Section 9.3 gives a brief review of technical analysis using `S+FinMetrics` functions. Section 9.4 discusses rolling regression using the `S+FinMetrics` function `rollOLS` and illustrates how `rollOLS` may be used for backtesting regression models. Section 9.5 describes rolling analysis of general models using the `S+FinMetrics` function `roll`.

Rolling analysis of financial time series is widely used in practice but the technique is seldom discussed in textbook treatments of time series analysis. Notable exceptions are Alexander (2001) and Dacorogna et. al. (2001). Rolling analysis techniques in finance are generally discussed in the technical analysis literature, but the statistical properties of backtesting technical analysis are rarely addressed. A comprehensive treatment of technical analysis indicators is given in Colby and Meyers (1988) and a critical evaluation of technical analysis is provided in Bauer and Dahlquist (1999). The econometric literature on evaluating the predictive accuracy of models through backtesting has matured over the last decade. The main reference is Diebold and Mariano (1995).

9.2 Rolling Descriptive Statistics

9.2.1 Univariate Statistics

Consider the analysis of a univariate time series y_t over a sample from $t = 1, \ldots, T$. Whether the mean and variance (or standard deviation) parameters of the distribution of y_t are constant over the entire sample is of interest. To assess parameter constancy, let n denote the width of a sub-sample or window and define the *rolling* sample means, variances and standard deviations

$$\hat{\mu}_t(n) = \frac{1}{n}\sum_{i=0}^{n-1} y_{t-i} \tag{9.1}$$

$$\hat{\sigma}_t^2(n) = \frac{1}{n-1}\sum_{i=0}^{n-1}(y_{t-i} - \hat{\mu}_t(n))^2 \tag{9.2}$$

$$\hat{\sigma}_t(n) = \sqrt{\hat{\sigma}_t^2(n)} \tag{9.3}$$

for windows $t = n, \ldots T$. The rolling mean and variance estimates at time t with window width n are the usual sample estimates using the most recent n observations. Provided the windows are rolled through the sample one observation at a time, there will be $T - n + 1$ rolling estimates of each parameter. The rolling mean $\hat{\mu}_t(n)$ is sometime called a n-period *simple moving average*.

Computing Rolling Descriptive Statistics Using the S-PLUS Function aggregateSeries

Consider the monthly continuously compounded returns on Microsoft stock over the period February 1990 through January 2001 computed from the monthly closing prices in the S+FinMetrics "timeSeries" singleIndex.dat

```
> msft.ret = getReturns(singleIndex.dat[,"MSFT"])
> start(msft.ret)
[1] Feb 1990
> end(msft.ret)
[1] Jan 2001
> nrow(msft.ret)
[1] 132
```

24-month rolling mean and standard deviations may be computed easily using the S-PLUS function aggregateSeries[1]

```
> roll.mean = aggregateSeries(msft.ret,moving=24,adj=1,FUN=mean)
> roll.sd = aggregateSeries(msft.ret,moving=24,adj=1,FUN=stdev)
```

The arguments moving=24, adj=1 and FUN=mean(stdev) tell the S-PLUS function aggregateSeries to evaluate the mean (stdev) function on a rolling window of size 24 and to adjust the output positions to the end of each window. roll.mean and roll.sd are "timeSeries" objects containing 109 rolling estimates:

```
> class(roll.mean)
```

[1]aggregateSeries is the method function of the generic S-PLUS function aggregate for objects of class "timeSeries" and "signalSeries".

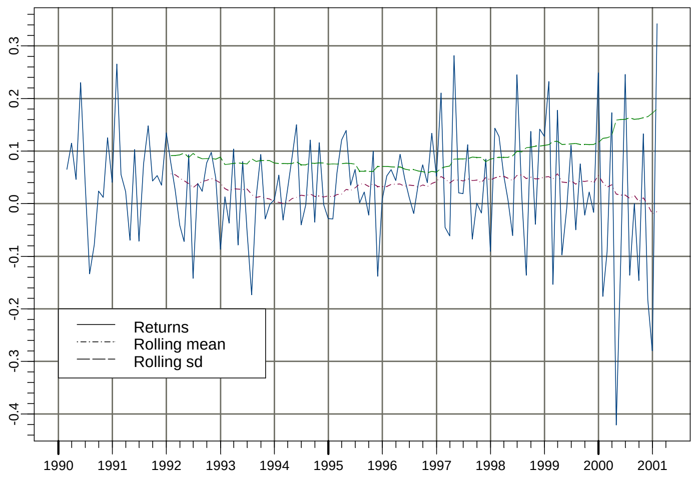


FIGURE 9.1. Monthly returns on Microsoft stock along with 24 month rolling means and standard deviations.

```
[1] "timeSeries"
> nrow(roll.mean)
[1] 109
> roll.mean[1:5]
 Positions       1
 Jan 1992  0.05671
 Feb 1992  0.05509
 Mar 1992  0.04859
 Apr 1992  0.04366
 May 1992  0.03795
```

The monthly returns along with the rolling means and standard deviations may be plotted together using

```
> plot(msft.ret,roll.mean,roll.sd,plot.args=list(lty=c(1,3,4)))
> legend(0,-0.2,legend=c("Returns","Rolling mean","Rolling sd"),
+ lty=c(1,3,4))
```

which is illustrated in Figure 9.1. The 24 month rolling estimates $\hat{\mu}_t(24)$ and $\hat{\sigma}_t(24)$ clearly vary over the sample. The rolling means start out around 5%, fall close to 0% in 1994, rise again to about 5% until 2000 and then fall below 0%. The rolling $\hat{\sigma}_t(24)$ values start out around 10%, fall slightly until 1997 and then begin to steadily rise for the rest of the sample. The end of sample value of $\hat{\sigma}_t(24)$ is almost twice as big as the beginning of sample value.

The moving average estimates (9.1) - (9.3) are one-sided backward looking estimates. The S-PLUS function `aggregateSeries` may also compute two-sided asymmetric moving averages by specifying a value between 0 and 1 for the optional argument `adj`. For example, to compute a 24 month symmetric two-sided simple moving average set `adj=0.5`

```
> roll.mean.5 = aggregateSeries(msft.ret,moving=24,adj=0.5,
+ FUN=mean)
> roll.mean.5[1:5]
 Positions      MSFT
 Feb 1991  0.056708
 Mar 1991  0.055095
 Apr 1991  0.048594
 May 1991  0.043658
 Jun 1991  0.037950
```

Instead of computing the rolling means and standard deviations in separate calls to `aggregateSeries`, they can be computed in a single call by supplying a user-written function to `aggregateSeries` that simply returns the mean and standard deviation. One such function is

```
> mean.sd = function (x) {
>     tmp1 = mean(x)
>     tmp2 = stdev(x)
>     ans = concat(tmp1,tmp2)
>     ans
> }
```

The call to `aggregateSeries` to compute the rolling means and standard deviations is then

```
> roll.mean.sd = aggregateSeries(msft.ret,moving=24,adj=1,
+ FUN=mean.sd,colnames=c("mean","sd"))
> roll.mean.sd[1:5,]
 Positions    mean      sd
 Jan 1992  0.05671 0.09122
 Feb 1992  0.05509 0.09140
 Mar 1992  0.04859 0.09252
 Apr 1992  0.04366 0.09575
 May 1992  0.03795 0.08792
```

Notice that the column names of `roll.mean.sd` are specified using optional argument `colnames=c("mean","sd")` in the call to `aggregateSeries`.

Standard error bands around the rolling estimates of μ and σ may be computed using the asymptotic formulas

$$\widehat{\mathrm{SE}}(\hat{\mu}_t(n)) = \frac{\hat{\sigma}_t(n)}{\sqrt{n}},\ \widehat{\mathrm{SE}}(\hat{\sigma}_t(n)) = \frac{\hat{\sigma}_t(n)}{\sqrt{2n}}$$

Using the rolling estimates in `roll.mean.sd`, the `S-PLUS` commands to compute and plot the rolling estimates along with approximate 95% confidence bands are

```
> lower.mean = roll.mean.sd[,"mean"]-
+ 2*roll.mean.sd[,"sd"]/sqrt(24)
> upper.mean = roll.mean.sd[,"mean"]+
+ 2*roll.mean.sd[,"sd"]/sqrt(24)
> lower.sd = roll.mean.sd[,"sd"]-
+ 2*roll.mean.sd[,"sd"]/sqrt(2*24)
> upper.sd = roll.mean.sd[,"sd"]+
+ 2*roll.mean.sd[,"sd"]/sqrt(2*24)
> par(mfrow=c(2,1))
> plot(roll.mean.sd[,"mean"],lower.mean,upper.mean,
+ main="24 month rolling means",plot.args=list(lty=c(1,2,2)))
> plot(roll.mean.sd[,"sd"],lower.sd,upper.sd,
+ main="24 month rolling standard deviations",
+ plot.args=list(lty=c(1,2,2)))
```

Figure 9.2 shows the results. In general, the rolling $\hat{\sigma}_t(24)$ values are estimated much more precisely than the rolling $\hat{\mu}_t(24)$ values.

The rolling means, variances and standard deviations are not the only rolling descriptive statistics of interest, particularly for asset returns. For risk management purposes, one may be interested in extreme values. Therefore, one may want to compute rolling minima and maxima. These may be computed using `aggregateSeries` with `FUN=min` and `FUN=max`.

Computing Rolling Means, Variances, Maxima and Minima Using the `S+FinMetrics` Functions `SMA`, `rollVar`, `rollMax` and `rollMin`

The `S-PLUS` function `aggregateSeries` is extremely flexible but not efficient for computing rolling means, variances, maxima and minima. The `S+FinMetrics` functions `SMA` (simple moving average), `rollVar`, `rollMax` and `rollMin` implement efficient algorithms for computing rolling means, variances, maxima and minima. The arguments expected by these functions are

```
> args(SMA)
function(x, n = 9, trim = T, na.rm = F)
> args(rollVar)
function(x, n = 9, trim = T, unbiased = T, na.rm = F)
```

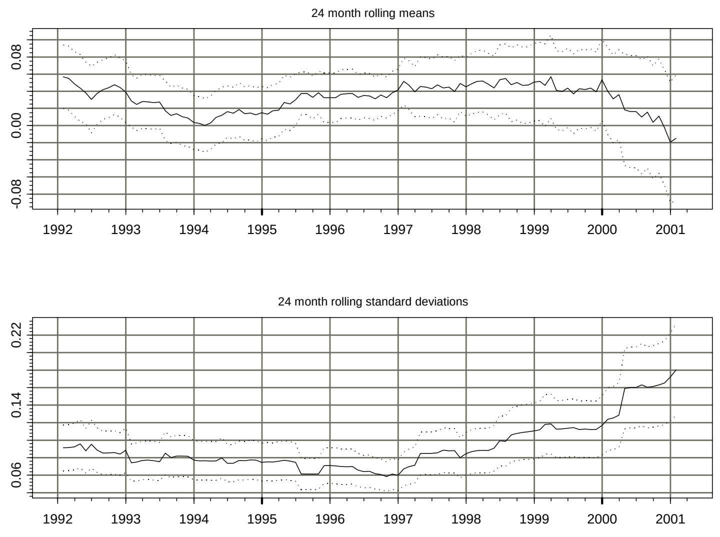


FIGURE 9.2. 24 month rolling estimates of $\hat{\mu}_t(24)$ and $\hat{\sigma}_t(24)$ for Microsoft with 95% confidence bands.

```
> args(rollMax)
function(x, n = 9, trim = T, na.rm = F)
> args(rollMin)
function(x, n = 9, trim = T, na.rm = F)
```

where `x` is a vector or univariate "`timeSeries`", `n` is the window width, `trim` determines if start-up values are trimmed from the output series and `na.rm` determines if missing values are to be removed. For `rollVar`, the option `unbiased=T` computes the unbiased variance estimator using $\frac{1}{n-1}$ as a divisor and `unbiased=F` computes the biased estimator using $\frac{1}{n}$.

To illustrate the use of `SMA`, `rollVar`, `rollMax` and `rollMin` 24 month rolling means, standard deviations, maxima and minima from the monthly returns on Microsoft are computed as

```
> roll2.mean = SMA(msft.ret,n=24)
> roll2.sd = sqrt(rollVar(msft.ret,n=24))
> roll.max = rollMax(msft.ret,n=24)
> roll.min = rollMin(msft.ret,n=24)
```

These estimates are identical to those computed using `aggregateSeries`, but the computation time required is much less. To compare the computation times in seconds within S-PLUS 6 for Windows the S-PLUS function dos.time may be used[2]

```
> dos.time(SMA(msft.ret,n=24))
[1] 0.05
> dos.time(aggregateSeries(msft.ret,moving=24,adj=1,FUN=mean))
[1] 4.23
> dos.time(sqrt(rollVar(msft.ret,n=24)))
[1] 0.06
> dos.time(aggregateSeries(msft.ret,moving=24,adj=1,FUN=stdev))
[1] 6.76
```

Example 49 *Computing rolling standard deviations from high frequency returns*

Rolling estimates of σ^2 and σ based on high frequency continuously compounded return data are often computed assuming the mean return is zero

$$\hat{\sigma}_t^2(n) = \frac{1}{n}\sum_{i=1}^{n} r_{t-i}^2$$

In this case the rolling estimates of σ^2 may be computed using the computationally efficient S+FinMetrics function SMA. For example, consider computing rolling estimates of σ based on the daily continuously compounded returns for Microsoft over the 10 year period from January 1991 through January 2001. The squared return data is computed from the daily closing price data in the S+FinMetrics "timeSeries" object DowJones30

```
> msft.ret2.d = getReturns(DowJones30[,"MSFT"],
+ type="continuous")^2
```

Rolling estimates of σ based on 25, 50 and 100 day windows are computed using SMA

```
> roll.sd.25 = sqrt(SMA(msft.ret2.d,n=25))
> roll.sd.50 = sqrt(SMA(msft.ret2.d,n=50))
> roll.sd.100 = sqrt(SMA(msft.ret2.d,n=100))
```

The rolling estimates $\hat{\sigma}_t(n)$ are illustrated in Figure 9.3 created using

```
> plot(roll.sd.25,roll.sd.50,roll.sd.100,
+ plot.args=(list(lty=c(1,3,4))))
> legend(0,0.055,legend=c("n=25","n=50","n=100"),
+ lty=c(1,3,4))
```

[2]The computations are carried out using S-PLUS 6 Professional Release 2 on a Dell Inspiron 3500 400MHz Pentium II with 96MB RAM.

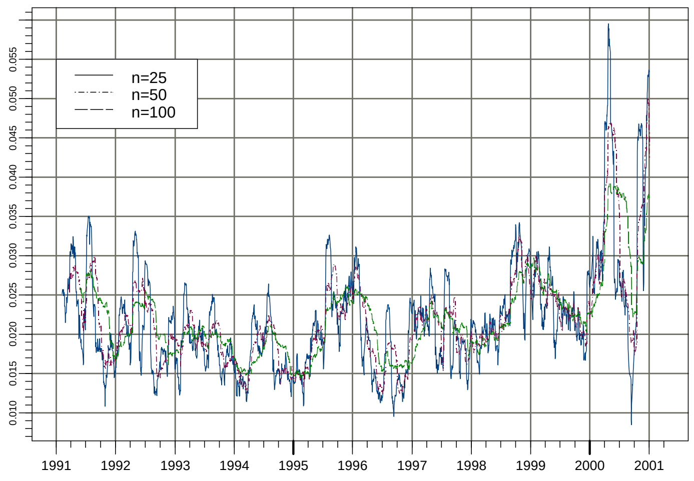


FIGURE 9.3. 25, 50 and 100 day rolling estimates of σ for the daily returns on Microsoft stock.

There is considerable variation in the rolling estimates of daily σ, and there appears to be a seasonal pattern with higher volatility in the summer months and lower volatility in the winter months.

9.2.2 Bivariate Statistics

Consider now the analysis of two univariate time series y_{1t} and y_{2t} over the sample from $t = 1, \ldots, T$. To assess if the covariance and correlation between y_{1t} and y_{2t} is constant over the entire sample the n-period rolling sample covariances and correlations

$$\begin{aligned}\hat{\sigma}_{12,t}(n) &= \frac{1}{n-1}\sum_{i=0}^{n-1}(y_{1t-i} - \hat{\mu}_{1t}(n))(y_{2t-i} - \hat{\mu}_{2t}(n)) \\ \hat{\rho}_{12,t}(n) &= \frac{\hat{\sigma}_{12,t}(n)}{\hat{\sigma}_{1t}(n)\hat{\sigma}_{2t}(n)}\end{aligned}$$

may be computed.

Example 50 *24 month rolling correlations between the returns on Microsoft and the S&P 500 index*

Consider the monthly continuously compounded returns on Microsoft stock and S&P 500 index over the period February 1990 through January 2001 computed from the monthly closing prices in the S+FinMetrics "timeSeries" object singleIndex.dat

```
> ret.ts = getReturns(singleIndex.dat,type="continuous")
> colIds(ret.ts)
[1] "MSFT"  "SP500"
```

The 24-month rolling correlations between the returns on Microsoft and S&P 500 index may be computed using the S-PLUS function aggregateSeries with a user specified function to compute the correlations. One such function is

```
> cor.coef = function(x) cor(x)[1,2]
```

The 24-month rolling correlations are then computed as

```
> smpl = positions(ret.ts)>=start(roll.cor)
> roll.cor = aggregateSeries(ret.ts,moving=24,together=T,
+ adj=1,FUN=cor.coef)
>  roll.cor[1:5]
 Positions      1
 Jan 1992  0.6549
 Feb 1992  0.6535
 Mar 1992  0.6595
 Apr 1992  0.6209
 May 1992  0.5479
```

In the call to aggregateSeries the argument together=T passes all of the columns of ret.ts to the function cor.coef instead of passing each column separately. The monthly returns on Microsoft and the S&P 500 index along with the rolling correlations are illustrated in Figure 9.4 which is created by

```
> par(mfrow=c(2,1))
> plot(ret.ts[smpl,],main="Returns on Microsoft and
+ S&P 500 index",
+ plot.args=list(lty=c(1,3)))
> legend(0,-0.2,legend=c("Microsoft","S&P 500"),
+ lty=c(1,3))
> plot(roll.cor,main="24-month rolling correlations")
```

At the beginning of the sample, the correlation between Microsoft and the S&P 500 is fairly high at 0.6. The rolling correlation declines steadily, hits

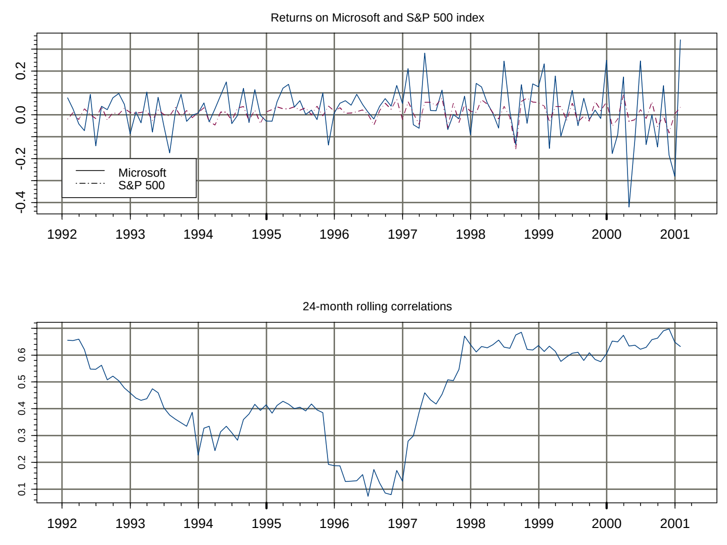


FIGURE 9.4. Returns on Microsoft and the S&P 500 index along with 24-month rolling correlations.

a low of about 0.1 at the beginning of 1997, then increases quickly to 0.6 and stabilizes at this value through the end of the sample[3].

9.2.3 Exponentially Weighted Moving Averages

The rolling descriptive statistics described in the previous sections are based on equally weighted moving averages of an observed time series y_t. Equally weighted moving averages are useful for uncovering periods of instability but may produce misleading results if used for short-term forecasting. This is because equally weighted averages are sensitive (not robust) to extreme values. To illustrate, consider $T = 100$ observations from a simulated time series $y_t \sim GWN(0, 1)$ with an outlier inserted at $t = 20$: i.e., $y_{20} = 10$. The data and rolling values $\hat{\mu}_t(10)$ and $\hat{\sigma}_t(10)$ are illustrated in Figure 9.5. Notice how the outlier at $t = 20$ inflates the rolling estimates $\hat{\mu}_t(10)$ and $\hat{\sigma}_t(10)$ for 9 periods.

[3]Approximate standard errors for the rolling correlations may be computed using

$$\widehat{SE}(\hat{\rho}_t(n)) = \sqrt{\frac{1 - \hat{\rho}_t(n)^2}{n}}$$

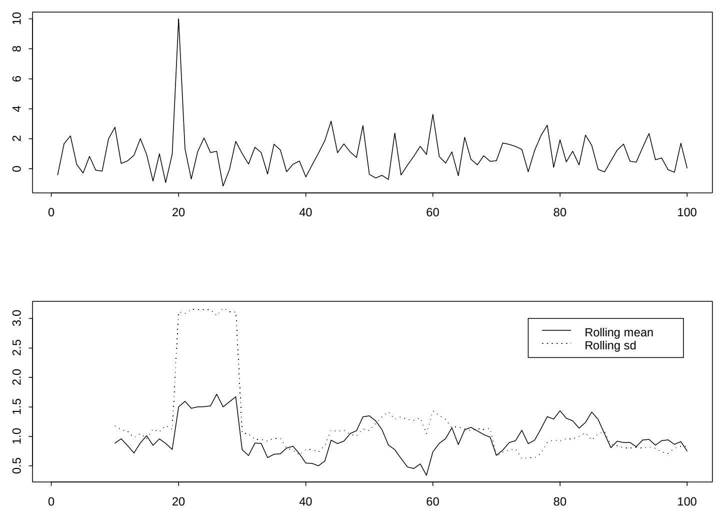


FIGURE 9.5. Effect of an outlier on equally weighted rolling estimates of μ and σ.

To mitigate the effects of extreme observations on moving average estimates the observations in a rolling window can be weighted differently. A common weighting scheme that puts more weight on the most recent observations is based on exponentially declining weights and the resulting weighted moving average is called an *exponentially weighted moving average* (EWMA). An n-period EWMA of a time series y_t is defined as

$$\tilde{\mu}_t(n) = \sum_{i=0}^{n-1} w_i \cdot y_{t-i},\ w_i = \frac{\lambda^{i-1}}{\sum_{i=0}^{n-1} \lambda^{i-1}}$$

where $0 < \lambda < 1$ is the decay parameter. As $n \to \infty$, $\lambda^n \to 0$, $w_n \to 0$, and the EWMA converges to

$$\tilde{\mu}_t(\lambda) = (1 - \lambda) \sum_{i=0}^{\infty} \lambda^i y_{t-i} \tag{9.4}$$

so the EWMA may be defined independently of the window width n. The EWMA in (9.4) may be efficiently computed using the recursion

$$\tilde{\mu}_t(\lambda) = (1 - \lambda) y_t + \lambda \tilde{\mu}_{t-1}(\lambda) \tag{9.5}$$

From (9.5), it is clear that the closer λ is to one the more weight is put on the the previous period's estimate relative to the current period's observation.

Therefore, λ may be interpreted as a persistence parameter. The recursive formula (9.5) requires a starting value $\mu_0(\lambda)$. Common choices are the first observation and the average over a local window.

EWMA estimates of descriptive statistics for continuously compounded asset returns are usually computed using high frequency data with the assumption that the mean returns are zero. Accordingly, the EWMA estimates of σ^2 and σ_{12} are

$$\begin{aligned} \tilde{\sigma}_t^2(\lambda) &= (1-\lambda)r_t^2 + \lambda\tilde{\sigma}_{t-1}^2(\lambda) \\ \tilde{\sigma}_{12,t}(\lambda) &= (1-\lambda)r_{1t}r_{2t} + \lambda\tilde{\sigma}_{12,t-1}(\lambda) \end{aligned} \tag{9.6}$$

where r_t denotes the continuously compounded return on an asset. The EWMA estimate of volatility (9.6) is in the form of a IGARCH(1,1) model without an constant term.

Computing EWMA Estimates Using the `S+FinMetrics` Function `EWMA`

EWMA estimates based on (9.5) may be efficiently computed using the `S+FinMetrics` function `EWMA`. The arguments expected by `EWMA` are

```
> args(EWMA)
function(x, n = 9, lambda = (n - 1)/(n + 1), start =
"average", na.rm = F)
```

where `x` is the data input, `n` is a window width, `lambda` is the decay parameter and `start` specifies the starting value for the recursion (9.5). The implied default value for λ is 0.8. Valid choices for `start` are `"average"` and `"first"`. The use of `EWMA` is illustrated with the following examples.

Example 51 *Outlier example*

Consider again the outlier example data shown in Figure 9.5. EWMA estimates of μ for $\lambda = 0.95$, 0.75 and 0.5 are computed and plotted in Figure 9.6 using

```
> ewma95.mean = EWMA(e,lambda=0.95)
> ewma75.mean = EWMA(e,lambda=0.75)
> ewma50.mean = EWMA(e,lambda=0.5)
> tsplot(ewma95.mean,ewma75.mean,ewma50.mean)
> legend(60,4,legend=c("lamda=0.95","lamda=0.75",
+ "lamda=0.50"),lty=1:3)
```

Notice that the EWMA estimates with $\lambda = 0.95$, which put the most weight on recent observations, are only minimally affected by the one-time outlier whereas the EWMA estimates with $\lambda = 0.75$ and 0.5 increase sharply at the date of the outlier.

Example 52 *EWMA estimates of standard deviations and correlations from high frequency data*

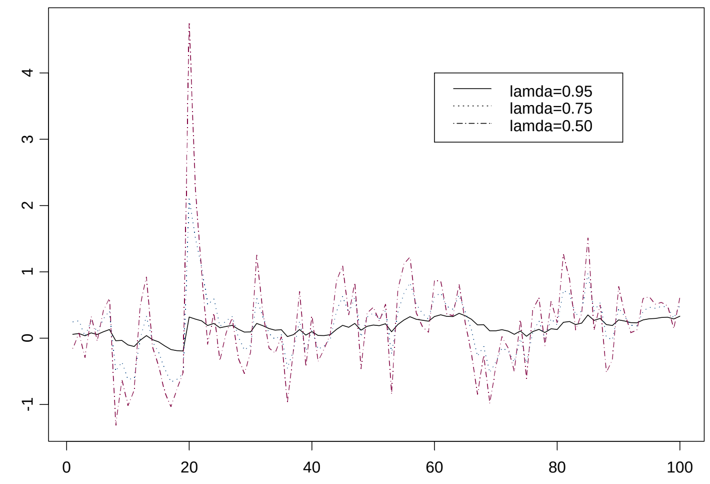


FIGURE 9.6. EWMA estimates of μ for outlier example data.

EWMA estimates of asset return standard deviations computed from high frequency data are commonly used as local or short-term estimates of volatility. Similarly, EWMA estimates of pairwise return correlations are often used to infer local interactions between assets. Indeed, J.P. Morgan's RiskMetrics® methodology is based on EWMA estimates of volatility and correlation. To illustrate, consider computing EWMA estimates of volatility and correlation with $\lambda = 0.95$ using daily closing price data on Microsoft and IBM stock over the five year period 1996-2000:

```
> smpl = (positions(DowJones30) >= timeDate("1/1/1996"))
> msft.ret.d = getReturns(DowJones30[smpl,"MSFT"])
> ibm.ret.d = getReturns(DowJones30[smpl,"IBM"])
> msft.ewma95.sd = sqrt(EWMA(msft.ret.d^2,lambda=0.95))
> ibm.ewma95.sd = sqrt(EWMA(ibm.ret.d^2,lambda=0.95))
> cov.ewma95 = EWMA(msft.ret.d*ibm.ret.d,lambda=0.95)
> cor.ewma95 = cov.ewma95/(msft.ewma95.sd*ibm.ewma95.sd)
> par(mfrow=c(2,1))
> plot(msft.ewma95.sd,ibm.ewma95.sd,
+ main="Rolling EWMA SD values",
+ plot.args=list(lty=c(1,3)))
> legend(0,0.055,legend=c("Microsoft","IBM"),lty=c(1,3))
> plot(cor.ewma95,main="Rolling EWMA correlation values")
```

Figure 9.7 shows the EWMA estimates of volatility and correlation. Daily

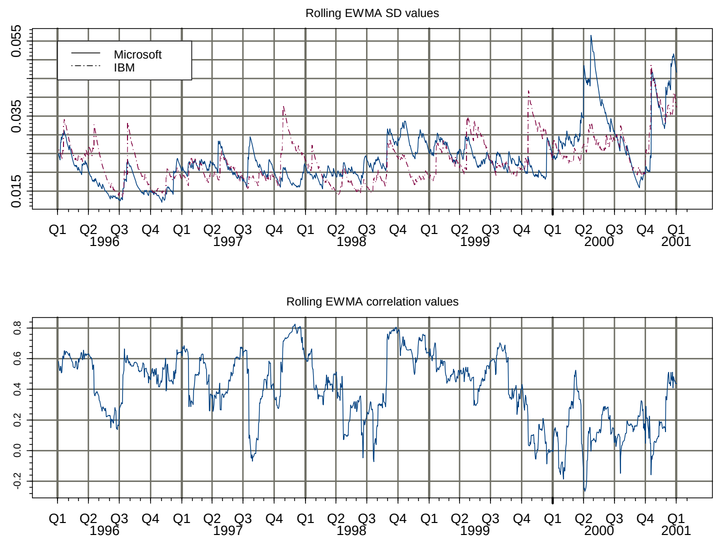


FIGURE 9.7. EWMA estimates of daily volatility and correlation with $\lambda = 0.95$.

volatility for Microsoft and IBM varies considerably, exhibiting apparent seasonality and an increasing trend. The daily correlations fluctuate around 0.5 for the first part of the sample and then drop to about 0.1 at the end of the sample.

9.2.4 Moving Average Methods for Irregularly Spaced High Frequency Data

The use of moving average and rolling methods on irregularly spaced or *inhomogeneous* high frequency time series data requires care. The moving average tools discussed so far are designed to work on regularly spaced or *homogeneous* time series data. Two approaches have been used to apply moving average methods to irregularly spaced data. The first approach converts the irregularly spaced data to regularly spaced data and then applies the tools for appropriate for regularly spaced data. The second approach, pioneered by Zumbach and Müller (2001), utilizes moving average methods specifically designed for irregularly spaced data.

Converting Inhomogeneous Time Series to Homogeneous Time Series

To illustrate the conversion of a inhomogeneous time series to a homogeneous time series, consider the transactions level data on 3M corporation stock for December 1999 in the S+FinMetrics data frame highFreq3M.df. As in Chapter 2, a "timeSeries" object may be created using

```
> td = timeDate(julian=(highFreq3M.df$trade.day-1),
+ ms=highFreq3M.df$trade.time*1000,
+ in.origin=c(month=12,day=1,year=1999),zone="GMT")
> hf3M.ts = timeSeries(pos=td,data=highFreq3M.df)
> hf3M.ts[1:20,]
          Positions trade.day trade.time trade.price
 12/1/99 9:33:32 AM 1         34412      94.69
 12/1/99 9:33:34 AM 1         34414      94.69
 12/1/99 9:33:34 AM 1         34414      94.69
...
 12/1/99 9:34:45 AM 1         34485      94.69
 12/1/99 9:34:47 AM 1         34487      94.63
...
```

The trade time is measured in second from midnight. Notice that many of the first trades took place at the same price and that there are instances of multiple transactions at the same time. The analysis is limited to the first three trading days of December

```
> smpl = positions(hf3M.ts) < timeDate("12/4/1999")
> hf3M.ts = hf3M.ts[smpl,]
```

The data in hf3M.ts may be made homogeneous by use of an interpolation method to align the irregularly spaced time sequence and associated data to a regularly spaced time sequence. For example, consider creating a homogeneous time series of five minute observations. Since the data in hf3M.ts may not be recorded at all five minute intervals, some interpolation scheme must be used to create the data. Two common interpolation schemes are: *previous tick interpolation*, and *linear interpolation*. The former method uses the most recent values, and the latter method uses observations bracketing the desired time. The S-PLUS functions align may be used to perform these interpolation schemes.

The function align takes a "timeSeries" and a "timeDate" vector of new positions to align to. An easy way to create a "timeDate" sequence of five minute observations covering the trading hours for 3M stock is to use the S-PLUS function aggregateSeries as follows:

```
> tmp = aggregateSeries(hf3M.ts,by="minutes",k.by=5,FUN=mean)
```

The positions slot of "timeSeries" tmp contains the desired five minute "timeDate" sequence:

```
> positions(tmp[1:4])
 [1] 12/1/99 9:30:00 AM  12/1/99 9:35:00 AM
 [3] 12/1/99 9:40:00 AM  12/1/99 9:45:00 AM
```

To align the 3M price data to the five minute time sequence using previous tick interpolation use

```
> hf3M.5min = align(hf3M.ts[,"trade.price"], positions(tmp),
+ how="before")
> hf3M.5min[1:5,]
          Positions trade.price
 12/1/99 9:30:00 AM    NA
 12/1/99 9:35:00 AM 94.63
 12/1/99 9:40:00 AM 94.75
 12/1/99 9:45:00 AM 94.50
 12/1/99 9:50:00 AM 94.31
```

To align the price data using linear interpolation use

```
> hf3M.5min = align(hf3M.ts[,"trade.price"], positions(tmp),
+ how="interp")
> hf3M.5min[1:5,]
          Positions trade.price
 12/1/99 9:30:00 AM    NA
 12/1/99 9:35:00 AM 94.65
 12/1/99 9:40:00 AM 94.75
 12/1/99 9:45:00 AM 94.42
 12/1/99 9:50:00 AM 94.26
```

The usual methods for the analysis of homogeneous data may now be performed on the newly created data. For example, to compute and plot an EWMA of price with $\lambda = 0.9$ use

```
> hf3M.5min.ewma = EWMA(hf3M.5min,lambda=0.9,na.rm=T)
> plot(hf3M.5min.ewma)
```

The resulting plot is shown in Figure 9.8.

Inhomogeneous Moving Average Operators

Zumbach and Müller (2001) presented a general framework for analyzing inhomogeneous time series. A detailed exposition of this framework is beyond the scope of this book. Only a brief description of the most fundamental inhomogeneous time series operators is presented and reader is referred to Zumbach and Müller (2001) or Dacorogna et. al. (2001) for technical details and further examples.

Zumbach and Müller (2001) distinguished between *microscopic* and *macroscopic* operations on inhomogeneous time series. A microscopic operation depends on the actual sampling times of the time series, whereas a macroscopic operator extracts an average over a specified range. Macroscopic operations on high frequency inhomogeneous time series are advantageous because they are essentially immune to small variations in the individual

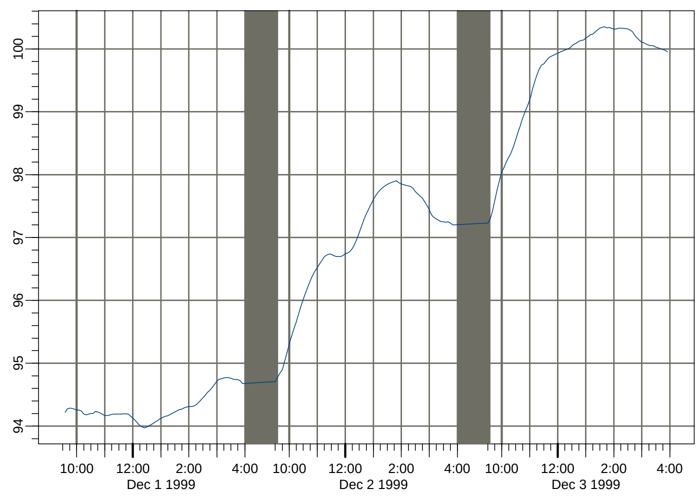


FIGURE 9.8. EWMA of five minute prices on 3M stock.

data observations and are better behaved and more robust than microscopic operations. The S+FinMetrics functions for analyzing inhomogeneous time series are based on a subset of the macroscopic operators discussed in Zumbach and Müller (2001). These functions are summarized in Table 9.1.

In general, given a continuous time signal $z(t)$, a macroscopic operator Ω can be defined as a *convolution* with a causal kernel $\omega(\cdot)$:

$$\Omega(t) = \int_0^t w(t-s)z(s)ds \tag{9.7}$$

for $t > 0$. Note that for a causal kernel $\omega(t) = 0$ for any $t < 0$, since future information cannot be utilized. In addition, it is usually required that

$$\int_0^\infty \omega(t)dt = 1$$

so that the operator can be interpreted as a weighted moving average of the signal $z(t)$. For example, the exponential moving average (EMA) operator is defined with an exponential kernel:

$$\omega(t) = \frac{e^{-t/\tau}}{\tau} \tag{9.8}$$

and it is easy to verify that

$$\int_0^\infty \frac{e^{-t/\tau}}{\tau} = 1$$

Function	Description
`iEMA`	Inhomogeneous EWMA
`iMA`	Inhomogeneous moving average
`iMNorm`	Inhomogeneous moving norm
`iMVar`	Inhomogeneous moving variance
`iMSD`	Inhomogeneous moving SD
`iMSkewness`	Inhomogeneous moving skewness
`iMKurtosis`	Inhomogeneous moving kurtosis
`iMCor`	Inhomogeneous moving correlation
`iDiff`	Inhomogeneous moving difference
`iEMA.kernel`	Kernel function for `iEMA`
`iMA.kernel`	Kernel function for `iMA`

TABLE 9.1. `S+FinMetrics` inhomogeneous time series function

The parameter τ can be shown to be the *range* of the EMA kernel.[4]

In reality, a time series signal is usually observed at discrete times. In addition, financial transactions level data are usually observed on irregular intervals. For the EMA operator, Zumbach and Müller suggest to use the following iterative formula to compute a discrete time approximation to (9.7):[5]

$$\mathrm{EMA}(t_n;\tau)=\mu\mathrm{EMA}(t_{n-1};\tau)+(1-\mu)z(t_n)+(\mu-\nu)[z(t_n)-z(t_{n-1})] \quad (9.9)$$

where

$$\mu=e^{-\alpha},\nu=(1-\mu)/\alpha$$

and

$$\alpha=(t_n-t_{n-1}).$$

Note that when α is very small, $e^{\alpha}\approx 1+\alpha$ and it can be shown that $\mu\approx\nu$. In this case, the above formula reduces to the same iteration for evenly spaced EWMA operator.

Using the basic EMA operator, different operators can be constructed. For example, Zumbach and Müller suggested that the basic EMA operator can be iterated a finite number of times to obtain an operator with a different kernel, denoted $\mathrm{EMA}(\tau,k)$. The $\mathrm{EMA}(\tau,k)$ operator can be summed to obtain the analog of the moving average (MA) operator for inhomogeneous time series:

$$\mathrm{MA}(\tau,k)=\frac{1}{k}\sum_{i=1}^{k}\mathrm{EMA}(s,i)$$

where $s=2\tau/(k+1)$ so that the range of $\mathrm{MA}(\tau,k)$ is equal to τ, independent of k.

[4]The range is defined as the first moment of the kernel, i.e., $\int_0^\infty \omega(t)tdt$.

[5]This formula is actually obtained by assuming linear interpolation between points. If previous tick interpolation is used, then $\nu=1$.

The `S+FinMetrics` functions `iEMA.kernel` and `iMA.kernel` can be used to plot the kernel functions for EMA and MA operators, while `iEMA` and `iMA` can be used to compute the EMA and MA operator for inhomogeneous time series. For example, the following code plots the EMA(τ, k) and MA(τ, k) kernel functions for $\tau = 1$ and $k = 1, 2, \cdots, 10$:

```
> par(mfrow=c(2,1))
> knl = iEMA.kernel(1, 1)
> plot(knl, type="l", main="EMA Kernel")
> for(i in 2:10) {
>   knl = iEMA.kernel(1, i)
>   lines(knl, lty=i)
> }

> knl = iMA.kernel(1, 1)
> plot(knl, type="l", main="MA Kernel")
> for(i in 2:10) {
>   knl = iMA.kernel(1, i)
>   lines(knl, lty=i)
> }
> par(mfrow=c(1,1))
```

and the resulting plot is shown in Figure 9.9. From the figure, it can be seen that when $k = 1$, EMA(τ, k) and MA$(\tau, 1)$ are equivalent by definition. However, as k gets larger, the kernel function of EMA(τ, k) becomes flatter, while the kernel function of MA$(\tau, 1)$ becomes more like a rectangle. In fact, Zumbach and Müller showed that the range of EMA(τ, k) is $k\tau$, while the range of MA$(\tau, 1)$ becomes a constant for $t \leq 2\tau$ as $k \to \infty$. As a result, to obtain an MA operator with window width equal to 9 (which corresponds to a range of 8, i.e., using 8 observations in the past), one sets $\tau = 4$ and k to a large number:

```
> iMA(1:100, 4, iter=10)
  [1]  1.000  1.084  1.305  1.662  2.150  2.761  3.481
  [8]  4.289  5.166  6.091  7.047  8.024  9.011 10.005
 [15] 11.002 12.001 13.000 14.000 15.000 16.000 17.000
 [22] 18.000 19.000 20.000 21.000 22.000 23.000 24.000
 [29] 25.000 26.000 27.000 28.000 29.000 30.000 31.000
 [36] 32.000 33.000 34.000 35.000 36.000 37.000 38.000
 [43] 39.000 40.000 41.000 42.000 43.000 44.000 45.000
 [50] 46.000 47.000 48.000 49.000 50.000 51.000 52.000
 [57] 53.000 54.000 55.000 56.000 57.000 58.000 59.000
 [64] 60.000 61.000 62.000 63.000 64.000 65.000 66.000
 [71] 67.000 68.000 69.000 70.000 71.000 72.000 73.000
 [78] 74.000 75.000 76.000 77.000 78.000 79.000 80.000
 [85] 81.000 82.000 83.000 84.000 85.000 86.000 87.000
```

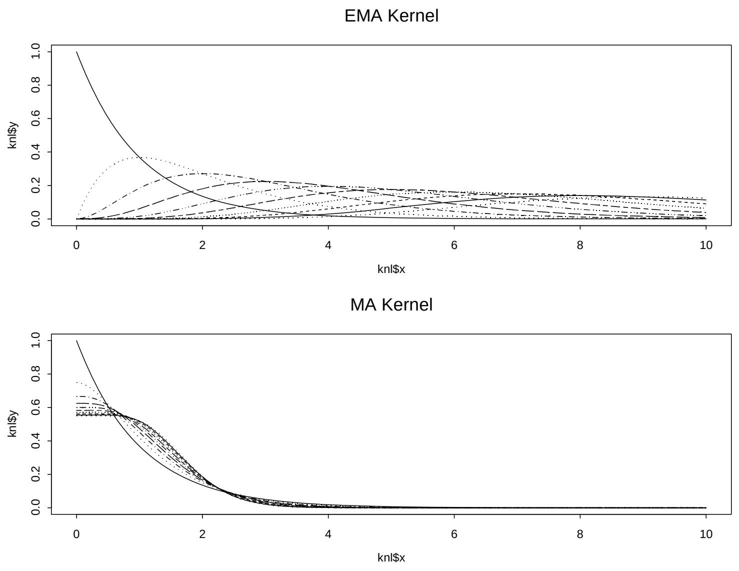


FIGURE 9.9. Kernel function for EMA and MA operators for inhomogeneous time series.

```
[92] 88.000 89.000 90.000 91.000 92.000 93.000 94.000
[99] 95.000 96.000
```

The S+FinMetrics iMA function requires at least two arguments: the first is the input series, and the second specifies the value for τ. In the above example, the optional argument iter is used to specify the number of iterations k; in addition, since the input series is not a "timeSeries" object, iMA treats it as evenly spaced and τ is in units of observations. If the input series is a "timeSeries" object, then τ should be specified in units of "business days". To illustrate the usage of iMA in this case, first create a "timeSeries" object representing the transaction price data from hf3M.ts created earlier[6]:

```
> smpl2 = positions(hf3M.ts) < timeDate("12/02/1999")
> hf3m.1min = aggregateSeries(hf3M.ts[smpl2,"trade.price"],
+ by="minutes", FUN=mean)
> hf3m.1min[103:110]
             Positions trade.price
 12/1/1999 11:28:00 AM 94.25000
```

[6]The S-PLUS function aggregateSeries is used to eliminate multiple transactions that occur at the same time. Currently, the S+FinMetrics inhomogeneous time series functions do not work if there are multiple observations with the same time stamp.

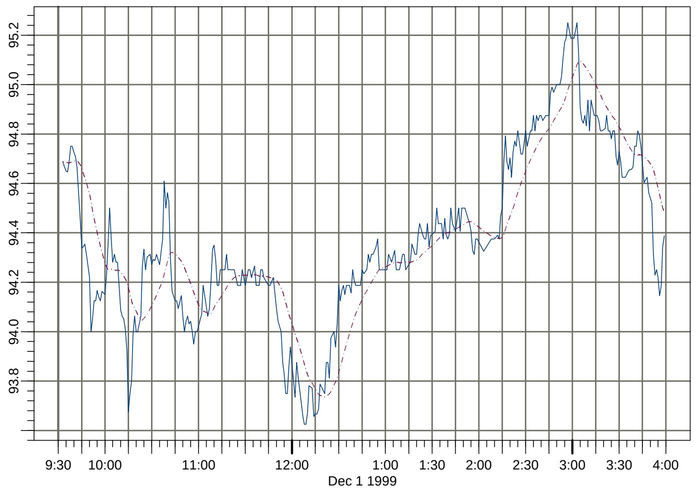


FIGURE 9.10. 20 minute moving average computed from `iMA` for 3M stock prices.

```
 12/1/1999 11:30:00 AM 94.18750
 12/1/1999 11:32:00 AM 94.25000
 12/1/1999 11:33:00 AM 94.25000
 12/1/1999 11:34:00 AM 94.21875
 12/1/1999 11:36:00 AM 94.26563
 12/1/1999 11:37:00 AM 94.18750
 12/1/1999 11:39:00 AM 94.18750
```

Note that the data is not evenly spaced. To obtain a 20 minute moving average of `hf3m.1min`, set $\tau = 10/(6.5 * 60)$ because there are 6.5 hours for the default trading hours (from 9:30 AM to 4:00 PM):

```
> hf3m.ma = iMA(hf3m.1min, 10/(6.5*60), iter=10)
> plot(seriesMerge(hf3m.1min, hf3m.ma),
+ plot.args=list(lty=c(1,3)))
```

The original data and the 20 minutes moving average `hf3m.ma` are plotted together in Figure 9.10.

9.2.5 Rolling Analysis of Miscellaneous Functions

The standard analysis tools for time series require the data to be stationary. Rolling analysis of descriptive statistics can give an indication of structural change or instability in the moments of a time series. Level shifts and

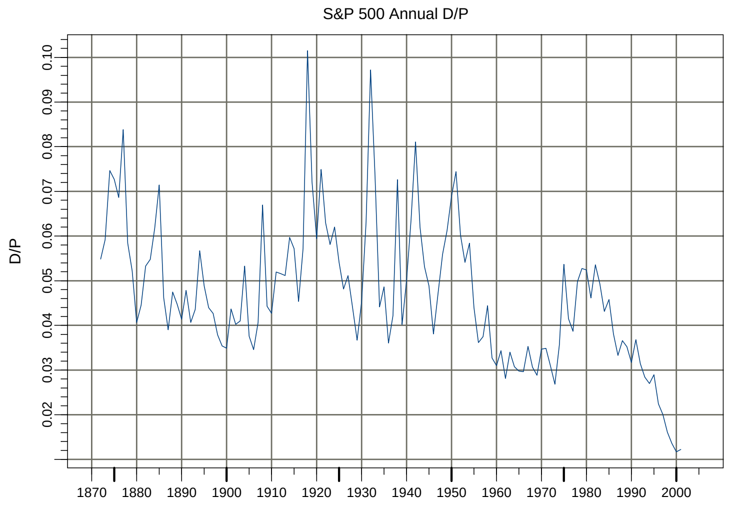


FIGURE 9.11. S & P 500 annual dividend/price ratio.

variance changes can usually be detected by rolling analyses. The `S-PLUS` function `aggregateSeries` may be used to perform rolling analysis with a variety of functions to uncover periods of instability and nonstationarity. The following example illustrates the use of `aggregateSeries` with the `S+FinMetrics` function `unitroot` to determine periods of unitroot nonstationarity of a time series.

Example 53 *Rolling unit root tests applied to annual dividend/price ratio*

Predictive regressions of asset returns on valuation ratios like dividend/ price or earnings/price require the valuation ratios to be stationary or, more generally, $I(0)$, for the regressions to be statistically valid. Since asset returns are $I(0)$, if valuation ratios are $I(1)$ then the predictive regressions are unbalanced and the results will be nonsensical. To illustrate, consider the annual dividend-price (D/P) ratio on S&P 500 index taken from the `S+FinMetrics` "`timeSeries`" `shiller.annual`

```
> dp.ratio = shiller.annual[,"dp.ratio"]
> plot(dp.ratio,main="S&P 500 Annual D/P",ylab="D/P")
```

shown in Figure 9.11. For most of the sample the annual D/P looks to be $I(0)$ with mean near 5%. However, there are long periods when the ratio stays above or below 5% suggesting periods of non-mean reverting (nonstationary) behavior. Also, there is a clear drop in the ratio at the end of the sample suggesting a fundamental change in the mean. Rolling unit root tests may be used to uncover periods of nonstationary behavior in D/P. To compute rolling ADF t-tests and normalized bias statistics using the S-PLUS function aggregateSeries create the following function adf.tests

```
> adf.tests = function(x, trend = "c", lags = 3)
> {
>     tmp1 = unitroot(x,trend=trend,lags=lags,statistic="t")
>     tmp2 = unitroot(x,trend=trend,lags=lags,statistic="n")
>     ans = concat(tmp1$sval,tmp2$sval)
>     ans
> }
```

The function adf.tests takes a time series x, passes it to the S+FinMetrics function unitroot twice and returns the ADF t-statistic and normalized bias statistic. Three lags are chosen for the tests based on a full sample analysis using the Ng-Perron backward selection procedure. Rolling unit root tests using a window of 50 years are then computed using the function aggregateSeries:

```
> roll.adf = aggregateSeries(dp.ratio,moving=50,adj=1,
+ FUN=adf.tests,colnames=c("t.test","norm.bias"))
```

The object roll.adf is a "timeSeries" containing the rolling unit root tests

```
> roll.adf[1:3,]
 Positions t.test norm.bias
 Dec 1920  -1.840 -13.24
 Dec 1921  -2.168 -15.24
 Dec 1922  -2.270 -16.03
```

Figure 9.12 is created using

```
> cvt.05 = qunitroot(0.05,trend="c",n.sample=50)
> cvn.05 = qunitroot(0.05,trend="c",statistic="n",
+ n.sample=50)
> par(mfrow=c(2,1))
> plot(roll.adf[,"t.test"], reference.grid=F,
+ main="Rolling ADF t-statistics")
> abline(h=cvt.05)
> plot(roll.adf[,"norm.bias"], reference.grid=F,
+ main="Rolling ADF normalized bias")
> abline(h=cvn.05)
```

and shows the rolling unit root tests along with 5% critical values. The results indicate that D/P is stationary mainly in the middle of the sample and becomes nonstationary toward the end of the sample. However, some

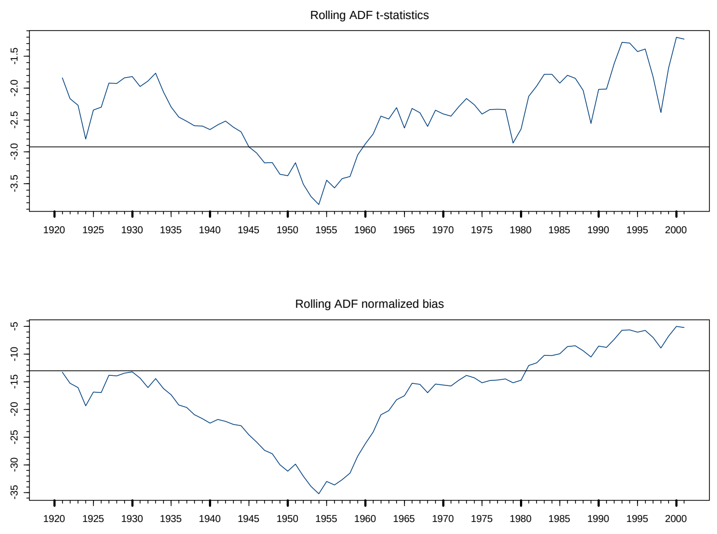


FIGURE 9.12. 50 year rolling ADF t-statistics and normalized bias statistics for the S&P 500 dividend-price ratio.

care must be used when interpreting the significance of the rolling unit root tests. The 5% critical values shown in the figures are appropriate for evaluating a single test and not a sequence of rolling tests. Critical values appropriate for rolling unit root tests are given in Banerjee, Lumsdaine and Stock (1992).

9.3 Technical Analysis Indicators

Technical analysis is, perhaps, the most widely used method for analyzing financial time series. Many of the most commonly used technical indicators are based on moving average techniques so it is appropriate to include a discussion of them here. A comprehensive survey of technical analysis is beyond the scope of this book. Useful references are Colby and Meyers (1988) and Bauer and Dahlquist (1999). The `S+FinMetrics` technical analysis indicators are implemented using the definitions in Colby and Meyers (1988). Broadly, the main technical indicators can be classified into four categories: price indicators, momentum indicators and oscillators, volatility indicators and volume indicators.

Function	**Description**
`TA.Bollinger`	Bollinger band
`TA.medprice`	Median price
`TA.typicalPrice`	Typical price
`TA.wclose`	Weighted close

TABLE 9.2. `S+FinMetrics` price indicators

9.3.1 Price Indicators

The `S+FinMetrics` price indicator functions are summarized in Table 9.2. To illustrate the use of these functions, consider the calculation of the typical daily price, which is defined to be the average of the highest, lowest and closing prices during the day, using the `S+FinMetrics` function `TA.typicalPrice`. The arguments expected by `TA.typicalPrice` are

```
> args(TA.typicalPrice)
function(high, low, close)
```

In order to compute this indicator, a data set with high, low and close prices is required. To compute the typical price for the Dow Jone Industrial Average over the period January 1, 1990 to February 20, 1990 using the `S-PLUS` "timeSeries" `djia`, use

```
> smpl = positions(djia) >= timeDate("1/1/1990")
> dj = djia[smpl,]
> tp.dj = TA.typicalPrice(dj[,"high"],
+ dj[,"low"],dj[,"close"])
> class(tp.dj)
[1] "timeSeries"
```

The typical price along with the high, low, open and close prices may be plotted together using

```
> plot.out = plot(dj[,1:4],plot.type="hloc")
> lines.render(positions(tp.dj),seriesData(tp.dj),
+ x.scale=plot.out$scale)
```

and the resulting plot is shown in Figure 9.13.

9.3.2 Momentum Indicators and Oscillators

The `S+FinMetrics` *momentum indicator and oscillator functions* are summarized in Table 9.3. For example, consider the popular *moving average convergence divergence* (MACD) indicator. This is an oscillator that represents the difference between two exponential moving averages. A signal line is computed as the exponential moving average of MACD. When the oscillator crosses above the signal line, it indicates a buy signal; when

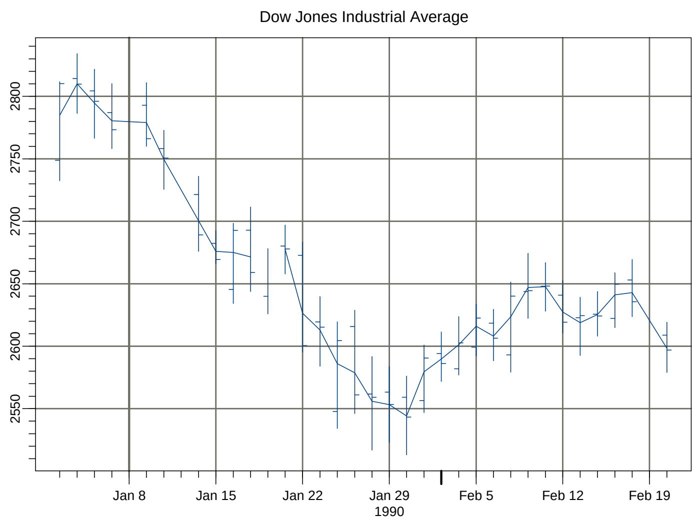


FIGURE 9.13. Typical price along with high, low, open and close prices for the Dow Jones Industrial Average.

the oscillator crosses below the signal line, it indicates a sell signal. The `S+FinMetrics` function `TA.macd` computes the MACD and has arguments

```
> args(TA.macd)
function(x, n.short = 12, n.long = 26, n.signal = 9, start
= "average", na.rm = F)
```

where `x` is a price series, `n.short` is a positive integer specifying the number of periods to be used for calculating the short window EWMA, `n.long` is a positive integer specifying the number of periods to be used for the calculating the long window EWMA, and `n.signal` is a positive integer

Function	**Description**
`TA.accel`	Acceleration
`TA.momentum`	Momentum
`TA.macd`	Moving average convergence divergence
`TA.roc`	Price rate of change
`TA.rsi`	Relative strength index
`TA.stochastic`	Stochastic Oscillator
`TA.williamsr`	Williams' %R
`TA.williamsad`	Williams' accumulation distribution

TABLE 9.3. `S+FinMetrics` momentum indicators

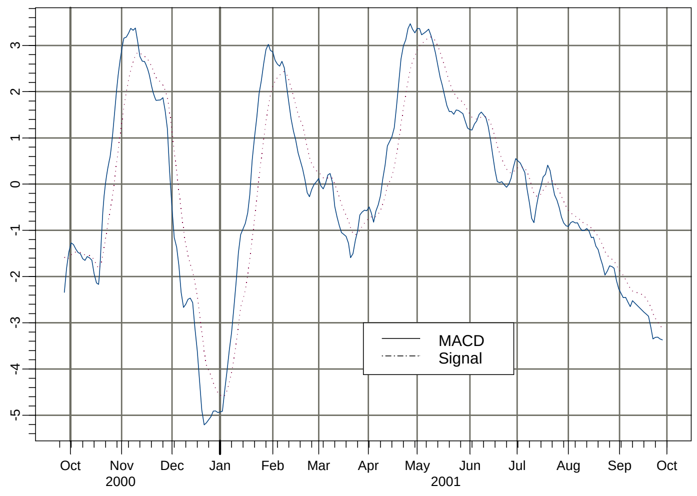


FIGURE 9.14. MACD and signal for daily closing prices on Microsoft stock.

Function	**Description**
`TA.adoscillator`	accumulation/distribution oscillator
`TA.chaikinv`	Chaikin's volatility
`TA.garmanKlass`	Garman-Klass estimator of volatility

TABLE 9.4. `S+FinMetrics` volatility indicator functions

giving the number of periods for the signal line. To compute and plot the MACD using daily closing prices on Microsoft use

```
> msft.macd = TA.macd(msft.dat[,"Close"])
> colIds(msft.macd) = c("MACD","Signal")
> plot(msft.macd,plot.args=list(lty=c(1:3)))
> legend(0.5,-3,legend=colIds(msft.macd),
+ lty=c(1,3))
```

Figure 9.14 shows the plot of the MACD and signal.

9.3.3 Volatility Indicators

The `S+FinMetrics` *volatility indicator functions* are summarized in Table 9.4. These functions compute estimates of volatility based on high, low, open and close information. For example, consider Chaikin's volatility indicator computed using the `S+FinMetrics` function `TA.chaikin`. It com-

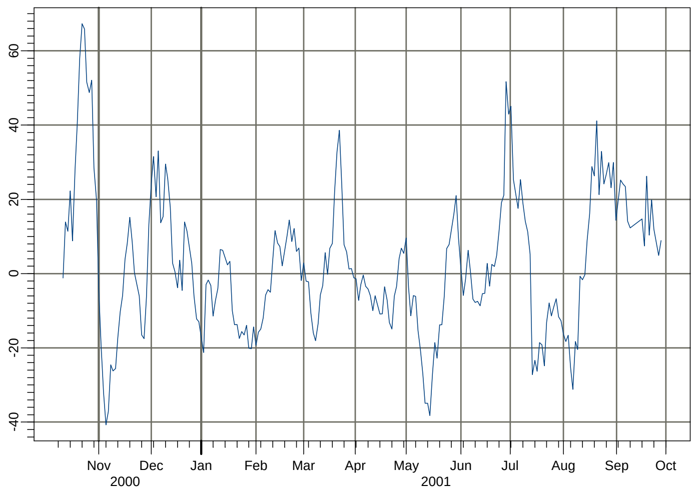


FIGURE 9.15. Chainkin's volatility estimate using the daily prices on Microsoft stock.

pares the spread between a security's high and low prices and quantifies volatility as a widening of the range between the high and low price. Let h_t and l_t represent the highest and lowest price for period t, respectively. Chaikin's volatility is calculated as the percentage change in the EWMA of $r_t = h_t - l_t$:

$$\frac{r_t - r_{t-nc}}{r_{t-nc}} \cdot 100$$

where nc is a positive number specifying the number of periods to use for computing the percentage change. To compute and plot Chaikin's volatility with $nc = 10$ and a ten day EWMA for the daily high and low price for Microsoft stock use

```
> msft.cv = TA.chaikinv(msft.dat[,"High"],
+ msft.dat[,"Low"],n.range=10,n.change=10)
> plot(msft.cv)
```

Figure 9.15 shows the estimated Chaikin volatility.

9.3.4 Volume Indicators

The `S+FinMetrics` *volume indicator functions* are summarized in Table 9.5. These indicators relate price movements with volume movements. To illustrate, consider the `S+FinMetrics` function `TA.adi` which computes the

Function	Description
`TA.adi`	Accumulation/distribution indicator
`TA.chaikino`	Chaikin oscillator
`TA.nvi`	Negative volume index
`TA.pvi`	Positive volume index
`TA.obv`	On balance volume
`TA.pvtrend`	Price-volume trend

TABLE 9.5. `S+FinMetrics` volume indicator functions

accumulations/distribution (A/D) indicator. This indicator associates price changes with volume as follows. Let c_t denote the closing price, h_t the highest price, l_t the lowest price, and v_t the trading volume for time t. The A/D indicator is the cumulative sum

$$AD_t = \sum_{i=1}^{t} \frac{c_i - l_i - (h_i - c_i)}{h_i - l_i} \cdot v_i$$

When AD_t moves up, it indicates that the security is being accumulated; when it moves down it indicates that the security is being distributed. To compute and plot the A/D indicator for Microsoft stock use

```
> msft.adi = TA.adi(msft.dat[,"High"],msft.dat[,"Low"],
+ msft.dat[,"Close"],msft.dat[,"Volume"])
> plot(msft.adi)
```

The resulting plot is shown in Figure 9.16.

9.4 Rolling Regression

For the linear regression model, rolling analysis may be used to assess the stability of the model's parameters and to provide a simple "poor man's" time varying parameter model. For a window of width $n < T$, the *rolling linear regression model* may be expressed as

$$\mathbf{y}_t(n) = \mathbf{X}_t(n)\boldsymbol{\beta}_t(n) + \boldsymbol{\varepsilon}_t(n),\ t = n, \ldots, T \tag{9.10}$$

where $\mathbf{y}_t(n)$ is an $(n \times 1)$ vector of observations on the response, $\mathbf{X}_t(n)$ is an $(n \times k)$ matrix of explanatory variables, $\boldsymbol{\beta}_t(n)$ is an $(k \times 1)$ vector of regression parameters and $\boldsymbol{\varepsilon}_t(n)$ is an $(n \times 1)$ vector of error terms. The n observations in $\mathbf{y}_t(n)$ and $\mathbf{X}_t(n)$ are the n most recent values from times $t-n+1$ to t. It is assumed that $n > k$. The rolling least squares estimates

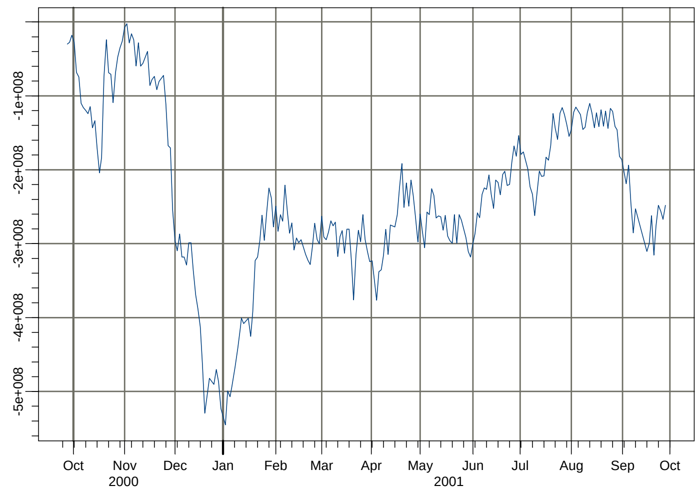


FIGURE 9.16. Accumulation/distribution indicator for Microsoft stock.

are

$$
\begin{aligned}
\hat{\boldsymbol{\beta}}_t(n) &= \left[\mathbf{X}_t(n)'\mathbf{X}_t(n)\right]^{-1}\mathbf{X}_t(n)'\mathbf{y}_t(n) \\
\hat{\sigma}_t^2(n) &= \frac{1}{n-k}\hat{\boldsymbol{\varepsilon}}_t(n)'\hat{\boldsymbol{\varepsilon}}_t(n) \\
&= \frac{1}{n-k}\left[\mathbf{y}_t(n)-\mathbf{X}_t(n)\hat{\boldsymbol{\beta}}_t(n)\right]'\left[\mathbf{y}_t(n)-\mathbf{X}_t(n)\hat{\boldsymbol{\beta}}_t(n)\right] \\
\widehat{avar}(\hat{\boldsymbol{\beta}}_t(n)) &= \hat{\sigma}_t^2(n)\cdot\left[\mathbf{X}_t(n)'\mathbf{X}_t(n)\right]^{-1}
\end{aligned}
$$

9.4.1 Estimating Rolling Regressions Using the S+FinMetrics Function rollOLS

The S+FinMetrics function rollOLS may be used to estimate general rolling regression models. rollOLS is based on the S+FinMetrics regression function OLS and implements efficient block updating algorithms for fast computation of rolling estimates. The arguments expected by rollOLS are

```
> args(rollOLS)
function(formula, data, subset, na.rm = F, method = "fit",
contrasts = NULL, start = NULL, end = NULL, width =
NULL, incr = 1, tau = 1e-010, trace = T, ...)
```

which are similar to those used by `OLS`. In particular, `AR` may be used in formulas to allow for lagged dependent variables and `tslag` and `pdl` may be used to allow for lagged independent variables. The argument `width` determines the rolling window width and the argument `incr` determines the increment size by which the windows are rolled through the sample. The output of `rollOLS` is an object of class "`rollOLS`" for which there are `print`, `summary`, `plot` and `predict` methods and extractor function `coefficients`. The use of `rollOLS` is illustrated with the following example.

Example 54 *Rolling estimation of CAPM for Microsoft*

Consider the estimation of the capital asset pricing model (CAPM) for an asset using rolling regression on the excess returns market model

$$r_t - r_{ft} = \alpha + \beta(r_{Mt} - r_{ft}) + \varepsilon_t,\ \varepsilon_t \sim WN(0, \sigma^2) \tag{9.11}$$

where r_t denotes the monthly return on an asset, r_{ft} denotes the 30 day T-bill rate, and r_{Mt} denotes the monthly return on a market portfolio proxy. The coefficient β measures the magnitude of market risk, and the CAPM imposes the restriction that $\alpha = 0$. Positive values of α indicate an average excess return above that predicted by the CAPM and negative values indicate an average return below that predicted by the CAPM. Rolling regression can be used to assess the stability of the CAPM regression over time and to uncover periods of time where an asset may have been overpriced or underpriced relative to the CAPM.

The monthly excess return data on Microsoft stock and S&P 500 index over the ten year period February 1990 through December 2000 are in the `S+FinMetrics` "`timeSeries`" object `excessReturns.ts`.

```
> colIds(excessReturns.ts)
[1] "MSFT"  "SP500"
> start(excessReturns.ts)
[1] Feb 1990
> end(excessReturns.ts)
[1] Dec 2000
```

The full sample CAPM estimates using the `S+FinMetrics` function `OLS` are

```
> ols.fit = OLS(MSFT~SP500,data=excessReturns.ts)
> summary(ols.fit)

Call:
OLS(formula = MSFT ~SP500, data = excessReturns.ts)

Residuals:
     Min      1Q  Median      3Q     Max
 -0.3101 -0.0620 -0.0024  0.0581  0.2260
```

```
Coefficients:
             Value Std. Error t value Pr(>|t|)
(Intercept) 0.0175 0.0081     2.1654  0.0322
      SP500 1.5677 0.2015     7.7788  0.0000

Regression Diagnostics:

         R-Squared 0.3193
Adjusted R-Squared 0.3140
Durbin-Watson Stat 2.1891

Residual standard error: 0.09095 on 129 degrees of freedom
Time period: from Feb 1990 to Dec 2000
F-statistic: 60.51 on 1 and 129 degrees of freedom, the p-va
lue is 2.055e-012
```

The estimated full sample β for Microsoft is 1.57, which indicates that Microsoft was riskier than the market. Also, the full sample estimate of α is significantly different from zero so, on average, the returns on Microsoft are larger than predicted by the CAPM.

Consider now the 24-month rolling regression estimates incremented by 1 month computed using `rollOLS`

```
> roll.fit = rollOLS(MSFT~SP500, data=excessReturns.ts,
+ width=24,incr=1)
Rolling Window #1: Total Rows = 24
Rolling Window #2: Total Rows = 25
Rolling Window #3: Total Rows = 26
...
Rolling Window #108: Total Rows = 131
```

To suppress the printing of the window count, specify `trace=F` in the call to `rollOLS`. The returned object `roll.fit` is of class "`rollOLS`" and has components

```
> names(roll.fit)
 [1] "width"     "incr"      "nwin"      "contrasts" "rdf"
 [6] "coef"      "stddev"    "sigma"     "terms"     "call"
[11] "positions"
```

The components `coef`, `stddev` and `sigma` give the estimated coefficients, standard errors, and residual standard deviations for each of the `nwin` regressions. The `positions` component gives the start and end date of the estimation sample.

The `print` method, invoked by typing the object's name, gives a brief report of the fit

```
> roll.fit

Call:
rollOLS(formula = MSFT ~SP500, data = excessReturns.ts,
width = 24, incr = 1)

Rolling Windows:
 number width increment
    108    24         1
Time period: from Feb 1990 to Dec 2000

Coefficients:
          (Intercept)  SP500
     mean 0.0221      1.2193
std. dev. 0.0120      0.4549

Coefficient Standard Deviations:
          (Intercept)  SP500
     mean 0.0177      0.5057
std. dev. 0.0034      0.1107

Residual Scale Estimate:
   mean std. dev.
 0.0827 0.0168
```

Regression estimates are computed for 108 rolling windows. The mean and standard deviation are computed for the estimates and for the estimated coefficient standard errors. The average and standard deviation of the $\hat{\alpha}$ values are 0.0221 and 0.0120, respectively, and the average and standard deviation of the SE($\hat{\alpha}$) values are 0.0177 and 0.0034, respectively. Hence, most of the $\hat{\alpha}$ values appear to be not significantly different from zero as predicted by the CAPM. The average and standard deviation of the $\hat{\beta}$ values are 1.2193 and 0.4549, respectively. The $\hat{\beta}$ values are quite variable and indicate that amount of market risk in Microsoft is not constant over time.

The rolling coefficient estimates for each rolling regression may be viewed using `summary`

```
> summary(roll.fit)

Call:
rollOLS(formula = MSFT ~ SP500, data = excessReturns.ts,
width = 24, incr = 1)

Rolling Windows:
 number width increment
```

```
    108    24          1
Time period: from Feb 1990 to Dec 2000

Coefficient: (Intercept)
           Value Std. Error t value Pr(>|t|)
Jan 1992 0.05075    0.01551   3.271 0.003492
Feb 1992 0.04897    0.01559   3.141 0.004751
Mar 1992 0.04471    0.01561   2.863 0.009035
...
Coefficient: SP500
          Value Std. Error t value  Pr(>|t|)
Jan 1992 1.3545     0.3322   4.077 0.0004993
Feb 1992 1.3535     0.3337   4.056 0.0005260
Mar 1992 1.3735     0.3332   4.123 0.0004472
...
```

or by using the `coef` extractor function. Notice that the first 24-month rolling estimates are computed for January 1992, which is 24 months after the sample start date of February 1990. The rolling estimates, however, are best viewed graphically using `plot`

```
> plot(roll.fit)

Make a plot selection (or 0 to exit):

1: plot: All
2: plot: Coef Estimates
3: plot: Coef Estimates with Confidence Intervals
4: plot: Residual Scale Estimates
Selection:
```

Plot selections 3 and 4 are illustrated in Figures 9.17 and 9.18. From the graphs of the rolling estimate it is clear that $\hat{\alpha}$ is significantly positive only at the very beginning of the sample. The $\hat{\beta}$ values are near unity for most windows and increase sharply at the end of the sample. However, the large standard errors make it difficult to determine if β is really changing over time. The residual scale estimates, $\hat{\sigma}$, increase sharply after 1999. This implies that the magnitude of the non-market risk in Microsoft increased after 1999.

In `rollOLS`, the optional argument `incr` sets the number of observations between the rolling blocks of data of length determined by `width`. Therefore, rolling regressions may be computed for arbitrary overlapping and non-overlapping blocks of data. For example, consider computing the CAPM estimates for Microsoft over the two non-overlapping but adjacent subsamples, February 1990 - June 1995 and July 1996 - November 2000 using `rollOLS`:

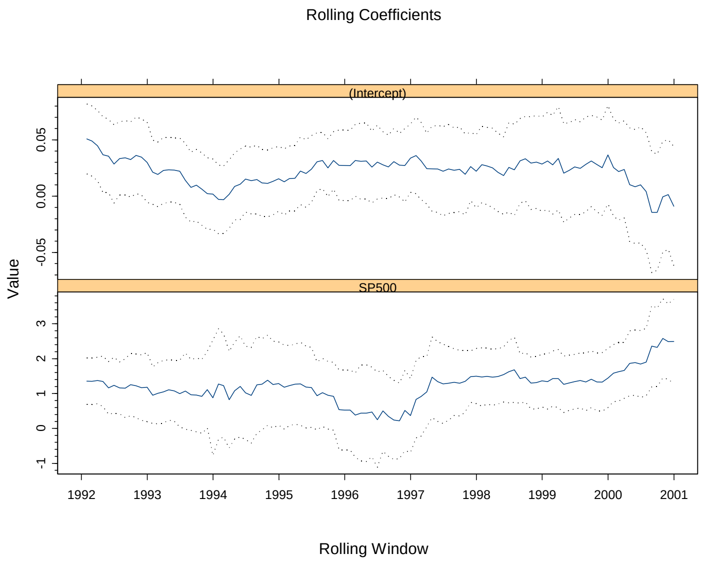


FIGURE 9.17. Rolling regression estimates of CAPM coefficients $\hat{\alpha}$ and $\hat{\beta}$ for Microsoft.

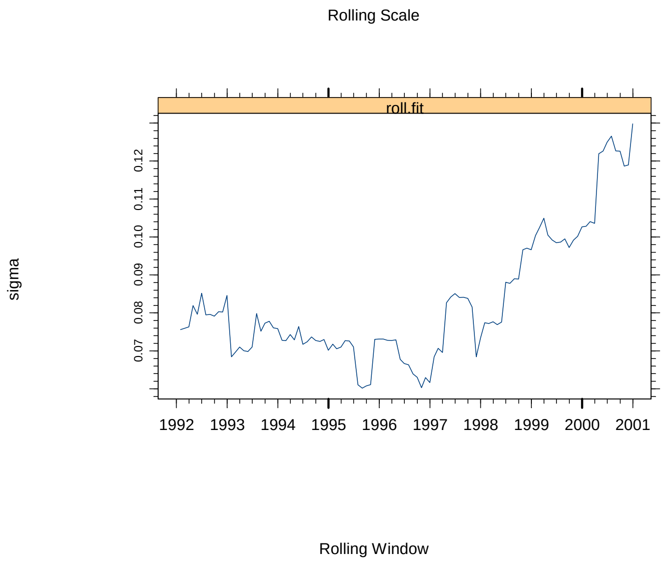


FIGURE 9.18. Rolling regression estimates of CAPM residual standard error for Microsoft.

```
> roll.fit2 = rollOLS(MSFT~SP500, data=excessReturns.ts,
+ width=65, incr=65)
Rolling Window #1: Total Rows = 65
Rolling Window #2: Total Rows = 130
> summary(roll.fit2)

Call:
rollOLS(formula = MSFT ~SP500, data = excessReturns.ts,
width = 65, incr = 65)

Rolling Windows:
 number width increment
      2    65        65
Time period: from Feb 1990 to Dec 2000

Coefficient: (Intercept)
           Value Std. Error t value Pr(>|t|)
Jun 1995 0.02765   0.009185  3.0100 0.003755
Nov 2000 0.01106   0.012880  0.8585 0.393851

Coefficient: SP500
         Value Std. Error t value   Pr(>|t|)
Jun 1995 1.339     0.2712   4.937 6.125e-006
Nov 2000 1.702     0.2813   6.050 8.739e-008
```

9.4.2 Rolling Predictions and Backtesting

Rolling regressions may be used to evaluate a model's predictive performance based on historical data using a technique commonly referred to as *backtesting*. To illustrate, consider the rolling regression model (9.10). The "out-of-sample" predictive performance of (9.10) is based on the rolling h-step predictions and prediction errors

$$\hat{y}_{t+h|t} = \mathbf{x}_{t+h}'\hat{\boldsymbol{\beta}}_t(n), \tag{9.12}$$

$$\hat{\varepsilon}_{t+h|t} = y_{t+h} - \hat{y}_{t+h|t} = y_{t+h} - \mathbf{x}_{t+h}'\hat{\boldsymbol{\beta}}_t(n) \tag{9.13}$$

The predictions are "out-of-sample" because $\hat{\boldsymbol{\beta}}_t(n)$ only uses data up to time t, whereas the predictions are for observations at times $t+h$ for $h>0$. The rolling predictions are adaptive since $\hat{\boldsymbol{\beta}}_t(n)$ is updated when t is increased. When $h=1$ there are $T-n$ rolling 1-step predictions $\{\hat{y}_{n+1|n}, \hat{y}_{n+2|n+1}, \ldots, \hat{y}_{T|T-1}\}$, when $h=2$ there are $T-n-1$ rolling 2-step predictions $\{\hat{y}_{n+2|n}, \hat{y}_{n+3|n+1}, \ldots, \hat{y}_{T|T-2}\}$ and so on.

Forecast Evaluation Statistics

The rolling forecasts (9.12) may be evaluated by examining the properties of the rolling forecast errors (9.13). Common evaluation statistics are

$$\begin{aligned}
\text{ME} &= \frac{1}{T-n-h+1}\sum_{t=n}^{T-h}\hat{\varepsilon}_{t+h|t} \qquad (9.14)\\
\text{MSE}(h) &= \frac{1}{T-n-h+1}\sum_{t=n}^{T-h}\hat{\varepsilon}_{t+h|t}^{2}\\
\text{RMSE}(h) &= \sqrt{MSE(h)}\\
\text{MAE}(h) &= \frac{1}{T-n-h+1}\sum_{t=n}^{T-h}\left|\hat{\varepsilon}_{t+h|t}\right|\\
\text{MAPE}(h) &= \frac{1}{T-n-h+1}\sum_{t=n}^{T-h}\left|\frac{\hat{\varepsilon}_{t+h|t}}{y_{t+h}}\right|
\end{aligned}$$

The first measure evaluates the bias of the forecasts, and the other measures evaluate bias and precision.

Example 55 *Backtesting the CAPM*

Consider again the estimation of the CAPM (9.11) for Microsoft using rolling regression. The rolling regression information is contained in the "`rollOLS`" object `roll.fit`. The rolling h-step predictions (9.12) may be computed using the generic `predict` method. For example, the rolling 1-step forecasts are computed as

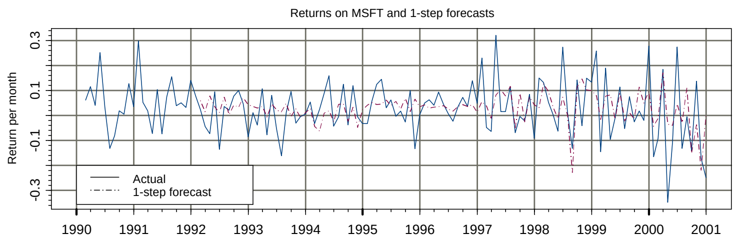
```
> roll.pred = predict(roll.fit,n.step=1)
> class(roll.pred)
[1] "listof"
> names(roll.pred)
[1] "1-Step-Ahead Forecasts"
```

The argument `n.step` determines the step length of the predictions. The object `roll.pred` is of class "`listof`" whose list component is a "`timeSeries`" object containing the rolling 1-step predictions:

```
> roll.pred[[1]]
 Positions           1
 Feb 1992   0.05994784
 Mar 1992   0.01481398
...
 Dec 2000  -0.00049354
```

The prediction errors (9.13) are then computed as

```
ehat.1step = excessReturns.ts[,"MSFT"]-roll.pred[[1]]
```



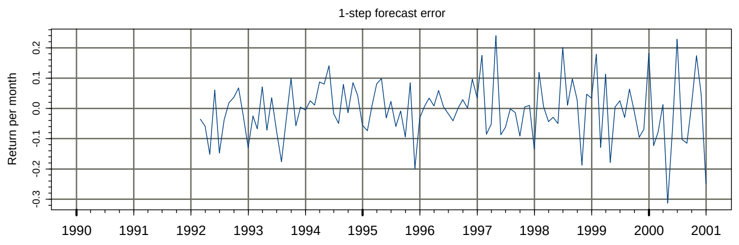


FIGURE 9.19. Monthly returns on Microsoft, 1-step rolling forecasts and forecast errors.

The monthly returns on Microsoft, 1-step forecasts and 1-step forecast errors are shown in Figure 9.19 created by

```
> par(mfrow=c(2,1))
> plot(excessReturns.ts[,"MSFT"], roll.pred[[1]],
+ main="Returns on MSFT and 1-step forecasts",
+ plot.args=list(lty=c(1,3)))
> legend(0, -0.2, legend=c("Actual","1-step forecast"),
+ lty=c(1,3))
> plot(ehat.1step,main="1-step forecast error")
```

The forecast evaluation statistics (9.14) may be computed as

```
> me.1step = mean(ehat.1step)
> mse.1step = as.numeric(var(ehat.1step))
> rmse.1step = sqrt(mse.1step)
> mae.1step = mean(abs(ehat.1step))
> mape.1step = mean(abs(ehat.1step/excessReturns.ts[,"MSFT"]),
+ na.rm=T)
```

To compute just the 2-step forecasts, specify `n.step=2` in the call to `predict`. To compute the 1-step and 2-step forecasts specify `n.step=1:2`

```
> roll.pred.12 = predict(roll.fit,n.steps=1:2)
> names(roll.pred.12)
[1] "1-Step-Ahead Forecasts" "2-Step-Ahead Forecasts"

> roll.pred.12[[1]]
 Positions           1
 Feb 1992   0.05994784
 Mar 1992   0.01481398
...
 Dec 2000  -0.00049354

> roll.pred.12[[2]]
 Positions          1
 Mar 1992   0.0165764
 Apr 1992   0.0823867
...
 Dec 2000  -0.0025076
```

Since the 1-step and 2-step predictions are components of the list object `roll.pred.12`, the `S-PLUS` function `lapply` may be used to simplify the computation of the forecast errors and evaluation statistics. To illustrate, supplying the user-defined function

```
> make.ehat = function(x,y) {
+    ans = y - x
+    ans[!is.na(ans),]
+ }
```

to `lapply` creates a "`named`" object containing the 1-step and 2-step forecasts errors as components:

```
> ehat.list = lapply(roll.pred.12, make.ehat,
+ excessReturns.ts[,"MSFT"])
> names(ehat.list)
[1] "1-Step-Ahead Forecasts" "2-Step-Ahead Forecasts"
```

The forecast evaluation statistics (9.14) may be computed for each component of `ehat.list` using `lapply` with the following user-defined function

```
> make.errorStats = function(x){
+    me = mean(x)
+    mse = as.numeric(var(x))
+    rmse = sqrt(mse)
+    mae = mean(abs(x))
+    ans = list(ME=me,MSE=mse,RMSE=rmse,MAE=mae)
+    ans
+ }

> errorStat.list = lapply(ehat.list,make.errorStats)
> unlist(errorStat.list)
 1-Step-Ahead Forecasts.ME 1-Step-Ahead Forecasts.MSE
                 -0.006165                   0.009283

 1-Step-Ahead Forecasts.RMSE 1-Step-Ahead Forecasts.MAE
                     0.09635                    0.07207

 2-Step-Ahead Forecasts.ME 2-Step-Ahead Forecasts.MSE
                 -0.007269                   0.009194

 2-Step-Ahead Forecasts.RMSE 2-Step-Ahead Forecasts.MAE
                     0.09589                    0.07187
```

The S-PLUS function sapply may be used instead of lapply to summarize the forecast error evaluation statistics:

```
> sapply(ehat.list,make.errorStats)
     1-Step-Ahead Forecasts 2-Step-Ahead Forecasts
[1,] -0.006165              -0.007269
[2,] 0.009283               0.009194
[3,] 0.09635                0.09589
[4,] 0.07207                0.07187
```

Comparing Predictive Accuracy

Backtesting is often used to compare the forecasting accuracy of two or more competing models. Typically, the forecast evaluation statistics (9.14) are computed for each model, and the model that produces the smallest set of statistics is judged to be the better model. Recently, Diebold and Mariano (1995) proposed a simple procedure using rolling h-step forecast errors for statistically determining if one model's forecast is more accurate than another's. Let $\hat{\varepsilon}^1_{t+h|t}$ and $\hat{\varepsilon}^2_{t+h|t}$ denote the h-step forecast errors from two competing models, and let N denote the number of h-step forecasts. The accuracy of each forecast is measured by a particular forecast evaluation or *loss function*

$$L(\hat{\varepsilon}^i_{t+h|t}),\ i = 1,2$$

Two popular loss functions are the *squared error loss* $L(\hat{\varepsilon}^i_{t+h|t}) = \left(\hat{\varepsilon}^i_{t+h|t}\right)^2$ and *absolute error loss* $L(\hat{\varepsilon}^i_{t+h|t}) = \left|\hat{\varepsilon}^i_{t+h|t}\right|$. To determine if one model forecasts better than another Diebold and Mariano (1995) suggested computing the loss differential

$$d_t = L(\hat{\varepsilon}^1_{t+h|t}) - L(\hat{\varepsilon}^2_{t+h|t})$$

and testing the null hypothesis of equal forecasting accuracy

$$H_0 : E[d_t] = 0$$

The Diebold-Mariano test statistic is the simple ratio

$$\mathrm{DM} = \frac{\bar{d}}{\widehat{\mathrm{lrv}}(\bar{d})^{1/2}} \tag{9.15}$$

where

$$\bar{d} = \frac{1}{N}\sum_{t=1}^{N} d_t$$

is the average loss differential, and $\widehat{\mathrm{lrv}}(\bar{d})$ is a consistent estimate of the long-run asymptotic variance of $\bar{d}$. Diebold and Mariano suggest computing $\widehat{\mathrm{lrv}}(\bar{d})$ using the Newey-West nonparametric estimator with a rectangular weight function and a lag truncation parameter equal to the forecast step length, h, less one. Diebold and Mariano showed that under the null hypothesis of equal predictive accuracy the DM statistic is asymptotically distributed $N(0,1)$.

Example 56 *Backtesting regression models for predicting asset returns*

To illustrate model comparison and evaluation by backtesting, consider the problem of predicting the annual real return on S&P 500 index using two different valuation ratios. The regression model is of the form

$$r_t = \alpha + \beta x_{t-1} + \varepsilon_t \tag{9.16}$$

where r_t denotes the natural logarithm of the annual real total return on S&P 500 index and x_t denotes the natural logarithm of a valuation ratio. The first valuation ratio considered is the dividend/price ratio and the second ratio is the earning/price ratio. The data are constructed from the `S+FinMetrics` “`timeSeries`” `shiller.annual` as follows:

```
> colIds(shiller.annual)
 [1] "price"         "dividend"      "earnings"
 [4] "cpi"           "real.price"    "real.dividend"
 [7] "real.earnings" "pe.10"         "dp.ratio"
[10] "dp.yield"
> # compute log of real data
> ln.p = log(shiller.annual[,"real.price"])
> colIds(ln.p) = "ln.p"
> ln.dpratio = log(dp.ratio)
> colIds(ln.dpratio) = "ln.dpratio"
> ln.epratio = -log(shiller.annual[,"pe.10"])
> ln.epratio = ln.epratio[!is.na(ln.epratio),]
> colIds(ln.epratio) = "ln.epratio"
> # compute cc real total returns - see CLM pg. 261
> ln.r = diff(ln.p) + log(1+exp(ln.dpratio[-1,]))
> colIds(ln.r) = "ln.r"
> stock.ts = seriesMerge(ln.p,ln.d,ln.dpratio,
+ ln.epratio,ln.r,pos=positions(ln.epratio))
> start(stock.ts)
[1] Dec 1881
> end(stock.ts)
[1] Dec 2000
```

Rolling regression estimates of (9.16) with the two valuation ratios using a 50 year window incremented by 1 year are computed as

```
> roll.dp.fit = rollOLS(ln.r~tslag(ln.dpratio),data=stock.ts,
+ width=50,incr=1)
Rolling Window #1: Total Rows = 50
Rolling Window #2: Total Rows = 51
...
> roll.ep.fit = rollOLS(ln.r~tslag(ln.epratio),data=stock.ts,
+ width=50,incr=1)
Rolling Window #1: Total Rows = 50
Rolling Window #2: Total Rows = 51
...
Rolling Window #70: Total Rows = 119
```

Figures 9.20 and 9.21 show the rolling coefficient estimates from the two models along with standard error bands. The rolling estimates of β for the two models are similar. For both models, the strongest evidence for return predictability occurs between 1960 and 1990. The value of β for the earning/price model appears to be different from zero during more periods than the value of β for the dividend/price model.

The rolling h-step predictions for $h = 1, \ldots, 5$ and prediction errors are

```
> roll.dp.pred = predict(roll.dp.fit,n.steps=1:5)
> roll.ep.pred = predict(roll.ep.fit,n.steps=1:5)
> ehat.dp.list = lapply(roll.dp.pred,make.ehat,
+ stock.ts[,"ln.r"])
> ehat.ep.list = lapply(roll.ep.pred,make.ehat,
+ stock.ts[,"ln.r"])
```

The forecast evaluation statistics are

```
> errorStats.dp.list = lapply(ehat.dp.list,make.errorStats)
> errorStats.ep.list = lapply(ehat.ep.list,make.errorStats)
> tmp = cbind(unlist(errorStats.dp.list),
+ unlist(errorStats.ep.list))
> colIds(tmp) = c("D/P","E/P")
> tmp
numeric matrix: 20 rows, 2 columns.
                                D/P     E/P
  1-Step-Ahead Forecasts.ME 0.03767 0.01979
```

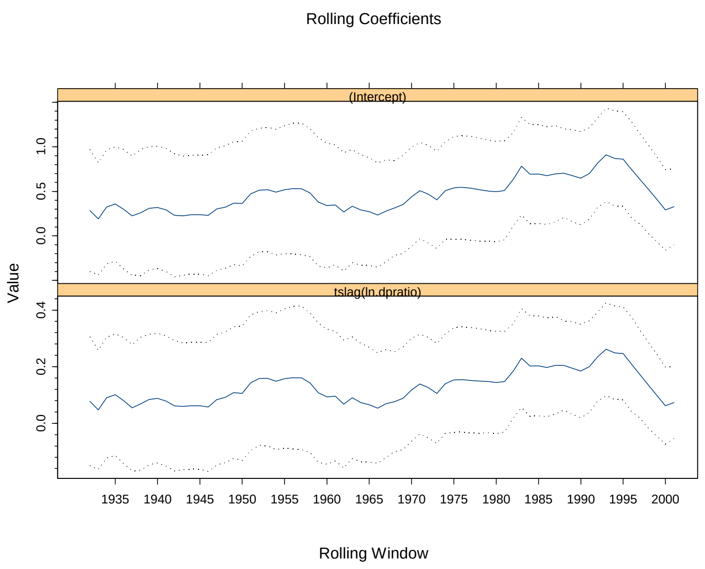


FIGURE 9.20. 50 year rolling regression estimates of (9.16) using dividend/price ratio.

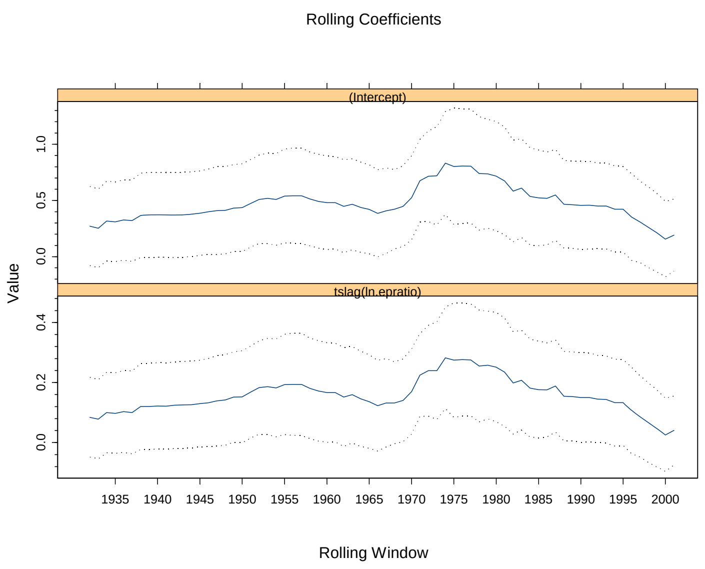


FIGURE 9.21. 50 year rolling regression estimates of (9.16) using earning/price.

```
 1-Step-Ahead Forecasts.MSE 0.03150 0.03139
1-Step-Ahead Forecasts.RMSE 0.17749 0.17718
 1-Step-Ahead Forecasts.MAE 0.14900 0.14556
  2-Step-Ahead Forecasts.ME 0.04424 0.02334
 2-Step-Ahead Forecasts.MSE 0.03223 0.03205
2-Step-Ahead Forecasts.RMSE 0.17952 0.17903
 2-Step-Ahead Forecasts.MAE 0.15206 0.14804
  3-Step-Ahead Forecasts.ME 0.04335 0.02054
 3-Step-Ahead Forecasts.MSE 0.03203 0.03180
3-Step-Ahead Forecasts.RMSE 0.17898 0.17832
 3-Step-Ahead Forecasts.MAE 0.14993 0.14731
  4-Step-Ahead Forecasts.ME 0.04811 0.02397
 4-Step-Ahead Forecasts.MSE 0.03292 0.03248
4-Step-Ahead Forecasts.RMSE 0.18143 0.18022
 4-Step-Ahead Forecasts.MAE 0.15206 0.14855
                                D/P     E/P
  5-Step-Ahead Forecasts.ME 0.04707 0.02143
 5-Step-Ahead Forecasts.MSE 0.03339 0.03255
5-Step-Ahead Forecasts.RMSE 0.18272 0.18043
 5-Step-Ahead Forecasts.MAE 0.15281 0.14825
```

The forecast evaluation statistics are generally smaller for the model using the earning/price ratio. The Diebold-Mariano statistics based on squared error and absolute error loss functions may be computed using

```
> for (i in 1:5) {
+   d.mse[,i] = ehat.dp.list[[i]]^2 - ehat.ep.list[[i]]^2
+   DM.mse[i] = mean(d.mse[,i])/sqrt(asymp.var(d.mse[,i],
+     bandwidth=i-1,window="rectangular"))
+   d.mae[,i] = abs(ehat.dp.list[[i]]) - abs(ehat.ep.list[[i]])
+   DM.mae[i] = mean(d.mae[,i])/sqrt(asymp.var(d.mae[,i],
+     bandwidth=i-1,window="rectangular"))
+ }
> names(DM.mse) = names(ehat.dp.list)
> names(DM.mae) = names(ehat.dp.list)
> cbind(DM.mse,DM.mae)
                        DM.mse  DM.mae
1-Step-Ahead Forecasts 0.07983 0.07987
2-Step-Ahead Forecasts 0.09038 0.08509
3-Step-Ahead Forecasts 0.07063 0.05150
4-Step-Ahead Forecasts 0.08035 0.06331
5-Step-Ahead Forecasts 0.07564 0.06306
```

Since the DM statistics are asymptotically standard normal, one cannot reject the null hypothesis of equal predictive accuracy at any reasonable significance level based on the 1-step through 5-step forecast errors for the two models.

9.5 Rolling Analysis of General Models Using the S+FinMetrics Function roll

The S-PLUS aggregateSeries function is appropriate for rolling analysis of simple functions and the S+FinMetrics function rollOLS handles rolling regression. The S+FinMetrics function roll is designed to perform rolling analysis of general S-PLUS modeling functions that take a formula argument describing the relationship between a response and explanatory variables and where the data, usually a data frame or "timeSeries" object with a data frame in the data slot, is supplied explicitly in a data argument. The arguments expected by roll are

```
> args(roll)
function(FUN, data, width, incr = 1, start = NULL, end =
NULL, na.rm = F, save.list = NULL, arg.data =
"data", trace = T, ...)
```

where FUN is the S-PLUS modeling function to be applied to each rolling window, data is the data argument to FUN which must be either a data frame or a "timeSeries" with a data frame in the data slot,width specifies the width of the rolling window and incr determines the increment size by which the windows are rolled through the sample. The argument save.list specifies the components of the object returned by FUN to save in the object returned by roll. If FUN requires more arguments in addition to data, for example a formula relating a response to a set of explanatory variables, then these arguments should be supplied in place of .... The use of roll is illustrated with the following examples.

Example 57 *Rolling regression*

In this example, the 24-month rolling regression estimation of the CAPM for Microsoft using the "timeSeries" excessReturns.ts is repeated using the S+FinMetrics function roll with FUN=OLS. OLS requires a formula argument for model specification and a data frame or "timeSeries" in data argument. The 24 month rolling CAPM estimates using roll are

```
> roll.fit = roll(FUN=OLS, data=excessReturns.ts,
+ width=24, incr=1, formula=MSFT~SP500)
Rolling Window #1: Total Rows = 24
Rolling Window #2: Total Rows = 25
...
Rolling Window #108: Total Rows = 131
```

```
> class(roll.fit)
[1] "roll"
```

The return `roll.fit` is an object of class "`roll`" for which there are no specific method functions. Since the `data` argument `excessReturns.ts` is a "`timeSeries`", the default components of `roll.fit` are the positions of the rolling windows and "`timeSeries`" objects containing the components that are produced by `OLS` for each of the windows:

```
> names(roll.fit)
 [1] "R"         "coef"      "df.resid"  "fitted"
 [5] "residuals" "assign"    "contrasts" "ar.order"
 [9] "terms"     "call"      "positions"
> class(roll.fit$coef)
[1] "timeSeries"
> nrow(roll.fit$coef)
[1] 108
> class(roll.fit$residuals)
[1] "timeSeries"
> nrow(roll.fit$residuals)
[1] 108
```

The first column of the "`timeSeries`" `roll.fit$coef` contains the rolling intercept estimates, and the second column contains the rolling slope estimates.

```
> roll.fit$coef[1:2,]
 Positions       1     2
 Jan 1992  0.05075 1.354
 Feb 1992  0.04897 1.353
```

The rows of the "`timeSeries`" `roll.fit$residuals` contain the residuals for the OLS fit on each 24-month window

```
> roll.fit$residuals[1:2,]
 Positions        1       2       3        4         5
 Jan 1992  0.007267 0.04021 0.03550 0.085707  0.004596
 Feb 1992  0.042014 0.03726 0.08757 0.006368 -0.164498

 ...

       24
  0.05834
 -0.03393
```

If only some of the components of `OLS` are needed for each rolling window, these components may be specified in the optional argument `save.list`. For example, to retain only the components `coef` and `residuals` over the rolling windows specify `save.list=c("coef","residuals")` in the call to `roll`:

```
> roll.fit = roll(FUN=OLS, data=excessReturns.ts,
+ width=24, incr=1, formula=MSFT~SP500,
+ save.list=c("coef","residuals"), trace=F)
> names(roll.fit)
[1] "coef"      "residuals" "call"      "positions"
```

9.6 References

Alexander, C. (2001). *Market Models: A Guide to Financial Data Analysis*. John Wiley & Sons, Chichester, UK.

Bauer, R.J. and J.R. Dahlquist (1999). *Techincal Market Indicators: Analysis & Performance*. John Wiley & Sons, New York.

Banerjee, A. R. Lumsdaine and J.H. Stock (1992). "Recursive and Sequential Tests of the Unit Root and Trend Break Hypothesis: Theory and International Evidence," *Journal of Business and Economic Statistics*, 10(3), 271-288.

Colby, R.W. and T.A Meyers (1988). *The Encyclopedia of Technical Market Indicators*. McGraw-Hill, New York.

Dacorogna, M.M., R. Gençay, U.A. Müller, R.B. Olsen, and O.V. Pictet (2001). *An Introduction to High-Frequency Finance*. Academic Press, San Diego.

Diebold, F.X. and R.S. Mariano (1995). "Comparing Predictive Accuracy," *Journal of Business and Economic Statistics*, 13, 253-263.

Shiller, R. (1998). *Irrational Exuberance*. Princeton University Press, Princeton, NJ.

Zumbach, G.O., and U.A. Müller (2001). "Operators on Inhomogeneous Time Series," *International Journal of Theoretical and Applied Finance*, 4, 147-178.

10
Systems of Regression Equations

10.1 Introduction

The previous chapters dealt with models for univariate financial time series. In many applications, it is desirable to model the joint behavior of multiple time series because of possible efficiency gains to the joint estimation of a system of time series models. For example, there may be complex interactions between the variables and/or the error structure across models. Univariate models cannot capture these interactions whereas multivariate models can. Furthermore, many equilibrium models for asset returns, like the capital asset pricing model (CAPM) or the arbitrage price model (APT), imply parameter restrictions that are common to the model representation of all assets. Hence, the testing of equilibrium asset pricing models requires the testing of cross equation parameter constraints, and the proper estimation of these models would impose these cross equation restrictions.

This chapter introduces methods for modeling and analyzing systems of linear and nonlinear regression equations. Section 10.2 describes Zellner's seemingly unrelated regression (SUR) system of regression equations that may be linked through common regressors, correlated error terms, or cross equation parameter restrictions. Section 10.3 describes the specification and estimation of linear SUR models and gives examples using the `S+FinMetrics` function `SUR`. Section 10.4 describes the specification and estimation of nonlinear SUR models and gives examples using the `S+FinMetrics` function `NLSUR`.

The SUR model was developed by Theil (1961) and Zellner (1962) and is described in most econometric textbooks. The nonlinear SUR model was developed by Gallant (1974). Greene (2000) gives a general overview of linear and nonlinear SUR models and Srivastava and Giles (1987) provides a thorough treatment. Burmeister and McElroy (1986) and Campbell, Lo, and MacKinlay (1997) describe the estimation and testing of systems of asset pricing models using SUR and nonlinear SUR models.

10.2 Systems of Regression Equations

Many applications in economics and finance involve a *system of linear regression equations* of the form

$$\begin{aligned} \mathbf{y}_1 &= \mathbf{X}_1\boldsymbol{\beta}_1 + \boldsymbol{\varepsilon}_1 \\ \mathbf{y}_2 &= \mathbf{X}_2\boldsymbol{\beta}_2 + \boldsymbol{\varepsilon}_2 \\ &\vdots \\ \mathbf{y}_M &= \mathbf{X}_M\boldsymbol{\beta}_M + \boldsymbol{\varepsilon}_M \end{aligned} \tag{10.1}$$

where $\mathbf{y}_i$ is a $(T \times 1)$ vector of dependent variables, $\mathbf{X}_i$ is a $(T \times k_i)$ matrix of explanatory variables and $\boldsymbol{\varepsilon}_i$ is a $(T \times 1)$ vector of errors for equations $i = 1, ..., M$. It is assumed that each $\mathbf{X}_i$ is exogenous, i.e., uncorrelated with $\boldsymbol{\varepsilon}_i$. Depending on the application, each $\mathbf{X}_i$ may be distinct or there may be common regressors across equations. The equations in (10.1) are potentially linked either through the covariance structure of the errors or through cross equation restrictions on the elements of $\boldsymbol{\beta}_i$ $(i = 1, ..., M)$, and estimation of the entire system generally produces more efficient estimates than the estimation of each equation individually.

Economic theory often implies a *system of nonlinear regression equations* of the form

$$\begin{aligned} \mathbf{y}_1 &= \mathbf{f}_1(\boldsymbol{\beta}_1, \mathbf{X}_1) + \boldsymbol{\varepsilon}_1 \\ \mathbf{y}_2 &= \mathbf{f}_2(\boldsymbol{\beta}_2, \mathbf{X}_2) + \boldsymbol{\varepsilon}_2 \\ &\vdots \\ \mathbf{y}_M &= \mathbf{f}_M(\boldsymbol{\beta}_M, \mathbf{X}_M) + \boldsymbol{\varepsilon}_M \end{aligned}$$

where $\mathbf{f}_i(\boldsymbol{\beta}_i, \mathbf{X}_i)$ is a $(T \times 1)$ vector containing the nonlinear function values $f_i(\boldsymbol{\beta}_i, \mathbf{x}_{it})$ for equations $i = 1, \ldots, M$. The functions f_i may be the same or different across equations, the error structures may be linked, and there may be cross equation restrictions on the elements of $\boldsymbol{\beta}_i$ $(i = 1, ..., M)$.

Some common applications in finance are illustrated below.

Example 58 *Exchange rate regressions*

Consider the system of M exchange rate regressions

$$\Delta s_{i,t+k} = \alpha_i + \gamma_i(f_{i,t}^k - s_{i,t}) + \varepsilon_{i,t+k},\ i = 1, \ldots, M \tag{10.2}$$

where $s_{i,t+k}$ represents the natural log of the spot exchange exchange rate for currency i (relative, say, to the U.S. dollar) at time $t+k$ and $f_{i,t}^k$ denotes the natural log of the forward exchange rate at time t for a forward contract in currency i that will deliver at time $t+k$. In terms of the previous notation for the system of linear regression equations, $y_{it} = s_{i,t+k}$, $\mathbf{x}_{it} = (1, f_{i,t}^k - s_{i,t})'$ and $\boldsymbol{\beta}_i = (\alpha_i, \gamma_i)'$, the only common regressor in the system is a vector of ones. The error terms, $\varepsilon_{i,t+k}$, are likely to be correlated contemporaneously across equations due to common random shocks affecting all exchange rates. That is, $E[\boldsymbol{\varepsilon}_i\boldsymbol{\varepsilon}_j'] = \sigma_{ij}\mathbf{I}_T$ where $E[\varepsilon_{it}\varepsilon_{js}] = \sigma_{ij}$ for $t = s$ and 0 otherwise. In the present context, this across equation correlation can be used to increase the efficiency of the parameter estimates for each equation.

Example 59 *The capital asset pricing model with a risk-free asset*

Consider the excess return single index model regression

$$R_{it} - r_{ft} = \alpha_i + \beta_i(R_{Mt} - r_{ft}) + \varepsilon_{it},\ i = 1, \ldots, M$$

where R_{it} denotes the return on asset i at time t, r_{ft} denotes the return on a risk-free asset, R_{Mt} denotes the return on an proxy for the "market portfolio" and ε_{it} denotes the residual return not explained by the "market" for asset i. In terms of the previous notation for the system of regression equations, $y_{it} = R_{it} - r_{ft}, \mathbf{x}_{it} = (1, R_{Mt} - r_{ft})'$, and $\boldsymbol{\beta}_i = (\alpha_i, \beta_i)'$ so that all regression equations share the same regressors. It is likely that the residual returns, ε_{it}, are contemporaneously correlated across assets due to common shocks not related to the "market". That is, $E[\varepsilon_i\varepsilon_j'] = \sigma_{ij}\mathbf{I}_T$ where $E[\varepsilon_{it}\varepsilon_{js}] = \sigma_{ij}$ for $t = s$ and 0 otherwise. However, unless there are across equation restrictions on $\boldsymbol{\beta}_i$, the fact that $\mathbf{x}_{it}$ is the same for each equation means that there will be no efficiency gain in estimating the parameters from exploiting the across equation error correlation. The capital asset pricing model (CAPM) imposes the restriction $\alpha_i = 0$ for all assets i so that $E[R_{it}] - r_{ft} = \beta_i(E[R_{Mt} - r_{ft})$. Testing the CAPM therefore involves a joint test of many cross equation zero restrictions.

Example 60 *The capital asset pricing model without a risk-free asset*

The CAPM formulation above assumes the existence of a risk-free asset. If there is no risk-free asset, then Black (1972) showed that the CAPM takes the form

$$E[R_{it}^{\text{real}}] - \gamma = \beta_i(E[R_{Mt}^{\text{real}}] - \gamma)$$

where R_{it}^{real} denotes the real return on asset i, R_{Mt}^{real} denotes the real return on the market, and γ denotes the unobservable return on a zero-beta portfolio. The Black form of the CAPM may be estimated from the system of nonlinear regression equations

$$R_{it}^{\text{real}} = (1-\beta_i)\gamma + \beta_i R_{Mt}^{\text{real}} + \varepsilon_{it},\ i = 1,\ldots,M \tag{10.3}$$

In terms of the above notation for systems of nonlinear regression equations, Black's restricted CAPM has $y_{it} = R_{it}^{\text{real}}$, $\mathbf{x}_{it} = (1, R_{Mt}^{\text{real}})'$, $\boldsymbol{\beta}_i = (\gamma, \beta_i)'$ and $f_i(\boldsymbol{\beta}_i, \mathbf{x}_{it}) = (1-\beta_i)\gamma + \beta_i R_{Mt}^{\text{real}}$. Notice that the parameter γ is common across all equations. The Black form of the CAPM may be tested by estimating the unrestricted system

$$R_{it}^{\text{real}} = \alpha_i + \beta_i R_{Mt}^{\text{real}} + \varepsilon_{it},\ i = 1,\ldots,M \tag{10.4}$$

and testing the M nonlinear cross equation restrictions $\alpha_i = (1-\beta_i)\gamma$.

10.3 Linear Seemingly Unrelated Regressions

The *seemingly unrelated regression* (SUR) model due to Theil (1961) and Zellner (1962) is the unrestricted system of M linear regression equations

$$\mathbf{y}_i = \mathbf{X}_i\boldsymbol{\beta}_i + \boldsymbol{\varepsilon}_i,\ i = 1,\ldots,M \tag{10.5}$$

where $\mathbf{y}_i$ is $(T\times 1)$, $\mathbf{X}_i$ is $(T\times k_i)$, $\boldsymbol{\beta}_i$ is $(k\times 1)$ and $\boldsymbol{\varepsilon}_i$ is $(T\times 1)$. The error terms are assumed to be contemporaneously correlated across equations but temporally uncorrelated: $E[\varepsilon_{it}\varepsilon_{js}] = \sigma_{ij}$ for $t = s$; 0 otherwise.

The M equations may be stacked to form the *giant regression* model

$$\begin{pmatrix} \mathbf{y}_1 \\ \vdots \\ \mathbf{y}_M \end{pmatrix} = \begin{pmatrix} \mathbf{X}_1 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \mathbf{X}_M \end{pmatrix} \begin{pmatrix} \boldsymbol{\beta}_1 \\ \vdots \\ \boldsymbol{\beta}_M \end{pmatrix} + \begin{pmatrix} \boldsymbol{\varepsilon}_1 \\ \vdots \\ \boldsymbol{\varepsilon}_M \end{pmatrix}$$

or

$$\underline{\mathbf{y}} = \underline{\mathbf{X}}\boldsymbol{\beta} + \underline{\boldsymbol{\varepsilon}} \tag{10.6}$$

where $\underline{\mathbf{y}}$ is $(MT\times 1)$, $\underline{\mathbf{X}}$ is $(MT\times K)$, $\boldsymbol{\beta}$ is $(K\times 1)$ and $\underline{\boldsymbol{\varepsilon}}$ is $(MT\times 1)$. Here $K = \sum_{i=1}^{M} k_i$ is the total number of regressors across all equations. The error term in the giant regression has non-diagonal covariance matrix

$$\mathbf{V} = E[\underline{\boldsymbol{\varepsilon}}\underline{\boldsymbol{\varepsilon}}'] = \boldsymbol{\Sigma}\otimes\mathbf{I}_T \tag{10.7}$$

where the $(M\times M)$ matrix $\boldsymbol{\Sigma}$ has elements σ_{ij}.

10.3.1 Estimation

Since $\mathbf{V}$ is not diagonal, least squares estimation of $\boldsymbol{\beta}$ in the giant regression (10.6) is not efficient. The *generalized least squares* (GLS) estimator of $\boldsymbol{\beta}$,

$$\begin{aligned} \hat{\boldsymbol{\beta}}_{\text{GLS}} &= (\underline{\mathbf{X}}'\mathbf{V}^{-1}\underline{\mathbf{X}})^{-1}\underline{\mathbf{X}}'\mathbf{V}^{-1}\underline{\mathbf{y}} \\ &= (\underline{\mathbf{X}}'(\boldsymbol{\Sigma}^{-1}\otimes\mathbf{I}_T)\underline{\mathbf{X}})^{-1}\underline{\mathbf{X}}'(\boldsymbol{\Sigma}^{-1}\otimes\mathbf{I}_T)\underline{\mathbf{y}} \end{aligned} \tag{10.8}$$

is efficient.

It can be shown (e.g., Greene 2000, Chap. 15) that if $\mathbf{X}_i = \mathbf{X}$ for all equations $i = 1, ..., M$ (i.e., all equations have the same regressors), or if the error covariance matrix $\mathbf{V}$ is diagonal and there are no cross equation restrictions on the values of $\boldsymbol{\beta}_i$ then least squares estimation of (10.5) equation by equation produces the GLS estimator (10.8).

Feasible GLS Estimation

The GLS estimator of $\boldsymbol{\beta}$ is usually not feasible since the covariance matrix $\mathbf{\Sigma}$, and hence $\mathbf{V}$, is generally not known. However, in the SUR model the elements of $\mathbf{\Sigma}$ can be consistently estimated by least squares estimation of (10.5) equation by equation using

$$\begin{aligned}\hat{\sigma}_{ij} &= df_{ij}^{-1}\hat{\boldsymbol{\varepsilon}}_i'\hat{\boldsymbol{\varepsilon}}_j \qquad (10.9)\\ &= df_{ij}^{-1}(\mathbf{y}_i - \mathbf{X}_i\hat{\boldsymbol{\beta}}_i)'(\mathbf{y}_j - \mathbf{X}_j\hat{\boldsymbol{\beta}}_j),\end{aligned}$$

producing $\hat{\mathbf{\Sigma}}$. In (10.9), df_{ij} represents the degrees of freedom used for the estimate $\hat{\sigma}_{ij}$. See the online help for `SUR` for more details on the different options for choosing df_{ij}. The *feasible generalized least squares* estimator (FGLS) is

$$\hat{\boldsymbol{\beta}}_{\mathrm{FGLS}} = (\underline{\mathbf{X}}'(\hat{\mathbf{\Sigma}}^{-1} \otimes \mathbf{I}_T)\underline{\mathbf{X}})^{-1}\underline{\mathbf{X}}'(\hat{\mathbf{\Sigma}}^{-1} \otimes \mathbf{I}_T)\underline{\mathbf{y}} \qquad (10.10)$$

and its asymptotic variance is consistently estimated by

$$\widehat{\mathrm{avar}}(\hat{\boldsymbol{\beta}}_{\mathrm{FGLS}}) = (\underline{\mathbf{X}}'(\hat{\mathbf{\Sigma}}^{-1} \otimes \mathbf{I}_T)\underline{\mathbf{X}})^{-1}$$

The FGLS estimator (10.10) is asymptotically equivalent to the GLS estimator (10.8).

Tests of linear hypotheses of the form $\mathbf{R}\boldsymbol{\beta} = \mathbf{r}$, which may incorporate cross equation linear restrictions, may be computed in the usual way with the Wald statistic

$$\mathrm{Wald} = (\mathbf{R}\hat{\boldsymbol{\beta}}_{\mathrm{FGLS}} - \mathbf{r})'\left[\mathbf{R}\widehat{\mathrm{avar}}(\hat{\boldsymbol{\beta}}_{\mathrm{FGLS}})\mathbf{R}'\right]^{-1}(\mathbf{R}\hat{\boldsymbol{\beta}}_{\mathrm{FGLS}} - \mathbf{r}) \qquad (10.11)$$

which is asymptotically distributed chi-square with degrees of freedom equal to the number of restrictions being tested under the null.

Iterated Feasible GLS Estimation

The estimate of $\mathbf{\Sigma}$ in FGLS estimation uses the inefficient least squares estimate of $\boldsymbol{\beta}_i$. The *iterated* FGLS estimator repeats the construction of the FGLS estimator using an updated estimator of $\mathbf{\Sigma}$ based on the FGLS estimator (10.10). That is, at each iteration updated estimates of σ_{ij} are computed as

$$\hat{\sigma}_{ij,\mathrm{FGLS}} = df_{ij}^{-1}(\mathbf{y}_i - \mathbf{X}_i\hat{\boldsymbol{\beta}}_{i,\mathrm{FGLS}})'(\mathbf{y}_j - \mathbf{X}_j\hat{\boldsymbol{\beta}}_{j,\mathrm{FGLS}})$$

and the resulting updated estimator of $\mathbf{\Sigma}$ is used to recompute the FGLS estimator. This process is iterated until $\hat{\boldsymbol{\beta}}_{\mathrm{FGLS}}$ no longer changes. If the error terms for each equation are Gaussian, it can be shown that the iterated FGLS estimator of $\boldsymbol{\beta}$ using $df_{ij} = T$ is the maximum likelihood estimator (MLE). It should be noted, however, that iteration does not improve the asymptotic properties of the FGLS estimator.

Maximum Likelihood Estimation

Although the MLE of $\boldsymbol{\beta}$ may be obtained by iterating the FGLS estimator, it is often computationally more efficient to compute the MLE directly. To conveniently express the likelihood function for the SUR model it is necessary to re-express the SUR model by grouping the data horizontally by observations instead of vertically by equations. The SUR model expressed this way is given by

$$\mathbf{y}_t' = \mathbf{x}_t'\mathbf{\Pi} + \boldsymbol{\varepsilon}_t',\ t = 1, \ldots, T$$

where $\mathbf{y}_t' = (y_{1t}, y_{2t}, \ldots, y_{Mt})$ is $(1 \times M)$ vector of dependent variables, $\mathbf{x}_t' = (x_{1t}, x_{2t}, \ldots, x_{Kt})$ is the $(1 \times K)$ vector containing all of the unique explanatory variables, $\mathbf{\Pi} = [\boldsymbol{\pi}_1, \boldsymbol{\pi}_2, \ldots, \boldsymbol{\pi}_M]$ is a $(K \times M)$ matrix where the $(K \times 1)$ vector $\boldsymbol{\pi}_i$ contains the coefficients on $\mathbf{x}_t'$ for the ith equation, and $\boldsymbol{\varepsilon}_t' = (\varepsilon_{1t}, \varepsilon_{2t}, \ldots, \varepsilon_{Mt})$ is a $(1 \times M)$ vector of error terms with covariance matrix $\mathbf{\Sigma}$. Note that since the ith equation may not have all of the variables as regressors so that some of the values in $\boldsymbol{\pi}_i$ may be equal to zero.

The log-likelihood function for a sample of size T is

$$\ln L(\boldsymbol{\beta}, \mathbf{\Sigma}) = -\frac{MT}{2}\ln(2\pi) - \frac{T}{2}\ln|\mathbf{\Sigma}| - \frac{1}{2}\sum_{t=1}^{T}\boldsymbol{\varepsilon}_t'\mathbf{\Sigma}^{-1}\boldsymbol{\varepsilon}_t$$

where $\boldsymbol{\beta}$ represents the appropriate non-zero elements of $\mathbf{\Pi}$. The log-likelihood function may be concentrated with respect to $\mathbf{\Sigma}$ giving

$$\ln L(\boldsymbol{\beta}) = -\frac{MT}{2}\left(\ln(2\pi) + 1\right) - \frac{T}{2}\ln\left(|\mathbf{\Sigma}(\boldsymbol{\beta})|\right) \tag{10.12}$$

where

$$\mathbf{\Sigma}(\boldsymbol{\beta}) = T^{-1}\sum_{t=1}^{T}\boldsymbol{\varepsilon}_t\boldsymbol{\varepsilon}_t' \tag{10.13}$$

is the MLE for $\mathbf{\Sigma}$ given $\boldsymbol{\beta}$. Hence, the MLE for $\boldsymbol{\beta}$ solves

$$\min_{\boldsymbol{\beta}} \frac{1}{2}\ln\left(|\mathbf{\Sigma}(\boldsymbol{\beta})|\right).$$

and the resulting estimator $\hat{\boldsymbol{\beta}}_{\mathrm{mle}}$ is equivalent to the iterated feasible GLS estimator with $df_{ij} = T$.

Likelihood Ratio Tests

The form of the concentrated log-likelihood function (10.12), implies that likelihood ratio (LR) tests for hypotheses about elements of $\boldsymbol{\beta}$ have the simple form

$$\mathrm{LR} = T\left(\ln\left(|\boldsymbol{\Sigma}(\tilde{\boldsymbol{\beta}}_{\mathrm{mle}})|\right) - \ln\left(|\boldsymbol{\Sigma}(\hat{\boldsymbol{\beta}}_{\mathrm{mle}})|\right)\right) \tag{10.14}$$

where $\tilde{\boldsymbol{\beta}}_{\mathrm{mle}}$ denotes the MLE imposing the restrictions under the null being tested and $\hat{\boldsymbol{\beta}}_{\mathrm{mle}}$ denotes the unrestricted MLE. The LR statistic (10.14) is asymptotically distributed chi-square with degrees of freedom equal to the number of restrictions being tested under the null.

10.3.2 *Analysis of SUR Models with the `S+FinMetrics` Function `SUR`*

The `S+FinMetrics` function `SUR` may be used for the estimation of linear SUR models without cross equation restrictions. The arguments for `SUR` are

```
> args(SUR)
function(formulas, data, subset, na.rm = F, start = NULL, end
= NULL, method = "fit", contrasts = NULL, df = 1,
tol = 1e-006, iterate = F, trace = T, ...)
```

Generally, the two specified arguments are `formulas`, which is a list containing the formulas for each equation in the SUR model, and `data`, which must be either a data frame, or a “`timeSeries`” object with a data frame in the `data` slot. Formulas are specified in the usual way with the response variables on the left hand side of the character and explanatory variables on the right hand side. If the variables in `formulas` can be directly accessed, e.g. through an attached data frame, then the `data` argument may be skipped. The default fitting method is one-step (not iterated) feasible GLS as in (10.10). To specify iterated feasible GLS set the optional argument `iterate=T`. In this case, the `trace` option controls printing of the iteration count and the `tol` option specifies the numerical tolerance of the convergence criterion. The optional argument `df` specifies the degrees of freedom parameter df_{ij} for the computation of $\hat{\sigma}_{ij}$. If `df=1`, the feasible GLS estimates are unbiased; if `df=2`, the feasible GLS estimates are minimum MSE estimates; if `df=3`, the sample size T is used as the degree of freedom; if `df=4`, $[(T-k_i)(T-k_j)]^{1/2}$ is used as the degree of freedom for the (i,j) element of the residual covariance matrix. To reproduce the MLE under Gaussian errors set `iterate=T` and `df=3`.

`SUR` produces an object of class “`SUR`” for which there are `print`, `summary` and `plot` methods as well as extractor functions `residuals`, `fitted.values` (or `fitted`), `coef`, and `vcov`. The use of `SUR` is illustrated using the following examples.

Example 61 *Testing efficiency in foreign exchange markets*

Consider estimating the system of exchange rate regressions (10.2) using monthly data on six currencies relative to the US dollar over the period August 1978 through June 1996. The data are in the "`timeSeries`" object `surex1.ts`, which is constructed from the data in the "`timeSeries`" `lexrates.dat`. The variables in `surex1.ts` are

```
> colIds(surex1.ts)
 [1] "USCN.FP.lag1" "USCNS.diff"   "USDM.FP.lag1" "USDMS.diff"
 [5] "USFR.FP.lag1" "USFRS.diff"   "USIL.FP.lag1" "USILS.diff"
 [9] "USJY.FP.lag1" "USJYS.diff"   "USUK.FP.lag1" "USUKS.diff"
```

The variables with extensions `.FP.lag1` are one month forward premia, $f_{i,t}^1 - s_{i,t}$, and variables with extensions `.diff` are future returns on spot currency, $\Delta s_{i,t+1}$. The list of formulas for the regressions in the system (10.2) is created using

```
> formula.list = list(USCNS.diff~USCN.FP.lag1,
+ USDMS.diff~USDM.FP.lag1,
+ USFRS.diff~USFR.FP.lag1,
+ USILS.diff~USIL.FP.lag1,
+ USJYS.diff~USJY.FP.lag1,
+ USUKS.diff~USUK.FP.lag1)
```

The command to compute the feasible GLS estimator of the SUR system over the period August 1978 through June 1996 is

```
> sur.fit = SUR(formula.list, data=surex1.ts,
+ start="Aug 1978", in.format="%m %Y")
> class(sur.fit)
[1] "SUR"
```

As usual, the `print` method is invoked by typing the name of the object and gives basic output:

```
> sur.fit

Seemingly Unrelated Regression:

Eq. 1: USCNS.diff ~USCN.FP.lag1

Coefficients:
 (Intercept) USCN.FP.lag1
 -0.0031     -1.6626
```

```
Degrees of freedom: 215 total; 213 residual
Time period: from Aug 1978 to Jun 1996
Residual scale estimate: 0.0135

Eq. 2: USDMS.diff ~USDM.FP.lag1

Coefficients:
 (Intercept) USDM.FP.lag1
 0.0006      0.5096

Degrees of freedom: 215 total; 213 residual
Time period: from Aug 1978 to Jun 1996
Residual scale estimate: 0.0358

...

Eq. 6: USUKS.diff ~USUK.FP.lag1

Coefficients:
 (Intercept) USUK.FP.lag1
 -0.0035     -1.2963

Degrees of freedom: 215 total; 213 residual
Time period: from Aug 1978 to Jun 1996
Residual scale estimate: 0.0344

Log determinant of residual covariance: -47.935
```

In the above output, the log determinant of residual covariance is the quantity $\ln\left(|\Sigma(\hat{\boldsymbol{\beta}}_{\mathrm{FGLS}})|\right)$. The forward rate is an unbiased predictor of the future spot rate if the coefficient on the forward premium is equal to 1. The results above suggest that unbiasedness holds only for the US/France exchange rate.

The `summary` method provides more detailed information about the fit including estimated standard errors of coefficients and fit measures for each equation

```
> summary(sur.fit)
Seemingly Unrelated Regression:

Eq. 1: USCNS.diff ~USCN.FP.lag1

Coefficients:
               Value Std. Error t value Pr(>|t|)
 (Intercept) -0.0031  0.0012    -2.5943  0.0101
```

```
USCN.FP.lag1 -1.6626  0.5883    -2.8263  0.0052

Regression Diagnostics:

         R-Squared 0.0300
Adjusted R-Squared 0.0254
Durbin-Watson Stat 2.2161

Degrees of freedom: 215 total; 213 residual
Time period: from Aug 1978 to Jun 1996
Residual scale estimate: 0.0135


  ...


Eq. 6: USUKS.diff ~USUK.FP.lag1

Coefficients:
               Value Std. Error t value Pr(>|t|)
 (Intercept) -0.0035  0.0027    -1.3256  0.1864
USUK.FP.lag1 -1.2963  0.6317    -2.0519  0.0414

Regression Diagnostics:

         R-Squared 0.0253
Adjusted R-Squared 0.0207
Durbin-Watson Stat 1.9062

Degrees of freedom: 215 total; 213 residual
Time period: from Aug 1978 to Jun 1996
Residual scale estimate: 0.0344

Log determinant of residual covariance: -47.935
```

Graphical summaries of each equation are provided by the `plot` method which produces a menu of plot choices:

```
> plot(sur.fit)

Make a plot selection (or 0 to exit):

1: plot: All
2: plot: Response and Fitted Values
3: plot: Residuals
4: plot: Normal QQplot of Residuals
5: plot: ACF of Residuals
```

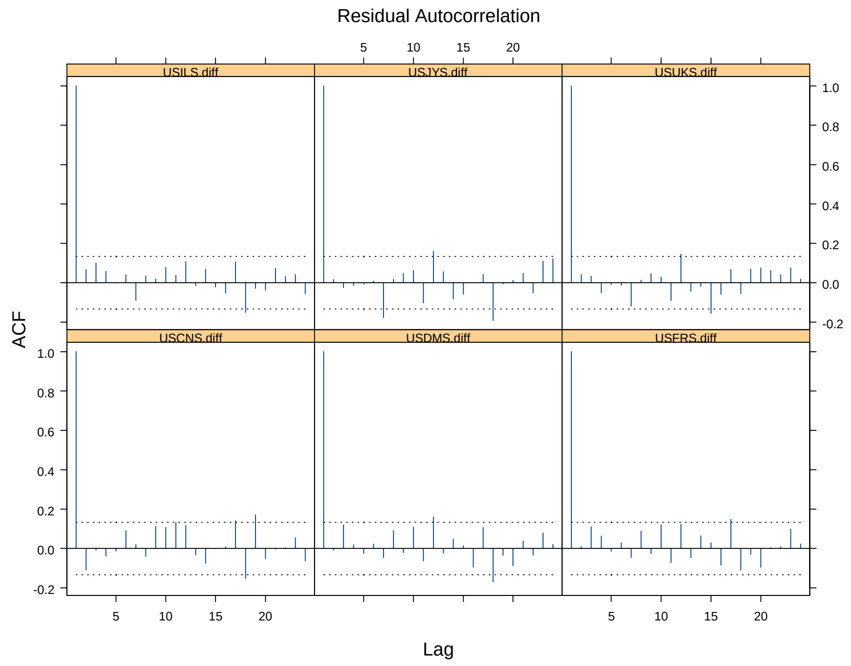


FIGURE 10.1. Residual ACF plots from SUR fit to exchange rate data.

```
6: plot: PACF of Residuals
7: plot: ACF of Squared Residuals
8: plot: PACF of Squared Residuals
Selection:
```

Plot choices 2-8 create multi-panel plots, one panel for each equation, using Trellis graphics. For example, Figure 10.1 shows the ACF of Residuals plot for the exchange rate data.

The above results are based on the non-iterated feasible GLS estimator (10.10). The iterated estimator is computed using

```
> sur.fit2 = SUR(formula.list, data=surex1.ts,
+ start="Aug 1978", in.format="%m %Y", iterate=T)
Iteration 1
Iteration 2
Iteration 3
Iteration 4
Iteration 5
Iteration 6
Iteration 7
Iteration 8
```

which converges after eight iterations. The non-iterated and iterated estimators may be easily compared using the `coef` extractor function:

```
> cbind(coef(sur.fit),coef(sur.fit2))
                    [,1]        [,2]
 (Intercept) -0.00312063 -0.00312126
USCN.FP.lag1 -1.66255897 -1.66303965
 (Intercept)  0.00058398  0.00035783
USDM.FP.lag1  0.50956590  0.65014949
 (Intercept)  0.00133327  0.00135930
USFR.FP.lag1  1.01512081  1.02834484
 (Intercept) -0.00058789 -0.00083921
USIL.FP.lag1  0.46173993  0.40852433
 (Intercept)  0.00778918  0.00744485
USJY.FP.lag1 -1.76416190 -1.63952144
 (Intercept) -0.00354947 -0.00334026
USUK.FP.lag1 -1.29625869 -1.19508947
```

There is not much difference between the two estimators.

The SUR estimator is more efficient than least squares equation-by-equation in this example provided the error terms across equations are correlated. The residual correlation matrix of the SUR fit (10.13) may be computed using

```
> sd.vals = sqrt(diag(sur.fit$Sigma))
> cor.mat = sur.fit$Sigma/outer(sd.vals,sd.vals)
> cor.mat
           USCNS.diff USDMS.diff USFRS.diff USILS.diff
USCNS.diff    1.00000    0.20187    0.19421    0.27727
USDMS.diff    0.20187    1.00000    0.97209    0.85884
USFRS.diff    0.19421    0.97209    1.00000    0.85090
USILS.diff    0.27727    0.85884    0.85090    1.00000
USJYS.diff    0.12692    0.61779    0.61443    0.50835
USUKS.diff    0.31868    0.71424    0.70830    0.72274

           USJYS.diff USUKS.diff
USCNS.diff    0.12692    0.31868
USDMS.diff    0.61779    0.71424
USFRS.diff    0.61443    0.70830
USILS.diff    0.50835    0.72274
USJYS.diff    1.00000    0.53242
USUKS.diff    0.53242    1.00000
```

Many of the estimated correlations are large so there appears to be an efficiency benefit to using SUR.

The forward rate unbiasedness implies that $\gamma_i = 1$ in (10.2) for $i = 1, \ldots, 6$. A formal test of the unbiasedness hypothesis for all six currencies simultaneously may be done using the Wald statistic (10.11) or the LR statistic (10.14). For the LR statistic, the iterated FGLS estimate should be used. The S-PLUS commands to compute the Wald statistic are

```
> bigR = matrix(0,6,12)
> bigR[1,2] = bigR[2,4] = bigR[3,6] = bigR[4,8] =
+ bigR[5,10] = bigR[6,12] = 1
> rr = rep(1,6)
> bHat = as.vector(coef(sur.fit))
> avar = bigR%*%vcov(sur.fit)%*%t(bigR)
> Wald = t((bigR%*%bHat-rr))%*%solve(avar)%*%(bigR%*%bHat-rr)
> Wald
       [,1]
[1,] 47.206
> 1-pchisq(Wald,6)
[1] 1.7025e-008
```

The data clearly reject the unbiased hypothesis. To compute the LR statistic (10.14), the restricted model with $\gamma_i = 1$ for $i = 1, \ldots, 6$ must first be computed. The restricted model takes the form

$$\Delta s_{i,t+k} - (f_{i,t}^k - s_{i,t}) = \alpha_i + \varepsilon_{i,t+k}$$

The S-PLUS commands to compute the restricted model are

```
> formula.list = list((USCNS.diff-USCN.FP.lag1)~1,
+ (USDMS.diff-USDM.FP.lag1)~1,
+ (USFRS.diff-USFR.FP.lag1)~1,
+ (USILS.diff-USIL.FP.lag1)~1,
+ (USJYS.diff-USJY.FP.lag1)~1,
+ (USUKS.diff-USUK.FP.lag1)~1)
> sur.fit2r = SUR(formula.list, data=surex1.ts,
+ start="Aug 1978", in.format="%m %Y", iterate=T)
Iteration 1
Iteration 2
> sur.fit2r

Seemingly Unrelated Regression:

Eq. 1: (USCNS.diff - USCN.FP.lag1) ~1

Coefficients:
 (Intercept)
 0.0004
```

```
Degrees of freedom: 215 total; 214 residual
Time period: from Aug 1978 to Jun 1996
Residual scale estimate: 0.014

...

Eq. 6: (USUKS.diff - USUK.FP.lag1) ~1

Coefficients:
 (Intercept)
 0.0012

Degrees of freedom: 215 total; 214 residual
Time period: from Aug 1978 to Jun 1996
Residual scale estimate: 0.0353

Log determinant of residual covariance: -47.61
```

The LR statistic (10.14) may then be computed using

```
> nobs = nrow(residuals(sur.fit2r))
> LR = nobs*(determinant(sur.fit2r$Sigma,log=T)$modulus-
+ determinant(sur.fit2$Sigma,log=T)$modulus)
> as.numeric(LR)
[1] 70.09
> 1-pchisq(LR,6)
[1] 3.912e-013
```

The LR statistic also confirms the rejection of the unbiasedness hypothesis.

10.4 Nonlinear Seemingly Unrelated Regression Models

The nonlinear SUR model is the system of M nonlinear regression equations

$$\mathbf{y}_i = \mathbf{f}_i(\boldsymbol{\beta}_i, \mathbf{X}_i) + \boldsymbol{\varepsilon}_i,\ i = 1, \ldots, M$$

where $\mathbf{y}_i$ is a $(T \times 1)$ vector of response variables, $\mathbf{f}_i(\boldsymbol{\beta}_i, \mathbf{X}_i)$ is a $(T \times 1)$ vector containing the nonlinear function values $f_i(\boldsymbol{\beta}_i, \mathbf{x}_{it})$, $\mathbf{X}_i$ is a $(T \times k_i)$ matrix of explanatory variables, and $\boldsymbol{\beta}_i$ is a $(k_i \times 1)$ vector of parameters. As with the linear SUR model, some of the explanatory variables in each $\mathbf{X}_i$ may be common across equations and some of the parameters in each $\boldsymbol{\beta}_i$ may also be common across equations. Without loss of generality, let $\mathbf{X}$ denote the $(T \times K)$ matrix of unique variables and let $\boldsymbol{\beta}$ denote the $(Q \times 1)$

vector of unique parameters,[1] and rewrite the nonlinear SUR system as

$$\mathbf{y}_i = \mathbf{f}_i(\boldsymbol{\beta}, \mathbf{X}) + \boldsymbol{\varepsilon}_i,\ i = 1, \ldots, M$$

The assumptions about the $(MT \times 1)$ system error vector $\underline{\boldsymbol{\varepsilon}} = (\boldsymbol{\varepsilon}_1' \ldots, \boldsymbol{\varepsilon}_M')'$ are the same as in the linear SUR model. That is, the covariance matrix of $\underline{\boldsymbol{\varepsilon}}$ is given by (10.7).

The estimation of nonlinear SUR models is detailed in Greene (2000) Chapter 15 and only a brief description is given here. The nonlinear FGLS estimator of $\boldsymbol{\beta}$ solves

$$\min_{\boldsymbol{\beta}} \sum_{i=1}^{M} \sum_{j=1}^{M} \hat{\sigma}^{ij} \left(\mathbf{y}_i - \mathbf{f}_i(\boldsymbol{\beta}, \mathbf{X})\right)' \left(\mathbf{y}_j - \mathbf{f}_j(\boldsymbol{\beta}, \mathbf{X})\right) \tag{10.15}$$

where σ^{ij} denotes the ijth element of Σ^{-1}. The nonlinear FGLS estimator utilizes initial estimates of σ^{ij} based on minimizing (10.15) with $\hat{\sigma}^{ij} = 1$. The iterated FGLS estimator minimizes (10.15) utilizing updated estimates of σ^{ij}. Assuming standard regularity conditions on the functions f_i the FGLS and iterated FGLS estimators are consistent and asymptotically normally distributed with asymptotic covariance matrix given by the inverse of the empirical Hessian matrix of (10.15).

10.4.1 Analysis of Nonlinear SUR Models with the `S+FinMetrics` Function `NLSUR`

The `S+FinMetrics` function `NLSUR` may be used to estimate general nonlinear SUR models as well as linear SUR models with parameter restrictions. The arguments for `NLSUR` are

```
> args(NLSUR)
function(formulas, data, na.rm = F, coef = NULL, start = NULL,
end = NULL, control = NULL, ...)
```

The usage of `NLSUR` is similar to that of `SUR`. The argument `formulas` contains a list of nonlinear model formulas for each equation in the SUR model. Nonlinear formulas are specified with the response on the left hand side of the ~ character and the nonlinear regression function specified on the right hand side. The K parameters of the nonlinear regression functions in the SUR model are specified explicitly by user specified coefficient values. For example, consider a "`timeSeries`" specified in the `data` argument with variables `y1`, `y2`, `y3`, `x1` and suppose the nonlinear SUR model has three

[1]In nonlinear models the number of parameters does not have to be equal to the number of variables.

equations

$$\begin{aligned} y_{1t} &= \beta_{10} + \beta_{11} x_{1t}^{\beta} + \varepsilon_{1t} \\ y_{2t} &= \beta_{20} + \beta_{21} x_{1t}^{\beta} + \varepsilon_{2t} \\ y_{3t} &= \beta_{30} + \beta_{31} x_{1t}^{\beta} + \varepsilon_{3t} \end{aligned}$$

The vector of $K = 7$ parameters of the SUR system are

$$\boldsymbol{\beta} = (\beta_{10}, \beta_{11}, \beta_{20}, \beta_{21}, \beta_{30}, \beta_{31}, \beta)'$$

Notice that the parameter β is common across the three equations. The formulas for the three equations above could be specified in a list as

```
> formula.list = list(y1~b10+b11*x1^b,
+ y2~b20+b21*x1^b,
+ y3~b30+b31*x1^b)
```

Notice that the user-specified coefficients `b10`, `b11`, `b20`, `b21`, `b30`, `b31`, `b` match the values in the parameter vector $\boldsymbol{\beta}$. The common parameter β across equations is given by the user specified coefficient `b`. Starting values for the coefficients may be specified in the named object `coef`. The parameter coefficients specified in the `formulas` argument must match the named elements in `coef`. For example, the starting values $\beta_{10} = \beta_{20} = \beta_{30} = 0, \beta_{11} = \beta_{21} = \beta_{31} = 1$ and $\beta = 0.5$ may be specified using

```
> start.vals = c(0,1,0,1,0,1,0.5)
> names(start.vals) = c("b10","b11","b20","b21","b30",
+ "b31","b")
```

Finally, the argument `control` is a list variable containing control parameters for the nonlinear optimization. These control parameters are the same as those used for the `S-PLUS` function `nlregb`. See the online help for `nlregb.control` for details.

`NLSUR` produces an object of class "`NLSUR`", that inherits from the class "`SUR`", for which there are `print`, `summary` and `plot` methods as well as extractor functions `residuals`, `fitted`, `coef` and `vcov`. The use of `NLSUR` is illustrated with the following examples.

Example 62 *Black CAPM model*

Consider estimating and testing the Black form of the CAPM (10.3) using monthly data on 16 assets over the five year period January 1983 through December 1987. The real return data are in the "`timeSeries`" `black.ts` which is constructed from the nominal return data in the "`timeSeries`" `berndt.dat` and the consumer price data in the "`timeSeries`" `CPI.dat`. The variables in `black.ts` are

```
> colIds(black.ts)
```

```
 [1] "BOISE"  "CITCRP" "CONED"  "CONTIL" "DATGEN" "DEC"
 [7] "DELTA"  "GENMIL" "GERBER" "IBM"    "MARKET" "MOBIL"
[13] "PANAM"  "PSNH"   "TANDY"  "TEXACO" "WEYER"
```

The variable `MARKET` is a value weighted index of all stocks on the NYSE and AMEX. The system (10.3) imposes $M = 16$ nonlinear cross equation restrictions on the intercept parameters: $\alpha_i = (1 - \beta_i)\gamma$. As a result, the parameter vector $\boldsymbol{\beta}$ has $K = 17$ elements: $\boldsymbol{\beta} = (\gamma, \beta_1, \ldots, \beta_{16})'$. A list of formulas for the 16 nonlinear regressions imposing the cross equation restrictions $\alpha_i = (1 - \beta_i)\gamma$ is

```
> formula.list = list(BOISE~(1-b1)*g + b1*MARKET,
+ CITCRP~(1-b2)*g + b2*MARKET,
+ CONED~(1-b3)*g + b3*MARKET,
+ CONTIL~(1-b4)*g + b4*MARKET,
+ DATGEN~(1-b5)*g + b5*MARKET,
+ DEC~(1-b6)*g + b6*MARKET,
+ DELTA~(1-b7)*g + b7*MARKET,
+ GENMIL~(1-b8)*g + b8*MARKET,
+ GERBER~(1-b9)*g + b9*MARKET,
+ IBM~(1-b10)*g + b10*MARKET,
+ MOBIL~(1-b11)*g + b11*MARKET,
+ PANAM~(1-b12)*g + b12*MARKET,
+ PSNH~(1-b13)*g + b13*MARKET,
+ TANDY~(1-b14)*g + b14*MARKET,
+ TEXACO~(1-b15)*g + b15*MARKET,
+ WEYER~(1-b16)*g + b16*MARKET)
```

The user specified coefficients `g`, `b1`, ..., `b16` represent the elements of the parameter vector $\boldsymbol{\beta}$, and the cross equation restrictions are imposed by expressing each intercept coefficient as the function `(1-bi)*g` for $i = 1, \ldots, 16$. The starting values $\gamma = 0$ and $\beta_i = 1$ $(i = 1, \ldots 16)$ for the estimation may be specified using

```
> start.vals = c(0,rep(1,16))
> names(start.vals) = c("g",paste("b",1:16,sep=""))
```

The FGLS nonlinear SUR estimator is computed using `NLSUR`

```
> nlsur.fit = NLSUR(formula.list,data=black.ts,
+ coef=start.vals,start="Jan 1983",in.format="%m %Y")
> class(nlsur.fit)
[1] "NLSUR"
```

The components of an “`NLSUR`” object are

```
> names(nlsur.fit)
 [1] "coef"       "objective"  "message"    "grad.norm"
 [5] "iterations" "r.evals"    "j.evals"    "scale"
 [9] "cov"        "call"       "parm.list"  "X.k"
[13] "residuals"  "fitted"     "Sigma"
```

The message component indicates that the nonlinear optimation converged:

```
> nlsur.fit$message
[1] "RELATIVE FUNCTION CONVERGENCE"
```

Since “NLSUR” objects inherit from “SUR” objects the print, summary and plot methods for “NLSUR” objects are identical to those for “SUR” objects. The print method gives basic fit information:

```
> nlsur.fit

Nonlinear Seemingly Unrelated Regression:

Eq. 1: BOISE ~(1 - b1) * g + b1 * MARKET

Coefficients:
     b1      g
 1.0120 0.0085

Degrees of freedom: 60 total; 58 residual
Time period: from Jan 1983 to Dec 1987
Residual scale estimate: 0.065

Eq. 2: CITCRP ~(1 - b2) * g + b2 * MARKET

Coefficients:
     b2      g
 1.0699 0.0085

Degrees of freedom: 60 total; 58 residual
Time period: from Jan 1983 to Dec 1987
Residual scale estimate: 0.0581

...

Eq. 16: WEYER ~(1 - b16) * g + b16 * MARKET

Coefficients:
    b16      g
 1.0044 0.0085

Degrees of freedom: 60 total; 58 residual
Time period: from Jan 1983 to Dec 1987
Residual scale estimate: 0.0574
```

```
Log determinant of residual covariance: -85.29
```

The estimated coefficients and their standard errors may be extracted using `coef` and `vcov`:

```
> std.ers = sqrt(diag(vcov(nlsur.fit)))
> cbind(coef(nlsur.fit),std.ers)
numeric matrix: 17 rows, 2 columns.
              std.ers
  g 0.008477 0.004642
 b1 1.012043 0.134400
 b2 1.069874 0.119842
 b3 0.028169 0.104461
 b4 1.479293 0.361368
 b5 1.133384 0.218596
 b6 1.099063 0.195965
 b7 0.704410 0.182686
 b8 0.547502 0.127458
 b9 0.960858 0.157903
b10 0.649761 0.096975
b11 0.741609 0.121843
b12 0.715984 0.265048
b13 0.205356 0.348101
b14 1.054715 0.185498
b15 0.574735 0.145838
b16 1.004  0.1186
```

More detailed information about the fit may be viewed using `summary`, and a graphical analysis of the fit may be created using `plot`. For example, Figure 10.2 shows the residuals from the 16 nonlinear equations.

The nonlinear restrictions implied by the Black form of the CAPM may be tested using a LR statistic. The unrestricted model (10.4) is specified using the formula list

```
> formula.list = list(BOISE~a1+b1*MARKET,
+ CITCRP~a2+b2*MARKET,
+ CONED~a3+b3*MARKET,
+ CONTIL~a4+b4*MARKET,
+ DATGEN~a5+b5*MARKET,
+ DEC~a6+b6*MARKET,
+ DELTA~a7+b7*MARKET,
+ GENMIL~a8+b8*MARKET,
+ GERBER~a9+b9*MARKET,
+ IBM~a10+b10*MARKET,
+ MOBIL~a11+b11*MARKET,
+ PANAM~a12+b12*MARKET,
```

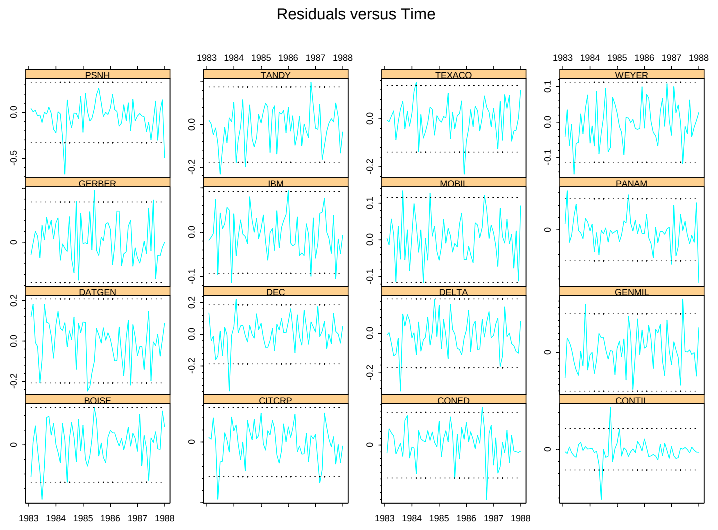


FIGURE 10.2. Estimated residuals from nonlinear SUR fit to Black's form of the CAPM.

```
+ PSNH~a13+b13*MARKET,
+ TANDY~a14+b14*MARKET,
+ TEXACO~a15+b15*MARKET,
+ WEYER~a16+b16*MARKET)
```

and is estimated using NLSUR with the starting values $\alpha_i = 0$ and $\beta_i = 1$ $(i = 1, \ldots, 16)$:

```
> start.vals = c(rep(0,16),rep(1,16))
> names(start.vals) =
+ c(paste("a",1:16,sep=""),paste("b",1:16,sep=""))
> nlsur.fit2 = NLSUR(formula.list,data=black.ts,
+ coef=start.vals,start="Jan 1983",in.format="%m %Y")
```

The LR statistic for testing the $M = 16$ nonlinear cross equation restrictions $\alpha_i = (1 - \beta_i)\gamma$ is computed using

```
> nobs = nrow(residuals(nlsur.fit2))
> LR = nobs*(determinant(nlsur.fit$Sigma,log=T)$modulus-
+ determinant(nlsur.fit2$Sigma,log=T)$modulus)
> as.numeric(LR)
[1] 15.86
> 1-pchisq(LR,16)
[1] 0.4625
```

The p-value of the test is 0.4627, and so the Black CAPM restrictions are not rejected at any reasonable significance level.

Example 63 *Estimation of exchange rate system with cross equation parameter restrictions*

Consider estimating the system of exchange rates (10.2), using the data described in the previous section, imposing the cross equation restriction that $\gamma_1 = \cdots = \gamma_M = \gamma$. The list of formulas for this restricted system may be constructed as

```
> formula.list = list(USCNS.diff~a1+g*USCN.FP.lag1,
+ USDMS.diff~a2+g*USDM.FP.lag1,
+ USFRS.diff~a3+g*USFR.FP.lag1,
+ USILS.diff~a4+g*USIL.FP.lag1,
+ USJYS.diff~a5+g*USJY.FP.lag1,
+ USUKS.diff~a6+g*USUK.FP.lag1)
```

Notice that the common parameter β is captured by the user-specified coefficient g. The starting values are chosen to be $\alpha_1 = \cdots = \alpha_6 = 0$ and $\gamma = 1$ and are specified using

```
> start.vals = c(rep(0,6),1)
> names(start.vals) = c(paste("a",1:6,sep=""),"g")
```

The FGLS estimator is computed using `NLSUR`

```
> nlsur.fit = NLSUR(formula.list, data=surex1.ts,
+ coef=start.vals, start="Aug 1978", in.format="%m %Y")
> nlsur.fit

Nonlinear Seemingly Unrelated Regression:

Eq. 1: USCNS.diff ~a1 + g * USCN.FP.lag1

Coefficients:
      a1       g
 -0.0005  0.3467

Degrees of freedom: 215 total; 213 residual
Time period: from Aug 1978 to Jun 1996
Residual scale estimate: 0.0138

...

Eq. 6: USUKS.diff ~a6 + g * USUK.FP.lag1

Coefficients:
      a6       g
 -0.0002  0.3467

Degrees of freedom: 215 total; 213 residual
Time period: from Aug 1978 to Jun 1996
Residual scale estimate: 0.035

Log determinant of residual covariance: -47.679
```

The estimate of the common parameter γ is 0.3467, and its asymptotic standard error is

```
> sqrt(diag(vcov(nlsur.fit)))[7]
[1] 0.16472
```

Hence, the data indicate that the common value of γ is less than 1. The LR statistic, however, rejects the common parameter restriction

```
> nobs = nrow(residuals(nlsur.fit))
> LR = nobs*(determinant(nlsur.fit$Sigma,log=T)$modulus
+ -determinant(sur.fit2$Sigma,log=T)$modulus)
> as.numeric(LR)
[1] 55.65
> 1 - pchisq(LR,6)
[1] 3.433e-010
```

10.5 References

Black, F. (1972). "Capital Market Equilibrium with Restricted Borrowing," *Journal of Business*, 44, 444-454.

Burmeister, E. and M.B. McElroy (1988). "Arbitrage Pricing Theory as a Restricted Nonlinear Multivariate Regression Model: ITNLSUR Estimates," *Journal of Business and Economic Statistics*, vol. 6(1), 28-42.

Campbell, J. A. Lo and C. MacKinlay (1997). *The Econometrics of Financial Markets*. Princeton University Press, Princeton, NJ.

Gallant, R.A. (1974). "Seemingly Unrelated Nonlinear Regressions," *Journal of Econometrics*, 3, 35-50.

Greene, W. (2000). *Econometric Analysis*, Fourth Edition. Prentice Hall, New Jersey.

Srivastava, V.K. and D.E.A. Giles (1987). *Seemingly Unrelated Regression Models: Estimation and Inference*. Marcel Dekker, New York.

Theil, H. (1961). *Economic Forecasts and Policy.* North Holland, Amsterdam.

Zellner, A. (1962). "An Efficient Method of Estimating Seemingly Unrelated Regressions and Tests of Aggregation Bias," *Journal of the American Statistical Association*, 57, 500-509.

11 Vector Autoregressive Models for Multivariate Time Series

11.1 Introduction

The *vector autoregression* (VAR) *model* is one of the most successful, flexible, and easy to use models for the analysis of multivariate time series. It is a natural extension of the univariate autoregressive model to dynamic multivariate time series. The VAR model has proven to be especially useful for describing the dynamic behavior of economic and financial time series and for forecasting. It often provides superior forecasts to those from univariate time series models and elaborate theory-based simultaneous equations models. Forecasts from VAR models are quite flexible because they can be made conditional on the potential future paths of specified variables in the model.

In addition to data description and forecasting, the VAR model is also used for structural inference and policy analysis. In structural analysis, certain assumptions about the causal structure of the data under investigation are imposed, and the resulting causal impacts of unexpected shocks or innovations to specified variables on the variables in the model are summarized. These causal impacts are usually summarized with impulse response functions and forecast error variance decompositions.

This chapter focuses on the analysis of covariance stationary multivariate time series using VAR models. The following chapter describes the analysis of nonstationary multivariate time series using VAR models that incorporate cointegration relationships.

This chapter is organized as follows. Section 11.2 describes specification, estimation and inference in VAR models and introduces the `S+FinMetrics` function `VAR`. Section 11.3 covers forecasting from VAR model. The discussion covers traditional forecasting algorithms as well as simulation-based forecasting algorithms that can impose certain types of conditioning information. Section 11.4 summarizes the types of structural analysis typically performed using VAR models. These analyses include Granger-causality tests, the computation of impulse response functions, and forecast error variance decompositions. Section 11.5 gives an extended example of VAR modeling. The chapter concludes with a brief discussion of Bayesian VAR models.

This chapter provides a relatively non-technical survey of VAR models. VAR models in economics were made popular by Sims (1980). The definitive technical reference for VAR models is Lütkepohl (1991), and updated surveys of VAR techniques are given in Watson (1994) and Lütkepohl (1999) and Waggoner and Zha (1999). Applications of VAR models to financial data are given in Hamilton (1994), Campbell, Lo and MacKinlay (1997), Cuthbertson (1996), Mills (1999) and Tsay (2001).

11.2 The Stationary Vector Autoregression Model

Let $\mathbf{Y}_t = (y_{1t}, y_{2t}, \ldots, y_{nt})'$ denote an $(n\times 1)$ vector of time series variables. The basic p-lag *vector autoregressive* (VAR(p)) model has the form

$$\mathbf{Y}_t = \mathbf{c} + \mathbf{\Pi}_1\mathbf{Y}_{t-1} + \mathbf{\Pi}_2\mathbf{Y}_{t-2} + \cdots + \mathbf{\Pi}_p\mathbf{Y}_{t-p} + \boldsymbol{\varepsilon}_t,\ t = 1, \ldots, T \quad (11.1)$$

where $\mathbf{\Pi}_i$ are $(n \times n)$ coefficient matrices and $\boldsymbol{\varepsilon}_t$ is an $(n \times 1)$ unobservable zero mean white noise vector process (serially uncorrelated or independent) with time invariant covariance matrix $\mathbf{\Sigma}$. For example, a bivariate VAR(2) model equation by equation has the form

$$\begin{aligned}\begin{pmatrix} y_{1t} \\ y_{2t} \end{pmatrix} &= \begin{pmatrix} c_1 \\ c_2 \end{pmatrix} + \begin{pmatrix} \pi_{11}^1 & \pi_{12}^1 \\ \pi_{21}^1 & \pi_{22}^1 \end{pmatrix}\begin{pmatrix} y_{1t-1} \\ y_{2t-1} \end{pmatrix} \quad (11.2) \\ &\quad + \begin{pmatrix} \pi_{11}^2 & \pi_{12}^2 \\ \pi_{21}^2 & \pi_{22}^2 \end{pmatrix}\begin{pmatrix} y_{1t-2} \\ y_{2t-2} \end{pmatrix} + \begin{pmatrix} \varepsilon_{1t} \\ \varepsilon_{2t} \end{pmatrix} \quad (11.3)\end{aligned}$$

or

$$\begin{aligned} y_{1t} &= c_1 + \pi_{11}^1 y_{1t-1} + \pi_{12}^1 y_{2t-1} + \pi_{11}^2 y_{1t-2} + \pi_{12}^2 y_{2t-2} + \varepsilon_{1t} \\ y_{2t} &= c_2 + \pi_{21}^1 y_{1t-1} + \pi_{22}^1 y_{2t-1} + \pi_{21}^2 y_{1t-1} + \pi_{22}^2 y_{2t-1} + \varepsilon_{2t} \end{aligned}$$

where $\text{cov}(\varepsilon_{1t}, \varepsilon_{2s}) = \sigma_{12}$ for $t = s$; 0 otherwise. Notice that each equation has the same regressors – lagged values of y_{1t} and y_{2t}. Hence, the VAR(p) model is just a *seemingly unrelated regression* (SUR) model with lagged variables and deterministic terms as common regressors.

In lag operator notation, the VAR(p) is written as

$$\mathbf{\Pi}(L)\mathbf{Y}_t = \mathbf{c} + \boldsymbol{\varepsilon}_t$$

where $\mathbf{\Pi}(L) = \mathbf{I}_n - \mathbf{\Pi}_1 L - ... - \mathbf{\Pi}_p L^p$. The VAR($p$) is stable if the roots of

$$\det\left(\mathbf{I}_n - \mathbf{\Pi}_1 z - \cdots - \mathbf{\Pi}_p z^p\right) = 0$$

lie outside the complex unit circle (have modulus greater than one), or, equivalently, if the eigenvalues of the companion matrix

$$\mathbf{F} = \begin{pmatrix} \mathbf{\Pi}_1 & \mathbf{\Pi}_2 & \cdots & \mathbf{\Pi}_n \\ \mathbf{I}_n & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \ddots & \mathbf{0} & \vdots \\ \mathbf{0} & \mathbf{0} & \mathbf{I}_n & \mathbf{0} \end{pmatrix}$$

have modulus less than one. Assuming that the process has been initialized in the infinite past, then a stable VAR(p) process is stationary and ergodic with time invariant means, variances, and autocovariances.

If $\mathbf{Y}_t$ in (11.1) is covariance stationary, then the unconditional mean is given by

$$\boldsymbol{\mu} = (\mathbf{I}_n - \mathbf{\Pi}_1 - \cdots - \mathbf{\Pi}_p)^{-1}\mathbf{c}$$

The *mean-adjusted* form of the VAR(p) is then

$$\mathbf{Y}_t - \boldsymbol{\mu} = \mathbf{\Pi}_1(\mathbf{Y}_{t-1} - \boldsymbol{\mu}) + \mathbf{\Pi}_2(\mathbf{Y}_{t-2} - \boldsymbol{\mu}) + \cdots + \mathbf{\Pi}_p(\mathbf{Y}_{t-p} - \boldsymbol{\mu}) + \boldsymbol{\varepsilon}_t$$

The basic VAR(p) model may be too restrictive to represent sufficiently the main characteristics of the data. In particular, other deterministic terms such as a linear time trend or seasonal dummy variables may be required to represent the data properly. Additionally, stochastic exogenous variables may be required as well. The general form of the VAR(p) model with deterministic terms and exogenous variables is given by

$$\mathbf{Y}_t = \mathbf{\Pi}_1\mathbf{Y}_{t-1} + \mathbf{\Pi}_2\mathbf{Y}_{t-2} + \cdots + \mathbf{\Pi}_p\mathbf{Y}_{t-p} + \mathbf{\Phi}\mathbf{D}_t + \mathbf{G}\mathbf{X}_t + \boldsymbol{\varepsilon}_t \qquad (11.4)$$

where $\mathbf{D}_t$ represents an $(l \times 1)$ matrix of deterministic components, $\mathbf{X}_t$ represents an $(m \times 1)$ matrix of exogenous variables, and Φ and $\mathbf{G}$ are parameter matrices.

Example 64 *Simulating a stationary VAR(1) model using* `S-PLUS`

A stationary VAR model may be easily simulated in `S-PLUS` using the `S+FinMetrics` function `simulate.VAR`. The commands to simulate $T = 250$ observations from a bivariate VAR(1) model

$$\begin{aligned} y_{1t} &= -0.7 + 0.7y_{1t-1} + 0.2y_{2t-1} + \varepsilon_{1t} \\ y_{2t} &= 1.3 + 0.2y_{1t-1} + 0.7y_{2t-1} + \varepsilon_{2t} \end{aligned}$$

with

$$\mathbf{\Pi}_1 = \begin{pmatrix} 0.7 & 0.2 \\ 0.2 & 0.7 \end{pmatrix}, \ \mathbf{c} = \begin{pmatrix} -0.7 \\ 1.3 \end{pmatrix}, \ \boldsymbol{\mu} = \begin{pmatrix} 1 \\ 5 \end{pmatrix}, \ \mathbf{\Sigma} = \begin{pmatrix} 1 & 0.5 \\ 0.5 & 1 \end{pmatrix}$$

and normally distributed errors are

```
> pi1 = matrix(c(0.7,0.2,0.2,0.7),2,2)
> mu.vec = c(1,5)
> c.vec = as.vector((diag(2)-pi1)%*%mu.vec)
> cov.mat = matrix(c(1,0.5,0.5,1),2,2)
> var1.mod = list(const=c.vec,ar=pi1,Sigma=cov.mat)
> set.seed(301)
> y.var = simulate.VAR(var1.mod,n=250,
+ y0=t(as.matrix(mu.vec)))
> dimnames(y.var) = list(NULL,c("y1","y2"))
```

The simulated data are shown in Figure 11.1. The VAR is stationary since the eigenvalues of $\mathbf{\Pi}_1$ are less than one:

```
> eigen(pi1,only.values=T)
$values:
[1] 0.9 0.5

$vectors:
NULL
```

Notice that the intercept values are quite different from the mean values of y_1 and y_2:

```
> c.vec
[1] -0.7  1.3
> colMeans(y.var)
     y1    y2
 0.8037 4.751
```

11.2.1 Estimation

Consider the basic VAR(p) model (11.1). Assume that the VAR(p) model is covariance stationary, and there are no restrictions on the parameters of the model. In SUR notation, each equation in the VAR(p) may be written as

$$\mathbf{y}_i = \mathbf{Z}\boldsymbol{\pi}_i + \mathbf{e}_i, \ i = 1, \ldots, n$$

where $\mathbf{y}_i$ is a $(T \times 1)$ vector of observations on the i^{th} equation, $\mathbf{Z}$ is a $(T \times k)$ matrix with t^{th} row given by $\mathbf{Z}_t' = (1, \mathbf{Y}_{t-1}', \ldots, \mathbf{Y}_{t-p}')$, $k = np + 1$, $\boldsymbol{\pi}_i$ is a $(k \times 1)$ vector of parameters and $\mathbf{e}_i$ is a $(T \times 1)$ error with covariance matrix $\sigma_i^2 \mathbf{I}_T$. Since the VAR(p) is in the form of a SUR model

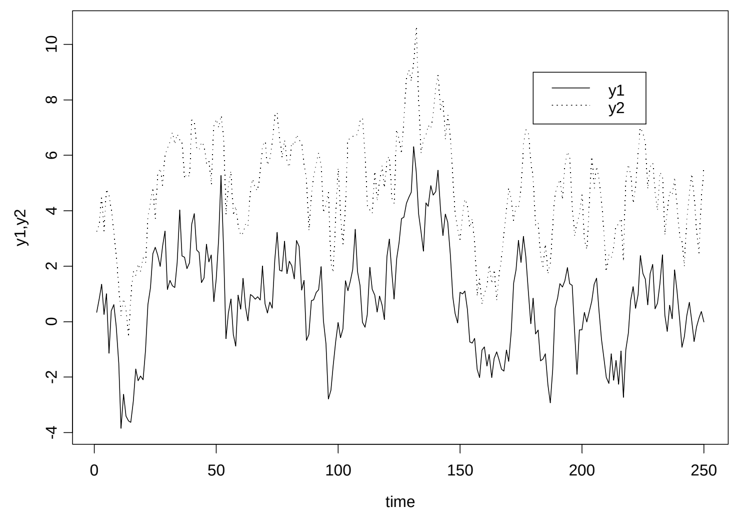


FIGURE 11.1. Simulated stationary VAR(1) model.

where each equation has the same explanatory variables, each equation may be estimated separately by ordinary least squares without losing efficiency relative to generalized least squares. Let $\hat{\mathbf{\Pi}} = [\hat{\boldsymbol{\pi}}_1, \ldots, \hat{\boldsymbol{\pi}}_n]$ denote the $(k \times n)$ matrix of least squares coefficients for the n equations.

Let $\text{vec}(\hat{\mathbf{\Pi}})$ denote the operator that stacks the columns of the $(n \times k)$ matrix $\hat{\mathbf{\Pi}}$ into a long $(nk \times 1)$ vector. That is,

$$\text{vec}(\hat{\mathbf{\Pi}}) = \begin{pmatrix} \hat{\boldsymbol{\pi}}_1 \\ \vdots \\ \hat{\boldsymbol{\pi}}_n \end{pmatrix}$$

Under standard assumptions regarding the behavior of stationary and ergodic VAR models (see Hamilton (1994) or Lütkepohl (1991)) $\text{vec}(\hat{\mathbf{\Pi}})$ is consistent and asymptotically normally distributed with asymptotic covariance matrix

$$\widehat{\text{avar}}(\text{vec}(\hat{\mathbf{\Pi}})) = \hat{\mathbf{\Sigma}} \otimes (\mathbf{Z}'\mathbf{Z})^{-1}$$

where

$$\hat{\mathbf{\Sigma}} = \frac{1}{T-k} \sum_{t=1}^{T} \hat{\boldsymbol{\varepsilon}}_t \hat{\boldsymbol{\varepsilon}}_t'$$

and $\hat{\boldsymbol{\varepsilon}}_t = \mathbf{Y}_t - \hat{\mathbf{\Pi}}'\mathbf{Z}_t$ is the multivariate least squares residual from (11.1) at time t.

11.2.2 Inference on Coefficients

The i^{th} element of $\text{vec}(\hat{\Pi})$, $\hat{\pi}_i$, is asymptotically normally distributed with asymptotic standard error given by the square root of i^{th} diagonal element of $\hat{\mathbf{\Sigma}} \otimes (\mathbf{Z}'\mathbf{Z})^{-1}$. Hence, asymptotically valid t-tests on individual coefficients may be constructed in the usual way. More general linear hypotheses of the form $\mathbf{R}\cdot\text{vec}(\mathbf{\Pi}) = \mathbf{r}$ involving coefficients across different equations of the VAR may be tested using the Wald statistic

$$\text{Wald} = (\mathbf{R}\cdot\text{vec}(\hat{\mathbf{\Pi}}) - \mathbf{r})' \left\{ \mathbf{R} \left[\widehat{\text{avar}}(\text{vec}(\hat{\mathbf{\Pi}})) \right] \mathbf{R}' \right\}^{-1} (\mathbf{R}\cdot\text{vec}(\hat{\mathbf{\Pi}}) - \mathbf{r}) \quad (11.5)$$

Under the null, (11.5) has a limiting $\chi^2(q)$ distribution where $q = \text{rank}(\mathbf{R})$ gives the number of linear restrictions.

11.2.3 Lag Length Selection

The lag length for the VAR(p) model may be determined using model selection criteria. The general approach is to fit VAR(p) models with orders $p = 0, ..., p_{\max}$ and choose the value of p which minimizes some model selection criteria. Model selection criteria for VAR(p) models have the form

$$IC(p) = \ln|\tilde{\mathbf{\Sigma}}(p)| \; + c_T \cdot \varphi(n, p)$$

where $\tilde{\mathbf{\Sigma}}(p) = T^{-1} \sum_{t=1}^{T} \hat{\varepsilon}_t \hat{\varepsilon}_t'$ is the residual covariance matrix *without a degrees of freedom correction* from a VAR(p) model, c_T is a sequence indexed by the sample size T, and $\varphi(n, p)$ is a penalty function which penalizes large VAR(p) models. The three most common information criteria are the Akaike (AIC), Schwarz-Bayesian (BIC) and Hannan-Quinn (HQ):

$$\begin{aligned}
\text{AIC}(p) &= \ln|\tilde{\mathbf{\Sigma}}(p)| + \frac{2}{T} pn^2 \\
\text{BIC}(p) &= \ln|\tilde{\mathbf{\Sigma}}(p)| + \frac{\ln T}{T} pn^2 \\
\text{HQ}(p) &= \ln|\tilde{\mathbf{\Sigma}}(p)| + \frac{2 \ln \ln T}{T} pn^2
\end{aligned}$$

The AIC criterion asymptotically overestimates the order with positive probability, whereas the BIC and HQ criteria estimate the order consistently under fairly general conditions if the true order p is less than or equal to $p_{\max}$. For more information on the use of model selection criteria in VAR models see Lütkepohl (1991) chapter four.

11.2.4 Estimating VAR Models Using the `S+FinMetrics` Function `VAR`

The `S+FinMetrics` function `VAR` is designed to fit and analyze VAR models as described in the previous section. `VAR` produces an object of class "`VAR`" for which there are `print`, `summary`, `plot` and `predict` methods as well as extractor functions `coefficients`, `residuals`, `fitted` and `vcov`. The calling syntax of `VAR` is a bit complicated because it is designed to handle multivariate data in matrices, data frames as well as "`timeSeries`" objects. The use of `VAR` is illustrated with the following example.

Example 65 *Bivariate VAR model for exchange rates*

This example considers a bivariate VAR model for $\mathbf{Y}_t = (\Delta s_t, fp_t)'$, where s_t is the logarithm of the monthly spot exchange rate between the US and Canada, $fp_t = f_t - s_t = i_t^{\mathrm{US}} - i_t^{\mathrm{CA}}$ is the forward premium or interest rate differential, and f_t is the natural logarithm of the 30-day forward exchange rate. The data over the 20 year period March 1976 through June 1996 is in the `S+FinMetrics` "`timeSeries`" `lexrates.dat`. The data for the VAR model are computed as

```
> dspot = diff(lexrates.dat[,"USCNS"])
> fp = lexrates.dat[,"USCNF"]-lexrates.dat[,"USCNS"]
> uscn.ts = seriesMerge(dspot,fp)
> colIds(uscn.ts) = c("dspot","fp")
> uscn.ts@title = "US/CN Exchange Rate Data"
> par(mfrow=c(2,1))
> plot(uscn.ts[,"dspot"],main="1st difference of US/CA spot
+ exchange rate")
> plot(uscn.ts[,"fp"],main="US/CN interest rate
+ differential")
```

Figure 11.2 illustrates the monthly return Δs_t and the forward premium fp_t over the period March 1976 through June 1996. Both series appear to be $I(0)$ (which can be confirmed using the `S+FinMetrics` functions `unitroot` or `stationaryTest`) with Δs_t much more volatile than fp_t. fp_t also appears to be heteroskedastic.

Specifying and Estimating the VAR(p) Model

To estimate a VAR(1) model for $\mathbf{Y}_t$ use

```
> var1.fit = VAR(cbind(dspot,fp)~ar(1),data=uscn.ts)
```

Note that the VAR model is specified using an `S-PLUS` formula, with the multivariate response on the left hand side of the `~` operator and the built-in AR term specifying the lag length of the model on the right hand side. The optional `data` argument accepts a data frame or "`timeSeries`" object with variable names matching those used in specifying the formula. If the data are in a "`timeSeries`" object or in an unattached data frame ("`timeSeries`" objects cannot be attached) then the `data` argument must be used. If the data are in a matrix then the `data` argument may be omitted. For example,

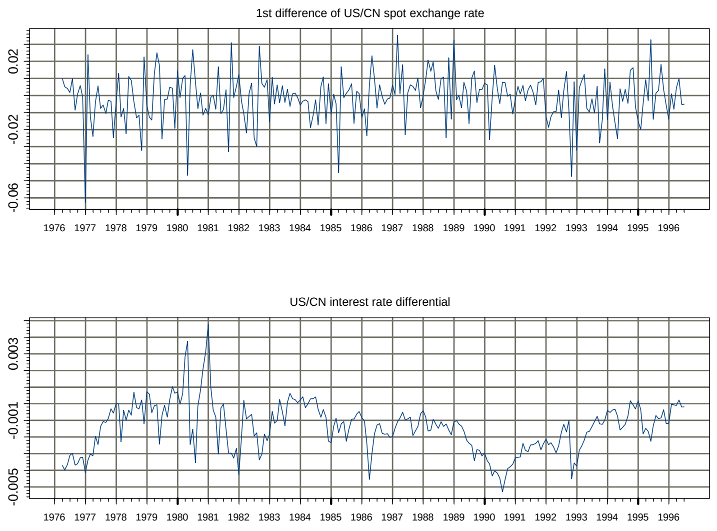


FIGURE 11.2. US/CN forward premium and spot rate.

```
> uscn.mat = as.matrix(seriesData(uscn.ts))
> var2.fit = VAR(uscn.mat~ar(1))
```

If the data are in a "`timeSeries`" object then the `start` and `end` options may be used to specify the estimation sample. For example, to estimate the VAR(1) over the sub-period January 1980 through January 1990

```
> var3.fit = VAR(cbind(dspot,fp)~ar(1), data=uscn.ts,
+ start="Jan 1980", end="Jan 1990", in.format="%m %Y")
```

may be used. The use of `in.format=\%m %Y"` sets the format for the date strings specified in the `start` and `end` options to match the input format of the dates in the positions slot of `uscn.ts`.

The VAR model may be estimated with the lag length p determined using a specified information criterion. For example, to estimate the VAR for the exchange rate data with p set by minimizing the BIC with a maximum lag $p_{\max} = 4$ use

```
> var4.fit = VAR(uscn.ts,max.ar=4, criterion="BIC")
> var4.fit$info
    ar(1) ar(2) ar(3) ar(4)
BIC -4028 -4013 -3994 -3973
```

When a formula is not specified and only a data frame, "`timeSeries`" or matrix is supplied that contains the variables for the VAR model, `VAR` fits all VAR(p) models with lag lengths p less than or equal to the value given to `max.ar`, and the lag length is determined as the one which minimizes the information criterion specified by the `criterion` option. The default criterion is `BIC` but other valid choices are `logL`, `AIC` and `HQ`. In the computation of the information criteria, a common sample based on `max.ar` is used. Once the lag length is determined, the VAR is re-estimated using the appropriate sample. In the above example, the BIC values were computed using the sample based on `max.ar=4` and $p = 1$ minimizes BIC. The VAR(1) model was automatically re-estimated using the sample size appropriate for $p = 1$.

Print and Summary Methods

The function `VAR` produces an object of class "`VAR`" with the following components.

```
> class(var1.fit)
[1] "VAR"
> names(var1.fit)
 [1] "R"         "coef"      "fitted"    "residuals"
 [5] "Sigma"     "df.resid"  "rank"      "call"
 [9] "ar.order"  "n.na"      "terms"     "Y0"
```

To see the estimated coefficients of the model use the `print` method:

```
> var1.fit

Call:
VAR(formula = cbind(dspot, fp) ~ar(1), data = uscn.ts)

Coefficients:
              dspot      fp
(Intercept) -0.0036 -0.0003
 dspot.lag1 -0.1254  0.0079
    fp.lag1 -1.4833  0.7938

Std. Errors of Residuals:
  dspot     fp
 0.0137 0.0009

Information Criteria:
  logL   AIC   BIC    HQ
  2058 -4104 -4083 -4096

                   total residual
Degree of freedom:   243      240
Time period: from Apr 1976 to Jun 1996
```

The first column under the label "`Coefficients`:" gives the estimated coefficients for the Δs_t equation, and the second column gives the estimated coefficients for the fp_t equation:

$$\begin{aligned}\Delta s_t &= -0.0036 - 0.1254 \cdot \Delta s_{t-1} - 1.4833 \cdot fp_{t-1} \\ fp_t &= -0.0003 + 0.0079 \cdot \Delta s_{t-1} + 0.7938 \cdot fp_{t-1}\end{aligned}$$

Since `uscn.ts` is a "`timeSeries`" object, the estimation time period is also displayed.

The `summary` method gives more detailed information about the fitted VAR:

```
> summary(var1.fit)

Call:
VAR(formula = cbind(dspot, fp) ~ar(1), data = uscn.ts)

Coefficients:
                dspot       fp
(Intercept)  -0.0036  -0.0003
  (std.err)   0.0012   0.0001
   (t.stat)  -2.9234  -3.2885

 dspot.lag1  -0.1254   0.0079
  (std.err)   0.0637   0.0042
   (t.stat)  -1.9700   1.8867

    fp.lag1  -1.4833   0.7938
  (std.err)   0.5980   0.0395
   (t.stat)  -2.4805  20.1049

Regression Diagnostics:
                dspot     fp
     R-squared 0.0365 0.6275
Adj. R-squared 0.0285 0.6244
  Resid. Scale 0.0137 0.0009

Information Criteria:
  logL   AIC   BIC    HQ
  2058 -4104 -4083 -4096

                   total residual
Degree of freedom:   243      240
Time period: from Apr 1976 to Jun 1996
```

In addition to the coefficient standard errors and t-statistics, `summary` also displays R^2 measures for each equation (which are valid because each equation is estimated by least squares). The summary output shows that the coefficients on Δs_{t-1} and fp_{t-1} in both equations are statistically significant at the 10% level and that the fit for the fp_t equation is much better than the fit for the Δs_t equation.

As an aside, note that the `S+FinMetrics` function `OLS` may also be used to estimate each equation in a VAR model. For example, one way to compute the equation for Δs_t using `OLS` is

```
> dspot.fit = OLS(dspot~ar(1)+tslag(fp),data=uscn.ts)
> dspot.fit

Call:
OLS(formula = dspot ~ar(1) + tslag(fp), data = uscn.ts)

Coefficients:
 (Intercept) tslag(fp)    lag1
 -0.0036     -1.4833   -0.1254

Degrees of freedom: 243 total; 240 residual
Time period: from Apr 1976 to Jun 1996
Residual standard error: 0.01373
```

Graphical Diagnostics

The `plot` method for “`VAR`” objects may be used to graphically evaluate the fitted VAR. By default, the `plot` method produces a menu of plot options:

```
> plot(var1.fit)

Make a plot selection (or 0 to exit):

1: plot: All
2: plot: Response and Fitted Values
3: plot: Residuals
4: plot: Normal QQplot of Residuals
5: plot: ACF of Residuals
6: plot: PACF of Residuals
7: plot: ACF of Squared Residuals
8: plot: PACF of Squared Residuals
Selection:
```

Alternatively, `plot.VAR` may be called directly. The function `plot.VAR` has arguments

```
> args(plot.VAR)
function(x, ask = T, which.plots = NULL, hgrid = F, vgrid
= F, ...)
```

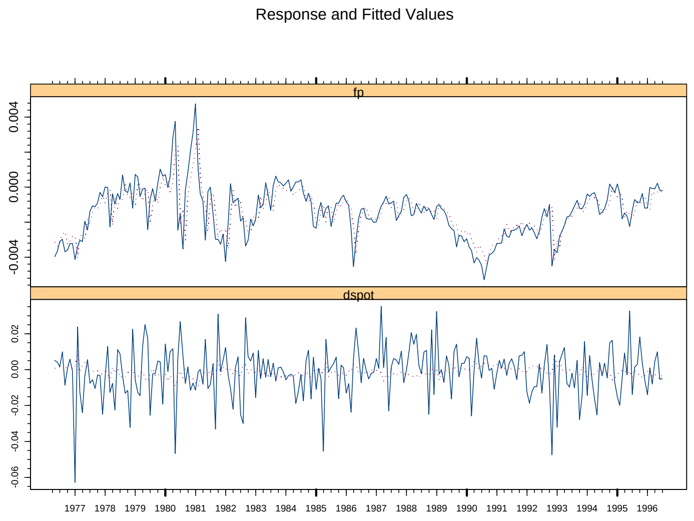


FIGURE 11.3. Response and fitted values from VAR(1) model for US/CN exchange rate data.

To create all seven plots without using the menu, set `ask=F`. To create the Residuals plot without using the menu, set `which.plot=2`. The optional arguments `hgrid` and `vgrid` control printing of horizontal and vertical grid lines on the plots.

Figures 11.3 and 11.4 give the Response and Fitted Values and Residuals plots for the VAR(1) fit to the exchange rate data. The equation for fp_t fits much better than the equation for Δs_t. The residuals for both equations look fairly random, but the residuals for the fp_t equation appear to be heteroskedastic. The qq-plot (not shown) indicates that the residuals for the Δs_t equation are highly non-normal.

Extractor Functions

The residuals and fitted values for each equation of the VAR may be extracted using the generic extractor functions `residuals` and `fitted`:

```
> var1.resid = resid(var1.fit)
> var1.fitted = fitted(var.fit)
> var1.resid[1:3,]
 Positions     dspot          fp
 Apr 1976  0.0044324 -0.00084150
 May 1976  0.0024350 -0.00026493
 Jun 1976  0.0004157  0.00002435
```

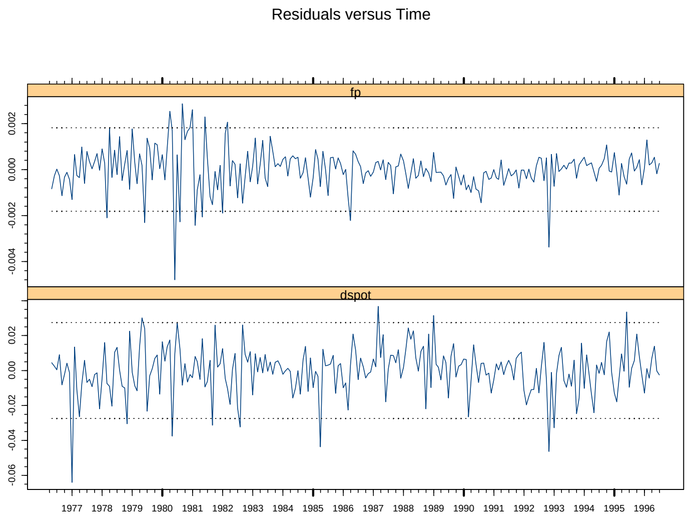


FIGURE 11.4. Residuals from VAR(1) model fit to US/CN exchange rate data.

Notice that since the data are in a "`timeSeries`" object, the extracted residuals and fitted values are also "`timeSeries`" objects.

The coefficients of the VAR model may be extracted using the generic `coef` function:

```
> coef(var1.fit)
                   dspot            fp
(Intercept) -0.003595149 -0.0002670108
 dspot.lag1 -0.125397056  0.0079292865
    fp.lag1 -1.483324622  0.7937959055
```

Notice that `coef` produces the (3×2) matrix $\hat{\mathbf{\Pi}}$ whose columns give the estimated coefficients for each equation in the VAR(1).

To test stability of the VAR, extract the matrix $\mathbf{\Pi}_1$ and compute its eigenvalues

```
> PI1 = t(coef(var1.fit)[2:3,])
> abs(eigen(PI1,only.values=T)$values)
[1] 0.7808 0.1124
```

Since the modulus of the two eigenvalues of $\mathbf{\Pi}_1$ are less than 1, the VAR(1) is stable.

Testing Linear Hypotheses

Now, consider testing the hypothesis that $\mathbf{\Pi}_1 = \mathbf{0}$ (i.e., $\mathbf{Y}_{t-1}$ does not help to explain $\mathbf{Y}_t$) using the Wald statistic (11.5). In terms of the columns of vec($\mathbf{\Pi}$) the restrictions are $\boldsymbol{\pi}_1 = (c_1, 0, 0)'$ and $\boldsymbol{\pi}_2 = (c_2, 0, 0)$ and may be expressed as $\mathbf{R}\text{vec}(\mathbf{\Pi}) = \mathbf{r}$ with

$$\mathbf{R} = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}, \mathbf{r} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

The Wald statistic is easily constructed as follows

```
> R = matrix(c(0,1,0,0,0,0,
+ 0,0,1,0,0,0,
+ 0,0,0,0,1,0,
+ 0,0,0,0,0,1),
+ 4,6,byrow=T)
> vecPi = as.vector(var1.fit$coef)
> avar = R%*%vcov(var1.fit)%*%t(R)
> wald = t(R%*%vecPi)%*%solve(avar)%*%(R%*%vecPi)
> wald
      [,1]
[1,] 417.1
> 1-pchisq(wald,4)
[1] 0
```

Since the p-value for the Wald statistic based on the $\chi^2(4)$ distribution is essentially zero, the hypothesis that $\mathbf{\Pi}_1 = \mathbf{0}$ should be rejected at any reasonable significance level.

11.3 Forecasting

Forecasting is one of the main objectives of multivariate time series analysis. Forecasting from a VAR model is similar to forecasting from a univariate AR model and the following gives a brief description.

11.3.1 Traditional Forecasting Algorithm

Consider first the problem of forecasting future values of $\mathbf{Y}_t$ when the parameters $\mathbf{\Pi}$ of the VAR(p) process are assumed to be known and there are no deterministic terms or exogenous variables. The best linear predictor, in terms of minimum mean squared error (MSE), of $\mathbf{Y}_{t+1}$ or 1-step forecast based on information available at time T is

$$\mathbf{Y}_{T+1|T} = \mathbf{c} + \mathbf{\Pi}_1 \mathbf{Y}_T + \cdots + \mathbf{\Pi}_p \mathbf{Y}_{T-p+1}$$

Forecasts for longer horizons h (h-step forecasts) may be obtained using the *chain-rule of forecasting* as

$$\mathbf{Y}_{T+h|T} = \mathbf{c} + \mathbf{\Pi}_1 \mathbf{Y}_{T+h-1|T} + \cdots + \mathbf{\Pi}_p \mathbf{Y}_{T+h-p|T}$$

where $\mathbf{Y}_{T+j|T} = \mathbf{Y}_{T+j}$ for $j \leq 0$. The h-step forecast errors may be expressed as

$$\mathbf{Y}_{T+h} - \mathbf{Y}_{T+h|T} = \sum_{s=0}^{h-1} \mathbf{\Psi}_s \boldsymbol{\varepsilon}_{T+h-s}$$

where the matrices $\mathbf{\Psi}_s$ are determined by recursive substitution

$$\mathbf{\Psi}_s = \sum_{j=1}^{p-1} \mathbf{\Psi}_{s-j} \mathbf{\Pi}_j \tag{11.6}$$

with $\mathbf{\Psi}_0 = \mathbf{I}_n$ and $\mathbf{\Pi}_j = 0$ for $j > p$.[1] The forecasts are unbiased since all of the forecast errors have expectation zero and the MSE matrix for $\mathbf{Y}_{t+h|T}$ is

$$\begin{aligned} \mathbf{\Sigma}(h) &= \text{MSE}\left(\mathbf{Y}_{T+h} - \mathbf{Y}_{T+h|T}\right) \\ &= \sum_{s=0}^{h-1} \mathbf{\Psi}_s \mathbf{\Sigma} \mathbf{\Psi}_s' \end{aligned} \tag{11.7}$$

Now consider forecasting $\mathbf{Y}_{T+h}$ when the parameters of the VAR(p) process are estimated using multivariate least squares. The best linear predictor of $\mathbf{Y}_{T+h}$ is now

$$\hat{\mathbf{Y}}_{T+h|T} = \hat{\mathbf{\Pi}}_1 \hat{\mathbf{Y}}_{T+h-1|T} + \cdots + \hat{\mathbf{\Pi}}_p \hat{\mathbf{Y}}_{T+h-p|T} \tag{11.8}$$

where $\hat{\mathbf{\Pi}}_j$ are the estimated parameter matrices. The h-step forecast error is now

$$\mathbf{Y}_{T+h} - \hat{\mathbf{Y}}_{T+h|T} = \sum_{s=0}^{h-1} \mathbf{\Psi}_s \boldsymbol{\varepsilon}_{T+h-s} + \left(\mathbf{Y}_{T+h} - \hat{\mathbf{Y}}_{T+h|T}\right) \tag{11.9}$$

and the term $\left(\mathbf{Y}_{T+h} - \hat{\mathbf{Y}}_{T+h|T}\right)$ captures the part of the forecast error due to estimating the parameters of the VAR. The MSE matrix of the h-step forecast is then

$$\hat{\mathbf{\Sigma}}(h) = \mathbf{\Sigma}(h) + \text{MSE}\left(\mathbf{Y}_{T+h} - \hat{\mathbf{Y}}_{T+h|T}\right)$$

[1]The `S+FinMetrics` fucntion `VAR.ar2ma` computes the Ψ_s matrices given the $\mathbf{\Pi}_j$ matrices using (11.6).

In practice, the second term MSE $\left(\mathbf{Y}_{T+h} - \hat{\mathbf{Y}}_{T+h|T}\right)$ is often ignored and $\hat{\mathbf{\Sigma}}(h)$ is computed using (11.7) as

$$\hat{\mathbf{\Sigma}}(h) = \sum_{s=0}^{h-1} \hat{\mathbf{\Psi}}_s \hat{\mathbf{\Sigma}} \hat{\mathbf{\Psi}}_s' \tag{11.10}$$

with $\hat{\mathbf{\Psi}}_s = \sum_{j=1}^{s} \hat{\mathbf{\Psi}}_{s-j} \hat{\mathbf{\Pi}}_j$. Lütkepohl (1991, Chap. 3) gave an approximation to MSE $\left(\mathbf{Y}_{T+h} - \hat{\mathbf{Y}}_{T+h|T}\right)$ which may be interpreted as a finite sample correction to (11.10).

Asymptotic $(1-\alpha) \cdot 100\%$ confidence intervals for the individual elements of $\hat{\mathbf{Y}}_{T+h|T}$ are then computed as

$$\left[\hat{y}_{k,T+h|T} - c_{1-\alpha/2}\hat{\sigma}_k(h),\ \hat{y}_{k,T+h|T} + c_{1-\alpha/2}\hat{\sigma}_k(h)\right]$$

where $c_{1-\alpha/2}$ is the $(1-\alpha/2)$ quantile of the standard normal distribution and $\hat{\sigma}_k(h)$ denotes the square root of the diagonal element of $\hat{\mathbf{\Sigma}}(h)$.

Example 66 *Forecasting exchange rates from a bivariate VAR*

Consider computing h-step forecasts, $h = 1, \ldots, 12$, along with estimated forecast standard errors from the bivariate VAR(1) model for exchange rates. Forecasts and forecast standard errors from the fitted VAR may be computed using the generic `S-PLUS` `predict` method

```
> uscn.pred = predict(var1.fit,n.predict=12)
```

The `predict` function recognizes `var1.fit` as a "`VAR`" object, and calls the appropriate method function `predict.VAR`. Alternatively, `predict.VAR` can be applied directly on an object inheriting from class "`VAR`". See the online help for explanations of the arguments to `predict.VAR`.

The output of `predict.VAR` is an object of class "`forecast`" for which there are `print`, `summary` and `plot` methods. To see just the forecasts, the `print` method will suffice:

```
> uscn.pred

Predicted Values:

                 dspot      fp
 1-step-ahead -0.0027 -0.0005
 2-step-ahead -0.0026 -0.0006
 3-step-ahead -0.0023 -0.0008
 4-step-ahead -0.0021 -0.0009
 5-step-ahead -0.0020 -0.0010
 6-step-ahead -0.0018 -0.0011
 7-step-ahead -0.0017 -0.0011
 8-step-ahead -0.0017 -0.0012
 9-step-ahead -0.0016 -0.0012
10-step-ahead -0.0016 -0.0013
11-step-ahead -0.0015 -0.0013
12-step-ahead -0.0015 -0.0013
```

The forecasts and their standard errors can be shown using `summary`:

```
> summary(uscn.pred)

Predicted Values with Standard Errors:

               dspot      fp
1-step-ahead -0.0027 -0.0005
   (std.err)  0.0137  0.0009
2-step-ahead -0.0026 -0.0006
   (std.err)  0.0139  0.0012

...

12-step-ahead -0.0015 -0.0013
    (std.err)  0.0140  0.0015
```

Lütkepohl's finite sample correction to the forecast standard errors computed from asymptotic theory may be obtained by using the optional argument `fs.correction=T` in the call to `predict.VAR`.

The forecasts can also be plotted together with the original data using the generic `plot` function as follows:

```
> plot(uscn.pred,uscn.ts,n.old=12)
```

where the `n.old` optional argument specifies the number of observations to plot from `uscn.ts`. If `n.old` is not specified, all the observations in `uscn.ts` will be plotted together with `uscn.pred`. Figure 11.5 shows the forecasts produced from the VAR(1) fit to the US/CN exchange rate data[2]. At the beginning of the forecast horizon the spot return is below its estimated mean value, and the forward premium is above its mean values. The spot return forecasts start off negative and grow slowly toward the mean, and the forward premium forecasts decline sharply toward the mean. The forecast standard errors for both sets of forecasts, however, are fairly large.

[2]Notice that the dates associated with the forecasts are not shown. This is the result of "`timeDate`" objects not having a well defined frequency from which to extrapolate dates.

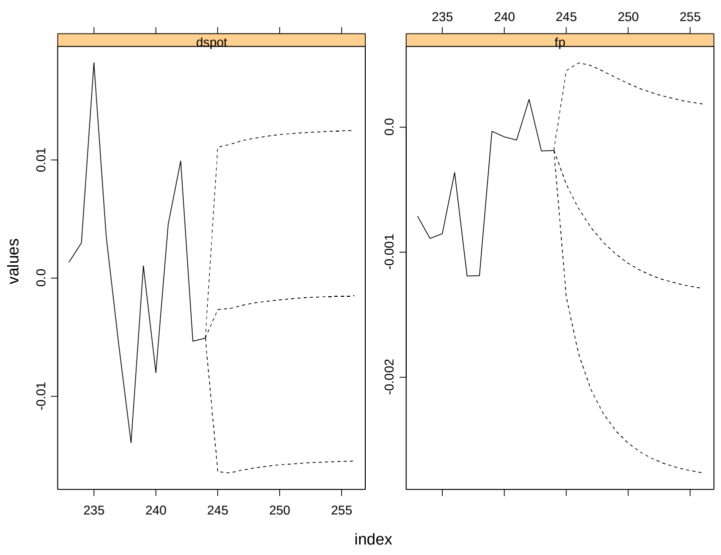


FIGURE 11.5. Predicted values from VAR(1) model fit to US/CN exchange rate data.

11.3.2 Simulation-Based Forecasting

The previous subsection showed how to generate multivariate forecasts from a fitted VAR model, using the chain-rule of forecasting (11.8). Since the multivariate forecast errors (11.9) are asymptotically normally distributed with covariance matrix (11.10), the forecasts of $\mathbf{Y}_{t+h}$ can be simulated by generating multivariate normal random variables with mean zero and covariance matrix (11.10). These simulation-based forecasts can be obtained by setting the optional argument `method` to `"mc"` in the call to `predict.VAR`.

When `method="mc"`, the multivariate normal random variables are actually generated as a vector of standard normal random variables scaled by the Cholesky factor of the covariance matrix (11.10). Instead of using standard normal random variables, one could also use the standardized residuals from the fitted VAR model. Simulation-based forecasts based on this approach are obtained by setting the optional argument method to `"bootstrap"` in the call to `predict.VAR`.

Example 67 *Simulation-based forecasts of exchange rate data from bivariate VAR*

The h-step forecasts ($h = 1, \ldots, 12$) for Δs_{t+h} and fp_{t+h} using the Monte Carlo simulation method are

```
> uscn.pred.MC = predict(var1.fit,n.predict=12,method="mc")
> summary(uscn.pred.MC)

Predicted Values with Standard Errors:

                dspot      fp
 1-step-ahead -0.0032 -0.0005
    (std.err)  0.0133  0.0009
 2-step-ahead -0.0026 -0.0006
    (std.err)  0.0133  0.0012
...
12-step-ahead -0.0013 -0.0013
    (std.err)  0.0139  0.0015
```

The Monte Carlo forecasts and forecast standard errors for fp_{t+h} are almost identical to those computed using the chain-rule of forecasting. The Monte Carlo forecasts for Δs_{t+h} are slightly different and the forecast standard errors are slightly larger than the corresponding values computed from the chain-rule.

The h-step forecasts computed from the bootstrap simulation method are

```
> uscn.pred.boot = predict(var1.fit,n.predict=12,
+ method="bootstrap")
> summary(uscn.pred.boot)

Predicted Values with Standard Errors:

                dspot      fp
 1-step-ahead -0.0020 -0.0005
    (std.err)  0.0138  0.0009
 2-step-ahead -0.0023 -0.0007
    (std.err)  0.0140  0.0012
...
12-step-ahead -0.0023 -0.0013
    (std.err)  0.0145  0.0015
```

As with the Monte Carlo forecasts, the bootstrap forecasts and forecast standard errors for fp_{t+h} are almost identical to those computed using the chain-rule of forecasting. The bootstrap forecasts for Δs_{t+h} are slightly different from the chain-rule and Monte Carlo forecasts. In particular, the bootstrap forecast standard errors are larger than corresponding values from the chain-rule and Monte Carlo methods.

The simulation-based forecasts described above are different from the traditional simulation-based approach taken in VAR literature, e.g., see

Runkle (1987). The traditional approach is implemented using the following procedure:

1. Obtain VAR coefficient estimates $\mathbf{\Pi}$ and residuals ε_t.

2. Simulate the fitted VAR model by Monte Carlo simulation or by bootstrapping the fitted residuals $\hat{\varepsilon}_t$.

3. Obtain new estimates of $\mathbf{\Pi}$ and forecasts of $\mathbf{Y}_{t+h}$ based on the simulated data.

The above procedure is repeated many times to obtain simulation-based forecasts as well as their confidence intervals. To illustrate this approach, generate 12-step ahead forecasts from the fitted VAR object `var1.fit` by Monte Carlo simulation using the `S+FinMetrics` function `simulate.VAR` as follows:

```
> set.seed(10)
> n.pred=12
> n.sim=100
> sim.pred = array(0,c(n.sim, n.pred, 2))
> y0 = seriesData(var1.fit$Y0)
> for (i in 1:n.sim) {
+    dat = simulate.VAR(var1.fit,n=243)
+    dat = rbind(y0,dat)
+    mod = VAR(dat~ar(1))
+    sim.pred[i,,] = predict(mod,n.pred)$values
+ }
```

The simulation-based forecasts are obtained by averaging the simulated forecasts:

```
> colMeans(sim.pred)
            [,1]       [,2]
 [1,] -0.0017917 -0.0012316
 [2,] -0.0017546 -0.0012508
 [3,] -0.0017035 -0.0012643
 [4,] -0.0016800 -0.0012741
 [5,] -0.0016587 -0.0012814
 [6,] -0.0016441 -0.0012866
 [7,] -0.0016332 -0.0012904
 [8,] -0.0016253 -0.0012932
 [9,] -0.0016195 -0.0012953
[10,] -0.0016153 -0.0012967
[11,] -0.0016122 -0.0012978
[12,] -0.0016099 -0.0012986
```

Comparing these forecasts with those in `uscn.pred` computed earlier, one can see that for the first few forecasts, these simulated forecasts are slightly different from the asymptotic forecasts. However, at larger steps, they approach the long run stable values of the asymptotic forecasts.

Conditional Forecasting

The forecasts algorithms considered up to now are unconditional multivariate forecasts. However, sometimes it is desirable to obtain forecasts of some variables in the system conditional on some knowledge of the future path of other variables in the system. For example, when forecasting multivariate macroeconomic variables using quarterly data from a VAR model, it may happen that some of the future values of certain variables in the VAR model are known, because data on these variables are released earlier than data on the other variables. By incorporating the knowledge of the future path of certain variables, in principle it should be possible to obtain more reliable forecasts of the other variables in the system. Another use of conditional forecasting is the generation of forecasts conditional on different "policy" scenarios. These scenario-based conditional forecasts allow one to answer the question: if something happens to some variables in the system in the future, how will it affect forecasts of other variables in the future?

`S+FinMetrics` provides a generic function `cpredict` for computing conditional forecasts, which has a method `cpredict.VAR` for "`VAR`" objects. The algorithms in `cpredict.VAR` are based on the conditional forecasting algorithms described in Waggoner and Zha (1999). Waggoner and Zha classified conditional information into "hard" conditions and "soft conditions". The hard conditions restrict the future values of certain variables at fixed values, while the soft conditions restrict the future values of certain variables in specified ranges. The arguments taken by `cpredict.VAR` are:

```
> args(cpredict.VAR)
function(object, n.predict = 1, newdata = NULL, olddata = NULL,
method = "mc", unbiased = T, variables.conditioned =
NULL, steps.conditioned = NULL, upper = NULL, lower =
NULL, middle = NULL, seed = 100, n.sim = 1000)
```

Like most `predict` methods in `S-PLUS`, the first argument must be a fitted model object, while the second argument, `n.predict`, specifies the number of steps to predict ahead. The arguments `newdata` and `olddata` can usually be safely ignored, unless exogenous variables were used in fitting the model.

With classical forecasts that ignore the uncertainty in coefficient estimates, hard conditional forecasts can be obtained in closed form as shown by Doan, Litterman and Sims (1984), and Waggoner and Zha (1999). To obtain hard conditional forecasts, the argument `middle` is used to specify fixed values of certain variables at certain steps. For example, to fix the

1-step ahead forecast of `dspot` in `var1.fit` at -0.005 and generate other predictions for 2-step ahead forecasts, use the following command:

```
> cpredict(var1.fit, n.predict=2, middle=-0.005,
+ variables="dspot", steps=1)

Predicted Values:

                dspot      fp
1-step-ahead -0.0050 -0.0005
2-step-ahead -0.0023 -0.0007
```

In the call to `cpredict`, the optional argument `variables` is used to specify the restricted variables, and `steps` to specify the restricted steps.

To specify a soft condition, the optional arguments `upper` and `lower` are used to specify the upper bound and lower bound, respectively, of a soft condition. Since closed form results are not available for soft conditional forecasts, either Monte Carlo simulation or bootstrap methods are used to obtain the actual forecasts. The simulations follow a similar procedure implemented in the function `predict.VAR`, except that a reject/accept method to sample from the distribution conditional on the soft conditions is used. For example, to restrict the range of the first 2-step ahead forecasts of `dspot` to be $(-0.004, -0.001)$ use:

```
> cpredict(var1.fit, n.predict=2, lower=c(-0.004, -0.004),
+ upper=c(-0.001, -0.001), variables="dspot",
+ steps=c(1,2))

Predicted Values:

                dspot      fp
1-step-ahead -0.0027 -0.0003
2-step-ahead -0.0029 -0.0005
```

11.4 Structural Analysis

The general VAR(p) model has many parameters, and they may be difficult to interpret due to complex interactions and feedback between the variables in the model. As a result, the dynamic properties of a VAR(p) are often summarized using various types of *structural analysis*. The three main types of structural analysis summaries are (1) *Granger causality tests*; (2) *impulse response functions*; and (3) *forecast error variance decompositions*. The following sections give brief descriptions of these summary measures.

11.4.1 Granger Causality

One of the main uses of VAR models is forecasting. The structure of the VAR model provides information about a variable's or a group of variables' forecasting ability for other variables. The following intuitive notion of a variable's forecasting ability is due to Granger (1969). If a variable, or group of variables, y_1 is found to be helpful for predicting another variable, or group of variables, y_2 then y_1 is said to *Granger-cause* y_2; otherwise it is said to *fail to Granger-cause* y_2. Formally, y_1 fails to Granger-cause y_2 if for all $s > 0$ the MSE of a forecast of $y_{2,t+s}$ based on $(y_{2,t}, y_{2,t-1}, \ldots)$ is the same as the MSE of a forecast of $y_{2,t+s}$ based on $(y_{2,t}, y_{2,t-1}, \ldots)$ and $(y_{1,t}, y_{1,t-1}, \ldots)$. Clearly, the notion of Granger causality does not imply true causality. It only implies forecasting ability.

Bivariate VAR Models

In a bivariate VAR(p) model for $\mathbf{Y}_t = (y_{1t}, y_{2t})'$, y_2 fails to Granger-cause y_1 if all of the p VAR coefficient matrices $\mathbf{\Pi}_1, \ldots, \mathbf{\Pi}_p$ are lower triangular. That is, the VAR(p) model has the form

$$\begin{aligned}\begin{pmatrix} y_{1t} \\ y_{2t} \end{pmatrix} &= \begin{pmatrix} c_1 \\ c_2 \end{pmatrix} + \begin{pmatrix} \pi_{11}^1 & 0 \\ \pi_{21}^1 & \pi_{22}^1 \end{pmatrix} \begin{pmatrix} y_{1t-1} \\ y_{2t-1} \end{pmatrix} + \cdots \\ &+ \begin{pmatrix} \pi_{11}^p & 0 \\ \pi_{21}^p & \pi_{22}^p \end{pmatrix} \begin{pmatrix} y_{1t-p} \\ y_{2t-p} \end{pmatrix} + \begin{pmatrix} \varepsilon_{1t} \\ \varepsilon_{2t} \end{pmatrix}\end{aligned}$$

so that all of the coefficients on lagged values of y_2 are zero in the equation for y_1. Similarly, y_1 fails to Granger-cause y_2 if all of the coefficients on lagged values of y_1 are zero in the equation for y_2. The p linear coefficient restrictions implied by Granger non-causality may be tested using the Wald statistic (11.5). Notice that if y_2 fails to Granger-cause y_1 and y_1 fails to Granger-cause y_2, then the VAR coefficient matrices $\mathbf{\Pi}_1, \ldots, \mathbf{\Pi}_p$ are diagonal.

General VAR Models

Testing for Granger non-causality in general n variable VAR(p) models follows the same logic used for bivariate models. For example, consider a VAR(p) model with $n = 3$ and $\mathbf{Y}_t = (y_{1t}, y_{2t}, y_{3t})'$. In this model, y_2 does not Granger-cause y_1 if all of the coefficients on lagged values of y_2 are zero in the equation for y_1. Similarly, y_3 does not Granger-cause y_1 if all of the coefficients on lagged values of y_3 are zero in the equation for y_1. These simple linear restrictions may be tested using the Wald statistic (11.5). The reader is encouraged to consult Lütkepohl (1991) or Hamilton (1994) for more details and examples.

Example 68 *Testing for Granger causality in bivariate VAR(2) model for exchange rates*

Consider testing for Granger causality in a bivariate VAR(2) model for $\mathbf{Y}_t = (\Delta s_t, fp_t)'$. Using the notation of (11.2), fp_t does not Granger cause Δs_t if $\pi_{12}^1 = 0$ and $\pi_{12}^2 = 0$. Similarly, Δs_t does not Granger cause fp_t if $\pi_{21}^1 = 0$ and $\pi_{21}^2 = 0$. These hypotheses are easily tested using the Wald statistic (11.5). The restriction matrix $\mathbf{R}$ for the hypothesis that fp_t does not Granger cause Δs_t is

$$\mathbf{R} = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

and the matrix for the hypothesis that Δs_t does not Granger cause fp_t is

$$\mathbf{R} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

The S-PLUS commands to compute and evaluate these Granger causality Wald statistics are

```
> var2.fit = VAR(cbind(dspot,fp)~ar(2),data=uscn.ts)
> # H0: fp does not Granger cause dspot
> R = matrix(c(0,0,1,0,0,0,0,0,0,0,
+ 0,0,0,0,1,0,0,0,0,0),
+ 2,10,byrow=T)
> vecPi = as.vector(coef(var2.fit))
> avar = R%*%vcov(var2.fit)%*%t(R)
> wald = t(R%*%vecPi)%*%solve(avar)%*%(R%*%vecPi)
> wald
         [,1]
[1,] 8.468844
> 1-pchisq(wald,2)
[1] 0.01448818

> R = matrix(c(0,0,0,0,0,0,1,0,0,0,
+ 0,0,0,0,0,0,0,0,1,0),
+ 2,10,byrow=T)
> vecPi = as.vector(coef(var2.fit))
> avar = R%*%vcov(var2.fit)%*%t(R)
> wald = t(R%*%vecPi)%*%solve(avar)%*%(R%*%vecPi)
> wald
      [,1]
[1,] 6.157
> 1-pchisq(wald,2)
[1] 0.04604
```

The p-values for the Wald tests indicate a fairly strong rejection of the null that fp_t does not Granger cause Δs_t but only a weak rejection of the null that Δs_t does not Granger cause fp_t. Hence, lagged values of fp_t appear to be useful for forecasting future values of Δs_t and lagged values of Δs_t appear to be useful for forecasting future values of fp_t.

11.4.2 Impulse Response Functions

Any covariance stationary VAR(p) process has a Wold representation of the form

$$\mathbf{Y}_t = \boldsymbol{\mu} + \boldsymbol{\varepsilon}_t + \boldsymbol{\Psi}_1\boldsymbol{\varepsilon}_{t-1} + \boldsymbol{\Psi}_2\boldsymbol{\varepsilon}_{t-2} + \cdots \tag{11.11}$$

where the $(n \times n)$ moving average matrices $\boldsymbol{\Psi}_s$ are determined recursively using (11.6). It is tempting to interpret the (i, j)-th element, ψ_{ij}^s, of the matrix $\boldsymbol{\Psi}_s$ as the dynamic multiplier or impulse response

$$\frac{\partial y_{i,t+s}}{\partial \varepsilon_{j,t}} = \frac{\partial y_{i,t}}{\partial \varepsilon_{j,t-s}} = \psi_{ij}^s,\ i, j = 1, \ldots, n$$

However, this interpretation is only possible if $\text{var}(\boldsymbol{\varepsilon}_t) = \boldsymbol{\Sigma}$ is a diagonal matrix so that the elements of $\boldsymbol{\varepsilon}_t$ are uncorrelated. One way to make the errors uncorrelated is to follow Sims (1980) and estimate the *triangular structural* VAR(p) model

$$\begin{aligned}
y_{1t} &= c_1 + \boldsymbol{\gamma}_{11}'\mathbf{Y}_{t-1} + \cdots + \boldsymbol{\gamma}_{1p}'\mathbf{Y}_{t-p} + \eta_{1t} \qquad (11.12)\\
y_{2t} &= c_1 + \beta_{21}y_{1t} + \boldsymbol{\gamma}_{21}'\mathbf{Y}_{t-1} + \cdots + \boldsymbol{\gamma}_{2p}'\mathbf{Y}_{t-p} + \eta_{2t}\\
y_{3t} &= c_1 + \beta_{31}y_{1t} + \beta_{32}y_{2t} + \boldsymbol{\gamma}_{31}'\mathbf{Y}_{t-1} + \cdots + \boldsymbol{\gamma}_{3p}'\mathbf{Y}_{t-p} + \eta_{3t}\\
&\vdots\\
y_{nt} &= c_1 + \beta_{n1}y_{1t} + \cdots + \beta_{n,n-1}y_{n-1,t} + \boldsymbol{\gamma}_{n1}'\mathbf{Y}_{t-1} + \cdots + \boldsymbol{\gamma}_{np}'\mathbf{Y}_{t-p} + \eta_{nt}
\end{aligned}$$

In matrix form, the triangular structural VAR(p) model is

$$\mathbf{B}\mathbf{Y}_t = \mathbf{c} + \boldsymbol{\Gamma}_1\mathbf{Y}_{t-1} + \boldsymbol{\Gamma}_2\mathbf{Y}_{t-2} + \cdots + \boldsymbol{\Gamma}_p\mathbf{Y}_{t-p} + \boldsymbol{\eta}_t \tag{11.13}$$

where

$$\mathbf{B} = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ -\beta_{21} & 1 & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots \\ -\beta_{n1} & -\beta_{n2} & \cdots & 1 \end{pmatrix} \tag{11.14}$$

is a lower triangular matrix with $1's$ along the diagonal. The algebra of least squares will ensure that the estimated covariance matrix of the error vector $\boldsymbol{\eta}_t$ is diagonal. The uncorrelated/orthogonal errors $\boldsymbol{\eta}_t$ are referred to as *structural* errors.

The triangular structural model (11.12) imposes the *recursive causal ordering*

$$y_1 \rightarrow y_2 \rightarrow \cdots \rightarrow y_n \tag{11.15}$$

The ordering (11.15) means that the contemporaneous values of the variables to the left of the arrow $\rightarrow$ affect the contemporaneous values of the variables to the right of the arrow but not vice-versa. These contemporaneous effects are captured by the coefficients β_{ij} in (11.12). For example, the ordering $y_1 \rightarrow y_2 \rightarrow y_3$ imposes the restrictions: y_{1t} affects y_{2t} and y_{3t} but y_{2t} and y_{3t} do not affect y_1; y_{2t} affects y_{3t} but y_{3t} does not affect y_{2t}. Similarly, the ordering $y_2 \rightarrow y_3 \rightarrow y_1$ imposes the restrictions: y_{2t} affects y_{3t} and y_{1t} but y_{3t} and y_{1t} do not affect y_2; y_{3t} affects y_{1t} but y_{1t} does not affect y_{3t}. For a VAR(p) with n variables there are $n!$ possible recursive causal orderings. Which ordering to use in practice depends on the context and whether prior theory can be used to justify a particular ordering. Results from alternative orderings can always be compared to determine the sensitivity of results to the imposed ordering.

Once a recursive ordering has been established, the Wold representation of $\mathbf{Y}_t$ based on the orthogonal errors $\boldsymbol{\eta}_t$ is given by

$$\mathbf{Y}_t = \boldsymbol{\mu} + \boldsymbol{\Theta}_0\boldsymbol{\eta}_t + \boldsymbol{\Theta}_1\boldsymbol{\eta}_{t-1} + \boldsymbol{\Theta}_2\boldsymbol{\eta}_{t-2} + \cdots \tag{11.16}$$

where $\boldsymbol{\Theta}_0 = \mathbf{B}^{-1}$ is a lower triangular matrix. The impulse responses to the orthogonal shocks η_{jt} are

$$\frac{\partial y_{i,t+s}}{\partial \eta_{j,t}} = \frac{\partial y_{i,t}}{\partial \eta_{j,t-s}} = \theta_{ij}^{s},\ i,j = 1,\ldots,n; s > 0 \tag{11.17}$$

where θ_{ij}^{s} is the (i,j) th element of $\boldsymbol{\Theta}_s$. A plot of θ_{ij}^{s} against s is called the *orthogonal impulse response function* (IRF) of y_i with respect to η_j. With n variables there are n^2 possible impulse response functions.

In practice, the orthogonal IRF (11.17) based on the triangular VAR(p) (11.12) may be computed directly from the parameters of the non triangular VAR(p) (11.1) as follows. First, decompose the residual covariance matrix $\boldsymbol{\Sigma}$ as

$$\boldsymbol{\Sigma} = \mathbf{ADA}'$$

where $\mathbf{A}$ is an invertible lower triangular matrix with $1's$ along the diagonal and $\mathbf{D}$ is a diagonal matrix with positive diagonal elements. Next, define the structural errors as

$$\boldsymbol{\eta}_t = \mathbf{A}^{-1}\boldsymbol{\varepsilon}_t$$

These structural errors are orthogonal by construction since $\mathrm{var}(\boldsymbol{\eta}_t) = \mathbf{A}^{-1}\boldsymbol{\Sigma}\mathbf{A}^{-1\prime} = \mathbf{A}^{-1}\mathbf{ADA}'\mathbf{A}^{-1\prime} = \mathbf{D}$. Finally, re-express the Wold representation (11.11) as

$$\begin{aligned}\mathbf{Y}_t &= \boldsymbol{\mu} + \mathbf{AA}^{-1}\boldsymbol{\varepsilon}_t + \boldsymbol{\Psi}_1\mathbf{AA}^{-1}\boldsymbol{\varepsilon}_{t-1} + \boldsymbol{\Psi}_2\mathbf{AA}^{-1}\boldsymbol{\varepsilon}_{t-2} + \cdots \\ &= \boldsymbol{\mu} + \boldsymbol{\Theta}_0\boldsymbol{\eta}_t + \boldsymbol{\Theta}_1\boldsymbol{\eta}_{t-1} + \boldsymbol{\Theta}_2\boldsymbol{\eta}_{t-2} + \cdots\end{aligned}$$

where $\boldsymbol{\Theta}_j = \boldsymbol{\Psi}_j\mathbf{A}$. Notice that the structural $\mathbf{B}$ matrix in (11.13) is equal to $\mathbf{A}^{-1}$.

Computing the Orthogonal Impulse Response Function Using the S+FinMetrics Function impRes

The orthogonal impulse response function (11.17) from a triangular structural VAR model (11.13) may be computed using the `S+FinMetrics` function `impRes`. The function `impRes` has arguments

```
> args(impRes)
function(x, period = NULL, std.err = "none", plot = F,
unbiased = T, order = NULL, ...)
```

where `x` is an object of class "`VAR`" and `period` specifies the number of responses to compute. By default, no standard errors for the responses are computed. To compute asymptotic standard errors for the responses, specify `std.err="asymptotic"`. To create a panel plot of all the response functions, specify `plot=T`. The default recursive causal ordering is based on the ordering of the variables in $\mathbf{Y}_t$ when the VAR model is fit. The optional argument `order` may be used to specify a different recursive causal ordering for the computation of the impulse responses. The argument `order` accepts a character vector of variable names whose order defines the recursive causal ordering. The output of `impRes` is an object of class "`impDecomp`" for which there are `print`, `summary` and `plot` methods. The following example illustrates the use of `impRes`.

Example 69 *IRF from VAR(1) for exchange rates*

Consider again the VAR(1) model for $\mathbf{Y}_t = (\Delta s_t, fp_t)'$. For the impulse response analysis, the initial ordering of the variables imposes the assumption that structural shocks to fp_t have no contemporaneous effect on Δs_t but structural shocks to Δs_t do have a contemporaneous effect on fp_t. To compute the four impulse response functions

$$\frac{\partial \Delta s_{t+h}}{\partial \eta_{1t}}, \ \frac{\partial \Delta s_{t+h}}{\partial \eta_{2t}}, \ \frac{\partial fp_{t+h}}{\partial \eta_{1t}}, \ \frac{\partial fp_{t+h}}{\partial \eta_{2t}}$$

for $h = 1, \ldots, 12$ we use `S+FinMetrics` function `impRes`. The first twelve impulse responses from the VAR(1) model for exchange rates are computed using

```
> uscn.irf = impRes(var1.fit, period=12, std.err="asymptotic")
```

The `print` method shows the impulse response values without standard errors:

```
> uscn.irf

Impulse Response Function:
(with responses in rows, and innovations in columns)
```

```
, , lag.0
        dspot      fp
dspot  0.0136  0.0000
   fp  0.0000  0.0009

, , lag.1
        dspot      fp
dspot -0.0018 -0.0013
   fp  0.0001  0.0007

, , lag.2
        dspot      fp
dspot  0.0000 -0.0009
   fp  0.0001  0.0006

...

, , lag.11
        dspot      fp
dspot  0.0000 -0.0001
   fp  0.0000  0.0001
```

The `summary` method will display the responses with standard errors and t-statistics. The `plot` method will produce a four panel Trellis graphics plot of the impulse responses

```
> plot(uscn.irf)
```

A plot of the impulse responses can also be created in the initial call to `impRes` by using the optional argument `plot=T`.

Figure 11.6 shows the impulse response functions along with asymptotic standard errors. The top row shows the responses of Δs_t to the structural shocks, and the bottom row shows the responses of fp_t to the structural shocks. In response to the first structural shock, η_{1t}, Δs_t initially increases but then drops quickly to zero after 2 months. Similarly, fp_t initially increases, reaches its peak response in 2 months and then gradually drops off to zero after about a year. In response to the second shock, η_{2t}, by assumption Δs_t has no initial response. At one month, a sharp drop occurs in Δs_t followed by a gradual return to zero after about a year. In contrast, fp_t initially increases and then gradually drops to zero after about a year.

The orthogonal impulse responses in Figure 11.6 are based on the recursive causal ordering $\Delta s_t \rightarrow fp_t$. It must always be kept in mind that this ordering identifies the orthogonal structural shocks η_{1t} and η_{2t}. If the ordering is reversed, then a different set of structural shocks will be identified, and these may give very different impulse response functions. To compute

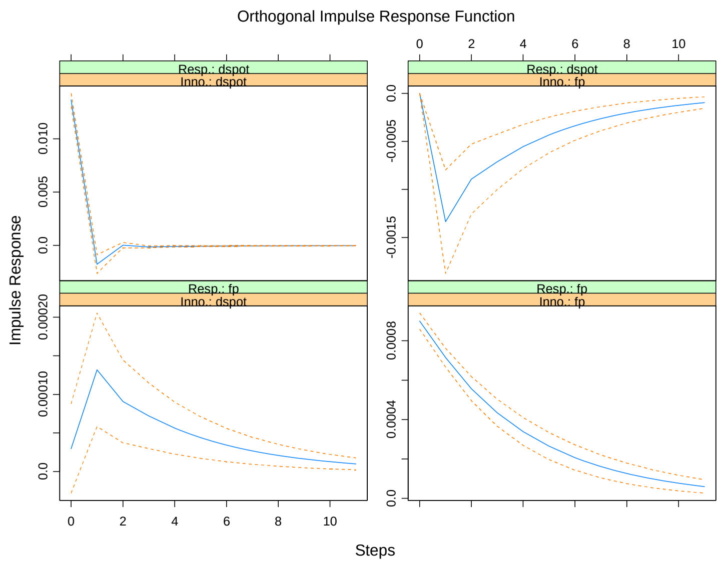


FIGURE 11.6. Impulse response function from VAR(1) model fit to US/CN exchange rate data with Δs_t ordered first.

the orthogonal impulse responses using the alternative ordering $fp_t \rightarrow \Delta s_t$ specify `order=c("fp","dspot")` in the call to `impRes`:

```
> uscn.irf2 = impRes(var1.fit,period=12,std.err="asymptotic",
+ order=c("fp","dspot"),plot=T)
```

These impulse responses are presented in Figure 11.7 and are almost identical to those computed using the ordering $\Delta s_t \rightarrow fp_t$. The reason for this response is that the reduced form VAR residuals $\hat{\varepsilon}_{1t}$ and $\hat{\varepsilon}_{2t}$ are almost uncorrelated. To see this, the residual correlation matrix may be computed using

```
> sd.vals = sqrt(diag(var1.fit$Sigma))
> cor.mat = var1.fit$Sigma/outer(sd.vals,sd.vals)
> cor.mat
         dspot       fp
dspot 1.000000 0.033048
   fp 0.033048 1.000000
```

Because of the near orthogonality in the reduced form VAR errors, the error in the Δs_t equation may be interpreted as an orthogonal shock to the exchange rate and the error in the fp_t equation may be interpreted as an orthogonal shock to the forward premium.

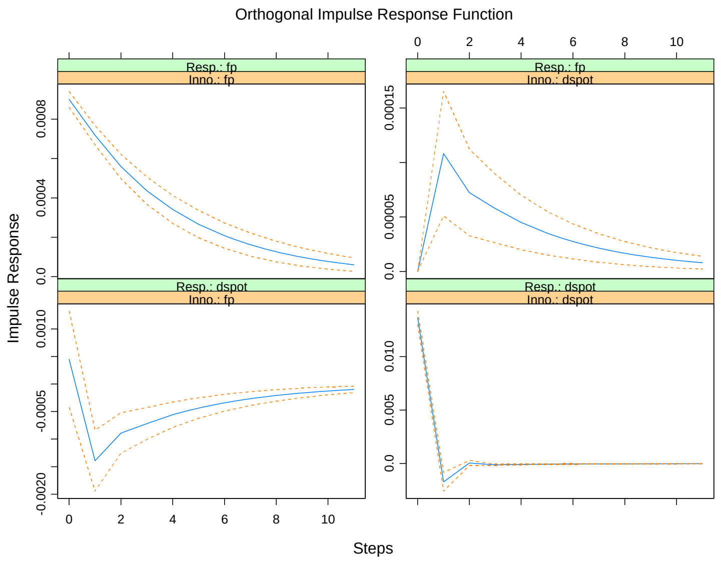


FIGURE 11.7. Impulse response function from VAR(1) model fit to US/CN exchange rate with fp_t ordered first.

11.4.3 Forecast Error Variance Decompositions

The *forecast error variance decomposition* (FEVD) answers the question: what portion of the variance of the forecast error in predicting $y_{i,T+h}$ is due to the structural shock η_j? Using the orthogonal shocks $\boldsymbol{\eta}_t$ the h-step ahead forecast error vector, with known VAR coefficients, may be expressed as

$$\mathbf{Y}_{T+h} - \mathbf{Y}_{T+h|T} = \sum_{s=0}^{h-1} \boldsymbol{\Theta}_s \boldsymbol{\eta}_{T+h-s}$$

For a particular variable $y_{i,T+h}$, this forecast error has the form

$$y_{i,T+h} - y_{i,T+h|T} = \sum_{s=0}^{h-1} \theta_{i1}^s \eta_{1,T+h-s} + \cdots + \sum_{s=0}^{h-1} \theta_{in}^s \eta_{n,T+h-s}$$

Since the structural errors are orthogonal, the variance of the h-step forecast error is

$$\text{var}(y_{i,T+h} - y_{i,T+h|T}) = \sigma_{\eta_1}^2 \sum_{s=0}^{h-1} (\theta_{i1}^s)^2 + \cdots + \sigma_{\eta_n}^2 \sum_{s=0}^{h-1} (\theta_{in}^s)^2$$

where $\sigma^2_{\eta j} = \text{var}(\eta_{jt})$. The portion of $\text{var}(y_{i,T+h} - y_{i,T+h|T})$ due to shock η_j is then

$$\text{FEVD}_{i,j}(h) = \frac{\sigma^2_{\eta_j} \sum_{s=0}^{h-1} \left(\theta^s_{ij}\right)^2}{\sigma^2_{\eta_1} \sum_{s=0}^{h-1} \left(\theta^s_{i1}\right)^2 + \cdots + \sigma^2_{\eta_n} \sum_{s=0}^{h-1} \left(\theta^s_{in}\right)^2},\ i,j = 1,\ldots,n \tag{11.18}$$

In a VAR with n variables there will be n^2 $\text{FEVD}_{i,j}(h)$ values. It must be kept in mind that the FEVD in (11.18) depends on the recursive causal ordering used to identify the structural shocks $\boldsymbol{\eta}_t$ and is not unique. Different causal orderings will produce different FEVD values.

Computing the FEVD Using the `S+FinMetrics` Function `fevDec`

Once a VAR model has been fit, the `S+FinMetrics` function `fevDec` may be used to compute the orthogonal FEVD. The function `fevDec` has arguments

```
> args(fevDec)
function(x, period = NULL, std.err = "none", plot = F,
unbiased = F, order = NULL, ...)
```

where `x` is an object of class "`VAR`" and `period` specifies the number of responses to compute. By default, no standard errors for the responses are computed and no plot is created. To compute asymptotic standard errors for the responses, specify `std.err="asymptotic"` and to plot the decompositions, specify `plot=T`. The default recursive causal ordering is based on the ordering of the variables in $\mathbf{Y}_t$ when the VAR model is fit. The optional argument `order` may be used to specify a different recursive causal ordering for the computation of the FEVD. The argument `order` accepts a text string vector of variable names whose order defines the recursive causal ordering. The output of `fevDec` is an object of class "`impDecomp`" for which there are `print`, `summary` and `plot` methods. The use of `fevDec` is illustrated with the following example.

Example 70 *FEVD from VAR(1) for exchange rates*

The orthogonal FEVD of the forecast errors from the VAR(1) model fit to the US/CN exchange rate data using the recursive causal ordering $\Delta s_t \rightarrow fp_t$ is computed using

```
> uscn.fevd = fevDec(var1.fit,period=12,
+ std.err="asymptotic")
> uscn.fevd

Forecast Error Variance Decomposition:
(with responses in rows, and innovations in columns)
```

```
, , 1-step-ahead
       dspot     fp
dspot 1.0000 0.0000
   fp 0.0011 0.9989

, , 2-step-ahead
       dspot     fp
dspot 0.9907 0.0093
   fp 0.0136 0.9864

...

, , 12-step-ahead
       dspot     fp
dspot 0.9800 0.0200
   fp 0.0184 0.9816
```

The `summary` method adds standard errors to the above output if they are computed in the call to `fevDec`. The `plot` method produces a four panel Trellis graphics plot of the decompositions:

```
> plot(uscn.fevd)
```

The FEVDs in Figure 11.8 show that most of the variance of the forecast errors for Δs_{t+s} at all horizons s is due to the orthogonal Δs_t innovations. Similarly, most of the variance of the forecast errors for fp_{t+s} is due to the orthogonal fp_t innovations.

The FEVDs using the alternative recursive causal ordering $fp_t \rightarrow \Delta s_t$ are computed using

```
> uscn.fevd2 = fevDec(var1.fit,period=12,
+ std.err="asymptotic",order=c("fp","dspot"),plot=T)
```

and are illustrated in Figure 11.9. Since the residual covariance matrix is almost diagonal (see analysis of IRF above), the FEVDs computed using the alternative ordering are almost identical to those computed with the initial ordering.

11.5 An Extended Example

In this example the causal relations and dynamic interactions among monthly real stock returns, real interest rates, real industrial production growth and the inflation rate is investigated using a VAR model. The analysis is similar to that of Lee (1992). The variables are in the `S+FinMetrics` "`timeSeries`" object `varex.ts`

```
> colIds(varex.ts)
```

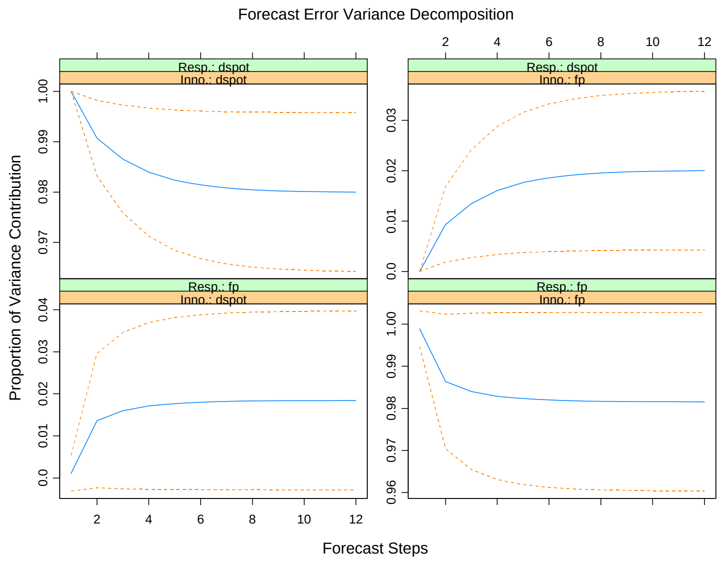


FIGURE 11.8. Orthogonal FEVDs computed from VAR(1) model fit to US/CN exchange rate data using the recursive causal ordering with Δs_t first.

```
[1] "MARKET.REAL" "RF.REAL" "INF" "IPG"
```

Details about the data are in the documentation slot of `varex.ts`

```
> varex.ts@documentation
```

To be comparable to the results in Lee (1992), the analysis is conducted over the postwar period January 1947 through December 1987

```
> smpl = (positions(varex.ts) >= timeDate("1/1/1947") &
+ positions(varex.ts) < timeDate("1/1/1988"))
```

The data over this period is displayed in Figure 11.10. All variables appear to be $I(0)$, but the real T-bill rate and the inflation rate appear to be highly persistent.

To begin the analysis, autocorrelations and cross correlations at leads and lags are computed using

```
> varex.acf = acf(varex.ts[smpl,])
```

and are illustrated in Figure 11.11. The real return on the market shows a significant positive first lag autocorrelation, and inflation appears to lead the real market return with a negative sign. The real T-bill rate is highly positively autocorrelated, and inflation appears to lead the real T-bill rate strongly with a negative sign. Inflation is also highly positively autocorrelated and, interestingly, the real T-bill rate appears to lead inflation with

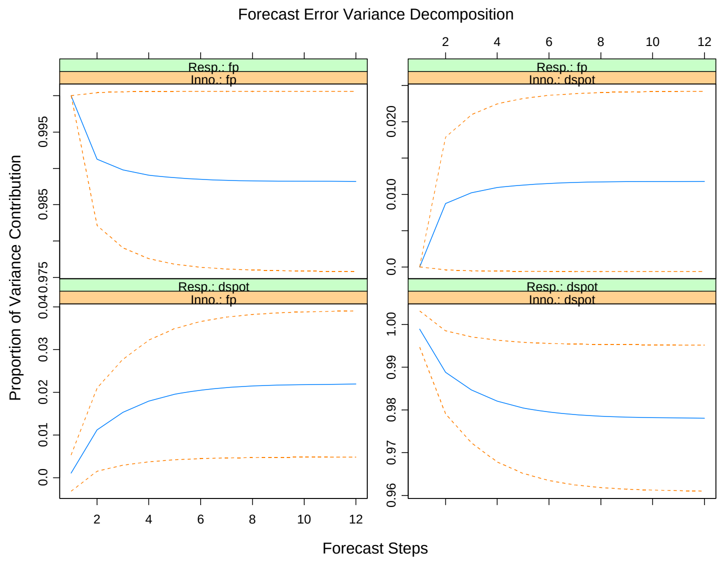


FIGURE 11.9. Orthogonal FEVDs from VAR(1) model fit to US/CN exchange rate data using recursive causal ordering with fp_t first.

a positive sign. Finally, industrial production growth is slightly positively autocorrelated, and the real market return appears to lead industrial production growth with a positive sign.

The VAR(p) model is fit with the lag length selected by minimizing the AIC and a maximum lag length of 6 months:

```
> varAIC.fit = VAR(varex.ts,max.ar=6,criterion="AIC",
+ start="Jan 1947",end="Dec 1987",
+ in.format="%m %Y")
```

The lag length selected by minimizing AIC is $p = 2$:

```
> varAIC.fit$info
      ar(1)  ar(2)  ar(3)  ar(4)  ar(5)  ar(6)
AIC -14832 -14863 -14853 -14861 -14855 -14862
> varAIC.fit$ar.order
[1] 2
```

The results of the VAR(2) fit are

```
> summary(varAIC.out)

Call:
VAR(data = varex.ts, start = "Jan 1947", end = "Dec 1987",
```

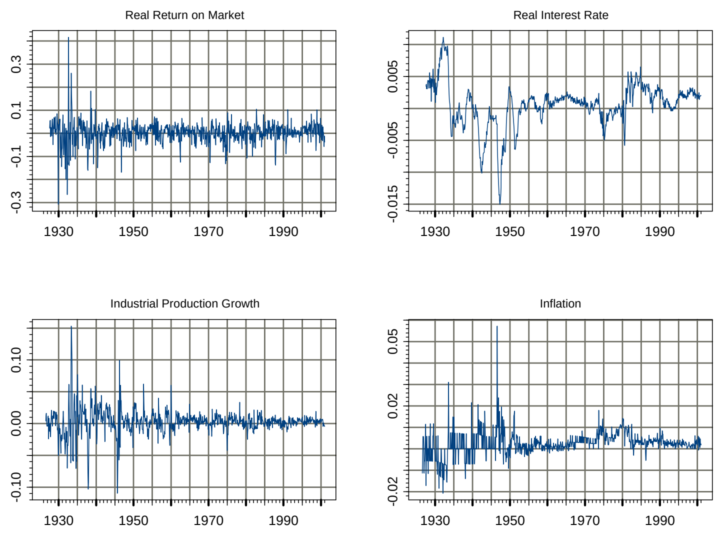


FIGURE 11.10. Monthly data on stock returns, interest rates, output growth and inflation.

```
max.ar = 6, criterion = "AIC", in.format = "%m %Y")

Coefficients:
                 MARKET.REAL  RF.REAL      INF     IPG
      (Intercept)    0.0074    0.0002   0.0010  0.0019
        (std.err)    0.0023    0.0001   0.0002  0.0007
         (t.stat)    3.1490    4.6400   4.6669  2.5819

 MARKET.REAL.lag1    0.2450    0.0001   0.0072  0.0280
        (std.err)    0.0470    0.0011   0.0042  0.0146
         (t.stat)    5.2082    0.0483   1.7092  1.9148

     RF.REAL.lag1    0.8146    0.8790   0.5538  0.3772
        (std.err)    2.0648    0.0470   0.1854  0.6419
         (t.stat)    0.3945   18.6861   2.9867  0.5877

         INF.lag1   -1.5020   -0.0710   0.4616 -0.0722
        (std.err)    0.4932    0.0112   0.0443  0.1533
         (t.stat)   -3.0451   -6.3147  10.4227 -0.4710

                 MARKET.REAL  RF.REAL      INF     IPG
         IPG.lag1   -0.0003    0.0031  -0.0143  0.3454
```

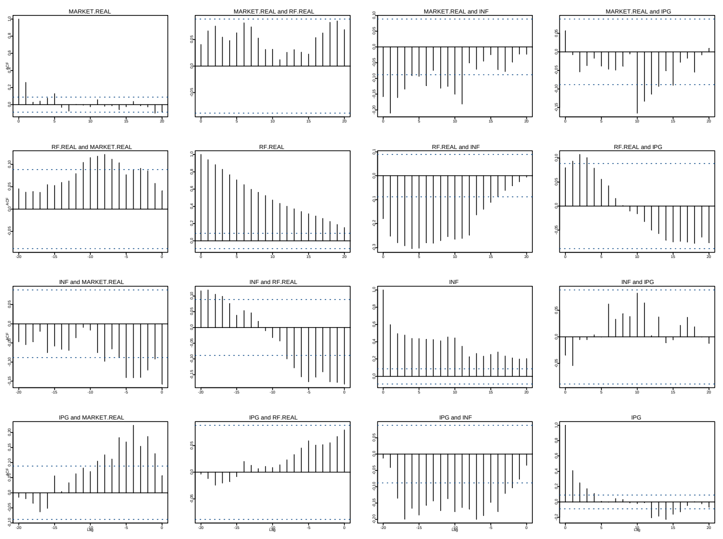


FIGURE 11.11. Autocorrelations and cross correlations at leads and lags of data in VAR model.

```
        (std.err)   0.1452      0.0033   0.0130   0.0452
         (t.stat)  -0.0018      0.9252  -1.0993   7.6501

MARKET.REAL.lag2  -0.0500      0.0022  -0.0066   0.0395
        (std.err)   0.0466      0.0011   0.0042   0.0145
         (t.stat)  -1.0727      2.0592  -1.5816   2.7276

    RF.REAL.lag2  -0.3481      0.0393  -0.5855  -0.3289
        (std.err)   1.9845      0.0452   0.1782   0.6169
         (t.stat)  -0.1754      0.8699  -3.2859  -0.5331

        INF.lag2  -0.0602      0.0079   0.2476  -0.0370
        (std.err)   0.5305      0.0121   0.0476   0.1649
         (t.stat)  -0.1135      0.6517   5.1964  -0.2245

          MARKET.REAL  RF.REAL      INF      IPG
 IPG.lag2  -0.1919      0.0028   0.0154   0.0941
(std.err)   0.1443      0.0033   0.0130   0.0449
 (t.stat)  -1.3297      0.8432   1.1868   2.0968

Regression Diagnostics:
               MARKET.REAL RF.REAL    INF    IPG
```

```
     R-squared 0.1031      0.9299  0.4109 0.2037
Adj. R-squared 0.0882      0.9287  0.4011 0.1905
  Resid. Scale 0.0334      0.0008  0.0030 0.0104

Information Criteria:
   logL    AIC    BIC     HQ
   7503 -14935 -14784 -14875

                   total residual
Degree of freedom:   489      480
Time period: from Mar 1947 to Nov 1987
```

The signs of the statistically significant coefficient estimates corroborate the informal analysis of the multivariate autocorrelations and cross lag autocorrelations. In particular, the real market return is positively related to its own lag but negatively related to the first lag of inflation. The real T-bill rate is positively related to its own lag, negatively related to the first lag of inflation, and positively related to the first lag of the real market return. Industrial production growth is positively related to its own lag and positively related to the first two lags of the real market return. Judging from the coefficients it appears that inflation Granger causes the real market return and the real T-bill rate, the real T-bill rate Granger causes inflation, and the real market return Granger causes the real T-bill rate and industrial production growth. These observations are confirmed with formal tests for Granger non-causality. For example, the Wald statistic for testing the null hypothesis that the real market return does not Granger-cause industrial production growth is

```
> bigR = matrix(0,2,36)
> bigR[1,29]=bigR[2,33]=1
> vecPi = as.vector(coef(varAIC.fit))
> avar = bigR%*%vcov(varAIC.fit)%*%t(bigR)
> wald = t(bigR%*%vecPi)%*%solve(avar)%*%(bigR%*%vecPi)
> as.numeric(wald)
[1] 13.82
> 1-pchisq(wald,2)
[1] 0.0009969
```

The 24-period IRF using the recursive causal ordering `MARKET.REAL` $\rightarrow$ `RF.REAL` $\rightarrow$ `IPG` $\rightarrow$ `INF` is computed using

```
> varAIC.irf = impRes(varAIC.fit,period=24,
+ order=c("MARKET.REAL","RF.REAL","IPG","INF"),
+ std.err="asymptotic",plot=T)
```

and is illustrated in Figure 11.12. The responses of `MARKET.REAL` to unexpected orthogonal shocks to the other variables are given in the first row of

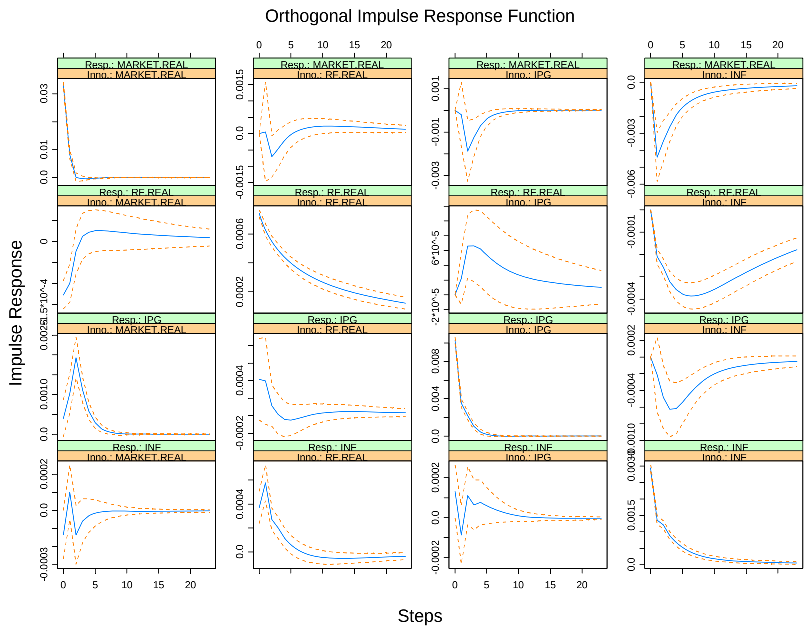


FIGURE 11.12. IRF using the recursive causal ordering `MARKET.REAL` $\longrightarrow$ `RF.REAL` $\longrightarrow$ `IPG` $\longrightarrow$ `INF`.

the figure. Most notable is the strong negative response of `MARKET.REAL` to an unexpected increase in inflation. Notice that it takes about ten months for the effect of the shock to dissipate. The responses of `RF.REAL` to the orthogonal shocks is given in the second row of the figure. `RF.REAL` also reacts negatively to an inflation shock and the effect of the shock is felt for about two years. The responses of `IPG` to the orthogonal shocks is given in the third row of the figure. Industrial production growth responds positively to an unexpected shock to `MARKET.REAL` and negatively to shocks to `RF.REAL` and `INF`. These effects, however, are generally short term. Finally, the fourth row gives the responses of `INF` to the orthogonal shocks. Inflation responds positively to a shock to the real T-bill rate, but this effect is short-lived.

The 24 month FEVD computed using the recursive causal ordering as specified by `MARKET.REAL` $\longrightarrow$ `RF.REAL` $\longrightarrow$ `IPG` $\longrightarrow$ `INF`,

```
> varAIC.fevd = fevDec(varAIC.out,period=24,
> order=c("MARKET.REAL","RF.REAL","IPG","INF"),
> std.err="asymptotic",plot=T)
```

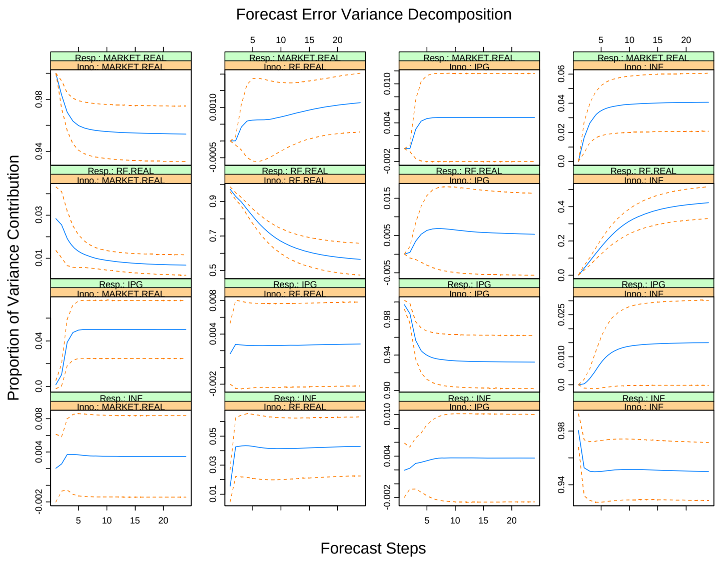


FIGURE 11.13. FEVDs using the recursive causal ordering `MARKET.REAL` $\rightarrow$ `RF.REAL` $\rightarrow$ `IPG` $\rightarrow$ `INF`.

is illustrated in Figure 11.13. The first row gives the variance decompositions for `MARKET.REAL` and shows that most of the variance of the forecast errors is due to own shocks. The second row gives the decompositions for `RF.REAL`. At short horizons, most of the variance is attributable to own shocks but at long horizons inflation shocks account for almost half the variance. The third row gives the variance decompositions for `IPG`. Most of the variance is due to own shocks and a small fraction is due to `MARKET.REAL` shocks. Finally, the fourth row shows that the forecast error variance of `INF` is due mostly to its own shocks.

The IRFs and FEVDs computed above depend on the imposed recursive causal ordering. However, in this example, the ordering of the variables will have little effect on the IRFs and FEVDs because the errors in the reduced form VAR are nearly uncorrelated:

```
> sd.vals = sqrt(diag(varAIC.out$Sigma))
> cor.mat = varAIC.out$Sigma/outer(sd.vals,sd.vals)
> cor.mat
            MARKET.REAL  RF.REAL      INF     IPG
MARKET.REAL     1.00000 -0.16855 -0.04518 0.03916
    RF.REAL    -0.16855  1.00000  0.13046 0.03318
        INF    -0.04518  0.13046  1.00000 0.04732
        IPG     0.03916  0.03318  0.04732 1.00000
```

11.6 Bayesian Vector Autoregression

VAR models with many variables and long lags contain many parameters. Unrestricted estimation of these models reqires lots of data and often the estimated parameters are not very precise, the results are hard to interpret, and forecasts may appear more precise than they really are because standard error bands do not account for parameter uncertainty. The estimates and forecasts can be improved if one has prior information about the structure of the model or the possible values of the parameters or functions of the parameters. In a classical framework, it is difficult to incorporate non-sample information into the estimation. Nonsample information is easy to incorporate in a Bayesian framework. A Bayesian framework also naturally incorporates parameter uncertainty into common measures of precision. This section briefly describes the Bayesian VAR modeling tools in `S+FinMetrics` and illustrates these tools with an example. Details of underlying Bayesian methods are given in Sims and Zha (1998) and Zha (1998).

11.6.1 An Example of a Bayesian VAR Model

`S+FinMetrics` comes with a “`timeSeries`” object `policy.dat`, which contains six U.S. macroeconomic variables:

```
> colIds(policy.dat)
[1] "CP"  "M2"  "FFR" "GDP" "CPI" "U"
```

which represent IMF’s index of world commodity prices, M2 money stock, federal funds rate, real GDP, consumer price index for urban consumers, and civilian unemployment rate. The data set contains monthly observations from January 1959 to March 1998. Tao Zha and his co-authors have analyzed this data set in a number of papers, for example see Zha (1998). To use the same time period as in Zha (1998), create a subset of the data:

```
> zpolicy.dat = policy.dat[1:264,]
> zpolicy.mat = as.matrix(seriesData(zpolicy.dat))
```

which contains monthly observations from January 1959 to December 1980.

Estimating a Bayesian VAR Model

To estimate a Bayesian vector autoregression model, use the `S+FinMetrics` function `BVAR`. For macroeconomic modeling, it is usually found that many trending macroeconomic variables have a unit root, and in some cases, they may also have a cointegrating relationship (as described in the next chapter). To incorporate these types of prior beliefs into the model, use the `unit.root.dummy` and `coint.dummy` optional arguments to the `BVAR` function, which add some dummy observations to the beginning of the data to reflect these beliefs:

```
> zpolicy.bar13 = BVAR(zpolicy.mat~ar(13), unit.root=T,
+ coint=T)
> class(zpolicy.bar13)
[1] "BVAR"
```

The returned object is of class "BVAR", which inherits from "VAR", so many method functions for "VAR" objects work similarly for "BVAR" objects, such as the extractor functions, impulse response functions, and forecast error variance decomposition functions.

The Bayesian VAR models are controlled through a set of hyper parameters, which can be specified using the optional argument control, which is usually a list returned by the function BVAR.control. For example, the tightness of the belief in the unit root prior and cointegration prior is specified by mu5 and mu6, respectively. To see what default values are used for these hyper parameters, use

```
> args(BVAR.control)
function(L0 = 0.9, L1 = 0.1, L2 = 1, L3 = 1, L4 = 0.05,
        mu5 = 5, mu6 = 5)
```

For the meanings of these hyper parameters, see the online help file for BVAR.control.

Adding Exogenous Variables to the Model

Other exogenous variables can be added to the estimation formula, just as for OLS and VAR functions. The BVAR function and related functions will automatically take that into consideration and return the coefficient estimates for those variables.

Unconditional Forecasts

To forecast from a fitted Bayesian VAR model, use the generic predict function, which automatically calls the method function predict.BVAR for an object inheriting from class "BVAR". For example, to compute 12-step ahead forecasts use

```
> zpolicy.bpred = predict(zpolicy.bar13,n.predict=12)
> class(zpolicy.bpred)
[1] "forecast"
> names(zpolicy.bpred)
[1] "values"  "std.err" "draws"
> zpolicy.bpred

Predicted Values:
```

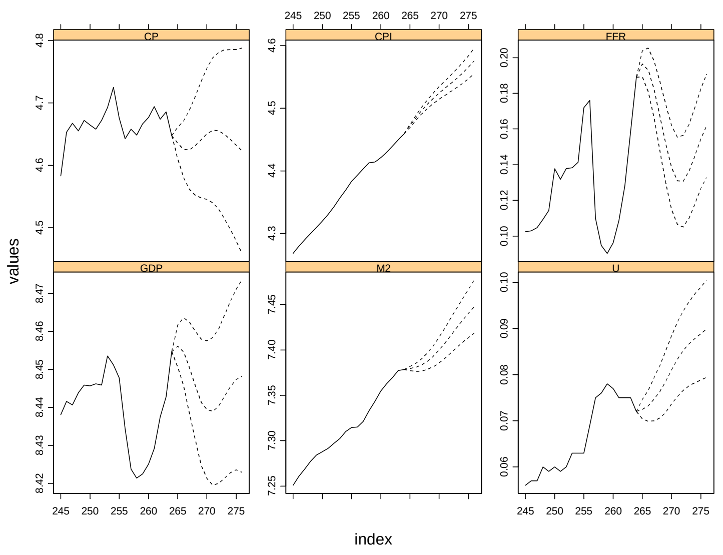


FIGURE 11.14. Forecasts from Bayesian VAR model.

```
                  CP     M2    FFR    GDP    CPI      U
 1-step-ahead 4.6354 7.3794 0.1964 8.4561 4.4714 0.0725
 2-step-ahead 4.6257 7.3808 0.1930 8.4546 4.4842 0.0732
 3-step-ahead 4.6247 7.3834 0.1823 8.4505 4.4960 0.0746
 4-step-ahead 4.6310 7.3876 0.1670 8.4458 4.5065 0.0763
 5-step-ahead 4.6409 7.3931 0.1515 8.4414 4.5160 0.0785
 6-step-ahead 4.6503 7.3998 0.1384 8.4394 4.5244 0.0810
 7-step-ahead 4.6561 7.4075 0.1309 8.4390 4.5321 0.0833
 8-step-ahead 4.6552 7.4159 0.1307 8.4403 4.5397 0.0852
 9-step-ahead 4.6496 7.4242 0.1362 8.4428 4.5475 0.0867
10-step-ahead 4.6415 7.4323 0.1451 8.4453 4.5561 0.0879
11-step-ahead 4.6321 7.4402 0.1546 8.4473 4.5655 0.0889
12-step-ahead 4.6232 7.4476 0.1618 8.4482 4.5753 0.0899
```

The forecasts can also be plotted along with the original data using

```
> plot(zpolicy.bpred, zpolicy.mat, n.old=20)
```

The resulting plot is shown in Figure 11.14. The Bayesian forecasts usually have wider error bands than classical forecasts, because they take into account the uncertainty in the coefficient estimates. To ignore the uncertainty in coefficient estimates, one can call the classical VAR `predict` method function, `predict.VAR`, directly instead of the generic `predict` function.

The forecasts from Bayesian VAR models are of class "`forecast`", and are computed using Monte Carlo integration. By default, 1000 simulation draws are used. To change the number of simulation draws and random seed, specify the `n.sim` and `seed` optional arguments, respectively. For forecasts from Bayesian VAR models, there is one more component in the returned object: `draws`, which contains all the simulated forecasts. This can be used to assess other statistical properties of the forecasts.

11.6.2 Conditional Forecasts

As mentioned earlier, conditional forecasts from classical VAR models ignore the uncertainty in estimated coefficients. In contrast, conditional forecasts from Bayesian VAR models take into account the uncertainty associated with estimated coefficients. To perform conditional forecasts from a fitted Bayesian VAR model, use the generic `cpredict` function. For example, if it is known that `FFR` in January 1981 is between 0.185 and 0.195, one can incorporate this (soft condition) information into the forecasts using:

```
> zpolicy.spred = cpredict(zpolicy.bar13, 12, steps=1,
+ variables="FFR", upper=0.195, lower=0.185)
> zpolicy.spred

Predicted Values:

                CP     M2    FFR    GDP    CPI      U
 1-step-ahead 4.6374 7.3797 0.1910 8.4554 4.4714 0.0729
 2-step-ahead 4.6286 7.3816 0.1855 8.4540 4.4840 0.0736
 3-step-ahead 4.6279 7.3850 0.1743 8.4498 4.4954 0.0752
 4-step-ahead 4.6349 7.3899 0.1587 8.4452 4.5057 0.0768
 5-step-ahead 4.6447 7.3960 0.1443 8.4414 4.5149 0.0791
 6-step-ahead 4.6525 7.4033 0.1324 8.4406 4.5231 0.0814
 7-step-ahead 4.6549 7.4114 0.1270 8.4412 4.5307 0.0835
 8-step-ahead 4.6523 7.4201 0.1283 8.4428 4.5383 0.0851
 9-step-ahead 4.6453 7.4284 0.1349 8.4457 4.5461 0.0864
10-step-ahead 4.6389 7.4365 0.1432 8.4482 4.5547 0.0876
11-step-ahead 4.6317 7.4444 0.1516 8.4501 4.5641 0.0885
12-step-ahead 4.6264 7.4519 0.1572 8.4511 4.5741 0.0896
```

For conditional forecasts with soft conditions, a Monte Carlo integration with acceptance/rejection method is used. By default, 1000 simulation draws are used. However, it may occur that only a small number of draws satisfy the soft conditions if the intervals are very tight. To see how many draws satisfied the soft conditions and thus are used for inference, simply check the dimension of the `draws` component of the returned object (see the on-line help file for `forecast.object` for details):

```
> dim(zpolicy.spred$draws)
[1] 372  72
```

In this case, only 372 out of 1000 simulation draws satisfied the conditions. To continue simulating from the posterior moment distribution, use the same command as before, with seed set to the current value of `.Random.seed`:

```
> zpolicy.spred2 = cpredict(zpolicy.bar13, 12, steps=1,
+ variables="FFR", upper=0.195, lower=0.185, seed=.Random.seed)
> dim(zpolicy.spred2$draws)
[1] 389  72
```

Note that the draws in `zpolicy.spred2` can be combined with the draws in `zpolicy.spred` to obtain an updated and more accurate estimate of conditional forecasts.

To ignore the coefficient uncertainty for the conditional forecasts, call the classical method function `cpredict.VAR` directly on a fitted Bayesian VAR object. The technique introduced above can also be used for classical prediction with soft conditions.

11.7 References

Campbell, J. A. Lo and C. MacKinlay (1997). *The Econometrics of Financial Markets.* Princeton University Press, Princeton, NJ.

Culbertson, K. (1996). *Quantitative Financial Economics: Stocks, Bonds and Foreign Exchange.* John Wiley & Sons, Chichester.

Doan, T. A., Litterman, R. B., and Sims, C. A. (1984). "Forecasting and Conditional Projection Using Realistic Prior Distributions," *Econometric Reviews*, 3, 1-100.

Granger, C.W.J. (1969). "Investigating Causal Relations by Econometric Models and Cross Spectral Methods," *Econometrica*, 37, 424-438.

Hamilton, J.D. (1994). *Time Series Analysis.* Princeton University Press, Princeton, NJ.

Lee, B.-S. (1992). "Causal Relations Among Stock Returns, Interest Rates, Real Activity, and Inflation," *Journal of Finance*, 47, 1591-1603.

Lutkepohl, H. (1991). *Introduction to Multiple Time Series Analysis.* Springer-Verlag, Berlin.

LUTKEPOHL, H. (1999). "Vector Autoregressions," unpublished manuscript, Institut für Statistik und Ökonometrie," Humboldt-Universität zu Berlin.

MILLS, T.C. (1999). *The Econometric Modeling of Financial Time Series, Second Edition.* Cambridge University Press, Cambridge.

RUNKLE, D. E. (1987). "Vector Autoregressions and Reality," *Journal of Business and Economic Statistics*, 5(4), 437-442.

SIMS, C.A. (1980). "Macroeconomics and Reality," *Econometrica*, 48, 1-48.

SIMS, C. A., AND ZHA, T. (1998). "Bayesian Methods for Dynamic Multivariate Models," *International Economic Review*, 39(4), 949-968.

STOCK, J.H. AND M.W. WATSON (2001). "Vector Autoregressions," *Journal of Economic Perspectives*, 15, 101-115.

TSAY, R. (2001). *Analysis of Financial Time Series.* John Wiley & Sons, New York.

WATSON, M. (1994). "Vector Autoregressions and Cointegration," in R.F. Engle and D. McFadden (eds.), *Handbook of Econometrics, Volume IV.* Elsevier Science Ltd., Amsterdam.

WAGGONER, D. F., AND ZHA, T. (1999). "Conditional Forecasts in Dynamic Multivariate Models," *Review of Economics and Statistics*, 81(4), 639-651.

ZHA, T. (1998). "Dynamic Multivariate Model for Use in Formulating Policy", *Economic Review*, Federal Reserve Bank of Atlanta, First Quarter, 1998.

12
Cointegration

12.1 Introduction

The regression theory of Chapter 6 and the VAR models discussed in the previous chapter are appropriate for modeling $I(0)$ data, like asset returns or growth rates of macroeconomic time series. Economic theory often implies equilibrium relationships between the levels of time series variables that are best described as being $I(1)$. Similarly, arbitrage arguments imply that the $I(1)$ prices of certain financial time series are linked. This chapter introduces the statistical concept of cointegration that is required to make sense of regression models and VAR models with $I(1)$ data.

The chapter is organized as follows. Section 12.2 gives an overview of the concepts of spurious regression and cointegration, and introduces the error correction model as a practical tool for utilizing cointegration with financial time series. Section 12.3 discusses residual-based tests for cointegration. Section 12.4 covers regression-based estimation of cointegrating vectors and error correction models. In Section 12.5, the connection between VAR models and cointegration is made, and Johansen's maximum likelihood methodology for cointegration modeling is outlined. Some technical details of the Johansen methodology are provided in the appendix to this chapter.

Excellent textbook treatments of the statistical theory of cointegration are given in Hamilton (1994), Johansen (1995) and Hayashi (2000). Applications of cointegration to finance may be found in Campbell, Lo, and

MacKinlay (1997), Mills (1999), Alexander (2001), Cochrane (2001), and Tsay (2001).

12.2 Spurious Regression and Cointegration

12.2.1 Spurious Regression

The time series regression model discussed in Chapter 6 required all variables to be $I(0)$. In this case, the usual statistical results for the linear regression model hold. If some or all of the variables in the regression are $I(1)$ then the usual statistical results may or may not hold[1]. One important case in which the usual statistical results do not hold is *spurious regression* when all the regressors are $I(1)$ and not cointegrated. The following example illustrates.

Example 71 *An illustration of spurious regression using simulated data*

Consider two independent and not cointegrated $I(1)$ processes y_{1t} and y_{2t} such that

$$y_{it} = y_{it-1} + \varepsilon_{it}, \text{ where } \varepsilon_{it} \sim GWN(0,1),\ i = 1,2$$

Following Granger and Newbold (1974), 250 observations for each series are simulated and plotted in Figure 12.1 using

```
> set.seed(458)
> e1 = rnorm(250)
> e2 = rnorm(250)
> y1 = cumsum(e1)
> y2 = cumsum(e2)
> tsplot(y1, y2, lty=c(1,3))
> legend(0, 15, c("y1","y2"), lty=c(1,3))
```

The data in the graph resemble stock prices or exchange rates. A visual inspection of the data suggests that the levels of the two series are positively related. Regressing y_{1t} on y_{2t} reinforces this observation:

```
> summary(OLS(y1~y2))

Call:
OLS(formula = y1 ~y2)
```

[1]A systematic technical analysis of the linear regression model with $I(1)$ and $I(0)$ variables is given in Sims, Stock and Watson (1990). Hamilton (1994) gives a nice summary of these results and Stock and Watson (1989) provides useful intuition and examples.

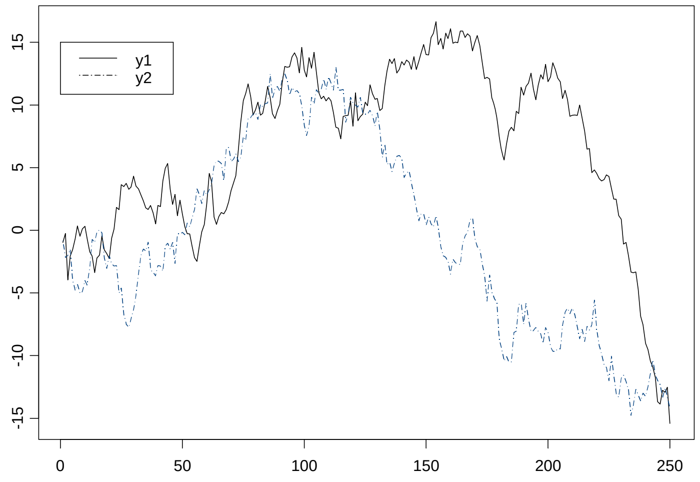


FIGURE 12.1. Two simulated independent $I(1)$ processes.

```
Residuals:
     Min      1Q  Median      3Q     Max
 -16.360  -4.352  -0.128   4.979  10.763

Coefficients:
              Value Std. Error t value Pr(>|t|)
(Intercept)  6.7445  0.3943    17.1033  0.0000
         y2  0.4083  0.0508     8.0352  0.0000

Regression Diagnostics:

         R-Squared 0.2066
Adjusted R-Squared 0.2034
Durbin-Watson Stat 0.0328

Residual standard error: 6.217 on 248 degrees of freedom
F-statistic: 64.56 on 1 and 248 degrees of freedom, the
p-value is 3.797e-014
```

The estimated slope coefficient is 0.408 with a large t-statistic of 8.035 and the regression R^2 is moderate at 0.201. The only suspicious statistic is the very low Durbin-Watson statistic suggesting strong residual autocorrelation. These statistics are representative of the spurious regression phenomenon with $I(1)$ that are not cointegrated. If Δy_{1t} is regressed on Δy_{2t} the correct relationship between the two series is revealed

```
> summary(OLS(diff(y1)~diff(y2)))

Call:
OLS(formula = diff(y1) ~diff(y2))

Residuals:
     Min      1Q  Median      3Q     Max
 -3.6632 -0.7706 -0.0074  0.6983  2.7184

Coefficients:
              Value Std. Error t value Pr(>|t|)
(Intercept) -0.0565  0.0669    -0.8447  0.3991
   diff(y2)  0.0275  0.0642     0.4290  0.6683

Regression Diagnostics:

         R-Squared  0.0007
Adjusted R-Squared -0.0033
Durbin-Watson Stat  1.9356

Residual standard error: 1.055 on 247 degrees of freedom
F-statistic: 0.184 on 1 and 247 degrees of freedom, the
p-value is 0.6683
```

Similar results to those above occur if $\text{cov}(\varepsilon_{1t}, \varepsilon_{2t}) \neq 0$. The levels regression remains spurious (no real long-run common movement in levels), but the differences regression will reflect the non-zero contemporaneous correlation between Δy_{1t} and Δy_{2t}.

Statistical Implications of Spurious Regression

Let $\mathbf{Y}_t = (y_{1t}, \ldots, y_{nt})'$ denote an $(n \times 1)$ vector of $I(1)$ time series that are not cointegrated. Using the partition $\mathbf{Y}_t = (y_{1t}, \mathbf{Y}_{2t}')'$, consider the least squares regression of y_{1t} on $\mathbf{Y}_{2t}$ giving the fitted model

$$y_{1t} = \hat{\boldsymbol{\beta}}_2' \mathbf{Y}_{2t} + \hat{u}_t \tag{12.1}$$

Since y_{1t} is not cointegrated with $\mathbf{Y}_{2t}$ (12.1) is a *spurious regression* and the true value of $\boldsymbol{\beta}_2$ is zero. The following results about the behavior of $\hat{\boldsymbol{\beta}}_2$ in the spurious regression (12.1) are due to Phillips (1986):

- $\hat{\boldsymbol{\beta}}_2$ does not converge in probability to zero but instead converges in distribution to a non-normal random variable not necessarily centered at zero. This is the spurious regression phenomenon.

- The usual OLS t-statistics for testing that the elements of $\boldsymbol{\beta}_2$ are zero diverge to $\pm\infty$ as $T \to \infty$. Hence, with a large enough sample it will appear that $\mathbf{Y}_t$ is cointegrated when it is not if the usual asymptotic normal inference is used.
- The usual R^2 from the regression converges to unity as $T \to \infty$ so that the model will appear to fit well even though it is misspecified.
- Regression with $I(1)$ data only makes sense when the data are cointegrated.

12.2.2 Cointegration

Let $\mathbf{Y}_t = (y_{1t}, \ldots, y_{nt})'$ denote an $(n \times 1)$ vector of $I(1)$ time series. $\mathbf{Y}_t$ is *cointegrated* if there exists an $(n \times 1)$ vector $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_n)'$ such that

$$\boldsymbol{\beta}'\mathbf{Y}_t = \beta_1 y_{1t} + \cdots + \beta_n y_{nt} \sim I(0) \tag{12.2}$$

In words, the nonstationary time series in $\mathbf{Y}_t$ are cointegrated if there is a linear combination of them that is stationary or $I(0)$. If some elements of $\boldsymbol{\beta}$ are equal to zero then only the subset of the time series in $\mathbf{Y}_t$ with non-zero coefficients is cointegrated. The linear combination $\boldsymbol{\beta}'\mathbf{Y}_t$ is often motivated by economic theory and referred to as a *long-run equilibrium* relationship. The intuition is that $I(1)$ time series with a long-run equilibrium relationship cannot drift too far apart from the equilibrium because economic forces will act to restore the equilibrium relationship.

Normalization

The cointegration vector $\boldsymbol{\beta}$ in (12.2) is not unique since for any scalar c the linear combination $c\boldsymbol{\beta}'\mathbf{Y}_t = \boldsymbol{\beta}^{*\prime}\mathbf{Y}_t \sim I(0)$. Hence, some *normalization* assumption is required to uniquely identify $\boldsymbol{\beta}$. A typical normalization is

$$\boldsymbol{\beta} = (1, -\beta_2, \ldots, -\beta_n)'$$

so that the cointegration relationship may be expressed as

$$\boldsymbol{\beta}'\mathbf{Y}_t = y_{1t} - \beta_2 y_{2t} - \cdots - \beta_n y_{nt} \sim I(0)$$

or

$$y_{1t} = \beta_2 y_{2t} + \cdots + \beta_n y_{nt} + u_t \tag{12.3}$$

where $u_t \sim I(0)$. In (12.3), the error term u_t is often referred to as the *disequilibrium error* or the *cointegrating residual*. In long-run equilibrium, the disequilibrium error u_t is zero and the long-run equilibrium relationship is

$$y_{1t} = \beta_2 y_{2t} + \cdots + \beta_n y_{nt}$$

Multiple Cointegrating Relationships

If the $(n\times 1)$ vector $\mathbf{Y}_t$ is cointegrated there may be $0 < r < n$ linearly independent cointegrating vectors. For example, let $n = 3$ and suppose there are $r = 2$ cointegrating vectors $\boldsymbol{\beta}_1 = (\beta_{11}, \beta_{12}, \beta_{13})'$ and $\boldsymbol{\beta}_2 = (\beta_{21}, \beta_{22}, \beta_{23})'$. Then $\boldsymbol{\beta}_1'\mathbf{Y}_t = \beta_{11}y_{1t} + \beta_{12}y_{2t} + \beta_{13}y_{3t} \sim I(0)$, $\boldsymbol{\beta}_2'\mathbf{Y}_t = \beta_{21}y_{1t} + \beta_{22}y_{2t} + \beta_{23}y_{3t} \sim I(0)$ and the (3×2) matrix

$$\mathbf{B}' = \begin{pmatrix} \boldsymbol{\beta}_1' \\ \boldsymbol{\beta}_2' \end{pmatrix} = \begin{pmatrix} \beta_{11} & \beta_{12} & \beta_{13} \\ \beta_{21} & \beta_{22} & \beta_{33} \end{pmatrix}$$

forms a *basis* for the space of cointegrating vectors. The linearly independent vectors $\boldsymbol{\beta}_1$ and $\boldsymbol{\beta}_2$ in the cointegrating basis $\mathbf{B}$ are not unique unless some normalization assumptions are made. Furthermore, any linear combination of $\boldsymbol{\beta}_1$ and $\boldsymbol{\beta}_2$, e.g. $\boldsymbol{\beta}_3 = c_1\boldsymbol{\beta}_1 + c_2\boldsymbol{\beta}_2$ where c_1 and c_2 are constants, is also a cointegrating vector.

Examples of Cointegration and Common Trends in Economics and Finance

Cointegration naturally arises in economics and finance. In economics, cointegration is most often associated with economic theories that imply equilibrium relationships between time series variables. The permanent income model implies cointegration between consumption and income, with consumption being the common trend. Money demand models imply cointegration between money, income, prices and interest rates. Growth theory models imply cointegration between income, consumption and investment, with productivity being the common trend. Purchasing power parity implies cointegration between the nominal exchange rate and foreign and domestic prices. Covered interest rate parity implies cointegration between forward and spot exchange rates. The Fisher equation implies cointegration between nominal interest rates and inflation. The expectations hypothesis of the term structure implies cointegration between nominal interest rates at different maturities. The equilibrium relationships implied by these economic theories are referred to as *long-run equilibrium* relationships, because the economic forces that act in response to deviations from equilibriium may take a long time to restore equilibrium. As a result, cointegration is modeled using long spans of low frequency time series data measured monthly, quarterly or annually.

In finance, cointegration may be a high frequency relationship or a low frequency relationship. Cointegration at a high frequency is motivated by arbitrage arguments. The *Law of One Price* implies that identical assets must sell for the same price to avoid arbitrage opportunities. This implies cointegration between the prices of the same asset trading on different markets, for example. Similar arbitrage arguments imply cointegration between spot and futures prices, and spot and forward prices, and bid and ask prices. Here the terminology long-run equilibrium relationship is somewhat misleading because the economic forces acting to eliminate arbitrage opportunities work very quickly. Cointegration is appropriately modeled using short spans of high frequency data in seconds, minutes, hours or days. Cointegration at a low frequency is motivated by economic equilibrium theories linking assets prices or expected returns to fundamentals. For example, the present value model of stock prices states that a stock's price is an expected discounted present value of its expected future dividends or earnings. This links the behavior of stock prices at low frequencies to the behavior of dividends or earnings. In this case, cointegration is modeled using low frequency data and is used to explain the long-run behavior of stock prices or expected returns.

12.2.3 Cointegration and Common Trends

If the $(n \times 1)$ vector time series $\mathbf{Y}_t$ is cointegrated with $0 < r < n$ cointegrating vectors then there are *$n - r$ common $I(1)$ stochastic trends.* To illustrate the duality between cointegration and common trends, let $\mathbf{Y}_t = (y_{1t}, y_{2t})' \sim I(1)$ and $\boldsymbol{\varepsilon}_t = (\varepsilon_{1t}, \varepsilon_{2t}, \varepsilon_{3t})' \sim I(0)$ and suppose that $\mathbf{Y}_t$ is cointegrated with cointegrating vector $\boldsymbol{\beta} = (1, -\beta_2)'$. This cointegration relationship may be represented as

$$\begin{aligned} y_{1t} &= \beta_2 \sum_{s=1}^{t} \varepsilon_{1s} + \varepsilon_{3t} \\ y_{2t} &= \sum_{s=1}^{t} \varepsilon_{1s} + \varepsilon_{2t} \end{aligned}$$

The common stochastic trend is $\sum_{s=1}^{t} \varepsilon_{1s}$. Notice that the cointegrating relationship annihilates the common stochastic trend:

$$\boldsymbol{\beta}'\mathbf{Y}_t = \beta_2 \sum_{s=1}^{t} \varepsilon_{1s} + \varepsilon_{3t} - \beta_2 \left(\sum_{s=1}^{t} \varepsilon_{1s} + \varepsilon_{2t} \right) = \varepsilon_{3t} - \beta_2 \varepsilon_{2t} \sim I(0).$$

12.2.4 Simulating Cointegrated Systems

Cointegrated systems may be conveniently simulated using Phillips' (1991) *triangular representation.* For example, consider a bivariate cointegrated system for $\mathbf{Y}_t = (y_{1t}, y_{2t})'$ with cointegrating vector $\boldsymbol{\beta} = (1, -\beta_2)'$. A triangular representation has the form

$$\begin{aligned} y_{1t} &= \beta_2 y_{2t} + u_t, \text{ where } u_t \sim I(0) && (12.4) \\ y_{2t} &= y_{2t-1} + v_t, \text{ where } v_t \sim I(0) && (12.5) \end{aligned}$$

The first equation describes the long-run equilibrium relationship with an $I(0)$ disequilibrium error u_t. The second equation specifies y_{2t} as the common stochastic trend with innovation v_t:

$$y_{2t} = y_{20} + \sum_{j=1}^{t} v_j.$$

In general, the innovations u_t and v_t may be contemporaneously and serially correlated. The time series structure of these innovations characterizes the short-run dynamics of the cointegrated system. The system (12.4)-(12.5) with $\beta_2 = 1$, for example, might be used to model the behavior of the logarithm of spot and forward prices, spot and futures prices or stock prices and dividends.

Example 72 *Simulated bivariate cointegrated system*

Consider simulating $T = 250$ observations from the system (12.4)-(12.5) using $\boldsymbol{\beta} = (1, -1)'$, $u_t = 0.75u_{t-1} + \varepsilon_t$, $\varepsilon_t \sim \text{iid}\, N(0, 0.5^2)$ and $v_t \sim \text{iid}\, N(0, 0.5^2)$. The S-PLUS code is

```
> set.seed(432)
> e = rmvnorm(250, mean=rep(0,2), sd=c(0.5,0.5))
> u.ar1 = arima.sim(model=list(ar=0.75), innov=e[,1])
> y2 = cumsum(e[,2])
> y1 = y2 + u.ar1
> par(mfrow=c(2,1))
> tsplot(y1, y2, lty=c(1,3),
+ main="Simulated bivariate cointegrated system",
+ sub="1 cointegrating vector, 1 common trend")
> legend(0, 7, legend=c("y1","y2"), lty=c(1,3))
> tsplot(u.ar1, main="Cointegrating residual")
```

Figure 12.2 shows the simulated data for y_{1t} and y_{2t} along with the cointegrating residual $u_t = y_{1t} - y_{2t}$. Since y_{1t} and y_{2t} share a common stochastic trend they follow each other closely. The impulse response function for u_t may be used to determine the speed of adjustment to long-run equilibrium. Since u_t is an AR(1) with $\phi = 0.75$ the half life of a shock is $\ln(0.5)/\ln(0.75) = 2.4$ time periods.

Next, consider a trivariate cointegrated system for $\mathbf{Y}_t = (y_{1t}, y_{2t}, y_{3t})'$. With a trivariate system there may be one or two cointegrating vectors. With one cointegrating vector there are two common stochastic trends and with two cointegrating vectors there is one common trend. A triangular representation with one cointegrating vector $\boldsymbol{\beta} = (1, -\beta_2, -\beta_3)'$ and two

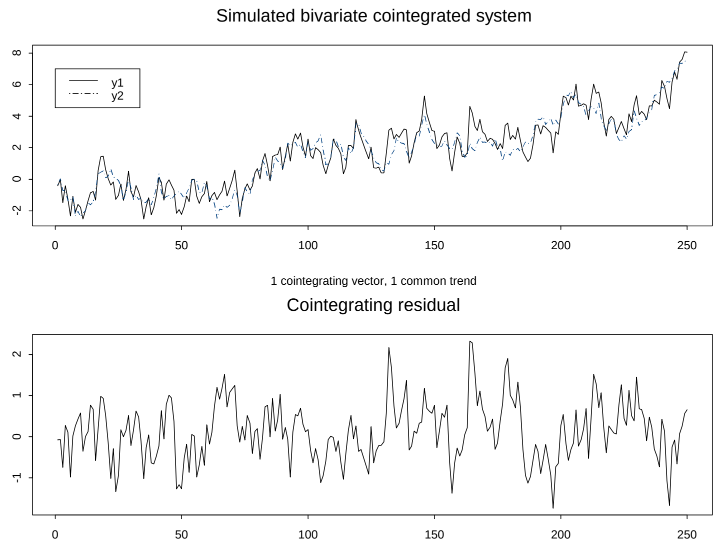


FIGURE 12.2. Simulated bivariate cointegrated system with $\boldsymbol{\beta} = (1, -1)'$.

stochastic trends is

$$\begin{aligned} y_{1t} &= \beta_2 y_{2t} + \beta_3 y_{3t} + u_t, \text{ where } u_t \sim I(0) \quad (12.6)\\ y_{2t} &= y_{2t-1} + v_t, \text{ where } v_t \sim I(0) \quad (12.7)\\ y_{3t} &= y_{3t-1} + w_t, \text{ where } w_t \sim I(0) \quad (12.8)\end{aligned}$$

The first equation describes the long-run equilibrium and the second and third equations specify the common stochastic trends. An example of a trivariate cointegrated system with one cointegrating vector is a system of nominal exchange rates, home country price indices and foreign country price indices. A cointegrating vector $\boldsymbol{\beta} = (1, -1, -1)'$ implies that the real exchange rate is stationary.

Example 73 *Simulated trivariate cointegrated system with 1 cointegrating vector*

The S-PLUS code for simulating $T = 250$ observation from (12.6)-(12.8) with $\boldsymbol{\beta} = (1, -0.5, -0.5)'$, $u_t = 0.75u_{t-1} + \varepsilon_t$, $\varepsilon_t \sim \text{iid}\, N(0, 0.5^2)$, $v_t \sim \text{iid}\, N(0, 0.5^2)$ and $w_t \sim \text{iid}\, N(0, 0.5^2)$ is

```
> set.seed(573)
> e = rmvnorm(250, mean=rep(0,3), sd=c(0.5,0.5,0.5))
> u1.ar1 = arima.sim(model=list(ar=0.75), innov=e[,1])
> y2 = cumsum(e[,2])
```

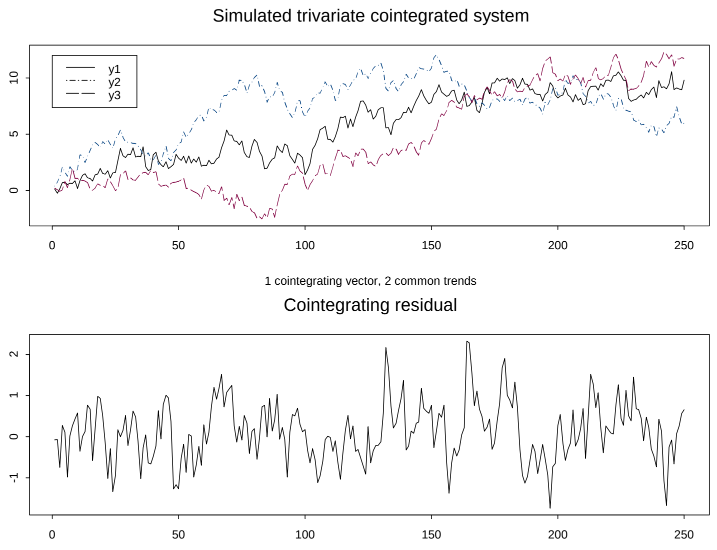


FIGURE 12.3. Simulated trivariate cointegrated system with one cointegrating vector $\boldsymbol{\beta} = (1, -0.5, -0.5)'$ and two stochastic trends.

```
> y3 = cumsum(e[,3])
> y1 = 0.5*y2 + 0.5*y3 + u1.ar1
> par(mfrow=c(2,1))
> tsplot(y1, y2, y3, lty=c(1,3,4),
+ main="Simulated trivariate cointegrated system",
+ sub="1 cointegrating vector, 2 common trends")
> legend(0, 12, legend=c("y1","y2","y3"), lty=c(1,3,4))
> tsplot(u.ar1, main="Cointegrating residual")
```

Figure 12.3 illustrates the simulated data. Here, y_{2t} and y_{3t} are the two independent common trends and $y_{1t} = 0.5y_{2t} + 0.5y_{3t} + u_t$ is the average of the two trends plus an AR(1) residual.

Finally, consider a trivariate cointegrated system with two cointegrating vectors and one common stochastic trend. A triangular representation for this system with cointegrating vectors $\boldsymbol{\beta}_1 = (1, 0, -\beta_{13})'$ and $\boldsymbol{\beta}_2 = (0, 1, -\beta_{23})'$ is

$$\begin{aligned} y_{1t} &= \beta_{13}y_{3t} + u_t, \text{ where } u_t \sim I(0) && (12.9) \\ y_{2t} &= \beta_{23}y_{3t} + v_t, \text{ where } v_t \sim I(0) && (12.10) \\ y_{3t} &= y_{3t-1} + w_t, \text{ where } w_t \sim I(0) && (12.11) \end{aligned}$$

Here the first two equations describe two long-run equilibrium relations and the third equation gives the common stochastic trend. An example in finance of such a system is the term structure of interest rates where y_3 represents the short rate and y_1 and y_2 represent two different long rates. The cointegrating relationships would indicate that the spreads between the long and short rates are stationary.

Example 74 *Simulated trivariate cointegrated system with 2 cointegrating vectors*

The S-PLUS code for simulating $T = 250$ observation from (12.9)-(12.11) with $\boldsymbol{\beta}_1 = (1, 0, -1)'$, $\beta_2 = (0, 1, -1)'$, $u_t = 0.75u_{t-1}+\varepsilon_t$, $\varepsilon_t \sim \text{iid}\, N(0, 0.5^2)$, $v_t = 0.75v_{t-1} + \eta_t$, $\eta_t \sim \text{iid}\, N(0, 0.5^2)$ and $w_t \sim \text{iid}\, N(0, 0.5^2)$ is

```
> set.seed(573)
> e = rmvnorm(250,mean=rep(0,3), sd=c(0.5,0.5,0.5))
> u.ar1 = arima.sim(model=list(ar=0.75), innov=e[,1])
> v.ar1 = arima.sim(model=list(ar=0.75), innov=e[,2])
> y3 = cumsum(e[,3])
> y1 = y3 + u.ar1
> y2 = y3 + v.ar1
> par(mfrow=c(2,1))
> tsplot(y1, y2, y3, lty=c(1,3,4),
+ main="Simulated trivariate cointegrated system",
+ sub="2 cointegrating vectors, 1 common trend")
> legend(0, 10, legend=c("y1","y2","y3"), lty=c(1,3,4))
> tsplot(u.ar1, v.ar1, lty=c(1,3),
+ main="Cointegrated residuals")
> legend(0, -1, legend=c("u","v"), lty=c(1,3))
```

12.2.5 Cointegration and Error Correction Models

Consider a bivariate $I(1)$ vector $\mathbf{Y}_t = (y_{1t}, y_{2t})'$ and assume that $\mathbf{Y}_t$ is cointegrated with cointegrating vector $\boldsymbol{\beta} = (1, -\beta_2)'$ so that $\boldsymbol{\beta}'\mathbf{Y}_t = y_{1t} - \beta_2 y_{2t}$ is $I(0)$. In an extremely influential and important paper, Engle and Granger (1987) showed that cointegration implies the existence of an *error correction model* (ECM) of the form

$$\begin{aligned}
\Delta y_{1t} &= c_1 + \alpha_1(y_{1t-1} - \beta_2 y_{2t-1}) && (12.12)\\
&\quad + \sum_j \psi_{11}^j \Delta y_{1t-j} + \sum_j \psi_{12}^j \Delta y_{2t-j} + \varepsilon_{1t}\\
\Delta y_{2t} &= c_2 + \alpha_2(y_{1t-1} - \beta_2 y_{2t-1}) && (12.13)\\
&\quad + \sum_j \psi_{21}^j \Delta y_{1t-j} + \sum_j \psi_{22}^2 \Delta y_{2t-j} + \varepsilon_{2t}
\end{aligned}$$

that describes the dynamic behavior of y_{1t} and y_{2t}. The ECM links the long-run equilibrium relationship implied by cointegration with the short-run dynamic adjustment mechanism that describes how the variables react

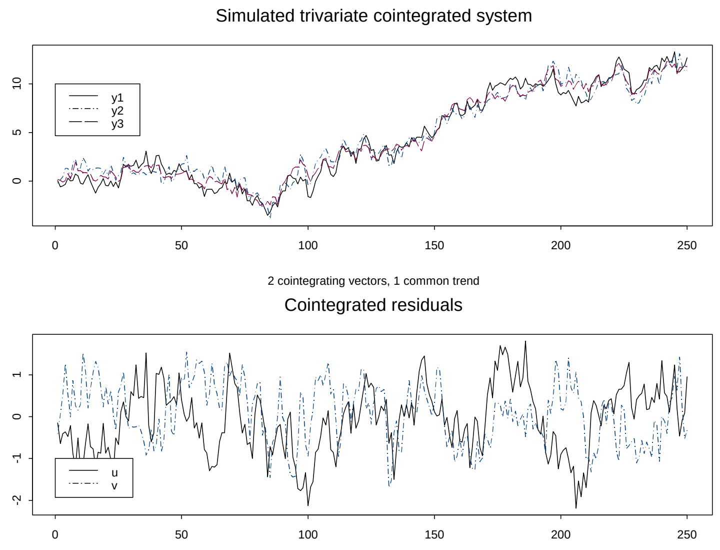


FIGURE 12.4. Simulated trivatiate cointegrated system with two cointegrating vectors $\boldsymbol{\beta}_1 = (1, 0, -1)'$, $\boldsymbol{\beta}_2 = (0, 1, -1)'$ and one common trend.

when they move out of long-run equilibrium. This ECM makes the concept of cointegration useful for modeling financial time series.

Example 75 *Bivariate ECM for stock prices and dividends*

As an example of an ECM, let s_t denote the log of stock prices and d_t denote the log of dividends and assume that $\mathbf{Y}_t = (s_t, d_t)'$ is $I(1)$. If the log dividend-price ratio is $I(0)$ then the logs of stock prices and dividends are cointegrated with $\boldsymbol{\beta} = (1, -1)'$. That is, the long-run equilibrium is

$$d_t = s_t + \mu + u_t$$

where μ is the mean of the log dividend-price ratio, and u_t is an $I(0)$ random variable representing the dynamic behavior of the log dividend-price ratio (disequilibrium error). Suppose the ECM has the form

$$\begin{aligned}\Delta s_t &= c_s + \alpha_s(d_{t-1} - s_{t-1} - \mu) + \varepsilon_{st} \\ \Delta d_t &= c_d + \alpha_d(d_{t-1} - s_{t-1} - \mu) + \varepsilon_{dt}\end{aligned}$$

where $c_s > 0$ and $c_d > 0$. The first equation relates the growth rate of dividends to the lagged disequilibrium error $d_{t-1} - s_{t-1} - \mu$, and the second equation relates the growth rate of stock prices to the lagged disequilibrium as well. The reactions of s_t and d_t to the disequilibrium error are captured by the *adjustment coefficients* α_s and α_d.

Consider the special case of (12.12)-(12.13) where $\alpha_d = 0$ and $\alpha_s = 0.5$. The ECM equations become

$$\begin{aligned}\Delta s_t &= c_s + 0.5(d_{t-1} - s_{t-1} - \mu) + \varepsilon_{st}, \\ \Delta d_t &= c_d + \varepsilon_{dt}.\end{aligned}$$

so that only s_t responds to the lagged disequilibrium error. Notice that $E[\Delta s_t|\mathbf{Y}_{t-1}] = c_s + 0.5(d_{t-1} - s_{t-1} - \mu)$ and $E[\Delta d_t|\mathbf{Y}_{t-1}] = c_d$. Consider three situations:

1. $d_{t-1} - s_{t-1} - \mu = 0$. Then $E[\Delta s_t|\mathbf{Y}_{t-1}] = c_s$ and $E[\Delta d_t|Y_{t-1}] = c_d$, so that c_s and c_d represent the growth rates of stock prices and dividends in long-run equilibrium.

2. $d_{t-1} - s_{t-1} - \mu > 0$. Then $E[\Delta s_t|\mathbf{Y}_{t-1}] = c_s + 0.5(d_{t-1} - s_{t-1} - \mu) > c_s$. Here the dividend yield has increased above its long-run mean (positive disequilibrium error) and the ECM predicts that s_t will grow faster than its long-run rate to restore the dividend yield to its long-run mean. Notice that the magnitude of the adjustment coefficient $\alpha_s = 0.5$ controls the speed at which s_t responds to the disequilibrium error.

3. $d_{t-1} - s_{t-1} - \mu < 0$. Then $E[\Delta s_t|\mathbf{Y}_{t-1}] = c_s + 0.5(d_{t-1} - s_{t-1} - \mu) < c_s$. Here the dividend yield has decreased below its long-run mean (negative disequilibrium error) and the ECM predicts that s_t will grow more slowly than its long-run rate to restore the dividend yield to its long-run mean.

In Case 1, there is no expected adjustment since the model was in long-run equilibrium in the previous period. In Case 2, the model was above long-run equilibrium last period so the expected adjustment in s_t is downward toward equilibrium. In Case 3, the model was below long-run equilibrium last period and so the expected adjustment is upward toward the equilibrium. This discussion illustrates why the model is called an error correction model. When the variables are out of long-run equilibrium, there are economic forces, captured by the adjustment coefficients, that push the model back to long-run equilibrium. The speed of adjustment toward equilibrium is determined by the magnitude of α_s. In the present example, $\alpha_s = 0.5$ which implies that roughly one half of the disequilibrium error is corrected in one time period. If $\alpha_s = 1$ then the entire disequilibrium is corrected in one period. If $\alpha_s = 1.5$ then the correction overshoots the long-run equilibrium.

12.3 Residual-Based Tests for Cointegration

Let the $(n \times 1)$ vector $\mathbf{Y}_t$ be $I(1)$. Recall, $\mathbf{Y}_t$ is cointegrated with $0 < r < n$ cointegrating vectors if there exists an $(r \times n)$ matrix $\mathbf{B}'$ such that

$$\mathbf{B}'\mathbf{Y}_t = \begin{pmatrix} \boldsymbol{\beta}_1'\mathbf{Y}_t \\ \vdots \\ \boldsymbol{\beta}_r'\mathbf{Y}_t \end{pmatrix} = \begin{pmatrix} u_{1t} \\ \vdots \\ u_{rt} \end{pmatrix} \sim I(0)$$

Testing for cointegration may be thought of as testing for the existence of long-run equilibria among the elements of $\mathbf{Y}_t$. Cointegration tests cover two situations:

- There is at most one cointegrating vector
- There are possibly $0 \leq r < n$ cointegrating vectors.

The first case was originally considered by Engle and Granger (1986) and they developed a simple two-step residual-based testing procedure based on regression techniques. The second case was originally considered by Johansen (1988) who developed a sophisticated sequential procedure for determining the existence of cointegration and for determining the number of cointegrating relationships based on maximum likelihood techniques. This section explains Engle and Granger's two-step procedure. Johansen's more general procedure will be discussed later on.

Engle and Granger's two-step procedure for determining if the $(n \times 1)$ vector $\boldsymbol{\beta}$ is a cointegrating vector is as follows:

- Form the cointegrating residual $\boldsymbol{\beta}'\mathbf{Y}_t = u_t$
- Perform a unit root test on u_t to determine if it is $I(0)$.

The null hypothesis in the Engle-Granger procedure is no-cointegration and the alternative is cointegration. There are two cases to consider. In the first case, the proposed cointegrating vector $\boldsymbol{\beta}$ is pre-specified (not estimated). For example, economic theory may imply specific values for the elements in $\boldsymbol{\beta}$ such as $\boldsymbol{\beta} = (1, -1)'$. The cointegrating residual is then readily constructed using the prespecified cointegrating vector. In the second case, the proposed cointegrating vector is estimated from the data and an estimate of the cointegrating residual $\hat{\boldsymbol{\beta}}'\mathbf{Y}_t = \hat{u}_t$ is formed. Tests for cointegration using a pre-specified cointegrating vector are generally much more powerful than tests employing an estimated vector.

12.3.1 Testing for Cointegration When the Cointegrating Vector Is Pre-specified

Let $\mathbf{Y}_t$ denote an $(n \times 1)$ vector of $I(1)$ time series, let $\boldsymbol{\beta}$ denote an $(n \times 1)$ prespecified cointegrating vector and let $u_t = \boldsymbol{\beta}'\mathbf{Y}_t$ denote the prespecified

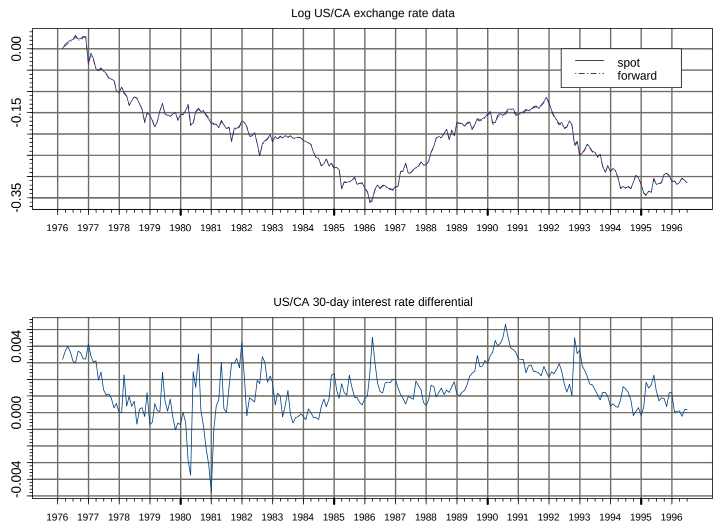


FIGURE 12.5. Log of US/CA spot and 30-day exchange rates and 30-day interest rate differential.

cointegrating residual. The hypotheses to be tested are

$$\begin{aligned} H_0 &: \quad u_t = \boldsymbol{\beta}'\mathbf{Y}_t \sim I(1) \text{ (no cointegration)} \qquad (12.14) \\ H_1 &: \quad u_t = \boldsymbol{\beta}'\mathbf{Y}_t \sim I(0) \text{ (cointegration)} \end{aligned}$$

Any unit root test statistic may be used to evaluate the above hypotheses. The most popular choices are the ADF and PP statistics. Cointegration is found if the unit root test rejects the no-cointegration null. It should be kept in mind, however, that the cointegrating residual may include deterministic terms (constant or trend) and the unit root tests should account for these terms accordingly. See Chapter 4 for details about the application of unit root tests.

Example 76 *Testing for cointegration between spot and forward exchange rates using a known cointegrating vector*

In international finance, the covered interest rate parity arbitrage relationship states that the difference between the logarithm of spot and forward exchange rates is equal to the difference between nominal domestic and foreign interest rates. It seems reasonable to believe that interest rate spreads are $I(0)$ which implies that spot and forward rates are cointegrated with cointegrating vector $\boldsymbol{\beta} = (1, -1)'$. To illustrate, consider the log monthly spot, s_t, and 30 day forward, f_t, exchange rates between the

US and Canada over the period February 1976 through June 1996 taken from the S+FinMetrics "timeSeries" object lexrates.dat

```
> uscn.s = lexrates.dat[,"USCNS"]
> uscn.s@title = "Log of US/CA spot exchange rate"
> uscn.f = lexrates.dat[,"USCNF"]
> uscn.f@title = "Log of US/CA 30-day forward exchange rate"
> u = uscn.s - uscn.f
> colIds(u) = "USCNID"
> u@title = "US/CA 30-day interest rate differential"
```

The interest rate differential is constructed using the pre-specified cointegrating vector $\boldsymbol{\beta} = (1, -1)'$ as $u_t = s_t - f_t$. The spot and forward exchange rates and interest rate differential are illustrated in Figure 12.5. Visually, the spot and forward exchange rates clearly share a common trend and the interest rate differential appears to be $I(0)$. In addition, there is no clear deterministic trend behavior in the exchange rates. The S+FinMetrics function unitroot may be used to test the null hypothesis that s_t and f_t are not cointegrated ($u_t \sim I(1)$). The ADF t-test based on 11 lags and a constant in the test regression leads to the rejection at the 5% level of the hypothesis that s_t and f_t are not cointegrated with cointegrating vector $\boldsymbol{\beta} = (1, -1)'$:

```
> unitroot(u, trend="c", method="adf", lags=11)

Test for Unit Root: Augmented DF Test

Null Hypothesis: there is a unit root
   Type of Test: t-test
 Test Statistic: -2.881
        P-value: 0.04914

Coefficients:
    lag1    lag2    lag3    lag4    lag5    lag6    lag7
 -0.1464 -0.1171 -0.0702 -0.1008 -0.1234 -0.1940  0.0128

    lag8    lag9   lag10   lag11 constant
 -0.1235  0.0550  0.2106 -0.1382  0.0002

Degrees of freedom: 234 total; 222 residual
Time period: from Jan 1977 to Jun 1996
Residual standard error: 8.595e-4
```

12.3.2 Testing for Cointegration When the Cointegrating Vector Is Estimated

Let $\mathbf{Y}_t$ denote an $(n \times 1)$ vector of $I(1)$ time series and let $\boldsymbol{\beta}$ denote an $(n \times 1)$ unknown cointegrating vector. The hypotheses to be tested are given in (12.14). Since $\boldsymbol{\beta}$ is unknown, to use the Engle-Granger procedure it must be first estimated from the data. Before $\boldsymbol{\beta}$ can be estimated some normalization assumption must be made to uniquely identify it. A common normalization is to specify the first element in $\mathbf{Y}_t$ as the dependent variable and the rest as the explanatory variables. Then $\mathbf{Y}_t = (y_{1t}, \mathbf{Y}_{2t}')'$ where $\mathbf{Y}_{2t} = (y_{2t}, \ldots, y_{nt})'$ is an $((n-1) \times 1)$ vector and the cointegrating vector is normalized as $\boldsymbol{\beta} = (1, -\boldsymbol{\beta}_2')'$. Engle and Granger proposed estimating the normalized cointegrating vector $\boldsymbol{\beta}_2$ by least squares from the regression

$$y_{1t} = c + \boldsymbol{\beta}_2'\mathbf{Y}_{2t} + u_t \tag{12.15}$$

and testing the no-cointegration hypothesis (12.14) with a unit root test using the estimated cointegrating residual

$$\hat{u}_t = y_{1t} - \hat{c} - \hat{\boldsymbol{\beta}}_2\mathbf{Y}_{2t} \tag{12.16}$$

where $\hat{c}$ and $\hat{\boldsymbol{\beta}}_2$ are the least squares estimates of c and $\boldsymbol{\beta}_2$. The unit root test regression in this case is without deterministic terms (constant or constant and trend). Phillips and Ouliaris (1990) showed that ADF and PP unit root tests (t-tests and normalized bias) applied to the estimated cointegrating residual (12.16) *do not* have the usual Dickey-Fuller distributions under the null hypothesis (12.14) of no-cointegration. Instead, due to the spurious regression phenomenon under the null hypothesis (12.14), the distribution of the ADF and PP unit root tests have asymptotic distributions that are functions of Wiener processes that depend on the deterministic terms in the regression (12.15) used to estimate $\boldsymbol{\beta}_2$ *and* the number of variables, $n-1$, in $\mathbf{Y}_{2t}$. These distributions are known as the *Phillips-Ouliaris* (PO) distributions, and are described in Phillips and Ouliaris (1990). To further complicate matters, Hansen (1992) showed the appropriate PO distributions of the ADF and PP unit root tests applied to the residuals (12.16) also depend on the trend behavior of y_{1t} and $\mathbf{Y}_{2t}$ as follows:

Case I: $\mathbf{Y}_{2t}$ and y_{1t} are both $I(1)$ without drift. The ADF and PP unit root test statistics follow the PO distributions, adjusted for a constant, with dimension parameter $n-1$.

Case II: $\mathbf{Y}_{2t}$ is $I(1)$ with drift and y_{1t} may or may not be $I(1)$ with drift. The ADF and PP unit root test statistics follow the PO distributions, adjusted for a constant and trend, with dimension parameter $n-2$. If $n-2=0$ then the ADF and PP unit root test statistics follow the DF distributions adjusted for a constant and trend.

Case III: $\mathbf{Y}_{2t}$ is $I(1)$ without drift and y_{1t} is $I(1)$ with drift. In this case, $\boldsymbol{\beta}_2$ should be estimated from the regression

$$y_{1t} = c + \delta t + \boldsymbol{\beta}_2'\mathbf{Y}_{2t} + u_t \tag{12.17}$$

The resulting ADF and PP unit root test statistics on the residuals from (12.17) follow the PO distributions, adjusted for a constant and trend, with dimension parameter $n-1$.

Computing Quantiles and P-values from the Phillips-Ouliaris Distributions Using the `S+FinMetrics` Functions `pcoint` and `qcoint`

The `S+FinMetrics` functions `qcoint` and `pcoint`, based on the response surface methodology of MacKinnon (1996), may be used to compute quantiles and p-values from the PO distributions. For example, to compute the 10%, 5% and 1% quantiles from the PO distribution for the ADF t-statistic, adjusted for a constant, with $n-1=3$ and a sample size $T=100$ use

```
> qcoint(c(0.1,0.05,0.01), n.sample=100, n.series=4,
+ trend="c", statistic="t")
[1] -3.8945 -4.2095 -4.8274
```

Notice that the argument `n.series` represents the total number of variables n. To adjust the PO distributions for a constant and trend set `trend="ct"`. To compute the PO distribution for the ADF normalized bias statistic set `statistic="n"`. The quantiles from the PO distributions can be very different from the quantiles from the DF distributions, especially if $n-1$ is large. To illustrate, the 10%, 5% and 1% quantiles from the DF distribution for the ADF t-statistic with a sample size $T=100$ are

```
> qunitroot(c(0.1,0.05,0.01), n.sample=100,
+ trend="c", statistic="t")
[1] -2.5824 -2.8906 -3.4970
```

The following examples illustrate testing for cointegration using an estimated cointegrating vector.

Example 77 *Testing for cointegration between spot and forward exchange rates using an estimated cointegrating vector*

Consider testing for cointegration between spot and forward exchange rates assuming the cointegrating vector is not known using the same data as in the previous example. Let $\mathbf{Y}_t = (s_t, f_t)'$ and normalize the cointegrating vector on s_t so that $\boldsymbol{\beta} = (1, -\beta_2)'$. The normalized cointegrating coefficient β_2 is estimated by least squares from the regression

$$s_t = c + \beta_2 f_t + u_t$$

giving the estimated cointegrating residual $\hat{u}_t = s_t - \hat{c} - \hat{\beta}_2 f_t$. The `OLS` function in `S+FinMetrics` is used to estimate the above regression:

```
> uscn.ts = seriesMerge(uscn.s,uscn.f)
> ols.fit = OLS(USCNS~USCNF,data=uscn.ts)
> ols.fit

Call:
OLS(formula = USCNS ~USCNF, data = uscn.ts)

Coefficients:
 (Intercept)  USCNF
 0.0023      1.0041

Degrees of freedom: 245 total; 243 residual
Time period: from Feb 1976 to Jun 1996
Residual standard error: 0.001444
```

The estimated value of β_2 is 1.004 and is almost identical to the value $\beta_2 = 1$ implied by covered interest parity. The estimated cointegrating residual $\hat{u}_t$ is extracted from the least squres fit using

```
> u.hat = residuals(ols.fit)
```

Next, the no-cointegration hypothesis (12.14) is tested using the ADF and PP t-tests. Because the mean of $\hat{u}_t$ is zero, the unit root test regressions are estimated without a constant or trend. The ADF t-statistic is computed using 11 lags, as in the previous example, and the PP t-statistic is computed using an automatic lag truncation parameter:

```
> adf.fit = unitroot(u.hat,trend="nc",method="adf",lags=11)
> adf.tstat = adf.fit$sval
> adf.tstat
   lag1
 -2.721

> pp.fit = unitroot(u.hat,trend="nc",method="pp")
> pp.tstat = pp.fit$sval
> pp.tstat
   lag1
 -5.416
```

The ADF t-statistic is -2.721 whereas the PP t-statistic is -5.416. Since s_t and f_t are both $I(1)$ without drift, the 10%, 5% and 1% quantiles from the approrpiate Phillips-Ouliaris distribution for the ADF t-statistic is

```
> qcoint(c(0.10,0.05,0.01),n.sample=nrow(uscn.s),n.series=2,
+ trend="c",statistic="t")
[1] -3.062 -3.361 -3.942
```

The no-cointegration null hypothesis is not rejected at the 10% level using the ADF t-statistic but is rejected at the 1% level using the PP t-statistic. The p-values for the ADF and PP t-statistics are

```
> pcoint(adf.tstat, n.sample=nrow(uscn.s), n.series=2,
+ trend="c", statistic="t")
[1] 0.1957

> pcoint(pp.tstat, n.sample=nrow(uscn.s), n.series=2,
+ trend="c", statistic="t")
[1] 0.00003925
```

12.4 Regression-Based Estimates of Cointegrating Vectors and Error Correction Models

12.4.1 Least Square Estimator

Least squares may be used to consistently estimate a normalized cointegrating vector. However, the asymptotic behavior of the least squares estimator is non-standard. The following results about the behavior of $\hat{\boldsymbol{\beta}}_2$ if $\mathbf{Y}_t$ is cointegrated are due to Stock (1987) and Phillips (1991):

- $T(\hat{\boldsymbol{\beta}}_2 - \boldsymbol{\beta}_2)$ converges in distribution to a non-normal random variable not necessarily centered at zero.

- The least squares estimate $\hat{\boldsymbol{\beta}}_2$ is consistent for $\boldsymbol{\beta}_2$ and converges to $\boldsymbol{\beta}_2$ at rate T instead of the usual rate $T^{1/2}$. That is, $\hat{\boldsymbol{\beta}}_2$ is *super consistent.*

- $\hat{\boldsymbol{\beta}}_2$ is consistent even if Y_{2t} is correlated with u_t so that there is no asymptotic simultaneity bias.

- In general, the asymptotic distribution of $T(\hat{\boldsymbol{\beta}}_2 - \boldsymbol{\beta}_2)$ is asymptotically biased and non-normal. The usual OLS formula for computing $\widehat{avar}(\hat{\boldsymbol{\beta}}_2)$ is incorrect and so the usual OLS standard errors are not correct.

- Even though the asymptotic bias goes to zero as T gets large $\hat{\boldsymbol{\beta}}_2$ may be substantially biased in small samples. The least squres estimator is also not efficient.

The above results indicate that the least squares estimator of the cointegrating vector $\boldsymbol{\beta}_2$ could be improved upon. A simple improvement is suggested by Stock and Watson (1993).

12.4.2 Stock and Watson's Efficient Lead/Lag Estimator

Stock and Watson (1993) provided a very simple method for obtaining an asymptotically efficient (equivalent to maximum likelihood) estimator for the normalized cointegrating vector $\boldsymbol{\beta}_2$ as well as a valid formula for computing its asymptotic variance[2].

Let $\mathbf{Y}_t = (y_{1t}, \mathbf{Y}_{2t}')'$ where $\mathbf{Y}_{2t} = (y_{2t}, \ldots, y_{nt})'$ is an $((n-1)\times 1)$ vector and let the cointegrating vector be normalized as $\boldsymbol{\beta} = (1, -\boldsymbol{\beta}_2')'$. Stock and Watson's efficient estimation procedure is:

- Augment the cointegrating regression of y_{1t} on $\mathbf{Y}_{2t}$ with appropriate deterministic terms $\mathbf{D}_t$ with p leads and lags of $\Delta\mathbf{Y}_{2t}$

$$\begin{aligned} y_{1t} &= \boldsymbol{\gamma}'\mathbf{D}_t + \boldsymbol{\beta}_2'\mathbf{Y}_{2t} + \sum_{j=-p}^{p} \boldsymbol{\psi}_j'\Delta\mathbf{Y}_{2t+j} + u_t \qquad (12.18)\\ &= \boldsymbol{\gamma}'\mathbf{D}_t + \boldsymbol{\beta}_2'\mathbf{Y}_{2t} + \boldsymbol{\psi}_p'\Delta\mathbf{Y}_{2t+p} + \cdots + \boldsymbol{\psi}_1'\Delta\mathbf{Y}_{2t+1} \\ &\quad + \boldsymbol{\psi}_0'\Delta\mathbf{Y}_{2t} + \boldsymbol{\psi}_{-1}'\Delta\mathbf{Y}_{2t-1} + \cdots + \boldsymbol{\psi}_{-p}'\Delta\mathbf{Y}_{2t-p} + u_t \end{aligned}$$

- Estimate the augmented regression by least squares. The resulting estimator of $\boldsymbol{\beta}_2$ is called the *dynamic OLS* estimator and is denoted $\hat{\boldsymbol{\beta}}_{2,\mathrm{DOLS}}$. It will be consistent, asymptotically normally distributed and efficient (equivalent to MLE) under certain assumptions (see Stock and Watson, 1993).

- Asymptotically valid standard errors for the individual elements of $\hat{\boldsymbol{\beta}}_{2,\mathrm{DOLS}}$ are given by the OLS standard errors from (12.18) multiplied by the ratio

$$\left(\frac{\hat{\sigma}_u^2}{\widehat{\mathrm{lrv}}(u_t)}\right)^{1/2}$$

  where $\hat{\sigma}_u^2$ is the OLS estimate of $\mathrm{var}(u_t)$ and $\widehat{\mathrm{lrv}}(u_t)$ is any consistent estimate of the long-run variance of u_t using the residuals $\hat{u}_t$ from (12.18). Alternatively, the Newey-West HAC standard errors may also be used.

Example 78 *DOLS estimation of cointegrating vector using exchange rate data*[3]

Let s_t denote the log of the monthly spot exchange rate between two currencies at time t and let f_t^k denote the log of the forward exchange rate at time t for delivery of foreign currency at time $t+k$. Under rational

[2]Hamilton (1994) chapter 19, and Hayashi (2000) chapter 10, give nice discussions of the Stock and Watson procedure.

[3]This example is based on Zivot (2000).

expectations and risk neutrality f_t^k is an unbiased predictor of s_{t+k}, the spot exchange rate at time $t+k$. That is

$$s_{t+k} = f_t^k + \varepsilon_{t+k}$$

where ε_{t+k} is a white noise error term. This is known as the *forward rate unbiasedness hypothesis* (FRUH). Assuming that s_t and f_t^k are $I(1)$ the FRUH implies that s_{t+k} and f_t^k are cointegrated with cointegrating vector $\boldsymbol{\beta} = (1, -1)'$. To illustrate, consider again the log monthly spot, s_t, and one month forward, f_t^1, exchange rates between the US and Canada over the period February 1976 through June 1996 taken from the S+FinMetrics "timeSeries" object lexrates.dat.The cointegrating vector between s_{t+1} and f_t^1 is estimated using least squares and Stock and Watson's dynamic OLS estimator computed from (12.18) with $y_{1t} = s_{t+1}$, $\mathbf{D}_t = 1$, $\mathbf{Y}_{2t} = f_t^1$ and $p = 3$. The data for the DOLS regression equation (12.18) are constucted as

```
> uscn.df = diff(uscn.f)
> colIds(uscn.df) = "D.USCNF"
> uscn.df.lags = tslag(uscn.df,-3:3,trim=T)
> uscn.ts = seriesMerge(uscn.s,uscn.f,uscn.df.lags)
> colIds(uscn.ts)
[1] "USCNS"         "USCNF"         "D.USCNF.lead3"
[4] "D.USCNF.lead2" "D.USCNF.lead1" "D.USCNF.lag0"
[7] "D.USCNF.lag1"  "D.USCNF.lag2"  "D.USCNF.lag3"
```

The least squares estimator of the normalized cointegrating coefficient β_2 computed using the S+FinMetrics function OLS is

```
> summary(OLS(tslag(USCNS,-1)~USCNF,data=uscn.ts,na.rm=T))

Call:
OLS(formula = tslag(USCNS, -1) ~USCNF, data = uscn.ts,
na.rm = T)

Residuals:
     Min      1Q  Median      3Q     Max
 -0.0541 -0.0072  0.0006  0.0097  0.0343

Coefficients:
               Value Std. Error  t value Pr(>|t|)
(Intercept)  -0.0048   0.0025    -1.9614   0.0510
      USCNF   0.9767   0.0110    88.6166   0.0000

Regression Diagnostics:

         R-Squared 0.9709
```

```
Adjusted R-Squared 0.9708
Durbin-Watson Stat 2.1610

Residual standard error: 0.01425 on 235 degrees of freedom
Time period: from Jun 1976 to Feb 1996
F-statistic: 7853 on 1 and 235 degrees of freedom,
the p-value is 0
```

Notice that in the regression formula, `tslag(USCN,-1)` computes s_{t+1}. The least squares estimate of β_2 is 0.977 with an estimated standard error of 0.011 indicating that f_t^1 underpredicts s_{t+1}. However, the usual formula for computing the estimated standard error is incorrect and should not be trusted.

The DOLS estimator of β_2 based on (12.18) is computed using

```
> dols.fit = OLS(tslag(USCNS,-1)~USCNF +
+ D.USCNF.lead3+D.USCNF.lead2+D.USCNF.lead1 +
+ D.USCNF.lag0+D.USCNF.lag1+D.USCNF.lag2+D.USCNF.lag3,
+ data=uscn.ts,na.rm=T)
```

The Newey-West HAC standard errors for the estimated coefficients are computed using `summary` with `correction="nw"`:

```
> summary(dols.fit,correction="nw")

Call:
OLS(formula = tslag(USCNS, -1) ~USCNF + D.USCNF.lead3 +
D.USCNF.lead2 + D.USCNF.lead1 + D.USCNF.lag0 +
D.USCNF.lag1 + D.USCNF.lag2 + D.USCNF.lag3, data =
uscn.ts, na.rm = T)

Residuals:
     Min      1Q  Median      3Q     Max
 -0.0061 -0.0008  0.0000  0.0009  0.0039

Coefficients:
                  Value Std. Error   t value  Pr(>|t|)
  (Intercept)    0.0023    0.0005     4.3948    0.0000
        USCNF    1.0040    0.0019   531.8862    0.0000
D.USCNF.lead3    0.0114    0.0063     1.8043    0.0725
D.USCNF.lead2    0.0227    0.0068     3.3226    0.0010
D.USCNF.lead1    1.0145    0.0090   112.4060    0.0000
 D.USCNF.lag0    0.0005    0.0073     0.0719    0.9427
 D.USCNF.lag1   -0.0042    0.0061    -0.6856    0.4937
 D.USCNF.lag2   -0.0056    0.0061    -0.9269    0.3549
 D.USCNF.lag3   -0.0014    0.0045    -0.3091    0.7575
```

```
Regression Diagnostics:

         R-Squared 0.9997
Adjusted R-Squared 0.9997
Durbin-Watson Stat 0.4461

Residual standard error: 0.001425 on 228 degrees of freedom
Time period: from Jun 1976 to Feb 1996
F-statistic: 101000 on 8 and 228 degrees of freedom,
the p-value is 0
```

The DOLS estimator of β_2 is 1.004 with a very small estimated standard error of 0.0019 and indicates that f_t^1 is essentially an unbiased predictor of the future spot rate s_{t+1}.

12.4.3 Estimating Error Correction Models by Least Squares

Consider a bivariate $I(1)$ vector $\mathbf{Y}_t = (y_{1t}, y_{2t})'$ and assume that $\mathbf{Y}_t$ is cointegrated with cointegrating vector $\boldsymbol{\beta} = (1, -\beta_2)'$ so that $\boldsymbol{\beta}'\mathbf{Y}_t = y_{1t} - \beta_2 y_{2t}$ is $I(0)$. Suppose one has a consistent estimate $\hat{\beta}_2$ (by OLS or DOLS) of the cointegrating coefficient and is interested in estimating the corresponding error correction model (12.12)-(12.13) for Δy_{1t} and Δy_{2t}. Because $\hat{\beta}_2$ is super consistent it may be treated as known in the ECM, so that the estimated disequilibrium error $y_{1t} - \hat{\beta}_2 y_{2t}$ may be treated like the known disequilibrium error $y_{1t} - \beta_2 y_{2t}$. Since all variables in the ECM are $I(0)$, the two regression equations may be consistently estimated using ordinary least squares (OLS). Alternatively, the ECM system may be estimated by seemingly unrelated regressions (SUR) to increase efficiency if the number of lags in the two equations are different.

Example 79 *Estimation of error correction model for exchange rate data*

Consider again the monthly log spot rate, s_t, and log forward rate, f_t, data between the U.S. and Canada. Earlier it was shown that s_t and f_t are cointegrated with an estimated cointegrating coefficient $\hat{\beta}_2 = 1.004$. Now consider estimating an ECM of the form (12.12)-(12.13) by least squares using the estimated disequilibrium error $s_t - 1.004 \cdot f_t$. In order to estimate the ECM, the number of lags of Δs_t and Δf_t needs to be determined. This may be done using test statistics for the significance of the lagged terms or model selection criteria like AIC or BIC. An initial estimation using one lag of Δs_t and Δf_t may be performed using

```
> u.hat = uscn.s - 1.004*uscn.f
> colIds(u.hat) = "U.HAT"
> uscn.ds = diff(uscn.s)
> colIds(uscn.ds) = "D.USCNS"
> uscn.df = diff(uscn.f)
> colIds(uscn.df) = "D.USCNF"
> uscn.ts = seriesMerge(uscn.s,uscn.f,uscn.ds,uscn.df,u.hat)
> ecm.s.fit = OLS(D.USCNS~tslag(U.HAT)+tslag(D.USCNS)
+ +tslag(D.USCNF),data=uscn.ts,na.rm=T)
> ecm.f.fit = OLS(D.USCNF~tslag(U.HAT)+tslag(D.USCNS)+
+ tslag(D.USCNF),data=uscn.ts,na.rm=T)
```

The estimated coefficients from the fitted ECM are

```
> ecm.s.fit

Call:
OLS(formula = D.USCNS ~tslag(U.HAT) + tslag(D.USCNS) + tslag(
D.USCNF), data = uscn.ts, na.rm = T)

Coefficients:
 (Intercept) tslag(U.HAT) tslag(D.USCNS) tslag(D.USCNF)
 -0.0050      1.5621       1.2683        -1.3877

Degrees of freedom: 243 total; 239 residual
Time period: from Apr 1976 to Jun 1996
Residual standard error: 0.013605

> ecm.f.fit

Call:
OLS(formula = D.USCNF ~tslag(U.HAT) + tslag(D.USCNS) + tslag(
D.USCNF), data = uscn.ts, na.rm = T)

Coefficients:
 (Intercept) tslag(U.HAT) tslag(D.USCNS) tslag(D.USCNF)
 -0.0054      1.7547       1.3595        -1.4702

Degrees of freedom: 243 total; 239 residual
Time period: from Apr 1976 to Jun 1996
Residual standard error: 0.013646
```

12.5 VAR Models and Cointegration

The Granger representation theorem links cointegration to error correction models. In a series of important papers and in a marvelous textbook, Soren Johansen firmly roots cointegration and error correction models in a vector autoregression framework. This section outlines Johansen's approach to cointegration modeling.

12.5.1 The Cointegrated VAR

Consider the levels VAR(p) model for the $(n \times 1)$ vector $\mathbf{Y}_t$

$$\mathbf{Y}_t = \mathbf{\Phi}\mathbf{D}_t + \mathbf{\Pi}_1\mathbf{Y}_{t-1} + \cdots + \mathbf{\Pi}_p\mathbf{Y}_{t-p} + \boldsymbol{\varepsilon}_t,\ t = 1, \ldots, T, \tag{12.19}$$

where $\mathbf{D}_t$ contains deterministic terms (constant, trend, seasonal dummies etc.). Recall, the VAR(p) model is stable if

$$\det(\mathbf{I}_n - \mathbf{\Pi}_1 z - \cdots - \mathbf{\Pi}_p z^p) = 0 \tag{12.20}$$

has all roots outside the complex unit circle. If (12.20) has a root on the unit circle then some or all of the variables in $\mathbf{Y}_t$ are $I(1)$ and they may also be cointegrated. Recall, $\mathbf{Y}_t$ is cointegrated if there exists some linear combination of the variables in $\mathbf{Y}_t$ that is $I(0)$. Suppose $\mathbf{Y}_t$ is $I(1)$ and possibly cointegrated. Then the VAR representation (22.11) is not the most suitable representation for analysis because the cointegrating relations are not explicitly apparent. The cointegrating relations become apparent if the levels VAR (22.11) is transformed to the *vector error correction model* (VECM)

$$\Delta\mathbf{Y}_t = \mathbf{\Phi}\mathbf{D}_t + \mathbf{\Pi}\mathbf{Y}_{t-1} + \mathbf{\Gamma}_1\Delta\mathbf{Y}_{t-1} + \cdots + \mathbf{\Gamma}_{p-1}\Delta\mathbf{Y}_{t-p+1} + \boldsymbol{\varepsilon}_t \tag{12.21}$$

where $\mathbf{\Pi} = \mathbf{\Pi}_1 + \cdots + \mathbf{\Pi}_p - \mathbf{I}_n$ and $\mathbf{\Gamma}_k = -\sum_{j=k+1}^{p}\mathbf{\Pi}_j$, $k = 1, \ldots, p-1$. The matrix $\mathbf{\Pi}$ is called the *long-run impact matrix* and $\mathbf{\Gamma}_k$ are the *short-run impact matrices*. Notice that the VAR parameters $\mathbf{\Pi}_i$ may be recovered from the VECM parameters $\mathbf{\Pi}$ and $\mathbf{\Gamma}_k$ via

$$\begin{aligned} \mathbf{\Pi}_1 &= \mathbf{\Gamma}_1 + \mathbf{\Pi} + \mathbf{I}_n, \\ \mathbf{\Pi}_k &= \mathbf{\Gamma}_k - \mathbf{\Gamma}_{k-1},\ k = 2, \ldots, p. \end{aligned} \tag{12.22}$$

In the VECM (12.21), $\Delta\mathbf{Y}_t$ and its lags are $I(0)$. The term $\mathbf{\Pi}\mathbf{Y}_{t-1}$ is the only one which includes potential $I(1)$ variables and for $\Delta\mathbf{Y}_t$ to be $I(0)$ it must be the case that $\mathbf{\Pi}\mathbf{Y}_{t-1}$ is also $I(0)$. Therefore, $\mathbf{\Pi}\mathbf{Y}_{t-1}$ must contain the cointegrating relations if they exist. If the VAR(p) process has unit roots then from (12.20) it is clear that $\mathbf{\Pi}$ is a singular matrix. If $\mathbf{\Pi}$ is singular then it has *reduced rank*; that is $\text{rank}(\mathbf{\Pi}) = r < n$. There are two cases to consider:

1. $\text{rank}(\mathbf{\Pi}) = 0$. This implies that $\mathbf{\Pi} = \mathbf{0}$ and $\mathbf{Y}_t$ is $I(1)$ and not cointegrated. The VECM (12.21) reduces to a VAR($p-1$) in first differences

$$\Delta\mathbf{Y}_t = \mathbf{\Phi}\mathbf{D}_t + \mathbf{\Gamma}_1\Delta\mathbf{Y}_{t-1} + \cdots + \mathbf{\Gamma}_{p-1}\Delta\mathbf{Y}_{t-p+1} + \boldsymbol{\varepsilon}_t.$$

2. $0 < \text{rank}(\mathbf{\Pi}) = r < n$. This implies that $\mathbf{Y}_t$ is $I(1)$ with r linearly independent cointegrating vectors and $n-r$ common stochastic trends

(unit roots)[4]. Since $\mathbf{\Pi}$ has rank r it can be written as the product

$$\underset{(n\times n)}{\mathbf{\Pi}} = \underset{(n\times r)}{\boldsymbol{\alpha}}\underset{(r\times n)}{\boldsymbol{\beta}}'$$

where $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are $(n \times r)$ matrices with $\text{rank}(\boldsymbol{\alpha}) = \text{rank}(\boldsymbol{\beta}) = r$. The rows of $\boldsymbol{\beta}'$ form a basis for the r cointegrating vectors and the elements of $\boldsymbol{\alpha}$ distribute the impact of the cointegrating vectors to the evolution of $\Delta\mathbf{Y}_t$. The VECM (12.21) becomes

$$\Delta\mathbf{Y}_t = \mathbf{\Phi}\mathbf{D}_t + \boldsymbol{\alpha}\boldsymbol{\beta}'\mathbf{Y}_{t-1} + \mathbf{\Gamma}_1\Delta\mathbf{Y}_{t-1} + \cdots + \mathbf{\Gamma}_{p-1}\Delta\mathbf{Y}_{t-p+1} + \boldsymbol{\varepsilon}_t, \tag{12.23}$$

where $\boldsymbol{\beta}'\mathbf{Y}_{t-1} \sim I(0)$ since $\boldsymbol{\beta}'$ is a matrix of cointegrating vectors.

It is important to recognize that the factorization $\mathbf{\Pi} = \boldsymbol{\alpha}\boldsymbol{\beta}'$ is not unique since for any $r \times r$ nonsingular matrix $\mathbf{H}$ we have

$$\boldsymbol{\alpha}\boldsymbol{\beta}' = \boldsymbol{\alpha}\mathbf{H}\mathbf{H}^{-1}\boldsymbol{\beta}' = (\mathbf{a}\mathbf{H})(\boldsymbol{\beta}\mathbf{H}^{-1\prime})' = \mathbf{a}^*\boldsymbol{\beta}^{*\prime}.$$

Hence the factorization $\mathbf{\Pi} = \boldsymbol{\alpha}\boldsymbol{\beta}'$ only identifies the space spanned by the cointegrating relations. To obtain unique values of $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}'$ requires further restrictions on the model.

Example 80 *A bivariate cointegrated VAR(1) model*

Consider the bivariate VAR(1) model for $\mathbf{Y}_t = (y_{1t}, y_{2t})'$

$$\mathbf{Y}_t = \mathbf{\Pi}_1\mathbf{Y}_{t-1} + \boldsymbol{\varepsilon}_t.$$

The VECM is

$$\Delta\mathbf{Y}_t = \mathbf{\Pi}\mathbf{Y}_{t-1} + \boldsymbol{\varepsilon}_t$$

where $\mathbf{\Pi} = \mathbf{\Pi}_1 - \mathbf{I}_2$. Assuming $\mathbf{Y}_t$ is cointegrated there exists a 2×1 vector $\boldsymbol{\beta} = (\beta_1, \beta_2)'$ such that $\boldsymbol{\beta}'\mathbf{Y}_t = \beta_1 y_{1t} + \beta_2 y_{2t}$ is $I(0)$. Using the normalization $\beta_1 = 1$ and $\beta_2 = -\beta$ the cointegrating relation becomes $\boldsymbol{\beta}'\mathbf{Y}_t = y_{1t} - \beta y_{2t}$. This normalization suggests the stochastic long-run equilibrium relation

$$y_{1t} = \beta y_{2t} + u_t$$

where u_t is $I(0)$ and represents the stochastic deviations from the long-run equilibrium $y_{1t} = \beta y_{2t}$.

Since $\mathbf{Y}_t$ is cointegrated with one cointegrating vector, $\text{rank}(\mathbf{\Pi}) = 1$ and can be decomposed as

$$\mathbf{\Pi} = \boldsymbol{\alpha}\boldsymbol{\beta}' = \begin{pmatrix} \alpha_1 \\ \alpha_2 \end{pmatrix}\begin{pmatrix} 1 & -\beta \end{pmatrix} = \begin{pmatrix} \alpha_1 & -\alpha_1\beta \\ \alpha_2 & -\alpha_2\beta \end{pmatrix}.$$

[4]To see that $\mathbf{Y}_t$ has $n-r$ common stochastic trends we have to look at the Beveridge-Nelson decomposition of the moving average representation of $\Delta\mathbf{Y}_t$.

The elements in the vector $\boldsymbol{\alpha}$ are interpreted as *speed of adjustment* coefficients. The cointegrated VECM for $\Delta\mathbf{Y}_t$ may be rewritten as

$$\Delta\mathbf{Y}_t = \boldsymbol{\alpha}\boldsymbol{\beta}'\mathbf{Y}_{t-1} + \boldsymbol{\varepsilon}_t. \tag{12.24}$$

Writing the VECM equation by equation gives

$$\begin{aligned}\Delta y_{1t} &= \alpha_1(y_{1t-1} - \beta y_{2t-1}) + \varepsilon_{1t},\\ \Delta y_{2t} &= \alpha_2(y_{1t-1} - \beta y_{2t-1}) + \varepsilon_{2t}.\end{aligned}$$

The first equation relates the change in y_{1t} to the lagged disequilibrium error $\boldsymbol{\beta}'\mathbf{Y}_{t-1} = (y_{1t-1} - \beta y_{2t-1})$ and the second equation relates the change in Δy_{2t} to the lagged disequilibrium error as well. Notice that the reactions of y_1 and y_2 to the disequilibrium errors are captured by the adjustment coefficients α_1 and α_2.

The stability conditions for the bivariate VECM are related to the stability conditions for the disequilibrium error $\boldsymbol{\beta}'\mathbf{Y}_t$. By pre-multiplying (12.24) by $\boldsymbol{\beta}'$, it is straightforward to show that $\boldsymbol{\beta}'\mathbf{Y}_t$ follows an AR(1) process

$$\boldsymbol{\beta}'\mathbf{Y}_t = (\mathbf{1} + \boldsymbol{\beta}'\boldsymbol{\alpha})\boldsymbol{\beta}'\mathbf{Y}_{t-1} + \boldsymbol{\beta}'\boldsymbol{\varepsilon}_t$$

or

$$u_t = \phi u_{t-1} + v_t$$

where $u_t = \boldsymbol{\beta}'\mathbf{Y}_t$, $\phi = 1 + \boldsymbol{\beta}'\boldsymbol{\alpha} = 1 + (\alpha_1 - \beta\alpha_2)$ and $v_t = \boldsymbol{\beta}'\boldsymbol{\varepsilon}_t = u_{1t} - \beta u_{2t}$. The AR(1) model for u_t is stable as long as $|\phi| = |1 + (\alpha_1 - \beta\alpha_2)| < 1$. For example, suppose $\beta = 1$. Then the stability condition is $|\phi| = |1 + (\alpha_1 - \alpha_2)| < 1$ which is satisfied if $\alpha_1 - \alpha_2 < 0$ and $\alpha_1 - \alpha_2 > -2$. If $\alpha_2 = 0$ then $-2 < \alpha_1 < 0$ is the required stability condition.

12.5.2 Johansen's Methodology for Modeling Cointegration

The basic steps in Johansen's methodology are:

- Specify and estimate a VAR(p) model for $\mathbf{Y}_t$.
- Construct likelihood ratio tests for the rank of $\mathbf{\Pi}$ to determine the number of cointegrating vectors.
- If necessary, impose normalization and identifying restrictions on the cointegrating vectors.
- Given the normalized cointegrating vectors estimate the resulting cointegrated VECM by maximum likelihood.

12.5.3 Specification of Deterministic Terms

Following Johansen (1995), the deterministic terms in (12.23) are restricted to the form

$$\mathbf{\Phi D}_t = \boldsymbol{\mu}_t = \boldsymbol{\mu}_0 + \boldsymbol{\mu}_1 t$$

If the deterministic terms are unrestricted then the time series in $\mathbf{Y}_t$ may exhibit quadratic trends and there may be a linear trend term in the cointegrating relationships. Restricted versions of the trend parameters $\boldsymbol{\mu}_0$ and $\boldsymbol{\mu}_1$ limit the trending nature of the series in $\mathbf{Y}_t$. The trend behavior of $\mathbf{Y}_t$ can be classified into five cases:

1. Model $H_2(r)$: $\boldsymbol{\mu}_t = 0$ (no constant). The restricted VECM is

$$\Delta\mathbf{Y}_t = \boldsymbol{\alpha}\boldsymbol{\beta}'\mathbf{Y}_{t-1} + \mathbf{\Gamma}_1\Delta\mathbf{Y}_{t-1} + \cdots + \mathbf{\Gamma}_{p-1}\Delta\mathbf{Y}_{t-p+1} + \boldsymbol{\varepsilon}_t,$$

   and all the series in $\mathbf{Y}_t$ are $I(1)$ without drift and the cointegrating relations $\boldsymbol{\beta}'\mathbf{Y}_t$ have mean zero.

2. Model $H_1^*(r)$: $\boldsymbol{\mu}_t = \boldsymbol{\mu}_0 = \boldsymbol{\alpha}\boldsymbol{\rho}_0$ (restricted constant). The restricted VECM is

$$\Delta\mathbf{Y}_t = \boldsymbol{\alpha}(\boldsymbol{\beta}'\mathbf{Y}_{t-1} + \boldsymbol{\rho}_0) + \mathbf{\Gamma}_1\Delta\mathbf{Y}_{t-1} + \cdots + \mathbf{\Gamma}_{p-1}\Delta\mathbf{Y}_{t-p+1} + \boldsymbol{\varepsilon}_t,$$

   the series in $\mathbf{Y}_t$ are $I(1)$ without drift and the cointegrating relations $\boldsymbol{\beta}'\mathbf{Y}_t$ have non-zero means $\boldsymbol{\rho}_0$.

3. Model $H_1(r)$: $\boldsymbol{\mu}_t = \boldsymbol{\mu}_0$ (unrestricted constant). The restricted VECM is

$$\Delta\mathbf{Y}_t = \boldsymbol{\mu}_0 + \boldsymbol{\alpha}\boldsymbol{\beta}'\mathbf{Y}_{t-1} + \mathbf{\Gamma}_1\Delta\mathbf{Y}_{t-1} + \cdots + \mathbf{\Gamma}_{p-1}\Delta\mathbf{Y}_{t-p+1} + \boldsymbol{\varepsilon}_t$$

   the series in $\mathbf{Y}_t$ are $I(1)$ with drift vector $\boldsymbol{\mu}_0$ and the cointegrating relations $\boldsymbol{\beta}'\mathbf{Y}_t$ may have a non-zero mean.

4. Model $H^*(r)$: $\boldsymbol{\mu}_t = \boldsymbol{\mu}_0 + \boldsymbol{\alpha}\boldsymbol{\rho}_1 t$ (restricted trend). The restricted VECM is

$$\begin{aligned}\Delta\mathbf{Y}_t &= \boldsymbol{\mu}_0 + \boldsymbol{\alpha}(\boldsymbol{\beta}'\mathbf{Y}_{t-1} + \boldsymbol{\rho}_1 t) \\ &\quad + \mathbf{\Gamma}_1\Delta\mathbf{Y}_{t-1} + \cdots + \mathbf{\Gamma}_{p-1}\Delta\mathbf{Y}_{t-p+1} + \boldsymbol{\varepsilon}_t\end{aligned}$$

   the series in $\mathbf{Y}_t$ are $I(1)$ with drift vector $\boldsymbol{\mu}_0$ and the cointegrating relations $\boldsymbol{\beta}'\mathbf{Y}_t$ have a linear trend term $\boldsymbol{\rho}_1 t$.

5. Model $H(r)$: $\boldsymbol{\mu}_t = \boldsymbol{\mu}_0 + \boldsymbol{\mu}_1 t$ (unrestricted constant and trend). The unrestricted VECM is

$$\Delta\mathbf{Y}_t = \boldsymbol{\mu}_0 + \boldsymbol{\mu}_1 t + \boldsymbol{\alpha}\boldsymbol{\beta}'\mathbf{Y}_{t-1} + \mathbf{\Gamma}_1\Delta\mathbf{Y}_{t-1} + \cdots + \mathbf{\Gamma}_{p-1}\Delta\mathbf{Y}_{t-p+1} + \boldsymbol{\varepsilon}_t,$$

   the series in $\mathbf{Y}_t$ are $I(1)$ with a linear trend (quadratic trend in levels) and the cointegrating relations $\boldsymbol{\beta}'\mathbf{Y}_t$ have a linear trend.

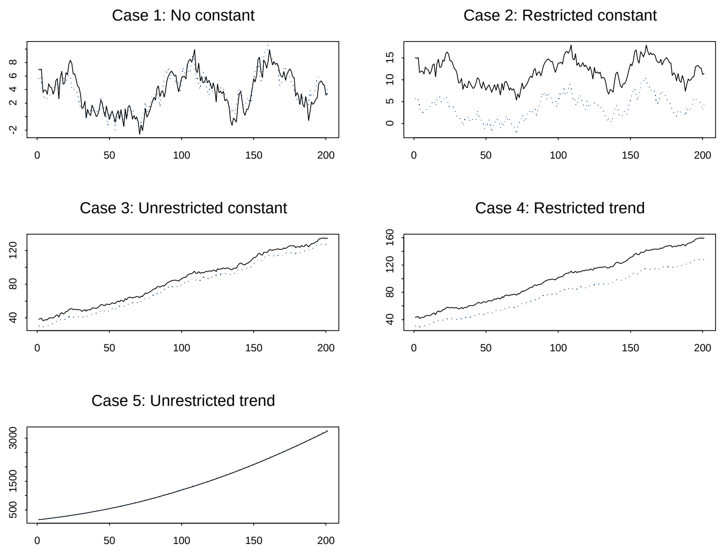


FIGURE 12.6. Simulated $\mathbf{Y}_t$ from bivariate cointegrated VECM for five trend cases.

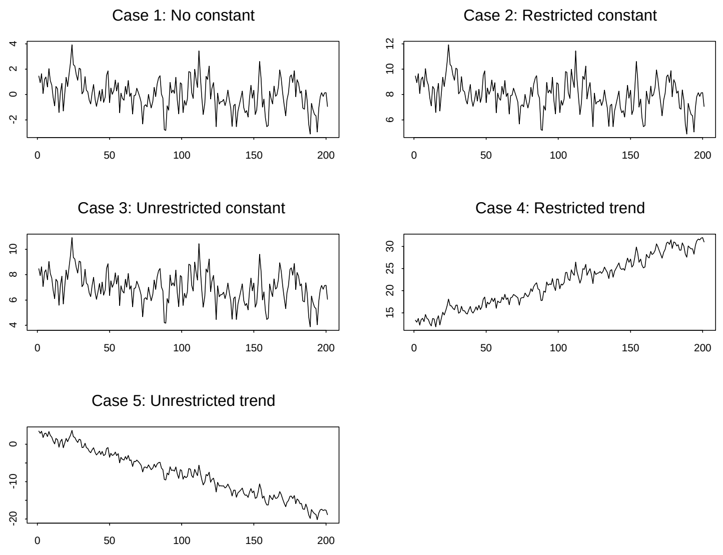


FIGURE 12.7. Simulated $\boldsymbol{\beta}'\mathbf{Y}_t$ from bivariate cointegrated VECM for five trend cases.

Simulated data from the five trend cases for a bivariate cointegrated VAR(1) model are illustrated in Figures 12.6 and 12.7. Case I is not really relevant for empirical work. The restricted contstant Case II is appropriate for non-trending $I(1)$ data like interest rates and exchange rates. The unrestriced constant Case III is appropriate for trending $I(1)$ data like asset prices, macroeconomic aggregates (real GDP, consumption, employment etc). The restricted trend case IV is also appropriate for trending $I(1)$ as in Case III. However, notice the deterministic trend in the cointegrating residual in Case IV as opposed to the stationary residual in case III. Finally, the unrestricted trend Case V is appropriate for $I(1)$ data with a quadratic trend. An example might be nominal price data during times of extreme inflation.

12.5.4 Likelihood Ratio Tests for the Number of Cointegrating Vectors

The unrestricted cointegrated VECM (12.23) is denoted $H(r)$. The $I(1)$ model $H(r)$ can be formulated as the condition that the rank of $\mathbf{\Pi}$ is less than or equal to r. This creates a nested set of models

$$H(0) \subset \cdots \subset H(r) \subset \cdots \subset H(n)$$

where $H(0)$ represents the non-cointegrated VAR model with $\mathbf{\Pi} = \mathbf{0}$ and $H(n)$ represents an unrestricted stationary VAR(p) model. This nested formulation is convenient for developing a sequential procedure to test for the number r of cointegrating relationships.

Since the rank of the long-run impact matrix $\mathbf{\Pi}$ gives the number of cointegrating relationships in $\mathbf{Y}_t$, Johansen formulates likelihood ratio (LR) statistics for the number of cointegrating relationships as LR statistics for determining the rank of $\mathbf{\Pi}$. These tests are based on the estimated eigenvalues $\hat{\lambda}_1 > \hat{\lambda}_2 > \cdots > \hat{\lambda}_n$ of the matrix $\mathbf{\Pi}$[5]. These eigenvalues also happen to equal the squared *canonical correlations* between $\Delta\mathbf{Y}_t$ and $\mathbf{Y}_{t-1}$ corrected for lagged $\Delta\mathbf{Y}_t$ and $\mathbf{D}_t$ and so lie between 0 and 1. Recall, the rank of $\mathbf{\Pi}$ is equal to the number of non-zero eigenvalues of $\mathbf{\Pi}$.

Johansen's Trace Statistic

Johansen's LR statistic tests the nested hypotheses

$$H_0(r) : r = r_0 \text{ vs. } H_1(r_0) : r > r_0$$

The LR statistic, called the *trace statistic*, is given by

$$\text{LR}_{\text{trace}}(r_0) = -T \sum_{i=r_0+1}^{n} \ln(1 - \hat{\lambda}_i)$$

[5]The calculation of the eigenvalues $\hat{\lambda}_i$ $(i = 1, \ldots, n)$ is described in the appendix.

If $\mathrm{rank}(\Pi) = r_0$ then $\hat{\lambda}_{r_0+1}, \ldots, \hat{\lambda}_n$ should all be close to zero and $\mathrm{LR}_{\mathrm{trace}}(r_0)$ should be small. In contrast, if $\mathrm{rank}(\mathbf{\Pi}) > r_0$ then some of $\hat{\lambda}_{r_0+1}, \ldots, \hat{\lambda}_n$ will be nonzero (but less than 1) and $\mathrm{LR}_{\mathrm{trace}}(r_0)$ should be large. The asymptotic null distribution of $\mathrm{LR}_{\mathrm{trace}}(r_0)$ is not chi-square but instead is a multivariate version of the Dickey-Fuller unit root distribution which depends on the dimension $n - r_0$ and the specification of the deterministic terms. Critical values for this distribution are tabulated in Osterwald-Lenum (1992) for the five trend cases discussed in the previous section for $n - r_0 = 1, \ldots, 10$.

Sequential Procedure for Determining the Number of Cointegrating Vectors

Johansen proposes a sequential testing procedure that consistently determines the number of cointegrating vectors. First test $H_0(r_0 = 0)$ against $H_1(r_0 > 0)$. If this null is not rejected then it is concluded that there are no cointegrating vectors among the n variables in Y_t. If $H_0(r_0 = 0)$ is rejected then it is concluded that there is at least one cointegrating vector and proceed to test $H_0(r_0 = 1)$ against $H_1(r_0 > 1)$. If this null is not rejected then it is concluded that there is only one cointegrating vector. If the null is rejected then it is concluded that there is at least two cointegrating vectors. The sequential procedure is continued until the null is not rejected.

Johansen's Maximum Eigenvalue Statistic

Johansen also derives a LR statistic for the hypotheses

$$H_0(r_0) : r = r_0 \text{ vs. } H_1(r_0) : r_0 = r_0 + 1$$

The LR statistic, called the maximum eigenvalue statistic, is given by

$$\mathrm{LR}_{\max}(r_0) = -T \ln(1 - \hat{\lambda}_{r_0+1})$$

As with the trace statistic, the asymptotic null distribution of $\mathrm{LR}_{\max}(r_0)$ is not chi-square but instead is a complicated function of Brownian motion, which depends on the dimension $n - r_0$ and the specification of the deterministic terms. Critical values for this distribution are tabulated in Osterwald-Lenum (1992) for the five trend cases discussed in the previous section for $n - r_0 = 1, \ldots, 10$.

Finite Sample Correction to LR Tests

Reinsel and Ahn (1992) and Reimars (1992) have suggested that the LR tests perform better in finite samples if the factor $T - np$ is used instead of T in the construction of the LR tests.

12.5.5 Testing Hypothesis on Cointegrating Vectors Using the S+FinMetrics Function coint

This section describes how to test for the number of cointegrating vectors, and how to perform certain tests for linear restrictions on the long-run $\boldsymbol{\beta}$ coefficients.

Testing for the Number of Cointegrating Vectors

The Johansen LR tests for determining the number of cointegrating vectors in multivariate time series may be computed using the `S+FinMetrics` function `coint`. The function `coint` has arguments

```
> args(coint)
function(Y, X = NULL, lags = 1, trend = "c", H = NULL,
b = NULL, save.VECM = T)
```

where `Y` is a matrix, data frame or "`timeSeries`" containing the $I(1)$ variables in $\mathbf{Y}_t$, `X` is a numeric object representing exogenous variables to be added to the VECM, `lags` denotes the number of lags in the VECM (one less than the number of lags in the VAR representation), `trend` determines the trend case specification, and `save.VECM` determines if the VECM information is to be saved. The arguments `H` and `b` will be explained later. The result of `coint` is an object of class "`coint`" for which there are `print` and `summary` methods. The use of `coint` is illustrated with the following examples.

Example 81 *Exchange rate data*

Consider testing for the number of cointegrating relations among the logarithms of the monthly spot and forward exchange rates in the "`timeSeries`" `uscn.ts` examined earlier. The first step is to determine the number of lags to use in the VECM. Using the `S+FinMetrics` function `VAR`, the lag length that minimizes the AIC with a maximum lag of 6 is $p = 2$:

```
> uscn.ts = seriesMerge(uscn.s, uscn.f)
> var.fit = VAR(uscn.ts,max.ar=6,criterion="AIC")
> var.fit$ar.order
[1] 2
```

The lag length for the VECM is then $p-1 = 1$. Since the monthly exchange rate data are not trending, the Johansen LR tests are computed assuming the restricted constant case II:

```
> coint.rc = coint(uscn.ts,trend="rc",lags=1)
> class(coint.rc)
[1] "coint"
> coint.rc
```

```
Call:
coint(Y = uscn.ts, lags = 1, trend = "rc")

Trend Specification:
H1*(r): Restricted constant

Trace tests significant at the 5% level are flagged by ' +'.
Trace tests significant at the 1% level are flagged by '++'.
Max Eigenvalue tests significant at the 5% level are flagged
by ' *'.
Max Eigenvalue tests significant at the 1% level are flagged
by '**'.

Tests for Cointegration Rank:
         Eigenvalue Trace Stat  95% CV  99% CV Max Stat
H(0)++**  0.0970    32.4687    19.9600 24.6000 24.8012
H(1)      0.0311     7.6675     9.2400 12.9700  7.6675

          95% CV  99% CV
H(0)++** 15.6700 20.2000
H(1)      9.2400 12.9700
```

Recall, the number of cointegrating vectors is equal to the number of nonzero eigenvalues of $\mathbf{\Pi}$. The two estimated eigenvalues are 0.0970 and 0.0311. The first row in the table gives $\mathrm{LR}_{\mathrm{trace}}(0)$ and $\mathrm{LR}_{\max}(0)$ for testing the null of $r_0 = 0$ cointegrating vectors as well as the 95% and 99% quantiles of the appropriate asymptotic distributions taken from the tables in Osterwald-Lenum (1992). Both the trace and maximum eigenvalue statistics reject the $r_0 = 0$ null at the 1% level. The second row in the table gives $\mathrm{LR}_{\mathrm{trace}}(1)$ and $\mathrm{LR}_{\max}(1)$ for testing the null of $r_0 = 1$. Neither statistic rejects the null that $r_0 = 1$.

The `summary` method gives the same output as `print` as well as the unnormalized cointegrating vectors, adjustment coefficients and the estimate of $\mathbf{\Pi}$.

Testing Linear Restrictions on Cointegrating Vectors

The `coint` function can also be used to test linear restrictions on the cointegrating vectors $\boldsymbol{\beta}$. Two types of restrictions are currently supported: the same linear restrictions on all cointegrating vectors in $\boldsymbol{\beta}$; some cointegrating vectors in $\boldsymbol{\beta}$ are assumed known. Following Johansen (1995), two examples are given illustrating how to use the `coint` function to test linear restrictions on $\boldsymbol{\beta}$.

Example 82 *Johansen's Danish data*

The "timeSeries" data set johansen.danish in S+FinMetrics contains the monthly Danish data used in Johansen (1995), with the columns LRM, LRY, LPY, IBO, IBE representing the log real money supply (m_t), log real income (y_t), log prices, bond rate (i_t^b) and deposit rate (i_t^d), respectively. Johansen (1995) considered testing the cointegrating relationship among m_t, y_t, i_t^b and i_t^d. A natural hypothesis is that the velocity of money is a function of the interest rates, or the cointegrating relation contains m_t and y_t only through the term $m_t - y_t$. For $\mathbf{R}' = (1, 1, 0, 0)$, this hypothesis can be represented as a linear restriction on $\boldsymbol{\beta}$:

$$H_0 : \mathbf{R}'\boldsymbol{\beta} = 0 \text{ or } \boldsymbol{\beta} = \mathbf{H}\boldsymbol{\Psi} \tag{12.25}$$

where $\boldsymbol{\Psi}$ are the unknown parameters in the cointegrating vectors $\boldsymbol{\beta}$, $\mathbf{H} = \mathbf{R}_\perp$ and $\mathbf{R}_\perp$ is the orthogonal complement of $\mathbf{R}$ such that $\mathbf{R}'\mathbf{R}_\perp = \mathbf{0}$. Johansen (1995) showed that the null hypothesis (12.25) against the alternative of unrestricted r cointegrating relations $H(r)$ can be tested using a likelihood ratio statistic, which is asymptotically distributed as a χ^2 with $r(n - s)$ degree of freedom where s is the number of columns in $\mathbf{H}$.

To test the hypothesis that the coefficients of m_t and y_t add up to zero in the cointegrating relations, Johansen (1995) considered a restricted constant model. In this case, $\mathbf{R}' = (1, 1, 0, 0, 0)$ since the restricted constant also enters the cointegrating space. Given the restriction matrix $\mathbf{R}$, the matrix $\mathbf{H}$ can be computed using the perpMat function in S+FinMetrics:

```
> R = c(1, 1, 0, 0, 0)
> H = perpMat(R)
> H
     [,1] [,2] [,3] [,4]
[1,]   -1    0    0    0
[2,]    1    0    0    0
[3,]    0    1    0    0
[4,]    0    0    1    0
[5,]    0    0    0    1
```

Now the test can be simply performed by passing the matrix $\mathbf{H}$ to the coint function:

```
> restr.mod1 = coint(johansen.danish[,c(1,2,4,5)],
+ trend="rc", H=H)
```

The result of the test can be shown by calling the generic print method on restr.mod1 with the optional argument restrictions=T:

```
> print(restr.mod1, restrictions=T)

Call:
coint(Y = johansen.danish[, c(1, 2, 4, 5)], trend = "rc",
      H = H)
```

```
Trend Specification:
H1*(r): Restricted constant

Tests for Linear Restriction on Coint Vectors:
Null hypothesis: the restriction is true
       Stat       Dist df P-value
H(1) 0.0346 chi-square  1  0.8523
H(2) 0.2607 chi-square  2  0.8778
H(3) 4.6000 chi-square  3  0.2035
H(4) 6.0500 chi-square  4  0.1954
```

For unrestricted sequential cointegration testing, the statistics in the output are labeled according to the null hypothesis, such as $H(0)$, $H(1)$, etc. However, when restrictions are imposed, the statistics in the output printed with `restrictions=T` are labeled according to the alternative hypothesis, such as $H(1)$, $H(2)$, etc. In the above output, the null hypothesis cannot be rejected against the alternatives of $H(1)$, $H(2)$, $H(3)$ and $H(4)$ at conventional levels of significance.

After confirming that the cointegrating coefficients on m_t and y_t add up to zero, it is interesting to see if $(1, -1, 0, 0, 0)$ actually is a cointegrating vector. In general, to test the null hypothesis that some cointegrating vectors in $\boldsymbol{\beta}$ are equal to $\mathbf{b}$:

$$H_0 : \boldsymbol{\beta} = (\mathbf{b}, \boldsymbol{\Psi})$$

where $\mathbf{b}$ is the $n \times s$ matrix of known cointegrating vectors and $\boldsymbol{\Psi}$ is the $n\times(r-s)$ matrix of unknown cointegrating vectors, Johansen (1995) showed that a likelihood ratio statistic can be used, which is asymptotically distributed as a χ^2 with $s(n-r)$ degrees of freedom. This test can be simply performed by setting the optional argument `b` to the known cointegrating vectors. For example,

```
> b = as.matrix(c(1,-1,0,0,0))
> restr.mod2 = coint(johansen.danish[,c(1,2,4,5)],
+ trend="rc", b=b)
> print(restr.mod2, restrictions=T)

Trend Specification:
H1*(r): Restricted constant

Tests for Known Coint Vectors:
Null hypothesis: the restriction is true
        Stat       Dist df P-value
H(1) 29.7167 chi-square  4  0.0000
H(2)  8.3615 chi-square  3  0.0391
```

```
H(3)  4.8759 chi-square  2  0.0873
H(4)  0.5805 chi-square  1  0.4461
```

Again, the statistics in the above output are labeled according to the alternative hypothesis $H(1)$, $H(2)$, etc. Although the hypothesis that the cointegrating coefficients on m_t and y_t sum up to zero cannot be rejected, $m_t - y_t$ does not seem to be stationary because the hypothesis of the known value of $\mathbf{b}$ is rejected at conventional levels of significance against $H(1)$ and $H(2)$.

12.5.6 Maximum Likelihood Estimation of the Cointegrated VECM

If it is found that $\text{rank}(\mathbf{\Pi}) = r$, $0 < r < n$, then the cointegrated VECM (12.23) becomes a reduced rank multivariate regression. The details of the maximum likelihood estimation of (12.23) under the reduced rank restriction $\text{rank}(\Pi) = r$ is briefly outlined in the Appendix to this chapter. There it is shown that

$$\hat{\boldsymbol{\beta}}_{\text{mle}} = (\hat{\mathbf{v}}_1, \ldots, \hat{\mathbf{v}}_r), \tag{12.26}$$

where $\hat{\mathbf{v}}_i$ are the eigenvectors associated with the eigenvalues $\hat{\lambda}_i$, and that the MLEs of the remaining parameters are obtained by multivariate least squares estimation of (12.23) with $\boldsymbol{\beta}$ replaced by $\hat{\boldsymbol{\beta}}_{\text{mle}}$.

Normalized Estimates

Recall, the factorization $\mathbf{\Pi} = \boldsymbol{\alpha}\boldsymbol{\beta}'$ is not unique and so the columns of $\hat{\boldsymbol{\beta}}_{\text{mle}}$ in (12.26) may be interpreted as linear combinations of the underlying cointegrating relations. For interpretations, it is often convenient to normalize or identify the cointegrating vectors by choosing a specific coordinate system in which to express the variables. One arbitrary way to do this, suggested by Johansen, is to solve for the triangular representation of the cointegrated system. The details of this normalization process is given in the appendix, and the `S+FinMetrics` function `VECM` utilizes this normalization scheme by default. The resulting normalized cointegrating vector is denoted $\hat{\boldsymbol{\beta}}_{c,\text{mle}}$. The normalization of the MLE for $\boldsymbol{\beta}$ to $\hat{\boldsymbol{\beta}}_{c,\text{mle}}$ will affect the MLE of $\boldsymbol{\alpha}$ but not the MLEs of the other parameters in the VECM.

It must be emphasized that it is not possible to estimate the individual elements of $\boldsymbol{\beta}$ without a specific normalization or identification scheme and that the normalization based on Phillips' triangular representation is arbitrary and the resulting normalized cointegrating vectors (12.29) may not have any economic meaning. Only in the case $r = 1$ can a unique cointegrating vector be found after normalization.

Example 83 *Unnormalzed MLEs for exchange rate data*

The unnormalized cointegrating vector assuming $r_0 = 1$ may also be extracted directly from the "`coint`" object:

```
> coint.rc$coint.vectors[1,]
     USCNS   USCNF Intercept*
 -739.0541 743.314   2.023532
```

Notice in the case of a restricted constant, the last coefficient in $\hat{\boldsymbol{\beta}}_{\text{mle}}$ is an estimate of the restricted constant. Normalizing on `USCNS` by dividing each element in $\hat{\boldsymbol{\beta}}_{\text{mle}}$ by -739.0541 gives

```
> coint.rc$coint.vectors[1,]/
+ as.numeric(-coint.rc$coint.vectors[1,1])
 USCNS    USCNF  Intercept*
    -1 1.005764 0.002738003
```

The normalized MLEs, $\hat{\boldsymbol{\beta}}_{c,\text{mle}} = (-1, 1.006)'$ and $\hat{\mu}_c = 0.0027$ are almost identical to the least squares estimates $\hat{\boldsymbol{\beta}} = (1, -1.004)'$ and $\hat{\mu} = 0.0023$ found earlier.

Asymptotic Distributions

Let $\hat{\boldsymbol{\beta}}_{c,\text{mle}}$ denote the MLE of the normalized cointegrating matrix $\boldsymbol{\beta}_c$. Johansen (1995) showed that $T(vec(\hat{\boldsymbol{\beta}}_{c,\text{mle}}) - vec(\boldsymbol{\beta}_c))$ is asymptotically (mixed) normally distributed and that a consistent estimate of the asymptotic covariance of $\hat{\boldsymbol{\beta}}_{c,\text{mle}}$ is given by

$$\widehat{avar}(vec(\hat{\boldsymbol{\beta}}_{c,\text{mle}})) =$$
$$T^{-1}(\mathbf{I}_n - \hat{\boldsymbol{\beta}}_{c,\text{mle}}\mathbf{c}')\mathbf{S}_{11}^{-1}(\mathbf{I}_n - \hat{\boldsymbol{\beta}}_{c,\text{mle}}\mathbf{c}')' \otimes \left(\hat{\boldsymbol{\alpha}}_{c,\text{mle}}'\hat{\boldsymbol{\Omega}}^{-1}\hat{\boldsymbol{\alpha}}_{c,\text{mle}}\right)^{-1} \quad (12.27)$$

Notice that this result implies that $\hat{\boldsymbol{\beta}}_{c,\text{mle}} \xrightarrow{p} \boldsymbol{\beta}_c$ at rate T instead of the usual rate $T^{1/2}$. Hence, like the least squares estimator, $\hat{\boldsymbol{\beta}}_{c,\text{mle}}$ is *super consistent*. However, unlike the least squares estimator, asymptotically valid standard errors may be compute using the square root of the diagonal elements of (12.27).

12.5.7 Maximum Likelihood Estimation of the Cointegrated VECM Using the `S+FinMetrics` Function `VECM`

Once the number of cointegrating vectors is determined from the `coint` function, the maximum likelihood estimates of the full VECM may be obtained using the `S+FinMetrics` function `VECM`. The arguments expected by `VECM` are

```
> args(VECM)
function(object, coint.rank = 1, coint.vec = NULL, X = NULL,
         unbiased = T, lags = 1, trend = "c", levels = F)
```

where `object` is either a "`coint`" object, usually produced by a call to the function `coint`, or a rectangular data object. If `object` is a "`coint`" object then `coint.rank` specifies the rank of $\mathbf{\Pi}$ to determine the number of cointegrating vectors to be used in the fitted VECM. The cointegrating vectors are then normalized using the Phillips' triangular representation described in the appendix. The lag length and trend specification for the VECM is obtained from the information in the "`coint`" object. The lag length in the fitted VECM, however, is one less than the lag length specified in the "`coint`" object. If `object` is a rectangular data object, then `coint.vec` must be assigned a matrix whose columns represent pre-specified cointegrating vectors. The argument `lags` is then used to specify the lag length of the VECM, and `trend` is used to set the trend specification. The optional argument `X` is used to specify any exogenous variables (e.g. dummy variables for events) other than a constant or trend. The optional argument `levels` determines if the VECM is to be fit to the levels $\mathbf{Y}_t$ or to the first differences $\Delta\mathbf{Y}_t$, and determines if forecasts are to be computed for the levels or the first differences. The result of `VECM` is an object of class "`VECM`", which inherits from "`VAR`" for which there are `print`, `summary`, `plot`, `cpredict` and `predict` methods and extractor functions `coef`, `fitted`, `residuals` and `vcov`. Since "`VECM`" objects inherit from "`VAR`" objects, most of the method and extractor functions for "`VECM`" objects work similarly to those for "`VAR`" objects. The use of `VECM` is illustrated with the following examples.

Example 84 *Maximum likelihood estimation of the VECM for exchange rate data*

Using the "`coint`" object `coint.rc` computed from the VAR(2) model with a restricted constant, the VECM(1) with a restricted constant for the exchange rate data is computed using

```
> vecm.fit = VECM(coint.rc)
> class(vecm.fit)
[1] "VECM"
> inherits(vecm.fit,"VAR")
[1] T
```

The `print` method gives the basic output

```
> vecm.fit

Call:
VECM(test = coint.rc)

Cointegrating Vectors:
           coint.1
     USCNS  1.0000
     USCNF -1.0058
Intercept* -0.0027

VECM Coefficients:
             USCNS   USCNF
   coint.1  1.7771  1.9610
USCNS.lag1  1.1696  1.2627
USCNF.lag1 -1.2832 -1.3679

Std. Errors of Residuals:
  USCNS  USCNF
 0.0135 0.0136

Information Criteria:
    logL     AIC     BIC      HQ
  2060.2 -4114.4 -4103.9 -4110.1

                   total residual
Degree of freedom:   243      240
Time period: from Apr 1976 to Jun 1996
```

The `print` method output is similar to that created by the `VAR` function. The output labeled `Cointegrating Vectors:` gives the estimated cointegrating vector coefficients normalized on the first variable in the specification of the VECM. To see standard errors for the estimated coefficients use the `summary` method

```
> summary(vecm.fit)

Call:
VECM(test = coint.rc)

Cointegrating Vectors:
             coint.1
              1.0000

     USCNF   -1.0058
 (std.err)    0.0031
  (t.stat) -326.6389

Intercept*   -0.0027
 (std.err)    0.0007
  (t.stat)   -3.9758


VECM Coefficients:
```

```
             USCNS   USCNF
   coint.1  1.7771  1.9610
 (std.err)  0.6448  0.6464
  (t.stat)  2.7561  3.0335

USCNS.lag1  1.1696  1.2627
 (std.err)  0.9812  0.9836
  (t.stat)  1.1921  1.2837

USCNF.lag1 -1.2832 -1.3679
 (std.err)  0.9725  0.9749
  (t.stat) -1.3194 -1.4030

Regression Diagnostics:
                USCNS  USCNF
     R-squared 0.0617 0.0689
Adj. R-squared 0.0538 0.0612
  Resid. Scale 0.0135 0.0136

Information Criteria:
    logL     AIC     BIC      HQ
  2060.2 -4114.4 -4103.9 -4110.1

                   total residual
Degree of freedom:   243      240
Time period: from Apr 1976 to Jun 1996
```

The VECM fit may be inspected graphically using the generic `plot` method

```
> plot(vecm.fit)

Make a plot selection (or 0 to exit):

1: plot: All
2: plot: Response and Fitted Values
3: plot: Residuals
4: plot: Normal QQplot of Residuals
5: plot: ACF of Residuals
6: plot: PACF of Residuals
7: plot: ACF of Squared Residuals
8: plot: PACF of Squared Residuals
9: plot: Cointegrating Residuals
10: plot: ACF of Cointegrating Residuals
11: plot: PACF of Cointegrating Residuals
12: plot: ACF of Squared Cointegrating Residuals
```

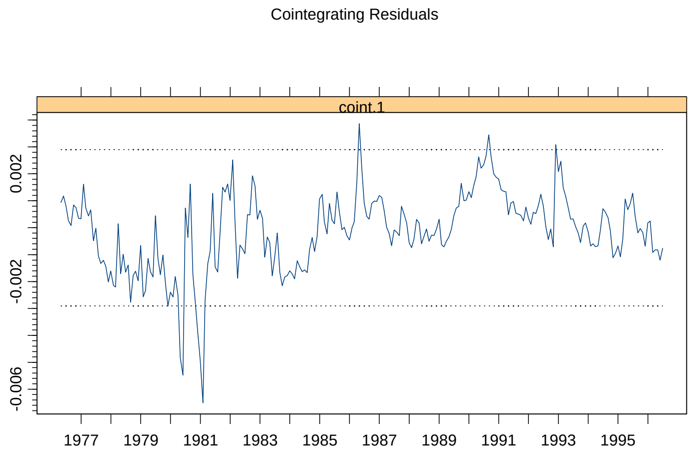


FIGURE 12.8. Cointegrating residual from maximum likelihood estimation of VECM(1) for exchange rate data.

```
13: plot: PACF of Squared Cointegrating Residuals
Selection:
```

The first eight plot options are the same as those created for a `"VAR"` object. The remaining plot options allow a graphical inspection of the cointegrating residual. For example, plot option 9 is illustrated in Figure 12.8. The estimated cointegrating residual appears to be $I(0)$.

Example 85 *Estimate VECM with pre-specified cointegrating vector*

For the exchange rate data, the MLE of the normalized cointegrating vector is close to $(1, -1)'$, and the estimate of the restricted constant is close to zero. These are the values implied by the FRUH. To estimate a VECM(1) imposing $\boldsymbol{\beta} = (1, -1)'$ and $\mu_t = 0$ use

```
> beta.true = as.matrix(c(1,-1, 0))
> dimnames(beta.true) = list(c("USCNS","USCNF","Intercept"),
+                            "coint.1")
> vecm.fruh.fit = VECM(uscn.ts, coint.vec = beta.true,
+                 lags = 1, trend = "rc")
```

Since the restricted constant lies in the cointegrating space, the last element of the pre-specified cointegrating vector is the value of the restricted constant. A summary of the restricted VECM fit is:

```
> summary(vecm.fruh.fit)

Call:
VECM(data = uscn.ts, coint.vec = beta.true, lags = 1, trend =
     "rc")

Cointegrating Vectors:
          coint.1
    USCNS       1
    USCNF      -1
Intercept       0

VECM Coefficients:
             USCNS   USCNF
   coint.1  0.0337  0.1354
 (std.err)  0.4442  0.4466
  (t.stat)  0.0758  0.3032

USCNS.lag1  2.1330  2.2781
 (std.err)  0.9535  0.9588
  (t.stat)  2.2371  2.3760

USCNF.lag1 -2.2226 -2.3573
 (std.err)  0.9472  0.9525
  (t.stat) -2.3465 -2.4748

Regression Diagnostics:
                USCNS  USCNF
     R-squared 0.0354 0.0393
Adj. R-squared 0.0273 0.0312
  Resid. Scale 0.0137 0.0138

Information Criteria:
      logL       AIC       BIC        HQ
  2053.662 -4101.324 -4090.845 -4097.103

                   total residual
Degree of freedom:   243      240
Time period: from Apr 1976 to Jun 1996
```

Notice that the VECM with the pre-specified cointegrating vector does not fit as well as the VECM using the estimated cointegrating vector.

12.5.8 *Forecasting from the VECM*

Forecasts from a VECM are computed by first transforming the VECM to a VAR using (12.22), and then using the forecasting algorithms for VAR models described in the previous chapter. For VECM models, one may forecast the changes in the variables, $\Delta \mathbf{Y}_t$, or the levels of the variables $\mathbf{Y}_t$. The generic `S+FinMetrics` functions `predict` and `cpredict` are used to compute unconditional and conditional forecasts from a “`VECM`” object. The following example illustrates the use of the `predict` method to compute forecasts for the differences and levels of the exchange rate data.

Example 86 *Forecasts from VECM fit to exchange rate data*

The “`VECM`” object `vecm.fit` was produced with the optional argument `levels=F`. Consequently, the `predict` method will produce forecasts for the changes in s_t and f_t. To compute h-step forecasts for Δs_t and Δf_t for $h = 1, \ldots, 12$ use

```
> vecm.fcst = predict(vecm.fit,n.predict=12)
> class(vecm.fcst)
[1] "forecast"
```

To see the forecast and forecast standard errors use

```
> summary(vecm.fcst)

Predicted Values with Standard Errors:

                USCNS   USCNF
 1-step-ahead -0.0105 -0.0110
    (std.err)  0.0136  0.0136
 2-step-ahead -0.0130 -0.0139
    (std.err)  0.0183  0.0183
...
12-step-ahead -0.0237 -0.0260
    (std.err)  0.0435  0.0432
```

By default, the forecasts are computed using the chain-rule of forecasting. To compute simulation-based forecasts use `method = "mc"` or `method = "bootstrap"` in the call to `predict`.

To see the forecasts with standard error bands along the original data use

```
> plot(vecm.fcst, xold=diff(uscn.ts), n.old=12)
```

Since the forecasts are of the first differenced data, the data passed to `xold` must be first differenced. The resulting plot is shown in Figure 12.9.

To compute forecasts for the levels s_t and f_t, re-fit the VECM with the optional argument `levels=T`

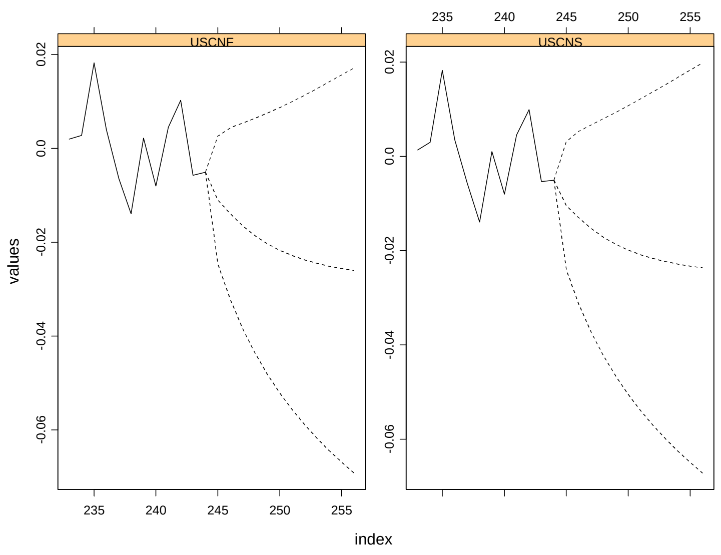


FIGURE 12.9. VECM forecasts of first differences of exchange rate data.

```
> vecm.fit.level = VECM(coint.rc, levels=T)
```

and then call the `predict` method as before

```
> vecm.fcst.level = predict(vecm.fit.level, n.predict=12)
> summary(vecm.fcst.level)

Predicted Values with Standard Errors:

                USCNS   USCNF
 1-step-ahead -0.3150 -0.3154
    (std.err)  0.0136  0.0136
 2-step-ahead -0.3157 -0.3161
    (std.err)  0.0183  0.0183
...
12-step-ahead -0.3185 -0.3193
    (std.err)  0.0435  0.0432
```

To plot the forecasts use

```
> plot(vecm.fcst.level, xold=uscn.ts, n.old=12)
```

The resulting plot is shown in Figure 12.10.

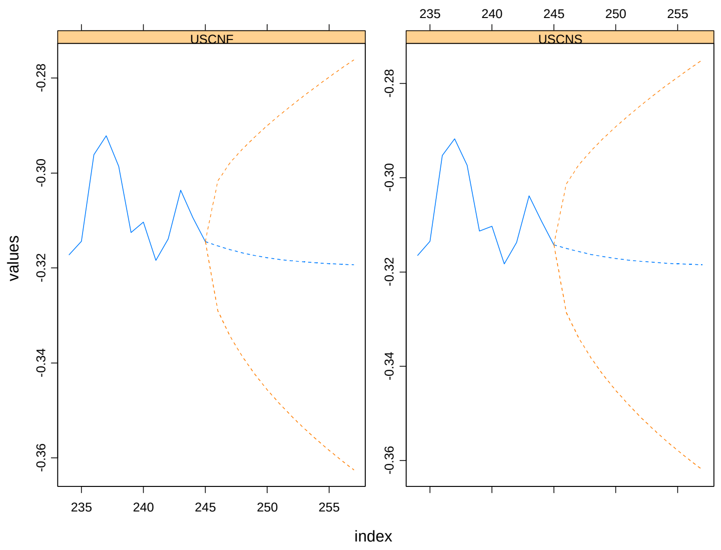


FIGURE 12.10. VECM forecasts of levels of exchange rate data.

12.6 Appendix: Maximum Likelihood Estimation of a Cointegrated VECM

The following brief discussion of maximum likelihood estimation of the cointegrated VECM (12.23) follows Hamilton (1994) and Johansen (1995). For simplicity, assume the absence of deterministic terms.

- Concentrate the likelihood function with respect to the error covariance matrix and short-run dynamics by estimating the regressions

$$\begin{aligned}\Delta\mathbf{Y}_t &= \hat{\mathbf{\Gamma}}_1\Delta\mathbf{Y}_{t-1}+\cdots+\hat{\mathbf{\Gamma}}_{p-1}\Delta\mathbf{Y}_{t-p+1}+\hat{\mathbf{U}}_t\\ \mathbf{Y}_t &= \hat{\mathbf{\Phi}}_1\Delta\mathbf{Y}_{t-1}+\cdots+\hat{\mathbf{\Phi}}_{p-1}\Delta\mathbf{Y}_{t-p+1}+\hat{\mathbf{V}}_t\end{aligned}$$

- Form the sample covariance matrices

$$\mathbf{S}_{00}=\frac{1}{T}\sum_{t=1}^{T}\hat{\mathbf{U}}_t\hat{\mathbf{U}}_t',\ \mathbf{S}_{01}=\frac{1}{T}\sum_{t=1}^{T}\hat{\mathbf{U}}_t\hat{\mathbf{V}}_t',\ \mathbf{S}_{11}=\frac{1}{T}\sum_{t=1}^{T}\hat{\mathbf{V}}_t\hat{\mathbf{V}}_t'$$

- Solve the eigenvalue problem

$$|\lambda\mathbf{S}_{11}-\mathbf{S}_{10}\mathbf{S}_{00}^{-1}\mathbf{S}_{01}|=0$$

giving ordered eigenvalues[6] $\hat{\lambda}_1 > \hat{\lambda}_2 > \cdots > \hat{\lambda}_n$ and associated eigenvectors $\hat{\mathbf{v}}_1, \hat{\mathbf{v}}_2, \ldots, \hat{\mathbf{v}}_n$ that satisfy

$$\begin{aligned} \hat{\lambda}_i \mathbf{S}_{11} \hat{\mathbf{v}}_i &= \mathbf{S}_{10} \mathbf{S}_{00}^{-1} \mathbf{S}_{01}, \ i = 1, \ldots, n \\ \hat{\mathbf{V}} \mathbf{S}_{11} \hat{\mathbf{V}} &= \mathbf{I}_n \end{aligned}$$

where $\hat{\mathbf{V}} = [\hat{\mathbf{v}}_1, \ldots, \hat{\mathbf{v}}_n]$

- The unnormalized MLE for the $(n \times r)$ matrix $\boldsymbol{\beta}$ based on $0 < r < n$ cointegrating vectors is given by the first r eigenvectors

$$\hat{\boldsymbol{\beta}}_{\text{mle}} = (\hat{\mathbf{v}}_1, \ldots, \hat{\mathbf{v}}_r)$$

- Form the normalized estimator $\hat{\boldsymbol{\beta}}_{c,\text{mle}}$ by imposing the appropriate normalizing and identifying restrictions. The MLE for the normalized estimator of α may be computed as

$$\hat{\boldsymbol{\alpha}}_{c,\text{mle}} = \mathbf{S}_{01} \hat{\boldsymbol{\beta}}_{c,\text{mle}}$$

- The maximum likelihood estimators for the remaining parameters may be obtained by multivariate least squares of the VECM with $\boldsymbol{\beta}$ replaced by $\hat{\boldsymbol{\beta}}_{c,\text{mle}}$

$$\Delta \mathbf{Y}_t = \boldsymbol{\alpha}_c \hat{\boldsymbol{\beta}}'_{c,\text{mle}} \mathbf{Y}_{t-1} + \boldsymbol{\Gamma}_1 \Delta \mathbf{Y}_{t-1} + \cdots + \boldsymbol{\Gamma}_{p-1} \Delta \mathbf{Y}_{t-p+1} + \boldsymbol{\varepsilon}_t$$

- The maximized value of the likelihood function based on r cointegrating vectors used in the construction of LR tests for the number of cointegrating vectors is

$$L_{\max}^{-2/T} \propto |\mathbf{S}_{00}| \prod_{i=1}^{r} (1 - \hat{\lambda}_i)$$

- Estimates of the orthogonal complements of $\boldsymbol{\alpha}_c$ and $\boldsymbol{\beta}_c$ are given by

$$\begin{aligned} \hat{\boldsymbol{\alpha}}_{c,\perp} &= \mathbf{S}_{00}^{-1} \mathbf{S}_{11} (\hat{\mathbf{v}}_{r+1}, \ldots, \hat{\mathbf{v}}_n) \\ \hat{\boldsymbol{\beta}}_{c,\perp} &= \mathbf{S}_{11} (\hat{\mathbf{v}}_{r+1}, \ldots, \hat{\mathbf{v}}_n) \end{aligned}$$

Let $\mathbf{c}$ be any $(n \times r)$ matrix such that $\boldsymbol{\beta}'\mathbf{c}$ has full rank. Then $\boldsymbol{\beta}$ may be normalized as

$$\boldsymbol{\beta}_c = \boldsymbol{\beta} (\mathbf{c}'\boldsymbol{\beta})^{-1}$$

[6]These eigenvalues are the squared canonical correlations between $\mathbf{Y}_t$ and $\Delta \mathbf{Y}_t$ corrected for $\Delta \mathbf{Y}_{t-1}, \ldots, \Delta \mathbf{Y}_{t-p+1}$. Johansen (1995) describes how to solve for the eigenvalues.

satisfying $\mathbf{c}'\boldsymbol{\beta}_c = \mathbf{I}_r$ provided $|\mathbf{c}'\boldsymbol{\beta}| \neq \mathbf{0}$. Johansen suggests setting

$$\mathbf{c} = (\mathbf{I}_r \vdots \mathbf{0})' \tag{12.28}$$

This choice of $\mathbf{c}$ corresponds to solving the cointegrating relations $\boldsymbol{\beta}'\mathbf{Y}_t$ for the first r variables. To see this, let $\mathbf{Y}_t = (\mathbf{Y}_{1t}', \mathbf{Y}_{2t}')'$, where $\mathbf{Y}_{1t}$ contains the first r variables and $\mathbf{Y}_{2t}$ contains the remaining $n-r$ variables, and let $\boldsymbol{\beta}' = (-\boldsymbol{\beta}_1 \vdots \boldsymbol{\beta}_2)$, where $\boldsymbol{\beta}_1$ is $(r \times r)$ and $\boldsymbol{\beta}_2$ is $(r \times (n-r))$. Then $\boldsymbol{\beta}'\mathbf{c} = -\boldsymbol{\beta}_1$ and

$$\boldsymbol{\beta}_c = \begin{pmatrix} \mathbf{I}_r \\ -\boldsymbol{\beta}_1^{-1}\boldsymbol{\beta}_2 \end{pmatrix} \tag{12.29}$$

provided $\boldsymbol{\beta}_1$ has full rank r.

Some examples will help clarify the normalization scheme described above. First, suppose there is only one cointegrating vector so that $r = 1$.Let the $(n \times 1)$ vector $\boldsymbol{\beta} = (-\beta_1, \beta_2, \ldots, \beta_n)'$ and define $\mathbf{c} = (1, 0, \ldots, 0)'$ so that $\boldsymbol{\beta}'\mathbf{c} = -\beta_1$ and $\boldsymbol{\beta}_c = (1, -\beta_2/\beta_1, \ldots, -\beta_n/\beta_1)'$ is the normalized cointegrating vector. Notice that this normalization requires $\beta_1 \neq 0$. Next, suppose there are two cointegrating vectors, $r = 2$, and let

$$\begin{aligned} \boldsymbol{\beta}' &= \begin{pmatrix} -\beta_{11} & -\beta_{12} & \beta_{13} & \ldots & \beta_{1n} \\ -\beta_{21} & -\beta_{22} & \beta_{23} & \ldots & \beta_{2n} \end{pmatrix} = \left(-\boldsymbol{\beta}_1 \vdots \boldsymbol{\beta}_2\right) \\ \mathbf{c}' &= \begin{pmatrix} 1 & 0 & 0 & \ldots & 0 \\ 0 & 1 & 0 & \ldots & 0 \end{pmatrix} = (\mathbf{I}_2 \vdots \mathbf{0}) \end{aligned}$$

such that $\boldsymbol{\beta}_1$ has full rank. Then $\boldsymbol{\beta}'\mathbf{c} = -\boldsymbol{\beta}_1$ and

$$\boldsymbol{\beta}_c' = \begin{pmatrix} 1 & 0 & \beta_{13}^* & \ldots & \beta_{1n}^* \\ 0 & 1 & \beta_{23}^* & \ldots & \beta_{2n}^* \end{pmatrix} = (\mathbf{I}_2 \vdots \boldsymbol{\beta}^*)$$

where $\boldsymbol{\beta}^* = -\boldsymbol{\beta}_1^{-1}\boldsymbol{\beta}_2$. The rows of $\boldsymbol{\beta}_c'$ are the normalized cointegrating vectors.

12.7 References

Alexander, C. (2001). *Market Models: A Guide to Financial Data Analysis*, John Wiley & Sons, Chichester, UK.

Cochrane, J. (2001). *Asset Pricing.* Princeton University Press, Princeton, NJ.

Engle, R.F. and C.W.J. Granger (1987). "Co-Integration and Error Correction: Representation, Estimation and Testing," *Econometrica*, 55, 251-276.

Granger, C.W.J. and P.E. Newbold (1974). "Spurious Regression in Econometrics," *Journal of Econometrics*, 2, 111-120.

Hamilton, J.D. (1994). *Time Series Analysis*, Princeton Unversity Press, Princeton, NJ.

Hansen, B.E. (1992). "Efficient Estimation and Testing of Cointegrating Vectors in the Presence of Deterministic Trends," *Journal of Econometrics*, 53, 87-121.

Hayashi, F. (2000). *Econometrics*, Princeton University Press, Princeton, NJ.

Johansen, S. (1988). "Statistical Analysis of Cointegration Vectors," *Journal of Economic Dynamics and Control*, 12, 231-254.

Johansen, S. (1995). *Likelihood Based Inference in Cointegrated Vector Error Correction Models*, Oxford University Press, Oxford.

MacKinnon, J. (1996). "Numerical Distribution Functions for Unit Root and Cointegration Tests," *Journal of Applied Econometrics*, 11, 601-618.

Mills, T. (1999). *The Econometric Analysis of Financial Time Series*, Cambridge University Press, Cambridge.

Osterwald-Lenum, M. (1992). "A Note with Quantiles of the Asymptotic Distribution of the Maximum Likelihood Cointegration Rank Statistics," *Oxford Bulletin of Economics and Statistics*,54, 461-472.

Phillips, P.C.B. (1986). "Understanding Spurious Regression in Econometrics," *Journal of Econometrics*, 33, 311-340.

Phillips, P.C.B. (1991). "Optimal Inference in Cointegrated Systems," *Econometrica*, 59, 283-306.

Phillips, P.C.B. and S. Ouliaris (1990). "Asymptotic Properties of Residual Based Tests for Cointegration," *Econometrica*, 58, 73-93.

Reimars, H.-E. (1992). "Comparisons of Tests for Multivariate Cointegration," *Statistical Papers*, 33, 335-359.

Reinsel, G.C. and S.K. Ahn (1992). "Vector Autoregression Models with Unit Roots and Reduced Rank Structure: Estimation, Likelihood Ratio Test, and Forecasting," *Journal of Time Series Analysis*, 13, 353-375.

Sims, C.A., J.H. Stock and M.W. Watson (1990). "Inference in Linear Time Series Models with Some Unit Roots," *Econometrica*, 58, 113-144.

Stock, J.H. (1987). "Asymptotic Properties of Least Squares Estimation of Cointegrating Vectors," *Econometrica*, 55, 1035-1056.

Stock, J.H. and M.W. Watson (1989). "Variable Trends in Economic Time Series," *Journal of Economic Perspectives*, Vol. 2(3), 147-174.

Stock, J.H. and M.W. Watson (1993). "A Simple Estimator of Cointegrating Vectors in Higher Order Integrated Systems," *Econometrica*, 61, 783-820.

Tsay, R. (2001). *The Analysis of Financial Time Series,* John Wiley & Sons, New York.

Zivot, E. (2000). "Cointegration and Forward and Spot Exchange Rate Regressions," *Journal of International Money and Finance*, 19, 785-812.

13 Multivariate GARCH Modeling

13.1 Introduction

When modeling multivariate economic and financial time series using vector autoregressive (VAR) models, squared residuals often exhibit significant serial correlation. For univariate time series, Chapter 7 indicates that the time series may be conditionally heteroskedastic, and GARCH models have been proved to be very successful at modeling the serial correlation in the second order moment of the underlying time series.

This chapter extends the univariate GARCH models to the multivariate context and shows how multivariate GARCH models can be used to model conditional heteroskedasticity in multivariate time series. In particular, it will focus on modeling and predicting the time varying volatility and volatility co-movement of multivariate time series. The multivariate GARCH models in `S+FinMetrics` are so general that they actually include the vector ARMA (VARMA) model as a special case.

To motivate multivariate GARCH models, Section 13.2 first introduces an exponentially weighted covariance estimate and shows how to estimate the optimal weight using the `mgarch` function in `S+FinMetrics`. Section 13.3 modifies exponentially weighted covariance estimates to obtain the popular diagonal VEC (DVEC) model. Section 13.4 illustrates how to use the `mgarch` function to estimate a multivariate GARCH model such as the DVEC model. Section 13.5 introduces some alternative formulations of multivariate GARCH models. Section 13.6 focuses on how to predict from multivariate GARCH models supported by `S+FinMetrics`. Sec-

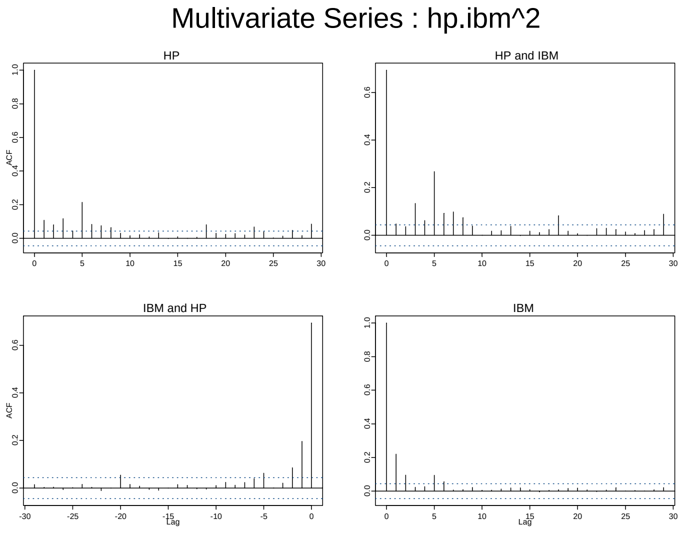


FIGURE 13.1. ACF of multivariate `hp.ibm^2`.

tion 13.7 gives a detailed explanation of the structure of "`garch.model`" and "`mgarch.model`" objects and shows how to use them to fine-tune or constrain a GARCH model, univariate or multivariate. Finally, section 13.8 illustrates how to simulate from selected multivariate GARCH models.

13.2 Exponentially Weighted Covariance Estimate

`S+FinMetrics` module comes with two "`timeSeries`" objects, `hp.s` and `ibm.s`, which represent daily stock returns of Hewlett-Packard and International Business Machine for the same time period. Chapter 7 shows that these financial return series usually exhibit little serial correlation, but squared returns are usually autocorrelated. In multivariate context, cross-correlations of the levels as well as the volatility of the time series are also of interest. Cross-correlation in the levels can be modeled using vector autoregression (VAR) as shown in the previous chapter. This chapter focuses on cross-correlation, or co-movement, of the volatility.

Just as in the univariate context, the existence of cross-correlation can be diagnosed using the `S-PLUS` function `acf`, which also takes a multivariate time series as an argument, to produce both autocorrelation and cross-correlation plots:

```
> hp.ibm = seriesMerge(hp.s, ibm.s)
> tmp = acf(hp.ibm^2)
```

Use the S-PLUS function seriesMerge, which is specifically designed for "timeSeries" objects, to create a multivariate time series. The plot is shown in Figure 13.1. Both the autocorrelation and cross-correlation of the second order moments are significant at least up to lag 5, which indicates that the covariance matrix of hp.ibm may be time varying and serially correlated.

Now let $\mathbf{y}_t$ be a $k \times 1$ vector of multivariate time series:

$$\mathbf{y}_t = \mathbf{c} + \boldsymbol{\epsilon}_t, \text{ for } t = 1, 2, \cdots, T \tag{13.1}$$

where $\mathbf{c}$ is the $k \times 1$ mean vector, and $\boldsymbol{\epsilon}_t$ is $k \times 1$ vector of white noise with zero mean. The sample covariance matrix is given by:

$$\boldsymbol{\Sigma} = \frac{1}{T-1} \sum_{t=1}^{T} (\mathbf{y}_t - \bar{\mathbf{y}})(\mathbf{y}_t - \bar{\mathbf{y}})'$$

where $\bar{\mathbf{y}}$ is the $k \times 1$ vector of sample mean. In the above calculation, the same weight $1/(T-1)$ is applied to the outer product of "demeaned" multivariate time series. To allow for time varying covariance matrix, in practice an *ad hoc* approach uses exponentially decreasing weights as follows:[1]

$$\begin{aligned}\boldsymbol{\Sigma}_t &= \lambda \boldsymbol{\epsilon}_{t-1}\boldsymbol{\epsilon}'_{t-1} + \lambda^2 \boldsymbol{\epsilon}_{t-2}\boldsymbol{\epsilon}'_{t-2} + \cdots \\ &= \sum_{i=1}^{\infty} \lambda^i \boldsymbol{\epsilon}_{t-i}\boldsymbol{\epsilon}'_{t-i}\end{aligned}$$

where $0 < \lambda < 1$ so that smaller weights are placed on observations further back into the past history. Since

$$\lambda + \lambda^2 + \cdots = \frac{\lambda}{1-\lambda}$$

the weights are usually scaled so that they sum up to one:

$$\boldsymbol{\Sigma}_t = (1-\lambda) \sum_{i=1}^{\infty} \lambda^{i-1} \boldsymbol{\epsilon}_{t-i}\boldsymbol{\epsilon}'_{t-i}. \tag{13.2}$$

The above equation can be easily rewritten to obtain the following recursive form for exponentially weighted covariance matrix:

$$\boldsymbol{\Sigma}_t = (1-\lambda)\boldsymbol{\epsilon}_{t-1}\boldsymbol{\epsilon}'_{t-1} + \lambda \boldsymbol{\Sigma}_{t-1} \tag{13.3}$$

which will be referred to as the EWMA model of time varying covariance. From the above equation, given λ and an initial estimate $\boldsymbol{\Sigma}_1$, the time varying exponentially weighted covariance matrices can be computed easily.

[1]This approach has recently been justified and exhaustively investigated by Foster and Nelson (1996), and Andreou and Ghysels (2002). Fleming, Kirby and Ostdiek (2001) applied this method for constructing portfolios.

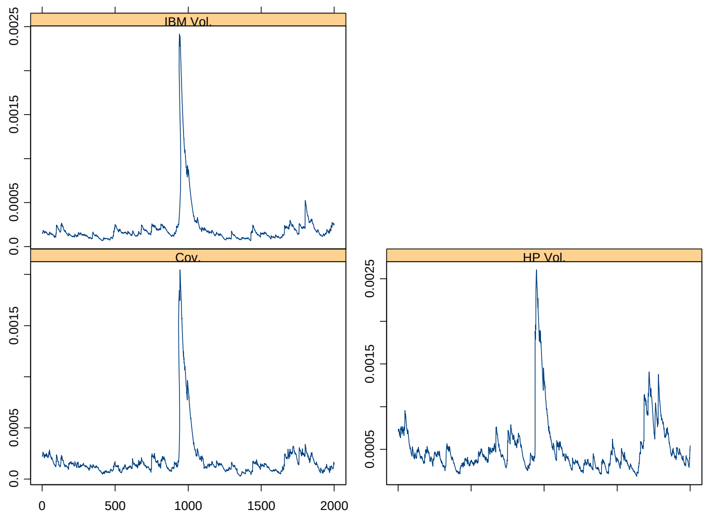


FIGURE 13.2. Exponentially weighted covariance estimate.

The S+FinMetrics function EWMA.cov can be used to compute the exponentially weighted covariance matrix. For example, to obtain the time varying covariance estimate of hp.ibm, use the following command:

```
> hp.ibm.cov = EWMA.cov(hp.ibm, lambda=0.9672375)
> seriesPlot(cbind(hp.ibm.cov[,1,1], hp.ibm.cov[,2,2],
+ hp.ibm.cov[,1,2]), one.plot=F,
+ strip.text=c("HP Vol.", "IBM Vol.", "Cov."))
```

The returned object hp.ibm.cov is an array of dimension $2000 \times 2 \times 2$ representing the time varying covariance matrices, since there are 2000 observations in hp.ibm. Then use the S+FinMetrics function seriesPlot to obtain a Trellis multivariate plot of the time varying covariance matrix as shown in Figure 13.2, where the large spikes in the middle correspond to the 1987 stock market crash.

In practice, the value of λ is usually chosen in an *ad hoc* way as typified by the RiskMetrics proposal. However, if one assumes that $\boldsymbol{\epsilon}_t$ in (13.1) follows a multivariate normal distribution with zero mean, and $\boldsymbol{\Sigma}_t = \text{Cov}_{t-1}(\boldsymbol{\epsilon}_t)$ is treated as the covariance of $\boldsymbol{\epsilon}_t$ conditional on the past history, then the log-likelihood function of the observed time series can be written as:

$$\log L = -\frac{kT}{2}\log(2\pi) - \frac{1}{2}\sum_{t=1}^{T}|\boldsymbol{\Sigma}_t| - \frac{1}{2}\sum_{t=1}^{T}(\mathbf{y}_t - \mathbf{c})'\boldsymbol{\Sigma}_t^{-1}(\mathbf{y}_t - \mathbf{c}). \quad (13.4)$$

Since $\mathbf{\Sigma}_t$ can be recursively calculated as in (13.3), the log-likelihood function can also be easily evaluated. Thus the mean vector $\mathbf{c}$ and λ can be treated as unknown model parameters and estimated using quasi-maximum likelihood estimation (MLE), given the initial value $\mathbf{\Sigma}_1$.

The `mgarch` function in `S+FinMetrics` actually allows the estimation of the above EWMA model using either (13.3) or an exact form of (13.2) with limited past history. The syntax of `mgarch` is very much similar to that of `garch` function. For example, to estimate the EWMA model as in (13.3), use the following command:

```
> hp.ibm.ewma = mgarch(hp.ibm~1, ~ewma1, trace=F)
> hp.ibm.ewma

Call:
mgarch(formula.mean = hp.ibm ~ 1, formula.var =  ~ ewma1,
       trace = F)

Mean Equation: hp.ibm ~ 1

Conditional Variance Equation:  ~ ewma1

Coefficients:

 C(1) 0.0005202
 C(2) 0.0004732
ALPHA 0.0327625
```

where the conditional variance formula is specified by `~ewma1`. In the output, `C(1)` and `C(2)` correspond to the 2×1 vector of $\mathbf{c}$ in (13.1), and `ALPHA` corresponds to $1 - \lambda$ in (13.3). This is why `lambda=0.9672375` is set in the earlier `EWMA.cov` example.

The EWMA model with an exact form of (13.2) can also be estimated by specifying `~ewma2` as the conditional variance formula. However, in that case, the coefficient labeled by `ALPHA` actually corresponds to λ in (13.2):

```
> mgarch(hp.ibm~1, ~ewma2, trace=F)

Call:
mgarch(formula.mean = hp.ibm ~ 1, formula.var =  ~ ewma2,
       trace = F)

Mean Equation: hp.ibm ~ 1

Conditional Variance Equation:  ~ ewma2

Coefficients:
 C(1) 0.0007369
 C(2) 0.0002603
ALPHA 0.9730018
```

13.3 Diagonal VEC Model

In the univariate context, the EWMA model introduced in the previous section reduces to:

$$\Sigma_t = (1-\lambda)\epsilon_{t-1}^2 + \lambda\Sigma_{t-1}$$

which is simply a GARCH$(1,1)$ model with $a_1 = 1-\lambda$, $b_1 = \lambda$ and thus $a_1 + b_1 = 1$. Since $a_1 + b_1$ corresponds to the AR(1) coefficient in the ARMA representation of GARCH models (see Section 22.16 in Chapter 7), the condition $a_1 + b_1 = 1$ implies that the GARCH model is not stationary in the weak sense.[2] Engle and Bollerslev (1986) termed this model the integrated GARCH (IGARCH) model in the univariate context.[3] Given the non-stationarity of IGARCH and EWMA models, they are sometimes not favored for modeling volatility.

To preserve the intuition behind EWMA models while allowing for a flexible and stationary model for time varying covariance, generalize the EWMA model as follows:

$$\boldsymbol{\Sigma}_t = \mathbf{A}_0 + \sum_{i=1}^{p} \mathbf{A}_i \otimes (\boldsymbol{\epsilon}_{t-i}\boldsymbol{\epsilon}_{t-i}') + \sum_{j=1}^{q} \mathbf{B}_j \otimes \boldsymbol{\Sigma}_{t-j} \tag{13.5}$$

where the symbol $\otimes$ stands for Hadamard product, i.e., element-by-element multiplication, and all the coefficient matrices have dimension $k \times k$. This model is first proposed by Bollerslev, Engle and Wooldridge (1988), and they called it the diagonal VEC, or DVEC(p,q) model.

[2]Unlike the unit root time series, a GARCH model may be strongly stationary, even when it is not weakly stationary. See Nelson (1990) and Bougerol and Picard (1992) for technical proof.

[3]In fact, the `mgarch` function can be called with a univariate time series using `~ewma1` as the conditional variance formula to estimate such an IGARCH model.

To appreciate the intuition behind DVEC model, consider the bivariate DVEC(1, 1) model:

$$\begin{aligned}\begin{bmatrix} \mathbf{\Sigma}_t^{(11)} & \\ \mathbf{\Sigma}_t^{(21)} & \mathbf{\Sigma}_t^{(22)} \end{bmatrix} &= \begin{bmatrix} \mathbf{A}_0^{(11)} & \\ \mathbf{A}_0^{(21)} & \mathbf{A}_0^{(22)} \end{bmatrix} \\ &+ \begin{bmatrix} \mathbf{A}_1^{(11)} & \\ \mathbf{A}_1^{(21)} & \mathbf{A}_1^{(22)} \end{bmatrix} \begin{bmatrix} \varepsilon_{t-1}^{(1)}\varepsilon_{t-1}^{(1)} & \\ \varepsilon_{t-1}^{(2)}\varepsilon_{t-1}^{(1)} & \varepsilon_{t-1}^{(2)}\varepsilon_{t-1}^{(2)} \end{bmatrix} \\ &+ \begin{bmatrix} \mathbf{B}_1^{(11)} & \\ \mathbf{B}_1^{(21)} & \mathbf{B}_1^{(22)} \end{bmatrix} \begin{bmatrix} \mathbf{\Sigma}_{t-1}^{(11)} & \\ \mathbf{\Sigma}_{t-1}^{(21)} & \mathbf{\Sigma}_{t-1}^{(22)} \end{bmatrix}\end{aligned}$$

where only the lower triangular part of the system is considered, with $\mathbf{X}^{(ij)}$ denoting the (i, j)-th element of matrix $\mathbf{X}$, and $\boldsymbol{\epsilon}^{(i)}$ the i-th element of vector $\boldsymbol{\epsilon}$. The above matrix notation can be rewritten as follows:[4]

$$\begin{aligned}\mathbf{\Sigma}_t^{(11)} &= \mathbf{A}_0^{(11)} + \mathbf{A}_1^{(11)}\boldsymbol{\epsilon}_{t-1}^{(1)}\boldsymbol{\epsilon}_{t-1}^{(1)} + \mathbf{B}_1^{(11)}\mathbf{\Sigma}_{t-1}^{(11)} \\ \mathbf{\Sigma}_t^{(21)} &= \mathbf{A}_0^{(21)} + \mathbf{A}_1^{(21)}\boldsymbol{\epsilon}_{t-1}^{(2)}\boldsymbol{\epsilon}_{t-1}^{(1)} + \mathbf{B}_1^{(21)}\mathbf{\Sigma}_{t-1}^{(21)} \\ \mathbf{\Sigma}_t^{(22)} &= \mathbf{A}_0^{(22)} + \mathbf{A}_1^{(22)}\boldsymbol{\epsilon}_{t-1}^{(2)}\boldsymbol{\epsilon}_{t-1}^{(2)} + \mathbf{B}_1^{(22)}\mathbf{\Sigma}_{t-1}^{(22)}\end{aligned}$$

so the (i, j)-th element of the time varying covariance matrix only depends on its own lagged element and the corresponding cross-product of errors. As a result, the volatility of each series follows a GARCH process, while the covariance process can also be treated as a GARCH model in terms of the cross-moment of the errors.

Since a covariance matrix must be symmetric, in practice it suffices to treat $\mathbf{\Sigma}_t$ as symmetric and only consider the lower triangular part of the system. A covariance matrix must be also positive semi-definite (PSD). However, $\mathbf{\Sigma}_t$ in the DVEC model cannot be guaranteed to be PSD, which is considered a weakness of the DVEC model. Section 13.5 will introduce other formulations of multivariate GARCH models that guarantee the time varying covariance matrix to be PSD.

13.4 Multivariate GARCH Modeling in `S+FinMetrics`

13.4.1 Multivariate GARCH Model Estimation

Section 13.2 showed that the `mgarch` function in `S+FinMetrics` can be used to estimate a multivariate GARCH model such as the EWMA model. It

[4]If these equations are written using matrix notation with a vector on the left hand side, then the coefficient matrices become diagonal matrices; thus this model is referred to as the diagonal VEC model.

can also be used to fit other types of multivariate GARCH models such as the DVEC model by using a different conditional variance formula. For example, to fit a DVEC$(1, 1)$ model to the bivariate time series `hp.ibm`, use the following command:

```
> hp.ibm.dvec = mgarch(hp.ibm~1, ~dvec(1,1), trace=F)
> class(hp.ibm.dvec)
[1] "mgarch"
> hp.ibm.dvec

Call:
mgarch(formula.mean = hp.ibm ~ 1, formula.var =  ~ dvec(1, 1),
       trace = F)

Mean Equation: hp.ibm ~ 1

Conditional Variance Equation:  ~ dvec(1, 1)

Coefficients:

          C(1) 7.018e-04
          C(2) 2.932e-04
       A(1, 1) 3.889e-05
       A(2, 1) 1.322e-05
       A(2, 2) 2.877e-05
 ARCH(1; 1, 1) 6.226e-02
 ARCH(1; 2, 1) 3.394e-02
 ARCH(1; 2, 2) 1.049e-01
GARCH(1; 1, 1) 8.568e-01
GARCH(1; 2, 1) 8.783e-01
GARCH(1; 2, 2) 7.421e-01
```

The returned object is of class "`mgarch`". Similar to "`garch`" objects, the `print` method shows the conditional mean equation, conditional variance equation, together with the estimated model coefficients. In the output, `C(i)` corresponds to the i-th element of $\mathbf{c}$ in (13.1), while `A(i,j)` corresponds to the (i, j)-th element of $\mathbf{A}_0$, `ARCH(i;j,k)` corresponds to the (j, k)-th element of $\mathbf{A}_i$, and `GARCH(j;i,k)` corresponds to the (i, k)-th element of $\mathbf{B}_j$ in (13.5).

As usual, use the `S-PLUS` function `names` to find out the component names of an "`mgarch`" object:

```
> names(hp.ibm.dvec)
 [1] "residuals"   "sigma.t"       "df.residual"  "coef"
 [5] "model"       "cond.dist"     "likelihood"   "opt.index"
 [9] "cov"         "std.residuals" "R.t"          "S.t"
```

```
[13] "prediction"  "call"          "series"
```

These components are similar to those of "garch" objects, and the on-line help file for mgarch provides details for them. For most components that a user is interested in, S+FinMetrics provides methods for generic functions such as coef, residuals, and vcov for extracting those components. For example, extract the estimated coefficients by calling the generic coef function:

```
> coef(hp.ibm.dvec)

          C(1) 7.017567e-04
          C(2) 2.932253e-04
       A(1, 1) 3.888696e-05
       A(2, 1) 1.322108e-05
       A(2, 2) 2.876733e-05
 ARCH(1; 1, 1) 6.225657e-02
 ARCH(1; 2, 1) 3.393546e-02
 ARCH(1; 2, 2) 1.048581e-01
GARCH(1; 1, 1) 8.567934e-01
GARCH(1; 2, 1) 8.783100e-01
GARCH(1; 2, 2) 7.421328e-01
```

Note that since only the lower triangular part of the system is considered for DVEC models, only that part of the coefficient matrices are shown here.

Similarly, call the generic vcov function to obtain the covariance matrix of the estimated coefficients. By default, the covariance matrix based on the outer product of gradients is returned. Just like in the univariate case, the covariance matrix based on the inverse of numerical Hessian and the robust covariance matrix can be obtained by setting the optional argument method to "op" and "qmle", respectively. For example, to obtain the robust standard error of the estimated coefficients, use the command:

```
> sqrt(diag(vcov(hp.ibm.dvec, method="qmle")))
 [1] 0.00048803101 0.00030789132 0.00003531643 0.00001088806
 [5] 0.00001685943 0.03070917257 0.02983075055 0.06630322823
 [9] 0.10170075535 0.09285527451 0.13273539264
```

Similar to the method functions for "garch" objects, residuals and sigma.t can be used to extract the model residuals and estimated volatility, respectively. If the original multivariate data is a "timeSeries" object, the extracted model residuals and conditional volatility will also be "timeSeries" objects with the same dimension. Note that in the multivariate case, the standardized residuals are computed as $\mathbf{\Sigma}_t^{-1/2}\boldsymbol{\epsilon}_t$, where $\mathbf{\Sigma}_t^{1/2}$ is the Cholesky factor of $\mathbf{\Sigma}_t$. To obtain the standardized residuals, set the optional argument standardize=T when calling the residuals function:

```
> residuals(hp.ibm.dvec, standardize=T)
```

The `sigma.t` function only extracts the conditional standard deviation of each series, and ignores the conditional covariance term. To obtain the conditional covariance or conditional correlation term, extract the `S.t` and `R.t` component, respectively. Both `S.t` and `R.t` are three dimensional arrays with dimension $T \times k \times k$.

13.4.2 Multivariate GARCH Model Diagnostics

The previous subsection showed how to estimate a multivariate GARCH model in `S+FinMetrics`, and how to extract various components of the fitted model. To assess the model fit, `S+FinMetrics` provides method functions for two generic functions: `summary` and `plot`, one for statistical summary and the other for visual diagnostics of the model fit.

For example, to obtain more detailed summary of `hp.ibm.dvec`, call the generic `summary` function:

```
> summary(hp.ibm.dvec)

Call:
mgarch(formula.mean = hp.ibm ~ 1, formula.var =  ~ dvec(1, 1),
       trace = F)

Mean Equation: hp.ibm ~ 1

Conditional Variance Equation:  ~ dvec(1, 1)

Conditional Distribution:  gaussian

------------------------------------------------------------

Estimated Coefficients:
------------------------------------------------------------
                   Value Std.Error t value  Pr(>|t|)
          C(1) 7.018e-04 4.630e-04   1.516 6.489e-02
          C(2) 2.932e-04 2.870e-04   1.022 1.536e-01
       A(1, 1) 3.889e-05 6.175e-06   6.297 1.860e-10
       A(2, 1) 1.322e-05 2.461e-06   5.372 4.345e-08
       A(2, 2) 2.877e-05 4.302e-06   6.687 1.469e-11
 ARCH(1; 1, 1) 6.226e-02 8.690e-03   7.164 5.498e-13
 ARCH(1; 2, 1) 3.394e-02 6.848e-03   4.955 3.916e-07
 ARCH(1; 2, 2) 1.049e-01 9.212e-03  11.382 0.000e+00
GARCH(1; 1, 1) 8.568e-01 1.762e-02  48.625 0.000e+00
GARCH(1; 2, 1) 8.783e-01 1.885e-02  46.589 0.000e+00
GARCH(1; 2, 2) 7.421e-01 2.966e-02  25.019 0.000e+00
```

```
--------------------------------------------------------------

AIC(11) = -21886.25
BIC(11) = -21824.64

Normality Test:
--------------------------------------------------------------
    Jarque-Bera P-value Shapiro-Wilk P-value
 HP       755.8       0       0.9891  0.7105
IBM      2606.3       0       0.9697  0.0000

Ljung-Box test for standardized residuals:
--------------------------------------------------------------
    Statistic P-value Chi^2-d.f.
 HP     18.57 0.09952         12
IBM     11.76 0.46511         12

Ljung-Box test for squared standardized residuals:
--------------------------------------------------------------
    Statistic P-value Chi^2-d.f.
 HP     11.43  0.4925         12
IBM      4.44  0.9741         12

Lagrange multiplier test:
--------------------------------------------------------------
      Lag 1   Lag 2   Lag 3  Lag 4  Lag 5   Lag 6   Lag 7
 HP -0.1990  0.2496 -0.7004  2.594 0.1039 -0.1167 -0.2286
IBM -0.7769 -0.9883 -0.5770 -1.198 0.4664 -0.2077 -0.4439

       Lag 8   Lag 9  Lag 10  Lag 11   Lag 12       C
 HP  0.09018 -0.7877 -0.1279 -0.9280 -0.03133  1.8549
IBM -0.26423 -0.5352 -0.6724  0.1852  0.02102 -0.0729

      TR^2 P-value F-stat P-value
 HP 11.914  0.4526  1.090  0.4779
IBM  4.522  0.9721  0.412  0.9947
```

By default, the `summary` method shows the standard errors and p-values of estimated coefficients, together with various tests on the standardized residuals for assessing the model fit. The standard errors and p-values are computed using the default covariance estimate. To use robust or numerical Hessian based standard errors to compute the p-values, the `summary` method takes an optional argument `method` just like the `vcov` method does.

All the tests performed on the standardized residuals can also be performed independently by using standard `S+FinMetrics` functions. In general, if the model is successful at modeling the serial correlation in the time series and the time varying aspect of covariance matrix, there should be no serial correlation left in both the first order and second order moments of standardized residuals. For example, to check that there is no serial correlation left in squared standardized residuals, use the following command:

```
> autocorTest(residuals(hp.ibm.dvec, standardize=T)^2, lag=12)

Test for Autocorrelation: Ljung-Box

Null Hypothesis: no autocorrelation

Test Statistics:
               HP     IBM
Test Stat 11.4299  4.4404
  p.value  0.4925  0.9741

Dist. under Null: chi-square with 12 degrees of freedom
   Total Observ.: 2000
```

which is the same as the test results returned by the `summary` method. Since the p-values for both series are much greater than the conventional 5% level, the null hypothesis that there is no autocorrelation left cannot be rejected.

Similarly, the LM test for ARCH effects can be performed on the multivariate standardized residuals:

```
> archTest(residuals(hp.ibm.dvec, standardize=T), lag=12)

Test for ARCH Effects: LM Test

Null Hypothesis: no ARCH effects

Test Statistics:
               HP     IBM
Test Stat 11.9136  4.5219
  p.value  0.4526  0.9721

Dist. under Null: chi-square with 12 degrees of freedom
   Total Observ.: 2000
```

which is also the same as the LM test returned by the `summary` method. The p-values for LM tests are very close to those of the autocorrelation tests, which confirms that the DVEC model is very successful at modeling the time varying aspect of covariance matrix.

Note the above tests are applied to each series separately, and they do not check the serial correlation of the cross-moment. Hence those tests are not really multivariate tests. However, the autocorTest function does have an option to produce a *multivariate portmanteau test* as proposed by Hosking (1980), which is a multivariate extension of the univariate Ljung-Box test. For example, to produce the multivariate test of squared standardized residuals, use the command:

```
> autocorTest(residuals(hp.ibm.dvec, standardize=T)^2,
+ lag=12, bycol=F)

Multivariate Portmanteau Test: Ljung-Box Type

Null Hypothesis: no serial correlation

Test Statistics:

Test Stat 42.4585
  p.value  0.6985

Dist. under Null: chi-square with 48 degrees of freedom
   Total Observ.: 2000
```

where the optional argument bycol is set to FALSE to use the Hosking's test. The autocorTest function sets bycol to TRUE by default, and thus tests the multivariate series column by column.

The goodness-of-fit of a multivariate GARCH model can also be assessed by calling the generic plot function on a fitted "mgarch" object. For example:

```
> plot(hp.ibm.dvec)

Make a plot selection (or 0 to exit):

1: plot: All
2: plot: Original Observations
3: plot: ACF of Observations
4: plot: ACF of Squared Observations
5: plot: Residuals
6: plot: Conditional SD
7: plot: Standardized Residuals
8: plot: ACF of Standardized Residuals
9: plot: ACF of Squared Standardized Residuals
10: plot: QQ-Plots of Standardized Residuals
Selection:
```

By selecting 9 the ACF of squared standardized residuals can be obtained, which is shown in Figure 13.3. After fitting the DVEC model, there is essentially little serial correlation left in the second order moments of

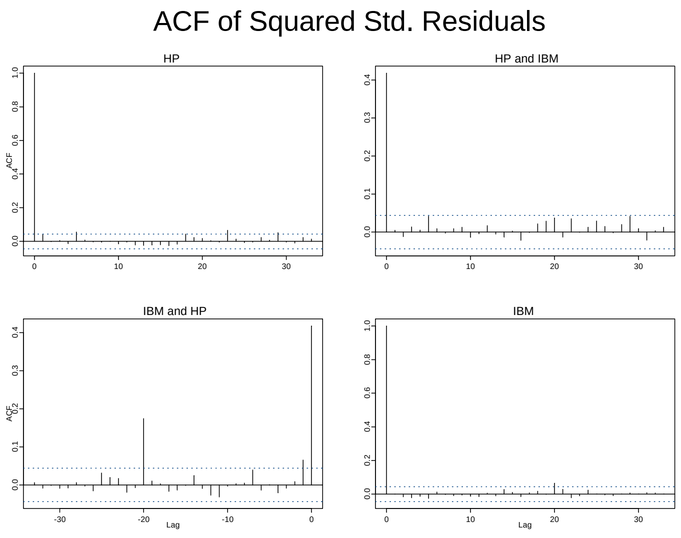


FIGURE 13.3. ACF of squared standardized residuals.

the residuals. Normal QQ-plot of standardized residuals can be obtained by selecting `10`, which is shown in Figure 13.4. There is significant deviation in the tails from the normal QQ-line for both residuals, which is also confirmed by the normality tests in the `summary` output shown earlier. Thus it seems that the normality assumption for the residuals may not be appropriate. Section 13.5.5 will show how to use alternative distributions in multivariate GARCH models.

Other plots can also be chosen to visualize the model fit. For example, choosing `6` plots the estimated conditional standard deviation as shown in Figure 13.5. For the bivariate time series `hp.ibm`, the time varying cross-correlation, which is contained in the `R.t` component of the fitted object, is also of interest. Since `R.t` is a three-dimensional array, use the following command to generate a time series of the conditional cross-correlation:

```
> hp.ibm.cross = hp.ibm.dvec$R.t[,1,2]
> hp.ibm.cross = timeSeries(hp.ibm.cross,pos=positions(hp.ibm))
> seriesPlot(hp.ibm.cross, strip="Conditional Cross Corr.")
```

The plot is shown in Figure 13.6. Although the conditional cross correlation between `hp.s` and `ibm.s` usually fluctuates around 0.5, it can suddenly drop down to 0.3 and then go back to 0.5 very quickly.

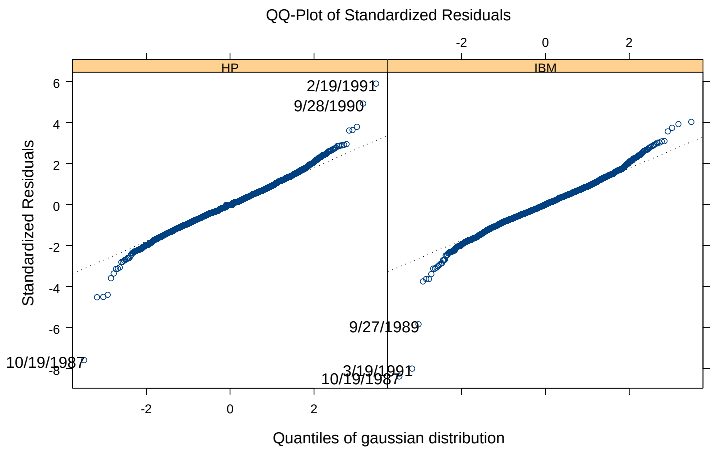


FIGURE 13.4. QQ-plot of standardized residuals.

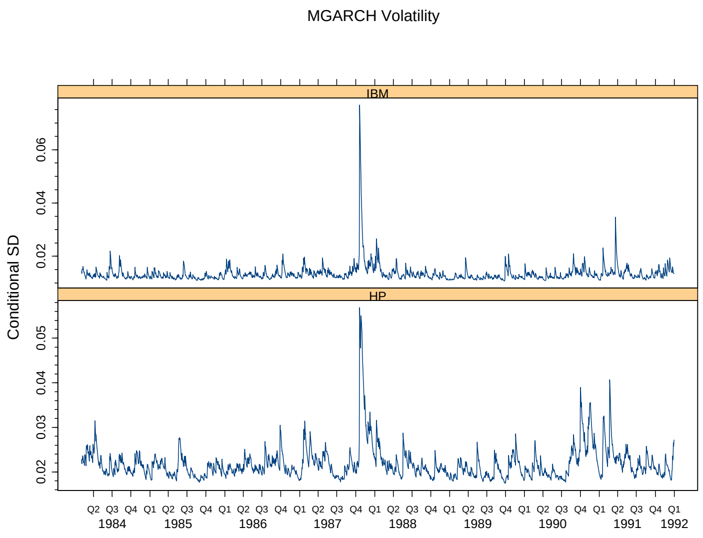


FIGURE 13.5. Multivariate conditional volatility: hp.ibm.

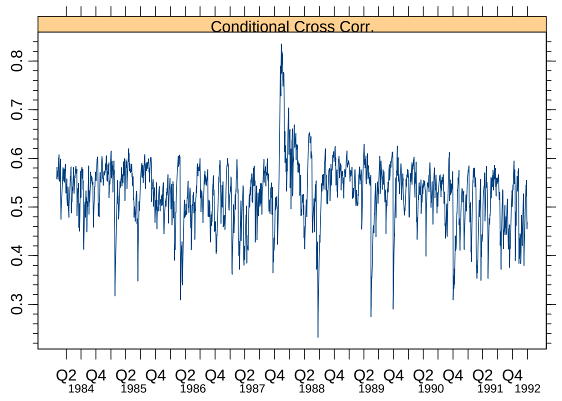


FIGURE 13.6. Conditional cross correlation: `hp.ibm`.

13.5 Multivariate GARCH Model Extensions

13.5.1 Matrix-Diagonal Models

Although the DVEC model provided a good model fit for the bivariate time series `hp.ibm`, the time varying covariance matrices are not guaranteed to be PSD given the formulation as in (13.5). Note that a sufficient condition for $\mathbf{\Sigma}_t$ to be PSD is that $\mathbf{A}_0$, $\mathbf{A}_i$ (for $i = 1, \cdots, p$) and $\mathbf{B}_j$ (for $j = 1, \cdots, q$) are all PSD. Based on this observation, Ding (1994) and Bollerslev, Engle and Nelson (1994) proposed to estimate the Cholesky factors of the coefficient matrices:

$$\mathbf{\Sigma}_t = \mathbf{A}_0\mathbf{A}_0' + \sum_{i=1}^{p}(\mathbf{A}_i\mathbf{A}_i') \otimes (\boldsymbol{\epsilon}_{t-i}\boldsymbol{\epsilon}_{t-i}') + \sum_{j=1}^{q}(\mathbf{B}_j\mathbf{B}_j') \otimes \mathbf{\Sigma}_{t-j} \tag{13.6}$$

where $\mathbf{A}_0$, $\mathbf{A}_i$ (for $i = 1, \cdots, p$) and $\mathbf{B}_j$ (for $j = 1, \cdots, q$) are all lower triangular matrices. This model will be referred to as the *matrix-diagonal* model.

The matrix-diagonal models can be further simplified by restricting $\mathbf{A}_i$ and $\mathbf{B}_j$ to be a vector, which results in:

$$\mathbf{\Sigma}_t = \mathbf{A}_0\mathbf{A}_0' + \sum_{i=1}^{p}(\mathbf{a}_i\mathbf{a}_i') \otimes (\boldsymbol{\epsilon}_{t-i}\boldsymbol{\epsilon}_{t-i}') + \sum_{j=1}^{q}(\mathbf{b}_j\mathbf{b}_j') \otimes \mathbf{\Sigma}_{t-j} \tag{13.7}$$

where $\mathbf{a}_i$ and $\mathbf{b}_j$ are $k \times 1$ vectors. Even simpler, use the following formulation:

$$\mathbf{\Sigma}_t = \mathbf{A}_0\mathbf{A}_0' + \sum_{i=1}^{p} a_i \otimes (\boldsymbol{\epsilon}_{t-i}\boldsymbol{\epsilon}_{t-i}') + \sum_{j=1}^{q} b_j \otimes \mathbf{\Sigma}_{t-j} \tag{13.8}$$

where a_i and b_j are positive scalars. It is easy to show that all the formulations given in (13.6), (13.7), and (13.8) guarantee that the time varying covariance matrix $\mathbf{\Sigma}_t$ is PSD. However, the simpler the model is, the more stringent restrictions are placed on the dynamics of the model.

The `mgarch` function in `S+FinMetrics` allows the estimation of all the above modifications of the DVEC model by using `~dvec.type.type(p,q)` as the conditional variance formula, where `type` can be `mat` for the (13.6) formulation, `vec` for the (13.7) formulation, or `scalar` for the (13.8) formulation, and the first `type` refers to the type of $\mathbf{A}_i$, the second `type` refers to the type of $\mathbf{B}_j$. Hence, one can use `mgarch` to estimate a multivariate GARCH model with different formulations for $\mathbf{A}_i$ and $\mathbf{B}_j$. For example, to estimate a multivariate GARCH model with the following covariance matrix formulation:

$$\mathbf{\Sigma}_t = \mathbf{A}_0\mathbf{A}_0' + \mathbf{A}_1\mathbf{A}_1' \otimes (\boldsymbol{\epsilon}_{t-i}\boldsymbol{\epsilon}_{t-i}') + b_1 \otimes \mathbf{\Sigma}_{t-j} \tag{13.9}$$

with $\mathbf{A}_0$ and $\mathbf{A}_1$ being lower triangular matrices and b_1 just a scalar, use the following conditional variance formula:

```
> mgarch(hp.ibm~1, ~dvec.mat.scalar(1,1), trace=F)

Call:
mgarch(formula.mean = hp.ibm ~ 1, formula.var =
       ~ dvec.mat.scalar(1, 1), trace = F)

Mean Equation: hp.ibm ~ 1

Conditional Variance Equation:  ~ dvec.mat.scalar(1, 1)

Coefficients:

          C(1) 0.0007500
          C(2) 0.0003268
       A(1, 1) 0.0099384
       A(2, 1) 0.0037295
       A(2, 2) 0.0044583
ARCH(1; 1, 1) 0.3215890
ARCH(1; 2, 1) 0.1984259
ARCH(1; 2, 2) 0.2958904
      GARCH(1) 0.6968114
```

Note that in the output the `GARCH(1)` coefficient corresponds to b_1, while `ARCH(1;i,j)` corresponds to the (i,j)-th element of $\mathbf{A}_1$ in (13.9).

13.5.2 BEKK Models

Although the DVEC model can be modified in various ways to ensure the time varying covariance matrices are PSD, the dynamics allowed in the conditional covariance matrix are still somewhat restricted. In particular, the conditional variance and covariance are only dependent on their own lagged element and the corresponding cross-product of shocks or error terms. For example, consider the bivariate time series `hp.ibm`. If there is a shock to `hp.s` in the current period, it will affect the conditional volatility of `hp.s` and the conditional correlation between `hp.s` and `ibm.s` in the next period. However, it will not directly affect the volatility of `ibm.s`.

The BEKK model, as formalized by Engle and Kroner (1995), provides an alternative formulation of the conditional variance equation:

$$\mathbf{\Sigma}_t = \mathbf{A}_0\mathbf{A}_0' + \sum_{i=1}^{p} \mathbf{A}_i(\boldsymbol{\epsilon}_{t-i}\boldsymbol{\epsilon}_{t-i}')\mathbf{A}_i' + \sum_{j=1}^{q} \mathbf{B}_j\mathbf{\Sigma}_{t-j}\mathbf{B}_j'$$

where $\mathbf{A}_0$ is a lower triangular matrix, but $\mathbf{A}_i$ $(i = 1, \cdots, p)$ and $\mathbf{B}_j$ $(j = 1, \cdots, q)$ are unrestricted square matrices. It is easy to show that $\mathbf{\Sigma}_t$ is guaranteed to be symmetric and PSD in the above formulation. Furthermore, the dynamics allowed in the BEKK model are richer than the DVEC model, which can be illustrated by considering the $(2,2)$ element of $\mathbf{\Sigma}_t$ in the BEKK$(1,1)$ model:

$$\begin{aligned}\mathbf{\Sigma}_t^{(22)} = \mathbf{A}_0^{(22)}\mathbf{A}_0^{(22)} &+ [\mathbf{A}_1^{(21)}\boldsymbol{\epsilon}_{t-1}^{(1)} + \mathbf{A}_1^{(22)}\boldsymbol{\epsilon}_{t-1}^{(2)}]^2 + \\ &[\mathbf{B}_1^{(21)}\mathbf{B}_1^{(21)}\mathbf{\Sigma}_{t-1}^{(11)} + 2\mathbf{B}_1^{(21)}\mathbf{B}_1^{(22)}\mathbf{\Sigma}_{t-1}^{(21)} + \mathbf{B}_1^{(22)}\mathbf{B}_1^{(22)}\mathbf{\Sigma}_{t-1}^{(22)}]\end{aligned}$$

where both $\boldsymbol{\epsilon}_{t-1}^{(1)}$ and $\boldsymbol{\epsilon}_{t-1}^{(2)}$ enter the equation. In addition, $\mathbf{\Sigma}_{t-1}^{(11)}$, the volatility of the first series, also has direct impacts on $\mathbf{\Sigma}_t^{(22)}$, the volatility of the second series. However, for the bivariate BEKK$(1,1)$ model, flexibility is achieved at the cost of two extra parameters, i.e., $\mathbf{A}_1^{(12)}$ and $\mathbf{B}_1^{(12)}$, which are not needed for the DVEC$(1,1)$ model. In general, a BEKK(p,q) model requires $k(k-1)(p+q)/2$ more parameters than a DVEC model of the same order.

One can fit a BEKK model by using `~bekk(p,q)` as the conditional variance formula. For example, to fit a BEKK$(1,1)$ model to the bivariate time series `hp.ibm`, use the following command:

```
> hp.ibm.bekk = mgarch(hp.ibm~1, ~bekk(1,1))
> hp.ibm.bekk
```

```
Call:
mgarch(formula.mean = hp.ibm ~ 1, formula.var =  ~ bekk(1, 1))

Mean Equation: hp.ibm ~ 1

Conditional Variance Equation:  ~ bekk(1, 1)

Coefficients:

          C(1)  0.0007782
          C(2)  0.0002870
       A(1, 1)  0.0077678
       A(2, 1) -0.0035790
       A(2, 2)  0.0046844
 ARCH(1; 1, 1)  0.2054901
 ARCH(1; 2, 1) -0.0287318
 ARCH(1; 1, 2) -0.0734735
 ARCH(1; 2, 2)  0.4169672
GARCH(1; 1, 1)  0.8078184
GARCH(1; 2, 1)  0.1277266
GARCH(1; 1, 2)  0.2867068
GARCH(1; 2, 2)  0.6954790
```

Note that in the output, the coefficient matrix $\mathbf{A}_1$ (the `ARCH(1;i,j)` coefficients) and $\mathbf{B}_1$ (the `GARCH(1;i,j)`) are not restricted.

Compare the conditional correlations between `hp.s` and `ibm.s` implied by the DVEC model and BEKK model as follows:

```
> seriesPlot(cbind(hp.ibm.dvec$R.t[,1,2],
+ hp.ibm.bekk$R.t[,1,2]),strip=c("DVEC Corr.","BEKK Corr."),
+ one.plot=F, layout=c(1,2,1))
```

The plot is shown in Figure 13.7, from which one can see that the conditional correlation implied by the BEKK model is more volatile than that implied by the DVEC model.

13.5.3 Univariate GARCH-based Models

For BEKK model, DVEC model and its modifications, the conditional covariance matrix is modeled directly. This approach can result in a large number of parameters since the covariance terms need to be modeled separately. Another approach in multivariate GARCH modeling is to transform the multivariate time series into uncorrelated time series and then apply the univariate GARCH models in Chapter 7 to each of those uncorrelated series. This subsection introduces three types of multivariate GARCH models in this fashion.

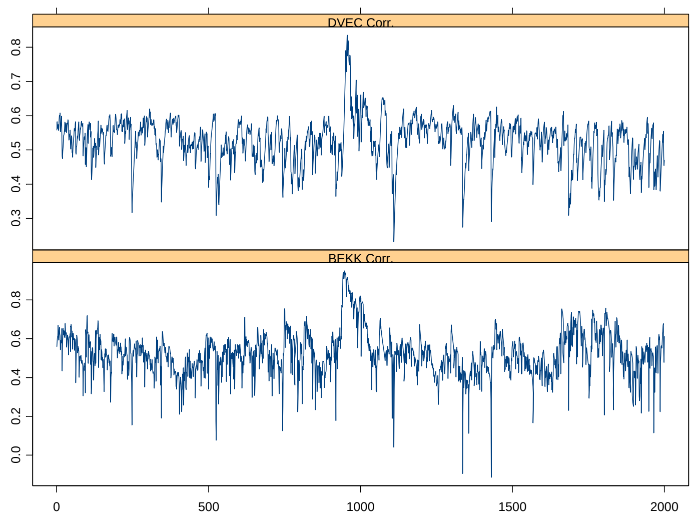


FIGURE 13.7. Comparison of conditional correlation: `hp.ibm`.

Constant Conditional Correlation Model

In general, a $k \times k$ covariance matrix $\mathbf{\Sigma}$ can be decomposed into the following form:

$$\mathbf{\Sigma} = \mathbf{\Delta R \Delta}$$

where $\mathbf{R}$ is the correlation matrix, $\mathbf{\Delta}$ is a diagonal matrix with the vector $(\sigma_1, \cdots, \sigma_k)$ on the diagonal, and σ_i is the standard deviation of the i-th series. Based on the observation that the correlation matrix of foreign exchange rate returns is usually constant over time, Bollerslev (1990) suggested modelling the time varying covariance matrix as follows:

$$\mathbf{\Sigma}_t = \mathbf{\Delta}_t \mathbf{R} \mathbf{\Delta}_t$$

where $\mathbf{R}$ is the constant conditional correlation matrix, and $\mathbf{\Delta}_t$ is the following diagonal matrix:

$$\mathbf{\Delta}_t = \begin{bmatrix} \sigma_{1t} & & \\ & \ddots & \\ & & \sigma_{kt} \end{bmatrix}$$

with σ_{it} following any univariate GARCH process, for $i = 1, \cdots, k$. This model is usually referred to as the *constant conditional correlation* (CCC) model.

The `mgarch` function can be used to estimate a CCC model with a GARCH(p, q) model for each series, by specifying `~ccc(p,q)` as the conditional variance formula. In addition, a more general formula such as `~ccc.type(p,q)` can also be used, where `type` can be any of the GARCH variants supported by the `garch` function.[5] For example, to use a two components model for each series when fitting a CCC model to the bivariate time series `hp.ibm`, use the following conditional variance formula:

```
> mgarch(hp.ibm~1, ~ccc.two.comp(1,1), trace=F)

Call:
mgarch(formula.mean = hp.ibm ~ 1, formula.var =
       ~ ccc.two.comp(1, 1), trace = F)

Mean Equation: hp.ibm ~ 1

Conditional Variance Equation:  ~ ccc.two.comp(1, 1)

Coefficients:

          C(1) 4.907e-04
          C(2) 1.844e-04
       A(1, 1) 8.722e-05
       A(2, 2) 6.579e-05
 ARCH(1; 1, 1) 8.102e-03
 ARCH(1; 2, 2) 9.621e-03
 ARCH(2; 1, 1) 9.669e-02
 ARCH(2; 2, 2) 9.582e-02
GARCH(1; 1, 1) 9.699e-01
GARCH(1; 2, 2) 9.654e-01
GARCH(2; 1, 1) 7.365e-01
GARCH(2; 2, 2) 7.271e-01

Conditional Constant Correlation Matrix:
        HP    IBM
 HP 1.0000 0.5582
IBM 0.5582 1.0000
```

When fitting a CCC model, `mgarch` function allows several alternatives for the estimation of the constant conditional correlation matrix $\mathbf{R}$ by setting the optional argument `cccor.choice`:

1. `cccor.choice=0`: The sample correlation matrix is used, and no further MLE estimation of $\mathbf{R}$ is carried out.

[5]See Section 7.9 in Chapter 7 for a summary of those specifications.

2. `cccor.choice=1`: The sample correlation matrix is used as the initial estimate, and the final estimate of $\mathbf{R}$ is obtained as part of the MLE method. This is the default value.

3. `cccor.choice=2`: The user supplies an initial correlation matrix estimate, and the final estimate of $\mathbf{R}$ is obtained as part of the MLE method. In this case, the user needs to supply the initial estimate with the optional argument `cccor.value`.

A potentially important use of the last choice is to obtain robustness toward multivariate outliers by using a robust initial covariance matrix estimate. The `covRob` function in `S-PLUS` `robust` library provides several robust covariance and correlation estimates.

Principal Component Model

In principal component analysis, it is well known that for any covariance matrix $\mathbf{\Sigma}$, one can always find an orthogonal matrix $\mathbf{\Lambda}$ and a diagonal matrix $\mathbf{\Delta}$ such that

$$\mathbf{\Lambda}\mathbf{\Delta}\mathbf{\Lambda}' = \mathbf{\Sigma}$$

where $\mathbf{\Lambda}$ is usually normalized so that $\mathbf{\Lambda}\mathbf{\Lambda}' = \mathbf{I}$ with $\mathbf{I}$ being an identity matrix. It can be shown that the diagonal elements of $\mathbf{\Delta}$ are the eigenvalues of $\mathbf{\Sigma}$, while the columns of $\mathbf{\Lambda}$ correspond to the eigenvectors of $\mathbf{\Sigma}$. Based on this result, the principal components of $\mathbf{y}_t$, which are defined as $\mathbf{z}_t = \mathbf{\Lambda}'\mathbf{y}_t$, have a diagonal covariance matrix. Ding (1994) describes the principal component GARCH model, which essentially models each principal component in $\mathbf{z}_t$ as a univariate GARCH model. This model is also proposed by Alexander (1998).

The `mgarch` function can be used to estimate a principal component model with a GARCH(p, q) model for principal component, by specifying `~prcomp(p,q)` as the conditional variance formula. Similar to the CCC model, a more general formula such as `~prcomp.type(p,q)` can also be used, where `type` can be any of the GARCH variants supported by the `garch` function. For example, to use a PGARCH$(1, 1, 1)$ model for each series when fitting the principal component model to the bivariate time series `hp.ibm`, use the following conditional variance formula:

```
> mgarch(hp.ibm~1, ~prcomp.pgarch(1,1,1), trace=F)

Call:
mgarch(formula.mean = hp.ibm ~ 1, formula.var =
       ~ prcomp.pgarch(1, 1, 1), trace = F)

Mean Equation: hp.ibm ~ 1

Conditional Variance Equation:  ~ prcomp.pgarch(1, 1, 1)
```

```
Coefficients:

          C(1) -3.519e-04
          C(2) -1.614e-05
       A(1, 1)  1.848e-03
       A(2, 2)  3.565e-04
 ARCH(1; 1, 1)  1.100e-01
 ARCH(1; 2, 2)  5.992e-02
GARCH(1; 1, 1)  8.380e-01
GARCH(1; 2, 2)  9.222e-01

Eigenvectors: (orthonormal transform matrix):
         HP     IBM
 HP -0.9054  0.4245
IBM -0.4245 -0.9054

Eigenvalues:
[1] 0.0006002 0.0001222
```

Pure Diagonal Model

Sometimes, the user may want to fit the same type of GARCH model to a large number of time series. The `mgarch` function also allows this type of univariate GARCH-based estimation, which totally ignores the correlation of the multivariate time series. For this purpose, any univariate GARCH specification can be used directly with the `mgarch` function. For example, to estimate a TGARCH$(1,1)$ model to both `hp.s` and `ibm.s` at the same time, use the following command:

```
> mgarch(hp.ibm~1, ~egarch(1,1), leverage=T, trace=F)

Call:
mgarch(formula.mean = hp.ibm ~ 1, formula.var = ~ egarch(1, 1),
       leverage = T, trace = F)

Mean Equation: hp.ibm ~ 1

Conditional Variance Equation:  ~ egarch(1, 1)

Coefficients:

          C(1)  0.0004561
          C(2)  0.0001810
       A(1, 1) -0.7959068
       A(2, 2) -0.9192535
```

```
 ARCH(1; 1, 1)  0.1618657
 ARCH(1; 2, 2)  0.1350345
GARCH(1; 1, 1)  0.9124564
GARCH(1; 2, 2)  0.9066042
  LEV(1; 1, 1)  0.0243099
  LEV(1; 2, 2) -0.1743824
```

Although the optional argument `leverage` can be used with any univariate GARCH-based models for `mgarch` function, it is ignored for BEKK, DVEC and its modifications.

13.5.4 ARMA Terms and Exogenous Variables

All the multivariate GARCH models considered so far have been restricted to a constant mean assumption. However, the `mgarch` function actually allows a more general model with a vector ARMA (VARMA) structure and optional inclusion of weakly exogenous variables in the conditional mean:

$$\mathbf{y}_t = \mathbf{c} + \sum_{i=1}^{r} \mathbf{\Phi}_i \mathbf{y}_{t-i} + \sum_{l=0}^{L} \boldsymbol{\beta}_l \mathbf{x}_{t-l} + \boldsymbol{\epsilon}_t + \sum_{j=1}^{s} \mathbf{\Theta}_j \boldsymbol{\epsilon}_{t-j} \qquad (13.10)$$

where $\mathbf{\Phi}_i$ are $k \times k$ autoregressive coefficient matrix, $\mathbf{\Theta}_j$ are $k \times k$ moving average coefficient matrix, $\mathbf{x}_t$ is the $m \times 1$ vector of weakly exogenous variables, and $\boldsymbol{\beta}_l$ is $k \times m$ coefficients of $\mathbf{x}_{t-l}$. Note that a distributed lag structure of $\mathbf{x}_t$ is allowed in the above equation by setting L to be a positive integer.

To include an AR(r), MA(s), or ARMA(r, s) term in the conditional mean, the user can simply add an `ar(r)`, `ma(s)`, or `arma(r,s)` term to the conditional mean formula. However, by default, $\mathbf{\Phi}_i$ and $\mathbf{\Theta}_j$ are restricted to be diagonal matrices for parsimonious reasons. This behavior can be changed by setting the optional argument `armaType` of the `mgarch` function. In particular, if `armaType="lower"`, then $\mathbf{\Phi}_i$ and $\mathbf{\Theta}_j$ are restricted to be lower triangular matrices; if `armaType="full"`, then $\mathbf{\Phi}_i$ and $\mathbf{\Theta}_j$ are not restricted. When weakly exogenous variables $\mathbf{x}_t$ are used, the optional argument `xlag` can be set to a positive integer to use a distributed lag structure.

Example 87 *Single factor model with multivariate GARCH errors*

Section 7.5 of Chapter 7 developed a single factor model with GARCH errors. Here that example is extended to multivariate context using the bivariate time series `hp.ibm`. The univariate example used daily returns on the value weighted New York Stock Exchange index as the "market returns" to estimate the "market beta". In practice, this market beta can be biased due to the serial correlation in the market returns. Hence, both

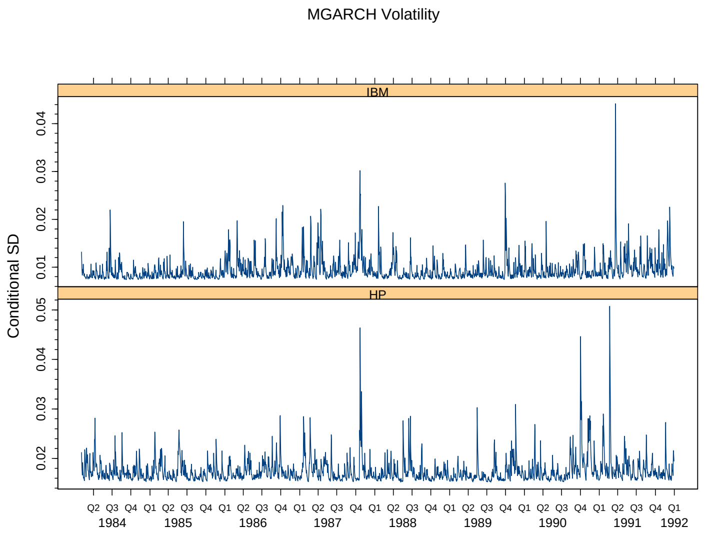


FIGURE 13.8. Idiosyncratic volatility of bivariate `hp.ibm`.

`nyse.s` and its first lag as regressors are included in the conditional mean equation, and the DVEC(1, 1) model is used in the conditional variance:

```
> hp.ibm.beta = mgarch(hp.ibm~seriesData(nyse.s), ~dvec(1,1),
+ xlag=1)
> summary(hp.ibm.beta)

Call:
mgarch(formula.mean = hp.ibm ~ seriesData(nyse.s),
       formula.var = ~ dvec(1, 1), xlag = 1)

Mean Equation: hp.ibm ~ seriesData(nyse.s)

Conditional Variance Equation:  ~ dvec(1, 1)

Conditional Distribution:  gaussian

------------------------------------------------------------

Estimated Coefficients:
------------------------------------------------------------
                    Value Std.Error  t value  Pr(>|t|)
          C(1)  7.860e-05 3.714e-04   0.2116 4.162e-01
```

```
         C(2) -3.343e-04 1.947e-04  -1.7166 4.311e-02
    X(0; 1, 1)  1.491e+00 2.867e-02  52.0032 0.000e+00
    X(0; 2, 1)  1.112e+00 1.751e-02  63.4896 0.000e+00
    X(1; 1, 1) -1.497e-01 3.233e-02  -4.6297 1.949e-06
    X(1; 2, 1) -1.802e-01 1.898e-02  -9.4945 0.000e+00
       A(1, 1)  1.028e-04 1.420e-05   7.2413 3.160e-13
       A(2, 1)  6.166e-06 4.520e-06   1.3642 8.633e-02
       A(2, 2)  3.117e-05 3.226e-06   9.6600 0.000e+00
 ARCH(1; 1, 1)  1.230e-01 1.812e-02   6.7878 7.482e-12
 ARCH(1; 2, 1)  5.030e-03 1.530e-02   0.3288 3.712e-01
 ARCH(1; 2, 2)  2.567e-01 2.155e-02  11.9125 0.000e+00
GARCH(1; 1, 1)  5.494e-01 5.543e-02   9.9112 0.000e+00
GARCH(1; 2, 1)  7.904e-01 1.483e-01   5.3285 5.511e-08
GARCH(1; 2, 2)  4.261e-01 4.432e-02   9.6126 0.000e+00

...
```

In the above output, the coefficient matrix $\boldsymbol{\beta}_0$ of `nyse.s` is denoted by `X(0;i,j)` and $\boldsymbol{\beta}_1$ of the first lag of `nyse.s` is denoted by `X(1;i,j)`. All those coefficients are very significant. Now compare the `GARCH(1;i,j)` coefficients with those of `hp.ibm.dvec`; after taking account of the market effects, the persistence in the GARCH volatilities has dropped quite a bit. The estimated conditional volatility can also be plotted as shown in Figure 13.8. Compare this with Figure 13.5: since the market effects are already taken into account in the above single factor model, the volatility in Figure 13.8 can be treated as the "idiosyncratic" volatility, while Figure 13.5 also includes the systematic market component.

Weakly exogenous variables are also allowed in the conditional variance equation for multivariate GARCH models. For example, for the DVEC(p, q) model, the general conditional variance equation is:

$$\mathbf{\Sigma}_t = \mathbf{A}_0 + \sum_{i=1}^{p} \mathbf{A}_i \otimes (\boldsymbol{\epsilon}_{t-i}\boldsymbol{\epsilon}'_{t-i}) + \sum_{j=1}^{q} \mathbf{B}_j \otimes \mathbf{\Sigma}_{t-j} + \mathbf{D} \cdot \mathbf{Z}_t \cdot \mathbf{D}' \tag{13.11}$$

where $\mathbf{Z}_t$ is a diagonal matrix with the $m \times 1$ weakly exogenous variable $(Z_{t1}, \cdots, Z_{tm})$ on the diagonal, and $\mathbf{D}$ is $k \times m$ coefficient matrix. Note that using this formulation, the regressor effects are guaranteed to be positive semi-definite as long as the regressors $\mathbf{Z}_t$ are non-negative.

Example 88 *Monday and Friday effects of volatility*

There is a conjecture that the volatility in stock markets may be higher on Mondays and Fridays. To investigate if this conjecture holds for the bivariate time series `hp.ibm`, build a dummy variable for those observations falling on a Monday or a Friday:

```
> weekdaysVec = as.integer(weekdays(positions(hp.ibm)))
> MonFriDummy = (weekdaysVec == 2 | weekdaysVec == 6)
```

Note that the integer representation of Monday in S-PLUS is two because Sunday is represented as one. Now add MonFriDummy as an exogenous variable in the conditional variance formula:

```
> hp.ibm.dummy = mgarch(hp.ibm~1, ~dvec(1,1)+MonFriDummy)
> summary(hp.ibm.dummy)

Call:
mgarch(formula.mean = hp.ibm ~ 1, formula.var =  ~ dvec(1, 1)
       + MonFriDummy)

Mean Equation: hp.ibm ~ 1

Conditional Variance Equation:  ~ dvec(1, 1) + MonFriDummy

Conditional Distribution:  gaussian

------------------------------------------------------------

Estimated Coefficients:
------------------------------------------------------------
                   Value Std.Error t value  Pr(>|t|)
          C(1) 6.953e-04 4.696e-04  1.4806 6.943e-02
          C(2) 2.659e-04 2.849e-04  0.9333 1.754e-01
       A(1, 1) 3.369e-05 8.612e-06  3.9124 4.723e-05
       A(2, 1) 7.384e-06 5.682e-06  1.2997 9.693e-02
       A(2, 2) 2.011e-05 5.214e-06  3.8565 5.934e-05
 ARCH(1; 1, 1) 6.400e-02 8.952e-03  7.1494 6.088e-13
 ARCH(1; 2, 1) 3.546e-02 7.029e-03  5.0443 2.482e-07
 ARCH(1; 2, 2) 1.076e-01 1.004e-02 10.7141 0.000e+00
GARCH(1; 1, 1) 8.600e-01 1.716e-02 50.1061 0.000e+00
GARCH(1; 2, 1) 8.805e-01 1.816e-02 48.4743 0.000e+00
GARCH(1; 2, 2) 7.472e-01 2.855e-02 26.1706 0.000e+00
        Z(1,1) 3.139e-03 2.440e-03  1.2865 9.921e-02
        Z(2,1) 4.426e-03 1.074e-03  4.1215 1.959e-05

...
```

In the above output, Z(1,1) denotes the coefficient of the dummy variable for hp.s, the p-value of which is higher than the conventional 5% level, and Z(2,1) denotes the coefficient for ibm.s, the p-value of which is very close to zero. So it seems that for IBM stocks, the volatility tends to be slightly higher on Mondays and Fridays.

13.5.5 Multivariate Conditional t-Distribution

In all the multivariate GARCH models fitted so far, it has been assumed that the residuals $\boldsymbol{\epsilon}_t$ follow a conditional multivariate normal distribution. The `mgarch` function also allows the residuals to follow a multivariate Student's t distribution.

If a k-dimensional random variable $\mathbf{u}_t$ follows a multivariate Student's t distribution with ν degrees of freedom and the scale matrix $\mathbf{S}_t$, the probability density function (PDF) of $\mathbf{u}_t$ is given by:

$$f(\mathbf{u}_t) = \frac{\Gamma[(\nu+k)/2]}{(\pi\nu)^{k/2}\Gamma(\nu/2)} \frac{|\mathbf{S}_t|^{-1/2}}{[1+\mathbf{u}_t'\mathbf{S}_t^{-1}\mathbf{u}_t/\nu]^{(\nu+k)/2}} \tag{13.12}$$

where $\Gamma(\cdot)$ is the gamma function. The covariance matrix of $\mathbf{u}_t$ is given by:

$$\mathrm{Cov}(\mathbf{u}_t) = \frac{\nu}{\nu-2}\mathbf{S}_t.$$

If the error term $\boldsymbol{\epsilon}_t$ is assumed in multivariate GARCH models follows a conditional multivariate Student's t distribution with ν degrees of freedom and $\mathrm{Cov}(\boldsymbol{\epsilon}_t) = \boldsymbol{\Sigma}_t$, obviously the scale matrix $\mathbf{S}_t$ should be chosen so that

$$\mathbf{S}_t = \frac{\nu-2}{\nu}\boldsymbol{\Sigma}_t.$$

By substituting the above relationship into (13.12), the user can easily derive the log-likelihood function for multivariate GARCH models with conditional multivariate Student's t distributed errors. The unknown model parameters can also be routinely estimated using maximum likelihood estimation.

To use the multivariate Student's t distribution with the `mgarch` function to estimate a multivariate GARCH model, simply set the optional argument `cond.dist` to `"t"`. For example:

```
> hp.ibm.dvec.t = mgarch(hp.ibm~1, ~dvec(1,1), cond.dist="t")
```

The estimated degree of freedom ν is contained in the `cond.dist` component of the returned object:

```
> hp.ibm.dvec.t$cond.dist
$cond.dist:
[1] "t"

$dist.par:
[1] 6.697768

$dist.est:
[1] T
```

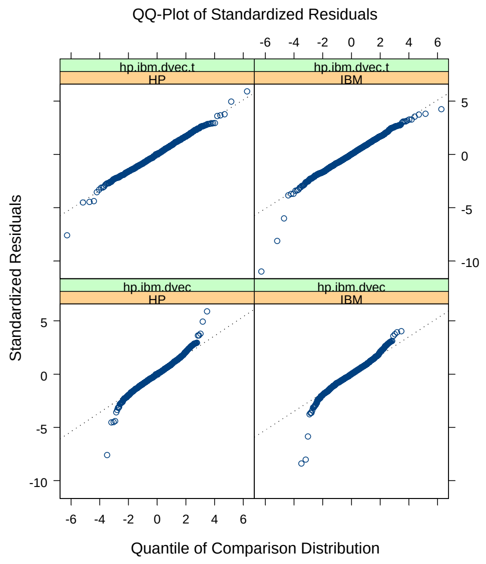


FIGURE 13.9. Comparison of QQ-plot using normal and Student-t distributions.

Compare this model with the one fitted using multivariate normal distribution:

```
> hp.ibm.comp = compare.mgarch(hp.ibm.dvec, hp.ibm.dvec.t)
> hp.ibm.comp
           hp.ibm.dvec hp.ibm.dvec.t
       AIC      -21886        -22231
       BIC      -21825        -22164
Likelihood       10954         11128
> plot(hp.ibm.comp, qq=T)
```

Obviously, the multivariate Student's t distribution provides a much better fit. This can also be confirmed by comparing the QQ-plot of standardized residuals, which is shown in Figure 13.9.

13.6 Multivariate GARCH Prediction

Predictions from multivariate GARCH models can be generated in a similar fashion to predictions from univariate GARCH models. Indeed, for the univariate GARCH-based models, such as CCC model and principal component model, the predictions are generated from the underlying univariate GARCH models and then converted to the scale of the original multivariate time series by using the appropriate transformation. This section focuses on predicting from DVEC model, because predicting from BEKK model can be performed similarly.

For multivariate GARCH models, predictions can be generated for both the levels of the original multivariate time series and its conditional covariance matrix. Predictions of the levels are obtained just as for vector autoregressive (VAR) models. Compared with VAR models, the predictions of the conditional covariance matrix from multivariate GARCH models can be used to construct more reliable confidence intervals for predictions of the levels.

To illustrate the prediction of conditional covariance matrix for multivariate GARCH models, consider the conditional variance equation for the DVEC$(1, 1)$ model:

$$\mathbf{\Sigma}_t = \mathbf{A}_0 + \mathbf{A}_1 \otimes (\boldsymbol{\epsilon}_{t-1}\boldsymbol{\epsilon}'_{t-1}) + \mathbf{B}_1 \otimes \Sigma_{t-1}$$

which is estimated over the time period $t = 1, 2, \cdots, T$. To obtain $E_T(\mathbf{\Sigma}_{T+k})$, use the forecasts of conditional covariance matrix at time $T + k$ for $k > 0$, given information at time T. For one-step-ahead prediction, it is easy to obtain:

$$\begin{aligned} E_T(\mathbf{\Sigma}_{T+1}) &= \mathbf{A}_0 + \mathbf{A}_1 \otimes E_T(\boldsymbol{\epsilon}_T\boldsymbol{\epsilon}'_T) + \mathbf{B}_1 \otimes E_T(\mathbf{\Sigma}_T) \\ &= \mathbf{A}_0 + \mathbf{A}_1 \otimes (\boldsymbol{\epsilon}_T\boldsymbol{\epsilon}'_T) + \mathbf{B}_1 \otimes \mathbf{\Sigma}_T \end{aligned}$$

since an estimate of $\boldsymbol{\epsilon}_T$ and $\mathbf{\Sigma}_T$ already exists after estimating the DVEC model. When $k = 2$, it can be shown that

$$\begin{aligned} E_T(\mathbf{\Sigma}_{T+2}) &= \mathbf{A}_0 + \mathbf{A}_1 \otimes E_T(\boldsymbol{\epsilon}_{T+1}\boldsymbol{\epsilon}'_{T+1}) + \mathbf{B}_1 \otimes E_T(\mathbf{\Sigma}_{T+1}) \\ &= \mathbf{A}_0 + (\mathbf{A}_1 + \mathbf{B}_1) \otimes E_T(\mathbf{\Sigma}_{T+1}). \end{aligned}$$

where $E_T(\mathbf{\Sigma}_{T+1})$ is obtained in the previous step. This procedure can be iterated to obtain $E_T(\mathbf{\Sigma}_{T+k})$ for $k > 2$.

The `predict` method for "`mgarch`" objects in `S+FinMetrics` implements the forecasting procedure for all the multivariate GARCH models supported by the `mgarch` function. The forecasts can be easily obtained by calling the generic `predict` function for an "`mgarch`" object with the desired number of forecasting periods. For example, to obtain 10-step-ahead forecasts from the BEKK model object `hp.ibm.bekk` fitted in Section 13.5, use the following command:

```
> hp.ibm.pred = predict(hp.ibm.bekk, 10)
> class(hp.ibm.pred)
[1] "predict.mgarch"
> names(hp.ibm.pred)
[1] "series.pred" "sigma.pred"  "R.pred"
```

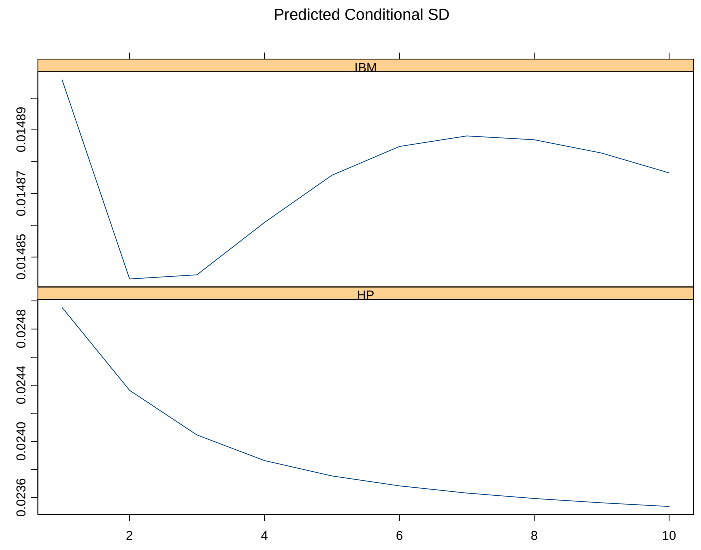


FIGURE 13.10. BEKK prediction of conditional standard deviations.

The returned object `hp.ibm.pred` is of class "`predict.mgarch`", and has three components: `series.pred` represents the forecasts of the levels of the time series, `sigma.pred` represents the forecasts of the conditional standard deviations, and `R.pred` represents the forecasts of the conditional correlation matrix. Note that the `sigma.pred` and `R.pred` components can be used together to obtain the forecasts of the conditional covariance matrix.

`S+FinMetrics` also implements a `plot` method for "`predict.mgarch`" objects, so that the multivariate forecasts can be visualized directly. For example, if the user calls the generic `plot` function directly on `hp.ibm.pred`:

```
> plot(hp.ibm.pred)

Make a plot selection (or 0 to exit):

1: plot: All
2: plot: Predicted Conditional Mean
3: plot: Predicted Conditional SD
Selection:
```

Selecting `3` will generate the plot of predicted conditional standard deviations, as shown in Figure 13.10, the confidence interval of the volatility forecasts should be obtained as well. Section 13.8 shows how to obtain a confidence interval using simulation-based forecasts.

13.7 Custom Estimation of GARCH Models

13.7.1 GARCH Model Objects

For both "garch" and "mgarch" objects, there is a model component which contains all the necessary model specific information about the fitted univariate or multivariate GARCH model. For example, for the univariate "garch" object ford.mod11 fitted in Section 7.4 of Chapter 7:

```
> class(ford.mod11$model)
[1] "garch.model"
> ford.mod11$model

Mean Equation: ford.s ~ 1

Conditional Variance Equation:  ~ garch(1, 1)

 ------------ Constants in mean ------------

     value which
 0.0007708     1

 ---------- Constants in variance ----------

     value which
 6.534e-06     1

 ------------------- ARCH ------------------

        value which
lag 1 0.07454     1

 ------------------ GARCH ------------------

       value which
lag 1 0.9102     1
```

So the model component of a "garch" object is of class "garch.model". Similarly, for the "mgarch" object hp.ibm.dvec fitted in this chapter:

```
> class(hp.ibm.dvec$model)
[1] "mgarch.model"
> hp.ibm.dvec$model

Mean Equation: hp.ibm ~ 1

Conditional Variance Equation:  ~ dvec(1, 1)
```

```
 ------------ Constants in mean ------------

          value which
v1 0.0007017567     T
v2 0.0002932253     T

 ---------- Constants in variance ----------

       v1.value     v2.value *** v1.which v2.which
v1 3.888696e-05 1.322108e-05 ***        T        T
v2 1.322108e-05 2.876733e-05 ***        T        T

 ------------------- ARCH ------------------

 Lag 1
     v1.value   v2.value *** v1.which v2.which
v1 0.06225657 0.03393546 ***        T        T
v2 0.03393546 0.10485809 ***        T        T

 ------------------ GARCH ------------------

 Lag 1
    v1.value  v2.value *** v1.which v2.which
v1 0.8567934 0.8783100 ***        T        T
v2 0.8783100 0.7421328 ***        T        T
```

So the `model` component of an "`mgarch`" object is of class "`mgarch.model`", which has similar structures to a "`garch.model`" object. This section will focus on "`mgarch.model`" objects, though all the things illustrated can also be applied to "`garch.model`" objects.

Since an "`mgarch.model`" object contains all the necessary information about a fitted GARCH model, this object can be saved or edited for many purposes.[6] The names of the components of an "`mgarch.model`" object can be obtained using the `S-PLUS` function `names`:

```
> names(hp.ibm.dvec$model)
[1] "c.which" "c.value" "MA"      "AR"      "arch"    "garch"
[7] "a.which" "a.value" "info"
```

[6]In the first release of `S+GARCH` module, there was a `revise` function which provides a graphical user interface for editing this object. However, the function was broken as the graphical user interface of S-PLUS went through several evolutions. Currently there is no `revise` function in `S+FinMetrics` module.

The component `c.value` contains the value of the constant terms in the conditional mean equation, while the component `a.value` contains the value of the constant terms in the conditional variance equation. The `MA`, `AR`, `arch` and `garch` components are lists themselves. For example:

```
> hp.ibm.dvec$model$arch
$order:
[1] 1

$value:
$lag.1:
           [,1]       [,2]
[1,] 0.06225657 0.03393546
[2,] 0.03393546 0.10485809


$which:
$lag.1:
     [,1] [,2]
[1,]    T    T
[2,]    T    T
```

Note that for each of the model coefficients, there is a corresponding `which` component that specifies if the coefficient is free to be estimated by MLE. If the `which` component is `1` or `TRUE`, then the corresponding coefficient is free to be estimated; otherwise, the corresponding coefficient is fixed at that value during MLE. The next subsection shows how these values can be edited for different purposes.

13.7.2 Revision of GARCH Model Estimation

For both univariate and multivariate GARCH models, the unknown model parameters are estimated using the BHHH algorithm (e.g., see Bollerslev, 1986). Both `garch` and `mgarch` functions take an optional argument `control`, which can be used to control certain numerical aspects of the BHHH algorithm. The defaults for those settings are provided in the online help file for `bhhh.control`.

Like many other nonlinear optimization algorithms, the BHHH algorithm performs local optimization in the sense that the optimal solution it finds may well be just a local optimum instead of the global optimum. To make sure that the global optimum has indeed been reached, start the algorithm using a few different starting values and see if they all lead to the same optimum. For this purpose, edit the `model` component of a fitted "`garch`" or "`mgarch`" object and use it as a new starting value.

Example 89 *Restarting multivariate GARCH estimation*

```
> bekk.mod = hp.ibm.bekk$model
> bekk.mod$a.value[2,1] = 0
> hp.ibm.bekk2 = mgarch(series=hp.ibm, model=bekk.mod)
```

Note that when a model object is supplied directly to the `mgarch` (or `garch`) function, the `series` argument must be used to supply the data. The user can easily verify that `hp.ibm.bekk2` reached a smaller log-likelihood value, so the original fit `hp.ibm.bekk` seems to be better.

Example 90 *Constraining multivariate GARCH estimation*

For some GARCH models, the user may want to fix certain parameters at certain values during maximum likelihood estimation. For example, most daily financial security returns seem to fluctuate around a zero mean. In this example, fix the constant terms in the conditional mean equation of `hp.ibm.bekk` to zero and re-estimate the model:

```
> bekk.mod = hp.ibm.bekk$model
> bekk.mod$c.value = rep(0,2)
> bekk.mod$c.which = rep(F,2)
> hp.ibm.bekk3 = mgarch(series=hp.ibm, model=bekk.mod)
> LR.stat = -2*(hp.ibm.bekk3$likelihood-
+ hp.ibm.bekk$likelihood)
```

Note that since the log-likelihood value of the fitted model is returned, a likelihood ratio (LR) test of the restrictions imposed in the above example can easily be performed.

The "`garch.model`" or "`mgarch.model`" object can be used for simulation. For example, simulation from fitted univariate GARCH models actually uses this component. The next section illustrates this usage for multivariate GARCH models.

13.8 Multivariate GARCH Model Simulation

`S+FinMetrics` provides a method of the generic function `simulate` for objects of class "`mgarch`". The method function, `simulate.mgarch`, can take a fitted "`mgarch`" object, or an "`mgarch.model` object, or simply a user specified list. This section illustrates how to create confidence intervals for the predictions of conditional standard deviations using simulations.

Example 91 *Simulation-based multivariate GARCH forecasts*

The function `simulate.mgarch` only supports those multivariate GARCH models of order $(1, 1)$, which should be enough for most applications. To simulate a multivariate GARCH process directly from a fitted "`mgarch`" object such as `hp.ibm.bekk`, call the generic function `simulate` as follows:

```
> hp.ibm.sim = simulate(hp.ibm.bekk, n=10)
```

where $n = 10$ specifies the length of the simulated time series. Since all the model specific information is contained in the `model` component of an "`mgarch`" object, which is an "`mgarch.model`" object as shown in the previous section, an "`mgarch.model` can also pass directly to the function `simulate.mgarch`. The following code example simulates 100 steps ahead from the end of estimation period in `hp.ibm.bekk`, and replicates the simulation 200 times:

```
> eps.start = residuals(hp.ibm.bekk)[2000,]@data
> V.start = hp.ibm.bekk$S.t[2000, , ]
> n.rep = 200
> hp.ibm.sim = array(0, c(100, 2, n.rep))

> set.seed(10)
> for (i in 1:n.rep) {
+   eps.pred = rbind(eps.start, rmvnorm(100))
+   tmp = simulate(hp.ibm.bekk, n=100, n.start=0,
+         etat=eps.pred, V.start=V.start)$V.t
+   hp.ibm.sim[, , i] = matrix(tmp,byrow=T,nrow=100)[,c(1,4)]
+ }

> hp.ibm.sim = sqrt(hp.ibm.sim)
> hp.ibm.simpred = rowMeans(hp.ibm.sim, dims=2)
> hp.ibm.simstde = rowStdevs(hp.ibm.sim, dims=2)
```

Note that to simulate the multivariate GARCH process using the last observation in the sample as the starting value, set `n.start=0` and `V.start` to the last estimated conditional covariance matrix. Similarly, the last estimated residual vector is used as the starting value in `eps.pred`, which is otherwise standard normal random variables. All the simulated conditional standard deviations are saved in `hp.ibm.sim`, which is a three dimensional array. The simulation-based forecasts of conditional standard deviations are computed as the average of `hp.ibm.sim`, and saved in the object `hp.ibm.simpred`, while `hp.ibm.simstde` contains the standard errors of those forecasts.

Finally, use the following code to plot confidence intervals around the simulation-based forecasts:

```
> par(mfrow=c(2,1))
> ci.upper = hp.ibm.simpred + 2*hp.ibm.simstde
> ci.lower = hp.ibm.simpred - 2*hp.ibm.simstde

> tsplot(cbind(hp.ibm.simpred[,1],ci.upper[,1],ci.lower[,1]))
> title("Forecasted HP Volatility", xlab="Time", ylab="SD")
```

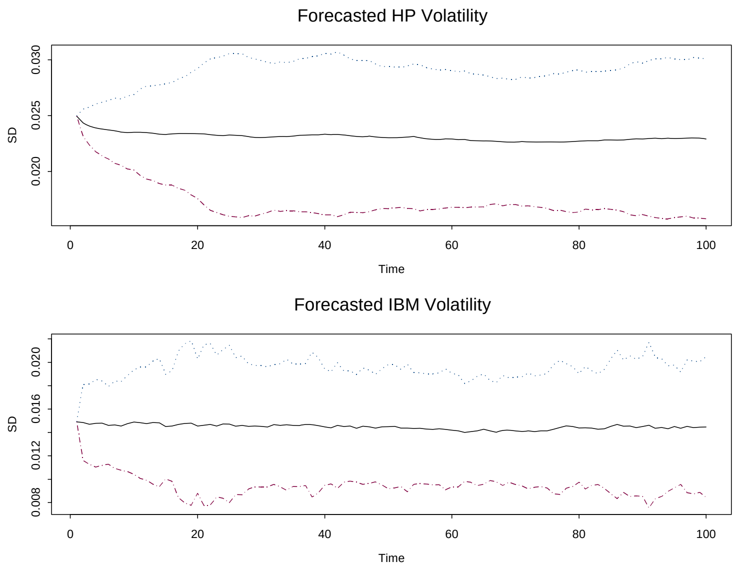


FIGURE 13.11. Simulation-based forecasts of BEKK model.

```
> tsplot(cbind(hp.ibm.simpred[,2],ci.upper[,2],ci.lower[,2]))
> title("Forecasted IBM Volatility", xlab="Time", ylab="SD")
> par(mfrow=c(1,1))
```

The plot shown in Figure 13.11 only used 200 replications, so the confidence intervals are a little rough. If more replications are used, the confidence intervals should be relatively smooth.

13.9 References

ALEXANDER, C. O. (1998). "Volatility and Correlation: Methods, Models and Applications," in C. O. Alexander (ed.) *Risk Management and Analysis: Measuring and Modeling Financial Risk*. John Wiley & Sons, New York.

ANDREOU, E., AND GHYSELS, E. (2002). "Rolling Volatility Estimators: Some new Theoretical, Simulation and Empirical Results," *Journal of Business and Economic Statistics*, 20(3), 363-376.

BOLLERSLEV, T. (1986). "Generalized Autoregressive Conditional Heteroskedasticity," *Journal of Econometrics*, 31, 307-327.

Bollerslev, T. (1990). "Modeling the Coherence in Short-run Nominal Exchange Rates: a Multivariate Generalized ARCH Model," *Review of Economics and Statistics*, 72, 498-505.

Bollerslev, T., Engle, R. F., and Nelson, D. B. (1994). "ARCH Models," in R. F. Engle and D. L. McFadden (eds.), *Handbook of Econometrics*, Vol. 4. Elsevier Science B. V., Amsterdam.

Bollerslev, T., Engle, R. F., and Wooldridge, J. M. (1988). "A Capital-Asset Pricing Model with Time-Varying Covariances," *Journal of Political Economy*, 96, 116-131.

Bougerol, P., and Picard, N. (1992). "Stationarity of GARCH Processes and of Some Nonnegative Time Series," *Journal of Econometrics*, 52, 115-127.

Ding, Z. (1994). "Time Series Analysis of Speculative Returns," *Ph.D. Thesis*, Department of Economics, University of California, San Diego.

Engle, R. F., and Bollerslev, T. (1986). "Modeling the Persistence of Conditional Variances," *Econometric Reviews*, 5, 1-50.

Engle, R. F., and Kroner, K. F. (1995). "Multivariate Simultaneous Generalized ARCH," *Econometric Theory*, 11, 122-150.

Fleming, J., Kirby, C., and Ostdiek, B. (2001). "The Economic Value of Volatility Timing," *Journal of Finance*, 56, 329-352.

Foster, D. P., and Nelson, D. B. (1996). "Continuous Record Asymptotics for Rolling Sample Variance Estimators," *Econometrica*, 64, 139-174.

Hosking, J. R. M. (1980). "The Multivariate Portmanteau Statistic," *Journal of the American Statistical Association*, 75, 602-608.

Nelson, D. B. (1990). "Stationarity and Persistence in the GARCH (1,1) Model," *Econometric Theory*, 6, 318-334.

14
State Space Models

14.1 Introduction

The state space modeling tools in `S+FinMetrics` are based on the algorithms in *SsfPack* 3.0 developed by Siem Jan Koopman and described in Koopman, Shephard and Doornik (1999, 2001)[1]. *SsfPack* is a suite of C routines for carrying out computations involving the statistical analysis of univariate and multivariate models in state space form. The routines allow for a variety of state space forms from simple time invariant models to complicated time-varying models. Functions are available to put standard models like ARMA and spline models in state space form. General routines are available for filtering, smoothing, simulation smoothing, likelihood evaluation, forecasting and signal extraction. Full details of the statistical analysis is provided in Durbin and Koopman (2001). This chapter gives an overview of state space modeling and the reader is referred to the papers by Koopman, Shephard and Doornik for technical details on the algorithms used in the `S+FinMetrics/SsfPack` functions.

This chapter is organized as follows. Section 14.2 describes the general state space model and state space representation required for the `S+FinMetrics/SsfPack` state space functions. Subsections describe the various `S+FinMetrics/SsfPack` functions for putting common time series models into state space form. The process of simulating observations from a given state space model is also covered. Section 14.3 summarizes the main

[1]Information about *Ssfpack* can be found at `http://www.ssfpack.com`.

algorithms used for the analysis of state space models. These include the Kalman filter, Kalman smoother, moment smoothing, disturbance smoothing and forecasting. Estimation of the unknown parameters in a state space model is described in Section 14.4. The chapter concludes with a short discussion of simulation smoothing.

Textbook treatments of state space models are given in Harvey (1989, 1993), Hamilton (1994), West and Harrison (1997), and Kim and Nelson (1999). Interesting applications of state space models in finance are given in Engle and Watson (1987), Bomhoff (1994), Duan and Simonato (1999), and Harvey, Ruiz, and Shephard (1994), Carmona (2004), and Chan (2002).

14.2 State Space Representation

Many dynamic time series models in economics and finance may be represented in *state space form*. Some common examples are ARMA models, time-varying regression models, dynamic linear models with unobserved components, discrete versions of continuous time diffusion processes, stochastic volatility models, non-parametric and spline regressions. The linear Gaussian *state space model* may be represented as the system of equations

$$\underset{m\times 1}{\boldsymbol{\alpha}_{t+1}} = \underset{m\times 1}{\mathbf{d}_t} + \underset{m\times m}{\mathbf{T}_t} \cdot \underset{m\times 1}{\boldsymbol{\alpha}_t} + \underset{m\times r}{\mathbf{H}_t} \cdot \underset{r\times 1}{\boldsymbol{\eta}_t} \tag{14.1}$$

$$\underset{N\times 1}{\boldsymbol{\theta}_t} = \underset{N\times 1}{\mathbf{c}_\mathbf{t}} + \underset{N\times m}{\mathbf{Z}_\mathbf{t}} \cdot \underset{m\times 1}{\boldsymbol{\alpha}_\mathbf{t}} \tag{14.2}$$

$$\underset{N\times 1}{\mathbf{y}_t} = \underset{N\times 1}{\boldsymbol{\theta}_t} + \underset{N\times N}{\mathbf{G}_t} \cdot \underset{N\times 1}{\boldsymbol{\varepsilon}_t} \tag{14.3}$$

where $t = 1, \ldots, n$ and

$$\boldsymbol{\alpha}_1 \sim N(\mathbf{a}, \mathbf{P}), \tag{14.4}$$

$$\boldsymbol{\eta}_t \sim \text{iid } N(0, \mathbf{I}_r) \tag{14.5}$$

$$\boldsymbol{\varepsilon}_t \sim \text{iid } N(\mathbf{0}, \mathbf{I}_N) \tag{14.6}$$

and it is assumed that

$$E[\boldsymbol{\varepsilon}_t \boldsymbol{\eta}_t'] = \mathbf{0}$$

In (14.4), $\mathbf{a}$ and $\mathbf{P}$ are fixed and known but that can be generalized. The *state vector* $\boldsymbol{\alpha}_t$ contains unobserved stochastic processes and unknown fixed effects and the *transition equation* (14.1) describes the evolution of the state vector over time using a first order Markov structure. The *measurement equation* (14.3) describes the vector of observations $\mathbf{y}_t$ in terms of the state vector $\boldsymbol{\alpha}_t$ through the signal $\boldsymbol{\theta}_t$ and a vector of disturbances $\boldsymbol{\varepsilon}_t$. It is assumed that the innovations in the transition equation and the innovations in the measurement equation are independent, but this assumption can be relaxed. The deterministic matrices $\mathbf{T}_t$, $\mathbf{Z}_t$, $\mathbf{H}_t$, $\mathbf{G}_t$ are called *system matrices* and are usually sparse selection matrices. The vectors $\mathbf{d}_t$ and $\mathbf{c}_t$ contain fixed components and may be used to incorporate known effects or known patterns into the model; otherwise they are equal to zero.

The state space model (14.1)-(14.6) may be compactly expressed as

$$\begin{aligned} \begin{pmatrix} \boldsymbol{\alpha}_{t+1} \\ \mathbf{y}_t \end{pmatrix} &= \underset{(m+N)\times 1}{\boldsymbol{\delta}_t} + \underset{(m+N)\times m}{\boldsymbol{\Phi}_t} \cdot \underset{m\times 1}{\boldsymbol{\alpha}_t} + \underset{(m+N)\times 1}{\mathbf{u}_t}, && (14.7) \\ \boldsymbol{\alpha}_1 &\sim N(\mathbf{a}, \mathbf{P}) && (14.8) \\ \mathbf{u}_t &\sim \text{iid } N(\mathbf{0}, \boldsymbol{\Omega}_t) && (14.9) \end{aligned}$$

where

$$\boldsymbol{\delta}_t = \begin{pmatrix} \mathbf{d}_t \\ \mathbf{c}_t \end{pmatrix}, \boldsymbol{\Phi}_t = \begin{pmatrix} \mathbf{T}_t \\ \mathbf{Z}_t \end{pmatrix}, \mathbf{u}_t = \begin{pmatrix} \mathbf{H}_t \boldsymbol{\eta}_t \\ \mathbf{G}_t \boldsymbol{\varepsilon}_t \end{pmatrix}, \boldsymbol{\Omega}_t = \begin{pmatrix} \mathbf{H}_t \mathbf{H}_t' & \mathbf{0} \\ \mathbf{0} & \mathbf{G}_t \mathbf{G}_t' \end{pmatrix}$$

The initial value parameters are summarized in the $(m+1) \times m$ matrix

$$\boldsymbol{\Sigma} = \begin{pmatrix} \mathbf{P} \\ \mathbf{a}' \end{pmatrix} \quad (14.10)$$

For multivariate models, i.e. $N > 1$, it is assumed that the $N \times N$ matrix $\mathbf{G}_t \mathbf{G}_t'$ is diagonal. In general, the system matrices in (14.7) are time varying.

14.2.1 Initial Conditions

The variance matrix $\mathbf{P}$ of the initial state vector $\boldsymbol{\alpha}_1$ is assumed to be of the form

$$\mathbf{P} = \mathbf{P}_* + \kappa \mathbf{P}_\infty \quad (14.11)$$

where $\mathbf{P}_\infty$ and $\mathbf{P}_*$ are symmetric $m \times m$ matrices with ranks r_∞ and r_*, respectively, and κ is a large scalar value, e.g. $\kappa = 10^6$. The matrix $\mathbf{P}_*$ captures the covariance structure of the stationary components in the initial state vector, and the matrix $\mathbf{P}_\infty$ is used to specify the initial variance matrix for nonstationary components. When the ith diagonal element of $\mathbf{P}_\infty$ is negative, the corresponding ith column and row of $\mathbf{P}_*$ are assumed to be zero, and the corresponding row and column of $\mathbf{P}_\infty$ will be taken into consideration. When some elements of state vector are nonstationary, the `S+FinMetrics/SsfPack` algorithms implement an “exact diffuse prior” approach as described in Durbin and Koopman (2001) and Koopman, Shephard and Doornik (2001).

14.2.2 State Space Representation in `S+FinMetrics/SsfPack`

State space models in `S+FinMetrics/SsfPack` utilize the compact representation (14.7) with initial value information (14.10). The following examples describe the specification of a state space model for use in the `S+FinMetrics/SsfPack` state space modeling functions.

Example 92 *State space representation of the local level model*

Consider the following simple model for the stochastic evolution of the logarithm of an asset price y_t

$$\begin{aligned} \alpha_{t+1} &= \alpha_t + \eta_t^*,\ \eta_t^* \sim \text{iid } N(0, \sigma_\eta^2) && (14.12) \\ y_t &= \alpha_t + \varepsilon_t^*,\ \varepsilon_t^* \sim \text{iid } N(0, \sigma_\varepsilon^2) && (14.13) \\ \alpha_1 &\sim N(a, P) && (14.14) \end{aligned}$$

where it is assumed that $E[\varepsilon_t^* \eta_t^*] = 0$. In the above model, the observed asset price y_t is the sum of two unobserved components, α_t and ε_t^*. The component α_t is the state variable and represents the fundamental value (signal) of the asset. The transition equation (14.12) shows that the fundamental values evolve according to a random walk. The component ε_t^* represents random deviations (noise) from the fundamental value that are assumed to be independent from the innovations to α_t. The strength of the signal in the fundamental value relative to the random deviation is measured by the signal-to-noise ratio of variances $q = \sigma_\eta^2/\sigma_\varepsilon^2$. The model (14.12)-(14.14) is called the *random walk plus noise model*, *signal plus noise model* or the *local level model*.[2]

The state space form (14.7) of the local level model has time invariant parameters

$$\boldsymbol{\delta} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \Phi = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \Omega = \begin{pmatrix} \sigma_\eta^2 & 0 \\ 0 & \sigma_\varepsilon^2 \end{pmatrix} \qquad (14.15)$$

with errors $\sigma_\eta \eta_t = \eta_t^*$ and $\sigma_\varepsilon \varepsilon_t = \varepsilon_t^*$. Since the state variable α_t is $I(1)$, the unconditional distribution of the initial state α_1 doesn't have finite variance. In this case, it is customary to set $a = E[\alpha_1] = 0$ and $P = \text{var}(\alpha_1)$ to some large positive number, e.g. $P = 10^7$, in (14.14) to reflect that no prior information is available. Using (14.11), the initial variance is specified with $P_* = 0$ and $P_\infty = 1$. Therefore, the initial state matrix (14.10) for the local level model has the form

$$\Sigma = \begin{pmatrix} -1 \\ 0 \end{pmatrix} \qquad (14.16)$$

where -1 implies that $P_\infty = 1$.

In `S+FinMetrics/SsfPack`, a state space model is specified by creating either a list variable with components giving the minimum components

[2]A detailed technical analysis of this model is given in Durbin and Koopman (2001), chapter 2.

State space parameter	List component name
$\boldsymbol{\delta}$	mDelta
$\boldsymbol{\Phi}$	mPhi
$\boldsymbol{\Omega}$	mOmega
$\boldsymbol{\Sigma}$	mSigma

TABLE 14.1. State space form list components

necessary for describing a particular state space form or by creating an "ssf" object. To illustrate, consider creating a list variable containing the state space parameters in (14.15)-(14.16), with $\sigma_\eta = 0.5$ and $\sigma_\varepsilon = 1$

```
> sigma.e = 1
> sigma.n = 0.5
> a1 = 0
> P1 = -1
> ssf.ll.list = list(mPhi=as.matrix(c(1,1)),
+ mOmega=diag(c(sigma.n^2,sigma.e^2)),
+ mSigma=as.matrix(c(P1,a1)))
> ssf.ll.list
$mPhi:
     [,1]
[1,]    1
[2,]    1

$mOmega:
     [,1] [,2]
[1,] 0.25    0
[2,] 0.00    1

$mSigma:
     [,1]
[1,]   -1
[2,]    0
```

In the list variable `ssf.ll.list`, the component names match the state space form parameters in (14.7) and (14.10). This naming convention, summarized in Table 14.1, must be used for the specification of any valid state space model. Also, notice the use of the coercion function `as.matrix`. This ensures that the dimensions of the state space parameters are correctly specified.

An "ssf" object may be created from the list variable `ssf.ll.list` using the `S+FinMetrics/SsfPack` function `CheckSsf`:

```
> ssf.ll = CheckSsf(ssf.ll.list)
> class(ssf.ll)
[1] "ssf"
```

```
> names(ssf.ll)
 [1] "mDelta"  "mPhi"    "mOmega"  "mSigma"  "mJPhi"
 [6] "mJOmega" "mJDelta" "mX"      "cT"      "cX"
[11] "cY"      "cSt"
> ssf.ll
$mPhi:
     [,1]
[1,]    1
[2,]    1

$mOmega:
     [,1] [,2]
[1,] 0.25    0
[2,] 0.00    1

$mSigma:
     [,1]
[1,]   -1
[2,]    0

$mDelta:
     [,1]
[1,]    0
[2,]    0

$mJPhi:
[1] 0

$mJOmega:
[1] 0

$mJDelta:
[1] 0

$mX:
[1] 0

$cT:
[1] 0

$cX:
[1] 0

$cY:
[1] 1
```

```
$cSt:
[1] 1

attr(, "class"):
[1] "ssf"
```

The function `CheckSsf` takes a list variable with a minimum state space form, coerces the components to matrix objects and returns the full parameterization of a state space model used in many of the `S+FinMetrics/SsfPack` state space modeling functions. See the online help for `CheckSsf` for descriptions of the components of an "`ssf`" object.

Example 93 *State space representation of a time varying parameter regression model*

Consider the Capital Asset Pricing Model (CAPM) with time varying intercept and slope

$$\begin{aligned} r_t &= \alpha_t + \beta_t r_{M,t} + \nu_t,\ \nu_t \sim GWN(0, \sigma_\nu^2) && (14.17) \\ \alpha_{t+1} &= \alpha_t + \xi_t,\ \xi_t \sim GWN(0, \sigma_\xi^2) && (14.18) \\ \beta_{t+1} &= \beta_t + \varsigma_t,\ \varsigma_t \sim GWN(0, \sigma_\varsigma^2) && (14.19) \end{aligned}$$

where r_t denotes the return on an asset in excess of the risk free rate, and $r_{M,t}$ denotes the excess return on a market index. In this model, both the abnormal excess return α_t and asset risk β_t are allowed to vary over time following a random walk specification. Let $\boldsymbol{\alpha}_t = (\alpha_t, \beta_t)'$, $y_t = r_t$, $\mathbf{x}_t = (1, r_{M,t})'$, $\mathbf{H}_t$ =diag$(\sigma_\xi, \sigma_\varsigma)'$ and $G_t = \sigma_\nu$. Then the state space form (14.7) of (14.17) - (14.19) is

$$\begin{pmatrix} \boldsymbol{\alpha}_{t+1} \\ y_t \end{pmatrix} = \begin{pmatrix} \mathbf{I}_2 \\ \mathbf{x}_t' \end{pmatrix} \boldsymbol{\alpha}_t + \begin{pmatrix} \mathbf{H}\boldsymbol{\eta}_t \\ G\varepsilon_t \end{pmatrix}$$

and has parameters

$$\boldsymbol{\Phi}_t = \begin{pmatrix} \mathbf{I}_2 \\ \mathbf{x}_t' \end{pmatrix},\ \boldsymbol{\Omega} = \begin{pmatrix} \sigma_\xi^2 & 0 & 0 \\ 0 & \sigma_\varsigma^2 & 0 \\ 0 & 0 & \sigma_\nu^2 \end{pmatrix} \qquad (14.20)$$

Since $\boldsymbol{\alpha}_t$ is $I(1)$ the initial state vector $\boldsymbol{\alpha}_1$ doesn't have finite variance so it is customary to set $\mathbf{a} = \mathbf{0}$ and $\mathbf{P} = \kappa \mathbf{I}_2$ where κ is large. Using (14.11), the initial variance is specified with $\mathbf{P}_* = \mathbf{0}$ and $\mathbf{P}_\infty = \mathbf{I}_2$. Therefore, the initial state matrix (14.10) for the time varying CAPM has the form

$$\boldsymbol{\Sigma} = \begin{pmatrix} -1 & 0 \\ 0 & -1 \\ 0 & 0 \end{pmatrix}$$

The state space parameter matrix $\mathbf{\Phi}_t$ in (14.20) has a time varying system element $\mathbf{Z}_t = \mathbf{x}_t'$. In `S+FinMetrics/SsfPack`, the specification of this time varying element in $\mathbf{\Phi}_t$ requires an index matrix $\mathbf{J}_\Phi$ and a data matrix $\mathbf{X}$ to which the indices in $\mathbf{J}_\Phi$ refer. The index matrix $\mathbf{J}_\Phi$ must have the same dimension as $\mathbf{\Phi}_t$. The elements of $\mathbf{J}_\Phi$ are all set to -1 except the elements for which the corresponding elements of $\mathbf{\Phi}_t$ are time varying. The non-negative index value indicates the column of the data matrix $\mathbf{X}$ which contains the time varying values.

The specification of the state space form for the time varying CAPM requires values for the variances σ_ξ^2, σ_ς^2, and σ_ν^2 as well as a data matrix $\mathbf{X}$ whose rows correspond with $\mathbf{Z}_t = \mathbf{x}_t' = (1, r_{M,t})$. For example, let $\sigma_\xi^2 = (0.01)^2$, $\sigma_\varsigma^2 = (0.05)^2$ and $\sigma_\nu^2 = (0.1)^2$ and construct the data matrix $\mathbf{X}$ using the excess return data in the `S+FinMetrics` "`timeSeries`" `excessReturns.ts`

```
> X.mat = cbind(1,
+ as.matrix(seriesData(excessReturns.ts[,"SP500"]))
```

The state space form may be created using

```
> Phi.t = rbind(diag(2),rep(0,2))
> Omega = diag(c((.01)^2,(.05)^2,(.1)^2))
> J.Phi = matrix(-1,3,2)
> J.Phi[3,1] = 1
> J.Phi[3,2] = 2
> Sigma = -Phi.t
> ssf.tvp.capm = list(mPhi=Phi.t,
+ mOmega=Omega,
+ mJPhi=J.Phi,
+ mSigma=Sigma,
+ mX=X.mat)
> ssf.tvp.capm
$mPhi:
     [,1] [,2]
[1,]    1    0
[2,]    0    1
[3,]    0    0

$mOmega:
       [,1]   [,2] [,3]
[1,] 0.0001 0.0000 0.00
[2,] 0.0000 0.0025 0.00
[3,] 0.0000 0.0000 0.01

$mJPhi:
     [,1] [,2]
```

Parameter index matrix	List component name
$\mathbf{J}_\delta$	`mJDelta`
$\mathbf{J}_\Phi$	`mJPhi`
$\mathbf{J}_\Omega$	`mJOmega`
Time varying component data matrix	**List component name**
$\mathbf{X}$	`mX`

TABLE 14.2. `S+FinMetrics` time varying state space components

```
[1,]   -1   -1
[2,]   -1   -1
[3,]    1    2

$mSigma:
     [,1] [,2]
[1,]   -1    0
[2,]    0   -1
[3,]    0    0

$mX:
numeric matrix: 131 rows, 2 columns.
         SP500
 1 1  0.002803
 2 1  0.017566
...
131 1 -0.0007548
```

Notice in the specification of $\mathbf{\Phi}_t$ the values associated with $\mathbf{x}_t'$ in the third row are set to zero. In the index matrix $\mathbf{J}_\Phi$, the (3,1) element is 1 and the (3,2) element is 2 indicating that the data for the first and second columns of $\mathbf{x}_t'$ come from the first and second columns of the component `mX`, respectively.

In the general state space model (14.7), it is possible that all of the system matrices $\boldsymbol{\delta}_t$, $\mathbf{\Phi}_t$ and $\mathbf{\Omega}_t$ have time varying elements. The corresponding index matrices $\mathbf{J}_\delta$, $\mathbf{J}_\Phi$ and $\mathbf{J}_\Omega$ indicate which elements of the matrices $\boldsymbol{\delta}_t$, $\mathbf{\Phi}_t$ and $\mathbf{\Omega}_t$ are time varying and the data matrix $\mathbf{X}$ contains the time varying components. The naming convention for these components is summarized in Table 14.2.

14.2.3 Missing Values

The `S+FinMetrics/SsfPack` state space modeling functions can handle missing values in the vector of response variables $\mathbf{y}_t$ in (14.3). Missing values are not allowed in the state space system matrices $\mathbf{\Phi}_t$, $\mathbf{\Omega}_t$, $\mathbf{\Sigma}$ and $\boldsymbol{\delta}_t$. Missing values are represented by `NA` in `S-PLUS`.

In the `S+FinMetrics/SsfPack` state space functions, the observation vector $\mathbf{y}_t$ with missing values will be be reduced to the vector $\mathbf{y}_t^\dagger$ without missing values and the measurement equation will be adjusted accordingly. For example, the measurement equation $\mathbf{y}_t = \mathbf{c}_t + \mathbf{Z}_t\boldsymbol{\alpha}_t + \mathbf{G}_t\boldsymbol{\varepsilon}_t$ with

$$\mathbf{y}_t = \begin{pmatrix} 5 \\ NA \\ 3 \\ NA \end{pmatrix},\ \mathbf{c}_t = \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \end{pmatrix},\ \mathbf{Z}_t = \begin{pmatrix} Z_{1,t} \\ Z_{2,t} \\ Z_{3,t} \\ Z_{4,t} \end{pmatrix},\ \mathbf{G}_t = \begin{pmatrix} G_{1,t} \\ G_{2,t} \\ G_{3,t} \\ G_{4,t} \end{pmatrix}$$

reduces to the measurement equation $\mathbf{y}_t^\dagger = \mathbf{c}_t^\dagger + \mathbf{Z}_t^\dagger\boldsymbol{\alpha}_t + \mathbf{G}_t^\dagger\boldsymbol{\varepsilon}_t$ with

$$\mathbf{y}_t^\dagger = \begin{pmatrix} 5 \\ 3 \end{pmatrix},\ \mathbf{c}_t^\dagger = \begin{pmatrix} 1 \\ 3 \end{pmatrix},\ \mathbf{Z}_t^\dagger = \begin{pmatrix} Z_{1,t} \\ Z_{3,t} \end{pmatrix},\ \mathbf{G}_t^\dagger = \begin{pmatrix} G_{1,t} \\ G_{3,t} \end{pmatrix}$$

The *SsfPack* algorithms in `S+FinMetrics` automatically replace the observation vector $\mathbf{y}_t$ with $\mathbf{y}_t^\dagger$ when missing values are encountered and the system matrices are adjusted accordingly.

14.2.4 `S+FinMetrics/SsfPack` Functions for Specifying the State Space Form for Some Common Time Series Models

`S+FinMetrics/SsfPack` has functions for the creation of the state space representation of some common time series models. These functions and models are described in the following sub-sections.

ARMA Models

Following Harvey (1993, Sec. 4.4), the ARMA(p,q) model with zero mean[3]

$$y_t = \phi_1 y_{t-1} + \cdots + \phi_p y_{t-p} + \xi_t + \theta_1\xi_{t-1} + \cdots + \theta_q\xi_{t-q} \tag{14.21}$$

may be put in state space form with transition and measurement equations

$$\begin{aligned} \boldsymbol{\alpha}_{t+1} &= \mathbf{T}\boldsymbol{\alpha}_t + \mathbf{H}\xi_t,\ \xi_t \sim N(0,\sigma_\varepsilon^2) \\ y_t &= \mathbf{Z}\boldsymbol{\alpha}_t \end{aligned}$$

[3]Note that the MA coefficients are specified with positive signs, which is the opposite of how the MA coefficients are specified for models estimated by the `S-PLUS` function `arima.mle`.

and time invariant system matrices

$$\begin{aligned}\mathbf{T} &= \begin{pmatrix} \phi_1 & 1 & 0 & \cdots & 0 \\ \phi_2 & 0 & 1 & & 0 \\ \vdots & & & \ddots & \vdots \\ \phi_{m-1} & 0 & 0 & & 1 \\ \phi_m & 0 & 0 & \cdots & \end{pmatrix}, \mathbf{H} = \begin{pmatrix} 1 \\ \theta_1 \\ \vdots \\ \theta_{m-1} \\ \theta_m \end{pmatrix}, \qquad (14.22) \\ \mathbf{Z} &= \begin{pmatrix} 1 & 0 & \cdots & 0 & 0 \end{pmatrix}\end{aligned}$$

where $\mathbf{d}, \mathbf{c}$ and $\mathbf{G}$ of the state space form (14.1)-(14.3) are all zero and $m = \max(p, q+1)$. The state vector $\boldsymbol{\alpha}_t$ has the form

$$\boldsymbol{\alpha}_t = \begin{pmatrix} y_t \\ \phi_2 y_{t-1} + \cdots + \phi_p y_{t-m+1} + \theta_1 \xi_t + \cdots + \theta_{m-1} \xi_{t-m+2} \\ \phi_3 y_{t-1} + \cdots + \phi_p y_{t-m+2} + \theta_2 \xi_t + \cdots + \theta_{m-1} \xi_{t-m+3} \\ \vdots \\ \phi_m y_{t-1} + \theta_{m-1} \xi_t \end{pmatrix} \qquad (14.23)$$

In compact state space form (14.7), the model is

$$\begin{aligned}\begin{pmatrix} \boldsymbol{\alpha}_{t+1} \\ y_t \end{pmatrix} &= \begin{pmatrix} \mathbf{T} \\ \mathbf{Z} \end{pmatrix} \boldsymbol{\alpha}_t + \begin{pmatrix} \mathbf{H} \\ 0 \end{pmatrix} \xi_t \\ &= \mathbf{\Phi} \boldsymbol{\alpha}_t + \mathbf{u}_t\end{aligned}$$

and

$$\mathbf{\Omega} = \begin{pmatrix} \sigma_\xi^2 \mathbf{H}\mathbf{H}' & 0 \\ 0 & 0 \end{pmatrix}$$

If y_t is stationary then $\boldsymbol{\alpha}_t \sim N(0, \mathbf{V})$ is the unconditional distribution of the state vector, and the covariance matrix $\mathbf{V}$ satisfies $\mathbf{V} = \mathbf{T}\mathbf{V}\mathbf{T}' + \boldsymbol{\sigma}_\xi^2 \mathbf{H}\mathbf{H}'$, which can be solved for the elements of $\mathbf{V}$. The initial value parameters are then

$$\mathbf{\Sigma} = \begin{pmatrix} \mathbf{V} \\ \mathbf{0}' \end{pmatrix}$$

The `S+FinMetrics/SsfPack` function `GetSsfArma` creates the state space system matrices for any univariate stationary and invertible ARMA model. The arguments expected by `GetSsfArma` are

```
> args(GetSsfArma)
function(ar = NULL, ma = NULL, sigma = 1, model = NULL)
```

where `ar` is the vector of p AR coefficients, `ma` is the vector of q MA coefficients, `sigma` is the innovation standard deviation σ_ξ, and `model` is a list with components giving the AR, MA and innovation standard deviation. If the arguments `ar`, `ma,` and `sigma` are specified, then `model` is ignored. The function `GetSsfArma` returns a list containing the system matrices $\mathbf{\Phi}$, $\mathbf{\Omega}$ and the initial value parameters $\mathbf{\Sigma}$.

Example 94 *AR(1) and ARMA(2,1)*

Consider the AR(1) model

$$y_t = 0.75y_{t-1} + \xi_t,\ \xi_t \sim GWN(0, (0.5)^2)$$

The state space form may be computed using

```
> ssf.ar1 = GetSsfArma(ar=0.75,sigma=.5)
> ssf.ar1
$mPhi:
     [,1]
[1,] 0.75
[2,] 1.00

$mOmega:
     [,1] [,2]
[1,] 0.25    0
[2,] 0.00    0

$mSigma:
       [,1]
[1,] 0.5714
[2,] 0.0000
```

In the component `mSigma`, the unconditional variance of the initial state α_1 is computed as $\text{var}(\alpha_1) = (0.5)^2/(1 - 0.75^2) = 0.5714$.

Next, consider the ARMA(2,1) model

$$y_t = 0.6y_{t-1} + 0.2y_{t-2} + \varepsilon_t - 0.2\varepsilon_{t-1},\ \varepsilon_t \sim GWN(0, 0.9)$$

The state space system matrices may be computed using

```
> arma21.mod = list(ar=c(0.6,0.2),ma=c(-0.2),sigma=sqrt(0.9))
> ssf.arma21 = GetSsfArma(model=arma21.mod)
> ssf.arma21
$mPhi:
     [,1] [,2]
[1,]  0.6    1
[2,]  0.2    0
[3,]  1.0    0

$mOmega:
      [,1]   [,2] [,3]
[1,]  0.90 -0.180    0
[2,] -0.18  0.036    0
[3,]  0.00  0.000    0
```

```
$mSigma:
        [,1]    [,2]
[1,] 1.58571 0.01286
[2,] 0.01286 0.09943
[3,] 0.00000 0.00000
```

The unconditional variance of the initial state vector $\boldsymbol{\alpha}_1 = (\alpha_{11}, \alpha_{12})'$ is in the top 2×2 block of `mSigma` and is

$$\text{var}(\boldsymbol{\alpha}_1) = \begin{pmatrix} 1.586 & 0.013 \\ 0.013 & 0.099 \end{pmatrix}.$$

Structural Time Series Models

The basic univariate unobserved components *structural time series model* (STSM) for a time series y_t has the form

$$y_t = \mu_t + \gamma_t + \psi_t + \xi_t \tag{14.24}$$

where μ_t represents the unobserved *trend* component, γ_t represents the unobserved *seasonal* component, ψ_t represents the unobserved *cycle* component, and ξ_t represents the unobserved *irregular* component.

The nonstationary trend component μ_t has the form of a *local linear trend*:

$$\mu_{t+1} = \mu_t + \beta_t + \eta_t,\ \eta_t \sim GWN(0, \sigma_\eta^2) \tag{14.25}$$
$$\beta_{t+1} = \beta_t + \varsigma_t,\ \varsigma_t \sim GWN(0, \sigma_\varsigma^2) \tag{14.26}$$

with $\mu_1 \sim N(0, \kappa)$ and $\beta_1 \sim N(0, \kappa)$ where κ is a large number, e.g. $\kappa = 10^6$. If $\sigma_\varsigma^2 = 0$ then μ_t follows a random walk with drift β_1. If both $\sigma_\varsigma^2 = 0$ and $\sigma_\eta^2 = 0$ then μ_t follows a linear deterministic trend.

The stochastic seasonal component γ_t has the form

$$\begin{aligned} S(L)\gamma_t &= \omega_t,\ \omega_t \sim GWN(0, \sigma_\omega^2) \\ S(L) &= 1 + L + \cdots + L^{s-1} \end{aligned} \tag{14.27}$$

where s gives the number of seasons. When $\sigma_\omega^2 = 0$, the seasonal component becomes fixed.

The stochastic cycle component ψ_t is specified as

$$\begin{pmatrix} \psi_{t+1} \\ \psi_{t+1}^* \end{pmatrix} = \rho \begin{pmatrix} \cos\lambda_c & \sin\lambda_c \\ -\sin\lambda_c & \cos\lambda_c \end{pmatrix} \begin{pmatrix} \psi_t \\ \psi_t^* \end{pmatrix} + \begin{pmatrix} \chi_t \\ \chi_t^* \end{pmatrix}, \tag{14.28}$$
$$\begin{pmatrix} \chi_t \\ \chi_t^* \end{pmatrix} \sim GWN\left(\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \sigma_\psi^2 (1-\rho^2) \mathbf{I}_2 \right)$$

where $\psi_0 \sim N(0, \sigma_\psi^2)$, $\psi_0^* \sim N(0, \sigma_\psi^2)$ and $cov(\psi_0, \psi_0^*) = 0$. The parameter $\rho \in (0, 1]$ is interpreted as a damping factor. The frequency of the cycle

Argument	STSM parameter
`irregular`	σ_η
`level`	σ_ξ
`slope`	σ_ς
`seasonalDummy`	σ_ω, s
`seasonalTrig`	σ_ω, s
`seasonalHS`	σ_ω, s
`cycle0`	$\sigma_\psi, \lambda_c, \rho$
$\vdots$	$\vdots$
`cycle9`	$\sigma_\psi, \lambda_c, \rho$

TABLE 14.3. Arguments to the `S+FinMetrics` function `GetSsfStsm`

is $\lambda_c = 2\pi/c$ and c is the period. When $\rho = 1$ the cycle reduces to a deterministic sine-cosine wave.

The `S+FinMetrics/SsfPack` function `GetSsfStsm` creates the state space system matrices for the univariate structural time series model (14.24). The arguments expected by `GetSsfStsm` are

```
> args(GetSsfStsm)
function(irregular = 1, level = 0.1, slope = NULL,
   seasonalDummy = NULL, seasonalTrig = NULL, seasonalHS
   = NULL, cycle0 = NULL, cycle1 = NULL, cycle2 = NULL,
   cycle3 = NULL, cycle4 = NULL, cycle5 = NULL, cycle6 =
   NULL, cycle7 = NULL, cycle8 = NULL, cycle9 = NULL)
```

These arguments are explained in Table 14.3.

Example 95 *Local level model*

The state space for the local level model (14.12)-(14.14) may be constructed using

```
> ssf.stsm = GetSsfStsm(irregular=1, level=0.5)
> class(ssf.stsm)
[1] "list"
> ssf.stsm
$mPhi:
     [,1]
[1,]    1
[2,]    1

$mOmega:
     [,1] [,2]
[1,] 0.25    0
[2,] 0.00    1
```

```
$mSigma:
     [,1]
[1,]   -1
[2,]    0
```

The arguments `irregular=1` and `level=0.5` specify $\sigma_\varepsilon = 1$ and $\sigma_\eta = 0.5$ in (14.13) and (14.12), respectively.

Regression Models

The linear regression model

$$y_t = \mathbf{x}_t'\boldsymbol{\beta} + \xi_t,\ \xi_t \sim GWN(0, \sigma_\xi^2),$$

where $\mathbf{x}_t$ is a $k \times 1$ data matrix and $\boldsymbol{\beta}$ is a $k \times 1$ fixed parameter vector, may be put in the state space

$$\begin{pmatrix} \boldsymbol{\alpha}_{t+1} \\ y_t \end{pmatrix} = \begin{pmatrix} \mathbf{I}_k \\ \mathbf{x}_t' \end{pmatrix} \boldsymbol{\alpha}_t + \begin{pmatrix} \mathbf{0} \\ \sigma_\xi \varepsilon_t \end{pmatrix} \tag{14.29}$$

The state vector satisfies $\boldsymbol{\alpha}_{t+1} = \boldsymbol{\alpha}_t = \boldsymbol{\beta}$. The state space system matrices are $\mathbf{T}_t = \mathbf{I}_k$, $\mathbf{Z}_t = \mathbf{x}_t'$, $\mathbf{G}_t = \sigma_\xi$ and $\mathbf{H}_t = 0$. The coefficient vector $\boldsymbol{\beta}$ is fixed and unknown so that the initial conditions are $\boldsymbol{\alpha}_1 \sim N(\mathbf{0}, \kappa\mathbf{I}_k)$ where κ is large. An advantage of analyzing the linear regression model in state space form is that recursive least squares estimates of the regression coefficient vector $\boldsymbol{\beta}$ are readily computed. Another advantage is that it is straightforward to allow some or all of the regression coefficients to be time varying.

The linear regression model with time varying parameters may be introduced by setting $\mathbf{H}_t$ not equal to zero in (14.29). For example, to allow all regressors to evolve as random walks set

$$\mathbf{H}_t = \begin{pmatrix} \sigma_{\beta_1} \\ \vdots \\ \sigma_{\beta_k} \end{pmatrix}$$

so that the state equation becomes

$$\boldsymbol{\alpha}_{t+1} = \boldsymbol{\alpha}_t + \mathbf{H}_t \boldsymbol{\eta}_t \tag{14.30}$$

More explicitly, since $\alpha_{i,t+1} = \alpha_{i,t} = \beta_{i,t}$ the state equation (14.30) implies

$$\beta_{i,t+1} = \beta_{i,t} + \sigma_{\beta_i} \cdot \eta_{i,t},\ i = 1, \ldots, k$$

If $\sigma_{\beta_i} = 0$ then β_i is constant.

The `S+FinMetrics/SsfPack` function `GetSsfReg` creates the state space system matrices for the linear regression model. The arguments expected by `GetSsfReg` are

```
> args(GetSsfReg)
function(mX)
```

where `mX` is a rectangular data object which represents the regressors in the model. The function `GetSsfReg` returns a list with components describing the minimal state space representation of the linear regression model.

Example 96 *Time trend regression model*

Consider the time trend regression model

$$y_t = \mu + \delta t + \xi_t,\ \xi_t \sim GWN(0, \sigma_\xi^2)$$

The state space form for a sample of size $n = 10$ is computed using

```
> ssf.reg = GetSsfReg(cbind(1,1:10))
> class(ssf.reg)
[1] "list"
> names(ssf.reg)
[1] "mPhi"   "mOmega" "mSigma" "mJPhi"  "mX"
> ssf.reg
$mPhi:
     [,1] [,2]
[1,]    1    0
[2,]    0    1
[3,]    0    0

$mOmega:
     [,1] [,2] [,3]
[1,]    0    0    0
[2,]    0    0    0
[3,]    0    0    1

$mSigma:
     [,1] [,2]
[1,]   -1    0
[2,]    0   -1
[3,]    0    0

$mJPhi:
     [,1] [,2]
[1,]   -1   -1
[2,]   -1   -1
[3,]    1    2

$mX:
      [,1] [,2]
```

```
 [1,]    1    1
 [2,]    1    2
 [3,]    1    3
 [4,]    1    4
 [5,]    1    5
 [6,]    1    6
 [7,]    1    7
 [8,]    1    8
 [9,]    1    9
[10,]    1   10
```

Since the system matrix $\mathbf{Z}_t = \mathbf{x}_t'$, the parameter $\mathbf{\Phi}_t$ is time varying and the index matrix $\mathbf{J}_\Phi$, represented by the component `mJPhi`, determines the association between the time varying data in $\mathbf{Z}_t$ and the data supplied in the component `mX`.

To specify a time trend regression with a time varying slope of the form

$$\delta_{t+1} = \delta_t + \zeta_t,\ \zeta_t \sim GWN(0, \sigma_\zeta^2) \tag{14.31}$$

one needs to specify a non-zero value for σ_ζ^2 in $\mathbf{\Omega}_t$. For example, if $\sigma_\zeta^2 = 0.5$ then

```
> ssf.reg.tvp = ssf.reg
> ssf.reg.tvp$mOmega[2,2] = 0.5
> ssf.reg.tvp$mOmega
     [,1] [,2] [,3]
[1,]    0  0.0    0
[2,]    0  0.5    0
[3,]    0  0.0    1
```

modifies the state space form for the time trend regression to allow a time varying slope of the form (14.31).

Regression Models with ARMA Errors

The ARMA(p, q) models created with `GetSsfArma` do not have deterministic terms (e.g., constant, trend, seasonal dummies) or exogenous regressors and are therefore limited. The general ARMA(p, q) model with exogenous regressors has the form

$$y_t = \mathbf{x}_t'\boldsymbol{\beta} + \xi_t$$

where ξ_t follows an ARMA(p, q) process of the form (14.21). Let $\boldsymbol{\alpha}_t$ be defined as in (14.23) and let

$$\boldsymbol{\alpha}_t^* = \begin{pmatrix} \boldsymbol{\alpha}_t \\ \boldsymbol{\beta}_t \end{pmatrix} \tag{14.32}$$

where $\boldsymbol{\beta}_t = \boldsymbol{\beta}$. Writing the state equation implied by (14.32) as $\boldsymbol{\alpha}_{t+1}^* = \mathbf{T}^*\boldsymbol{\alpha}_t^* + \mathbf{H}^*\boldsymbol{\eta}_t$ and let

$$\begin{aligned} \mathbf{T}^* &= \begin{bmatrix} \mathbf{T} & 0 \\ 0 & \mathbf{I}_k \end{bmatrix}, \ \mathbf{H}^* = \begin{bmatrix} \mathbf{H} \\ \mathbf{0} \end{bmatrix}, \\ \mathbf{Z}_t^* &= \begin{pmatrix} 1 & 0 & \ldots & 0 & \mathbf{x}_t' \end{pmatrix} \end{aligned}$$

where $\mathbf{T}$ and $\mathbf{H}$ are defined in (14.23). Then the state space form of the regression model with ARMA errors is

$$\begin{pmatrix} \boldsymbol{\alpha}_{t+1} \\ y_t \end{pmatrix} = \begin{pmatrix} \mathbf{T}^* \\ \mathbf{Z}_t^* \end{pmatrix} \boldsymbol{\alpha}_t^* + \begin{pmatrix} \mathbf{H}^*\boldsymbol{\eta}_t \\ 0 \end{pmatrix}$$

The `S+FinMetrics/SsfPack` function `GetSsfRegArma` creates the state space system matrices for the linear regression model with ARMA errors. The arguments expected by `GetSsfRegArma` are

```
> args(GetSsfRegArma)
function(mX, ar = NULL, ma = NULL, sigma = 1, model = NULL)
```

where `mX` is a rectangular data object which represents the regressors in the model, and the remaining arguments are the same as those for `GetSsfArma`. The function `GetSsfRegArma` returns a list with components describing the minimal state space representation of the linear regression model.

Example 97 *Time trend regression with AR(2) errors*

The state space form of the time trend regression with AR(2) errors

$$\begin{aligned} y_t &= \mu + \delta t + \xi_t, \\ \xi_t &= \phi_1\xi_{t-1} + \phi_2\xi_{t-2} + \nu_t, \ \nu_t \sim GWN(0, \sigma_\xi^2) \end{aligned}$$

may be computed using

```
> ssf.reg.ar2 = GetSsfRegArma(cbind(1,1:10),
+ ar=c(1.25,-0.5))
> ssf.reg.ar2
$mPhi:
      [,1] [,2] [,3] [,4]
[1,]  1.25    1    0    0
[2,] -0.50    0    0    0
[3,]  0.00    0    1    0
[4,]  0.00    0    0    1
[5,]  1.00    0    0    0

$mOmega:
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    0    0    0    0
```

```
[2,]    0    0    0    0    0
[3,]    0    0    0    0    0
[4,]    0    0    0    0    0
[5,]    0    0    0    0    0

$mSigma:
        [,1]   [,2] [,3] [,4]
[1,]  4.364 -1.818    0    0
[2,] -1.818  1.091    0    0
[3,]  0.000  0.000   -1    0
[4,]  0.000  0.000    0   -1
[5,]  0.000  0.000    0    0

$mJPhi:
     [,1] [,2] [,3] [,4]
[1,]   -1   -1   -1   -1
[2,]   -1   -1   -1   -1
[3,]   -1   -1   -1   -1
[4,]   -1   -1   -1   -1
[5,]   -1   -1    1    2

$mX:
      [,1] [,2]
 [1,]    1    1
 [2,]    1    2
...
[10,]    1   10
```

Nonparametric Cubic Spline Model

Suppose the continuous time process $y(t)$ is observed at discrete time points $t_1, \ldots, t_n$. Define $\delta_i = t_i - t_{i-1} \geq 0$ as the time duration between observations. The goal of the *nonparametric cubic spline model* is to estimate a smooth signal $\mu(t)$ from $y(t)$ using

$$y(t) = \mu(t) + \varepsilon(t)$$

where $\varepsilon(t)$ is a stationary error. The signal $\mu(t)$ is extracted by minimizing

$$\sum_{i=1}^{n}(y(t_i) - \mu(t_i))^2 + q^{-1} \int \left(\frac{\partial^2 \mu(t)}{\partial t^2} \right)^2 dt$$

where the second term is a penalty term that forces $\mu(t)$ to be "smooth"[4], and q may be interpreted as a signal-to-noise ratio. The resulting function $\mu(t)$ may be expressed as the structural time series model

$$\begin{aligned} \mu(t_{i+1}) &= \mu(t_i) + \delta_i \beta(t_i) + \eta(t_i) \\ \beta(t_{i+1}) &= \beta(t_i) + \varsigma(t_i) \end{aligned} \tag{14.33}$$

where

$$\begin{pmatrix} \eta(t_i) \\ \varsigma(t_i) \end{pmatrix} \text{~} N \left[\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \sigma_\varsigma^2 \delta_i \begin{pmatrix} \frac{1}{3}\delta_i^2 & \frac{1}{2}\delta_i \\ \frac{1}{2}\delta_i & 1 \end{pmatrix} \right]$$

Combining (14.33) with the measurement equation

$$y(t_i) = \mu(t_i) + \varepsilon(t_i)$$

where $\varepsilon(t_i) \sim N(0, \sigma_\varepsilon^2)$ and is independent of $\eta(t_i)$ and $\varsigma(t_i)$, gives the state space form for the nonparametric cubic spline model. The state space system matrices are

$$\mathbf{\Phi}_t = \begin{pmatrix} 1 & \delta_i \\ 0 & 1 \\ 1 & 0 \end{pmatrix}, \ \mathbf{\Omega}_t = \begin{pmatrix} \frac{q\delta_t^2}{3} & \frac{q\delta_t^2}{2} & 0 \\ \frac{q\delta_t^2}{2} & q\delta_t & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

When the observations are equally spaced, δ_i is constant and the above system matrices are time invariant.

The `S+FinMetrics/SsfPack` function `GetSsfSpline` creates the state space system matrices for the nonparametric cubic spline model. The arguments expected by `GetSsfSpline` are

```
> args(GetSsfSpline)
function(snr = 1, delta = 0)
```

where `snr` is the signal-to-noise ratio q, and `delta` is a numeric vector containing the time durations δ_i $(i = 1, \ldots, n)$. If `delta=0` then δ_i is assumed to be equal to unity and the time invariant form of the model is created. The function `GetSsfSpline` returns a list with components describing the minimal state space representation of the nonparametric cubic spline model.

Example 98 *State space form of non-parametric cubic spline model*

The default non-parametric cubic spline model with $\delta_t = 1$ is created using

[4]This process can be interpreted as an interpolation technique and is similar to the technique used in the `S+FinMetrics` functions `interpNA` and `hpfilter`. See also the smoothing spline method described in chapter sixteen.

```
> GetSsfSpline()
$mPhi:
     [,1] [,2]
[1,]    1    1
[2,]    0    1
[3,]    1    0

$mOmega:
       [,1] [,2] [,3]
[1,] 0.3333  0.5    0
[2,] 0.5000  1.0    0
[3,] 0.0000  0.0    1

$mSigma:
     [,1] [,2]
[1,]   -1    0
[2,]    0   -1
[3,]    0    0

$mJPhi:
NULL

$mJOmega:
NULL

$mX:
NULL
```

For unequally spaced observations

```
> t.vals = c(2,3,5,9,12,17,20,23,25)
> delta.t = diff(t.vals)
```

and $q = 0.2$, the state space form is

```
> GetSsfSpline(snr=0.2,delta=delta.t)
$mPhi:
     [,1] [,2]
[1,]    1    1
[2,]    0    1
[3,]    1    0

$mOmega:
        [,1] [,2] [,3]
[1,] 0.06667  0.1    0
[2,] 0.10000  0.2    0
[3,] 0.00000  0.0    1
```

```
$mSigma:
     [,1] [,2]
[1,]   -1    0
[2,]    0   -1
[3,]    0    0

$mJPhi:
     [,1] [,2]
[1,]   -1    1
[2,]   -1   -1
[3,]   -1   -1

$mJOmega:
     [,1] [,2] [,3]
[1,]    4    3   -1
[2,]    3    2   -1
[3,]   -1   -1   -1

$mX:
        [,1]   [,2]  [,3] [,4]  [,5] [,6] [,7]   [,8]
[1,] 1.00000 2.0000 4.000  3.0 5.000  3.0  3.0 2.0000
[2,] 0.20000 0.4000 0.800  0.6 1.000  0.6  0.6 0.4000
[3,] 0.10000 0.4000 1.600  0.9 2.500  0.9  0.9 0.4000
[4,] 0.06667 0.5333 4.267  1.8 8.333  1.8  1.8 0.5333
```

14.2.5 Simulating Observations from the State Space Model

Once a state space model has been specified, it is often interesting to draw simulated values from the model. The S+FinMetrics/SsfPack function SsfSim may be used for such a purpose. The arguments expected from SsfSim are

```
> args(SsfSim)
function(ssf, n = 100, mRan = NULL, a1 = NULL)
```

where ssf represents either a list with components giving a minimal state space form or a valid "ssf" object, n is the number of simulated observations, mRan is user-specified matrix of disturbances, and a1 is the initial state vector. The use of SsfSim is illustrated with the following examples.

Example 99 *Simulating observations from the local level model*

To generate 250 observations on the state variable α_{t+1} and observations y_t in the local level model (14.12) - (14.14) use

```
> set.seed(112)
```

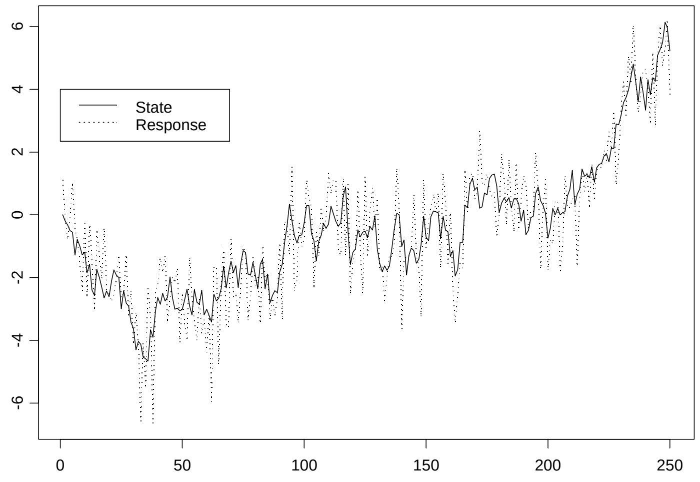


FIGURE 14.1. Simulated values from local level model created using the `S+FinMetrics` function `SsfSim`.

```
> ll.sim = SsfSim(ssf.ll.list,n=250)
> class(ll.sim)
[1] "matrix"
> colIds(ll.sim)
[1] "state"    "response"
```

The function `SsfSim` returns a matrix containing the simulated state variables α_{t+1} and observations y_t. These values are illustrated in Figure 14.1 created using

```
> tsplot(ll.sim)
> legend(0,4,legend=c("State","Response"),lty=1:2)
```

Example 100 *Simulating observations from CAPM with time varying parameters*

When simulating observations from a state space form with a data matrix component `mX` using the function `SsfSim`, the number of simulated values must match the number of rows of `mX`. The state space form for the CAPM with time varying parameters created earlier uses a data matrix `mX` with $n = 131$ observations

```
> nrow(ssf.tvp.capm$mX)
[1] 131
```

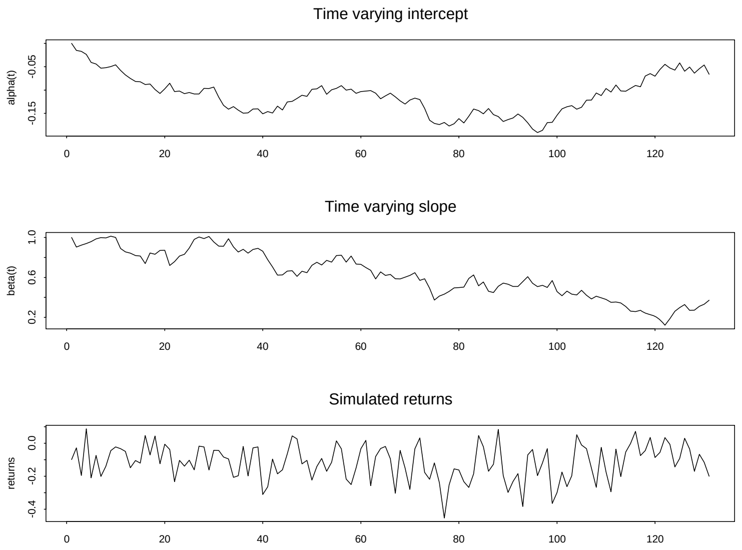


FIGURE 14.2. Simulated state and response values from the CAPM with time varying parameters state space form `ssf.tvp`.

The state variables are the time varying intercept α_{t+1} and the time varying β_{t+1}. Natural initial values for these parameters are $\alpha_1 = 0$ and $\beta_1 = 1$. Using these initial values, $n = 131$ observations are generated from the CAPM with time varying parameters using

```
> set.seed(444)
> tvp.sim = SsfSim(ssf.tvp.capm,n=nrow(X.mat),a1=c(0,1))
> colIds(tvp.sim)
[1] "state.1"  "state.2"  "response"
```

The simulated state and response variables are illustrated in Figure 14.2 created using

```
> par(mfrow=c(3,1))
> tsplot(tvp.sim[,"state.1"],main="Time varying intercept",
+ ylab="alpha(t)")
> tsplot(tvp.sim[,"state.2"],main="Time varying slope",
+ ylab="beta(t)")
> tsplot(tvp.sim[,"response"],main="Simulated returns",
+ ylab="returns")
```

14.3 Algorithms

14.3.1 Kalman Filter

The Kalman filter is a recursive algorithm for the evaluation of moments of the normally distributed state vector $\boldsymbol{\alpha}_{t+1}$ conditional on the observed data $\mathbf{Y}_t = (y_1, \ldots, y_t)$. To describe the algorithm, let $\mathbf{a}_t = E[\boldsymbol{\alpha}_t|\mathbf{Y}_{t-1}]$ denote the conditional mean of $\boldsymbol{\alpha}_t$ based on information available at time $t-1$ and let $\mathbf{P}_t = \text{var}(\boldsymbol{\alpha}_t|\mathbf{Y}_{t-1})$ denote the conditional variance of α_t.

The *filtering* or *updating* equations of the Kalman filter compute $\mathbf{a}_{t|t} = E[\boldsymbol{\alpha}_t|\mathbf{Y}_t]$ and $\mathbf{P}_{t|t} = \text{var}(\boldsymbol{\alpha}_t|\mathbf{Y}_t)$ using

$$\mathbf{a}_{t|t} = \mathbf{a}_t + \mathbf{K}_t\mathbf{v}_t \tag{14.34}$$

$$\mathbf{P}_{t|t} = \mathbf{P}_t - \mathbf{P_t Z_t'}\mathbf{K}_t' \tag{14.35}$$

where

$$\mathbf{v}_t = \mathbf{y}_t - \mathbf{c}_t - \mathbf{Z}_t\mathbf{a}_t \tag{14.36}$$

$$\mathbf{F}_t = \mathbf{Z}_t\mathbf{P}_t\mathbf{Z}_t' + \mathbf{G}_t\mathbf{G}_t' \tag{14.37}$$

$$\mathbf{K}_t = \mathbf{P}_t\mathbf{Z}_t'\mathbf{F}_t^{-1} \tag{14.38}$$

The variable $\mathbf{v}_t$ is the *measurement equation innovation* or prediction error, $\mathbf{F}_t = \text{var}(\mathbf{v}_t)$ and $\mathbf{K}_t$ is the *Kalman gain* matrix.

The *prediction* equations of the Kalman filter compute $\mathbf{a}_{t+1}$ and $\mathbf{P}_{t+1}$ using

$$\mathbf{a}_{t+1} = \mathbf{T}_t\mathbf{a}_{t|t} \tag{14.39}$$

$$\mathbf{P}_{t+1} = \mathbf{T}_t\mathbf{P}_{t|t}\mathbf{T}_t' + \mathbf{H}_t\mathbf{H}_t' \tag{14.40}$$

In the Kalman filter recursions, if there are missing values in $\mathbf{y}_t$ then $\mathbf{v}_t = \mathbf{0}$, $\mathbf{F}_t^{-1} = \mathbf{0}$ and $\mathbf{K}_t = \mathbf{0}$. This allows out-of-sample forecasts of $\boldsymbol{\alpha}_t$ and $\mathbf{y}_t$ to be computed from the updating and prediction equations.

14.3.2 Kalman Smoother

The Kalman filtering algorithm is a forward recursion which computes one-step ahead estimates $\mathbf{a}_{t+1}$ and $\mathbf{P}_{t+1}$ based on $\mathbf{Y}_t$ for $t = 1, \ldots, n$. The *Kalman smoothing* algorithm is a backward recursion which computes the mean and variance of specific conditional distributions based on the full data set $\mathbf{Y}_n = (y_1, \ldots, y_n)$. The *smoothing equations* are

$$\mathbf{r}_t^* = \mathbf{T}_t'\mathbf{r}_t,\ \mathbf{N}_t^* = \mathbf{T}_t\mathbf{N}_t\mathbf{T}_t',\ \mathbf{K}_t^* = \mathbf{N}_t^*\mathbf{K}_t \tag{14.41}$$

$$\mathbf{e}_t = \mathbf{F}_t^{-1}\mathbf{v}_t - \mathbf{K}_t'\mathbf{r}_t^*,\ \mathbf{D}_t = \mathbf{F}_t^{-1} + \mathbf{K}_t\mathbf{K}_t^{*\prime}$$

and the *backwards updating equations* are

$$\mathbf{r}_{t-1} = \mathbf{Z}_t'\mathbf{e}_t + \mathbf{r}_t^*,\ \mathbf{N}_{t-1} = \mathbf{Z}_t'\mathbf{D}_t\mathbf{Z}_t - < \mathbf{K}_t^*\mathbf{Z}_t > + \mathbf{N}_t^* \tag{14.42}$$

for $t = n, \ldots, 1$ with initializations $\mathbf{r}_n = 0$ and $\mathbf{N}_n = 0$. For any square matrix $\mathbf{A}$, the operator $< \mathbf{A} >= \mathbf{A} + \mathbf{A}'$. The values $\mathbf{r}_t$ are called *state smoothing residuals* and the values $\mathbf{e}_t$ are called *response smoothing residuals.* The recursions (14.41) and (14.42) are somewhat non-standard. Durbin and Koopman (2001) show how they may be re-expressed in more standard form.

14.3.3 Smoothed State and Response Estimates

The smoothed estimates of the state vector $\boldsymbol{\alpha}_t$ and its variance matrix are denoted $\hat{\boldsymbol{\alpha}}_t = E[\boldsymbol{\alpha}_t|\mathbf{Y}_n]$ and $\text{var}(\hat{\boldsymbol{\alpha}}_t|\mathbf{Y}_n)$, respectively. The smoothed estimate $\hat{\boldsymbol{\alpha}}_t$ is the optimal estimate of $\boldsymbol{\alpha}_t$ using all available information $\mathbf{Y}_n$, whereas the filtered estimate $\hat{\mathbf{a}}_{t|t}$ is the optimal estimate only using information available at time $t, \mathbf{Y}_t$. The computation of $\hat{\boldsymbol{\alpha}}_t$ and its variance from the Kalman smoother algorithm is described in Durbin and Koopman (2001).

The smoothed estimate of the response $\mathbf{y}_t$ and its variance are computed using

$$\begin{aligned} \hat{\mathbf{y}}_t &= \hat{\boldsymbol{\theta}}_t = E[\boldsymbol{\theta}_t|\mathbf{Y}_n] = \mathbf{c}_t + \mathbf{Z}_t\hat{\boldsymbol{\alpha}}_t \\ \text{var}(\hat{\mathbf{y}}_t|\mathbf{Y}_n) &= \mathbf{Z}_t\text{var}(\hat{\boldsymbol{\alpha}}_t|\mathbf{Y}_n)\mathbf{Z}_t' \end{aligned}$$

14.3.4 Smoothed Disturbance Estimates

The smoothed disturbance estimates are the estimates of the measurement equations innovations $\boldsymbol{\varepsilon}_t$ and transition equation innovations $\boldsymbol{\eta}_t$ based on all available information $\mathbf{Y}_n$, and are denoted $\hat{\boldsymbol{\varepsilon}}_t = E[\boldsymbol{\varepsilon}_t|\mathbf{Y}_n]$ and $\hat{\boldsymbol{\eta}}_t = E[\boldsymbol{\eta}_t|\mathbf{Y}_n]$, respectively. The computation of $\hat{\boldsymbol{\varepsilon}}_t$ and $\hat{\boldsymbol{\eta}}_t$ from the Kalman smoother algorithm is described in Durbin and Koopman (2001). These smoothed disturbance estimates are useful for parameter estimation by maximum likelihood and for diagnostic checking. See chapter seven in Durbin and Koopman (2001) for details.

14.3.5 Forecasting

The Kalman filter prediction equations (14.39) - (14.40) produces one-step ahead predictions of the state vector, $\mathbf{a}_{t+1} = E[\boldsymbol{\alpha}_{t+1}|\mathbf{Y}_t]$, along with prediction variance matrices $\mathbf{P}_{t+1}$. Out of sample predictions, together with associated mean square errors, can be computed from the Kalman filter prediction equations by extending the data set $\mathbf{y}_1, \ldots, \mathbf{y}_n$ with a set of missing values. When y_τ is missing, the Kalman filter reduces to the prediction step described above. As a result, a sequence of m missing values at the end of the sample will produce a set of h-step ahead forecasts for $h = 1, \ldots, m$.

14.3.6 S+FinMetrics/SsfPack Implementation of State Space Modeling Algorithms

The S+FinMetrics/SsfPack function KalmanFil implements the Kalman filter forward recursions in a computationally efficient way, see Koopman, Shephard and Doornik (2001). It produces an object of class "KalmanFil" for which there are print and plot methods. The S+FinMetrics/SsfPack function KalmanSmo computes the Kalman smoother backwards recursions, and produces an object of class "KalmanSmo" for which there are print and plot methods. The functions KalmanFil and KalmanSmo are primarily used by other S+FinMetrics/SsfPack state space functions that require the output from the Kalman filter and Kalman smoother.

Filtered and smoothed estimates of $\boldsymbol{\alpha}_t$ and $\mathbf{y}_t$, with estimated variances, as well as smoothed estimates of $\boldsymbol{\varepsilon}_t$ and $\boldsymbol{\eta}_t$, with estimated variances, are computed using the S+FinMetrics/SsfPack function SsfMomentEst. The result of SsfMomentEst is an object of class "SsfMomentEst" for which there is only a plot method. The function SsfMomentEst may also be used to compute out-of-sample forecasts and forecast variances of $\boldsymbol{\alpha}_t$ and y_t.

The use of the S+FinMetrics/SsfPack functions for implementing the state space algorithms are illustrated with the following examples.

Example 101 *State space algorithms applied to local level model*

Consider the simulated data for the local level model (14.12) - (14.14) in the object ll.sim computed earlier. The response variable y_t is extracted using

```
> y.ll = ll.sim[,"response"]
```

Kalman Filter

The Kalman filter recursions for the simulated data from the local level model are obtained using the S+FinMetrics/SsfPack function KalmanFil with the optional argument task="STFIL" (which stands for state filtering)

```
> KalmanFil.ll = KalmanFil(y.ll,ssf.ll,task="STFIL")
> class(KalmanFil.ll)
[1] "KalmanFil"
```

The function KalmanFil takes as input a vector of response data and either a list describing the minimal state space form or a valid "ssf" object. The result of KalmanFil is an object of class "KalmanFil" with components

```
> names(KalmanFil.ll)
 [1] "mOut"         "innov"        "std.innov"
 [4] "mGain"        "loglike"      "loglike.conc"
 [7] "dVar"         "mEst"         "mOffP"
[10] "task"         "err"          "call"
```

A complete explanation of the components of a “KalmanFil” object is given in the online help for KalmanFil. These components are mainly used by other S+FinMetrics/SsfPack functions and are only briefly discussed here. The component mOut contains the basic Kalman filter output.

```
> KalmanFil.ll$mOut
numeric matrix: 250 rows, 3 columns.
          [,1]   [,2]   [,3]
 [1,]  0.00000 1.0000 0.0000
 [2,] -1.28697 0.5556 0.4444
...
[250,] -1.6371 0.3904 0.6096
```

The first column of mOut contains the prediction errors v_t, the second column contains the Kalman gains, K_t, and the last column contains the inverses of the prediction error variances, F_t^{-1}. Since task="STFIL" the filtered estimates $a_{t|t}$ and $y_{t|t} = Z_t a_{t|t}$ are in the component mEst

```
> KalmanFil.ll$mEst
numeric matrix: 250 rows, 4 columns.
          [,1]     [,2]   [,3]   [,4]
 [1,]  1.10889  1.10889 1.0000 1.0000
 [2,]  0.39390  0.39390 0.5556 0.5556
...
[250,] 4.839 4.839 0.3904 0.3904
```

The plot method allows for a graphical analysis of the Kalman filter output

```
> plot(KalmanFil.ll)

Make a plot selection (or 0 to exit):

1: plot: all
2: plot: innovations
3: plot: standardized innovations
4: plot: innovation histogram
5: plot: normal QQ-plot of innovations
6: plot: innovation ACF
Selection:
```

The standardized innovations v_t/F_t are illustrated in Figure 14.3.

Kalman Smoother

The Kalman smoother backwards recursions for the simulated data from the local level model are obtained using the S+FinMetrics/SsfPack function KalmanSmo

```
> KalmanSmo.ll = KalmanSmo(KalmanFil.ll,ssf.ll)
```

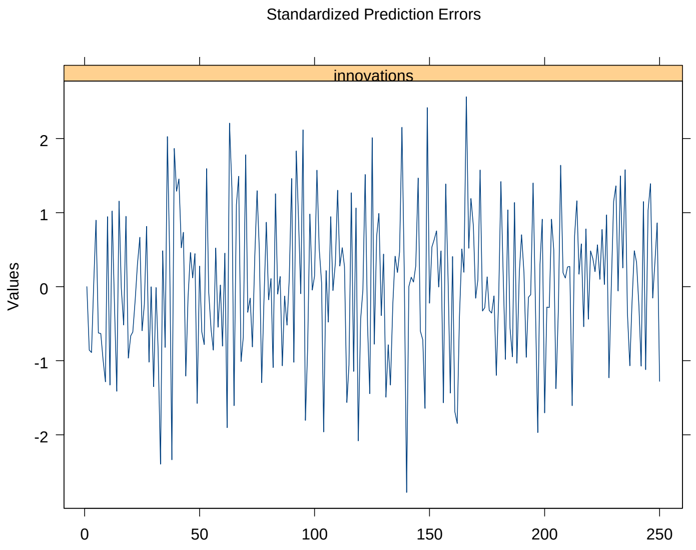


FIGURE 14.3. Standardized innovations from Kalman filter applied to simulated data from local level model.

```
> class(KalmanSmo.ll)
[1] "KalmanSmo"
```

The function `KalmanSmo` takes as input an object of class “`KalmanFil`” and an associated list variable containing the state space form used to produce the “`KalmanFil`” object. The result of `KalmanSmo` is an object of class “`KalmanSmo`” with components

```
> names(KalmanSmo.ll)
[1] "state.residuals"    "response.residuals"
[3] "state.variance"     "response.variance"
[5] "aux.residuals"      "scores"
[7] "call"
```

The component `state.residuals` contains the smoothing residuals from the state equation, `response.residuals` contains the smoothing residuals from the measurement equation. The corresponding variances of these residuals are in the components `state.variance` and `response.variance`. A multi-panel timeplot of the standardized residuals in the component `aux.residuals`, illustrated in Figure 14.4, is created with the `plot` method

```
> plot(KalmanSmo.ll,layout=c(1,2))
```

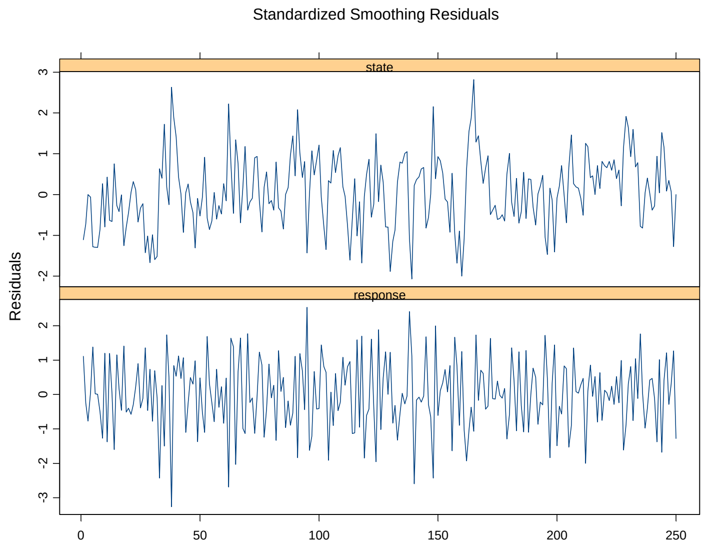


FIGURE 14.4. Standardized smoothing residuals from Kalman smoother recursions computed from simulated data from local level model.

Filtered and Smoothed Moment Estimates

Filtered and smoothed estimates of $\boldsymbol{\alpha}_t$ and y_t with corresponding estimates of variances may be computed using the `S+FinMetrics/SsfPack` function `SsfMomentEst`. To compute filtered estimates, call `SsfMomentEst` with the argument `task="STFIL"` (which stands for state filtering)

```
> FilteredEst.ll = SsfMomentEst(y.ll,ssf.ll,task="STFIL")
> class(FilteredEst.ll)
[1] "SsfMomentEst"
> names(FilteredEst.ll)
[1] "state.moment"      "state.variance"
[3] "response.moment"   "response.variance"
[5] "task"
```

The function `SsfMomentEst` takes as input a vector of response data and either a list describing the minimal state space form or a valid “`ssf`” object. The result of `SsfMomentEst` is an object of class “`SsfMomentEst`” for which there is only a `plot` method. The filtered estimates $a_{t|t}$ and $y_{t|t} = c_t + Z_t a_{t|t}$ are in the components `state.moment` and `response.moment`, respectively, and the corresponding filtered variance estimates are in the components `state.variance` and `response.variance`. From the measurement equation (14.13) in the local level model, $a_{t|t} = y_{t|t}$

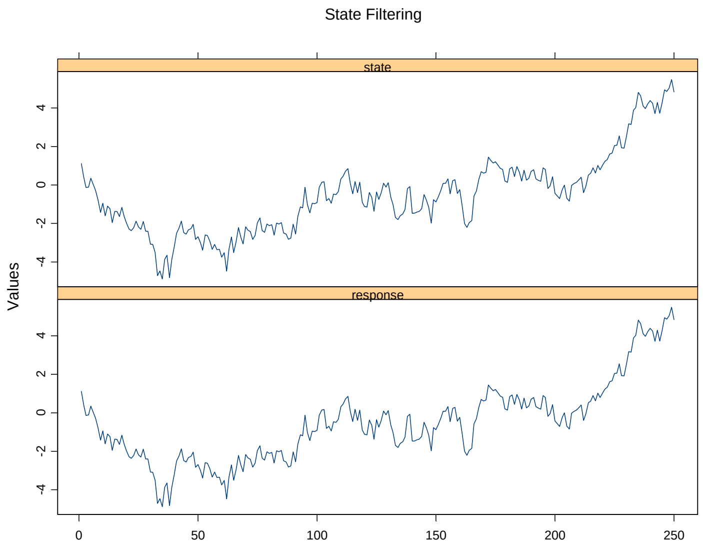


FIGURE 14.5. Filtered estimates of α_t and y_t computed from simulated data from local level model.

```
> FilteredEst.ll$state.moment[1:5]
[1]  1.1089  0.3939 -0.1389 -0.1141  0.3461
> FilteredEst.ll$response.moment[1:5]
[1]  1.1089  0.3939 -0.1389 -0.1141  0.3461
```

The `plot` method creates a multi-panel timeplot, illustrated in Figure 14.5, of the estimates of α_t and y_t

```
> plot(FilteredEst.ll,layout=c(1,2))
```

A plot of the filtered state estimates with 95% confidence intervals may be created using

```
> upper.state = FilteredEst.ll$state.moment +
+ 2*sqrt(FilteredEst.ll$state.variance)
> lower.state = FilteredEst.ll$state.moment -
+ 2*sqrt(FilteredEst.ll$state.variance)
> tsplot(FilteredEst.ll$state.moment,upper.state,lower.state,
+ lty=c(1,2,2),ylab="filtered state")
```

and is shown in Figure 14.6.

The smoothed estimates $\hat{\alpha}_t$ and $\hat{y}_t$ along with estimated variances may be computed using `SsfMomentEst` with `task="STSMO"` (state smoothing)

```
> SmoothedEst.ll = SsfMomentEst(y.ll,ssf.ll.list,task="STSMO")
```

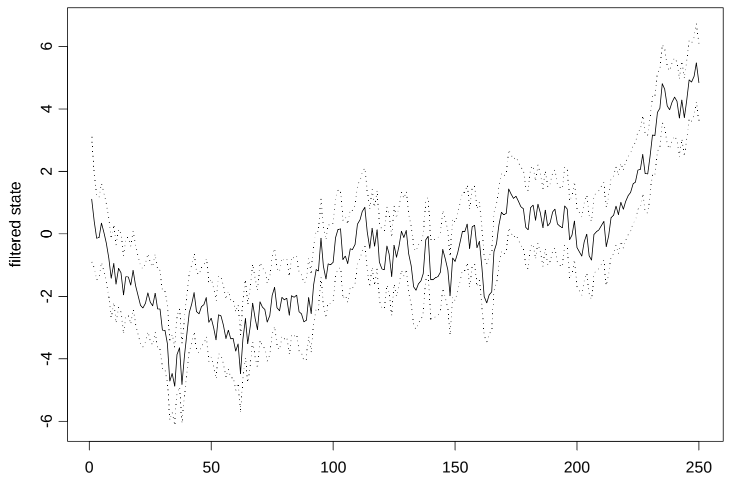


FIGURE 14.6. Filtered estimates of α_t with 95% confidence intervals computed from simulated values from local level model.

In the local level model, $\hat{\alpha}_t = \hat{y}_t$

```
> SmoothedEst.ll$state.moment[1:5]
[1]  0.24281  0.02629 -0.13914 -0.13925 -0.15455
> SmoothedEst.ll$response.moment[1:5]
[1]  0.24281  0.02629 -0.13914 -0.13925 -0.15455
```

The smoothed state estimates with 95% confidence bands are illustrated in Figure 14.7. Compared to the filtered state estimates, the smoothed estimates are "smoother" and the confidence bands are slightly smaller.

Smoothed estimates of α_t and y_t without estimated variances may be obtained using the S+FinMetrics/SsfPack function SsfCondDens with the argument task="STSMO" (which stands for state smoothing)

```
> smoothedEst.ll = SsfCondDens(y.ll,ssf.ll.list,task="STSMO")
> class(smoothedEst.ll)
[1] "SsfCondDens"
> names(smoothedEst.ll)
[1] "state"    "response" "task"
```

The object smoothedEst.ll is of class "SsfCondDens" with components state, giving the smoothed state estimates $\hat{\alpha}_t$, response, which gives the smoothed response estimates $\hat{y}_t$, and task, naming the task performed. The

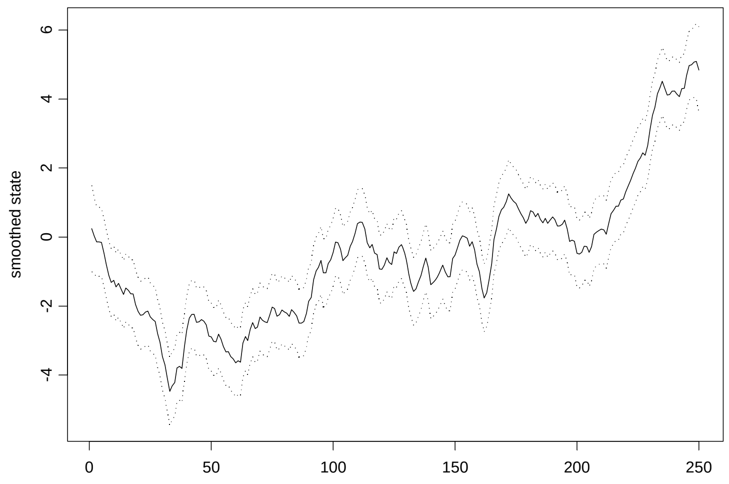


FIGURE 14.7. Smoothed estimates of α_t with 95% confidence intervals computed from simulated values from local level model.

smoothed estimates $\hat{y}_t$ and $\hat{\alpha}_t$ may be visualized using the `plot` method for "`SsfCondDens`" objects

```
> plot(smoothedEst.ll)
```

The resulting plot has the same format at the plot shown in Figure 14.5.

Smoothed Disturbance Estimates

The smoothed disturbance estimates $\hat{\varepsilon}_t$ and $\hat{\eta}_t$ may be computed using `SsfMomentEst` with the optional argument `task="DSSMO"` (which stands for disturbance smoothing)

```
> disturbEst.ll = SsfMomentEst(y.ll,ssf.ll,task="DSSMO")
> names(disturbEst.ll)
[1] "state.moment"      "state.variance"
[3] "response.moment"   "response.variance"
[5] "task"
```

The estimates $\hat{\eta}_t$ are in the component `state.moment`, and the estimates $\hat{\varepsilon}_t$ are in the component `response.moment`. These estimates may be visualized using the `plot` method.

Koopman, Shephard and Doornik (1999) pointed out that, in the local level model, the standardized smoothed disturbances

$$\frac{\hat{\eta}_t}{\sqrt{\widehat{\mathrm{var}}(\hat{\eta}_t)}},\ \frac{\hat{\varepsilon}_t}{\sqrt{\widehat{\mathrm{var}}(\hat{\varepsilon}_t)}} \tag{14.43}$$

may be interpreted as t-statistics for impulse intervention variables in the transition and measurement equations, respectively. Consequently, large values of (14.43) indicate outliers and/or structural breaks in the local level model.

Forecasting

To produce out-of-sample h-step ahead forecasts $y_{t+h|t}$ for $h = 1, \ldots, 5$ a sequence of 5 missing values is appended to the end of the response vector `y.ll`

```
> y.ll.new = c(y.ll,rep(NA,5))
```

The forecast values and mean squared errors are computed using the function `SsfMomentEst` with the argument `task="STPRED"`

```
> PredictedEst.ll = SsfMomentEst(y.ll.new,ssf.ll,task="STPRED")
> y.ll.fcst = PredictedEst.ll$response.moment
> fcst.var = PredictedEst.ll$response.variance
```

The actual values, forecasts and 95% confidence bands are illustrated in Figure 14.8 created by

```
> upper = y.ll.fcst + 2*sqrt(fcst.var)
> lower = y.ll.fcst - 2*sqrt(fcst.var)
> upper[1:250] = lower[1:250] = NA
> tsplot(y.ll.new[240:255],y.ll.fcst[240:255],
+ upper[240:255],lower[240:255],lty=c(1,2,2,2))
```

14.4 Estimation of State Space Models

14.4.1 Prediction Error Decomposition of Log-Likelihood

The *prediction error decomposition* of the log-likelihood function for the unknown parameters $\boldsymbol{\varphi}$ of a state space model may be conveniently computed using the output of the Kalman filter

$$\begin{aligned} \ln L(\boldsymbol{\varphi}|Y_n) &= \sum_{t=1}^{n} \ln f(\mathbf{y}_t|\mathbf{Y}_{t-1};\boldsymbol{\varphi}) \qquad (14.44) \\ &= -\frac{nN}{2}\ln(2\pi) - \frac{1}{2}\sum_{t=1}^{n}\left(\ln|\mathbf{F}_t| \ + \mathbf{v}_t'\mathbf{F}_t^{-1}\mathbf{v}_t\right) \end{aligned}$$

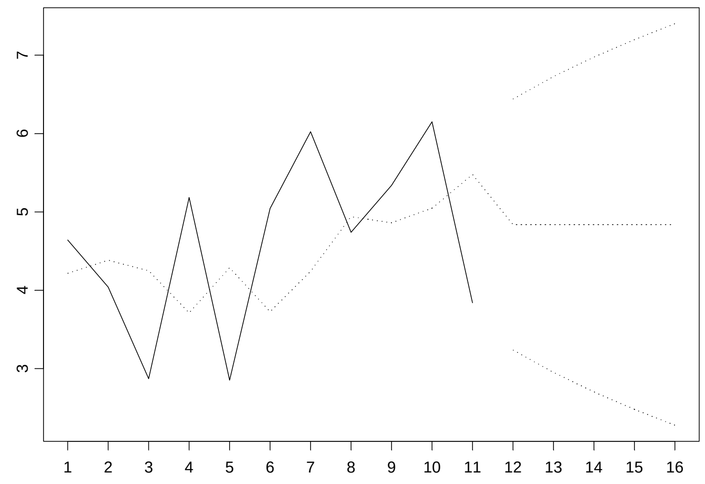


FIGURE 14.8. Actual values, h-step forecasts and 95% confidence intervals for y_t from local level model.

where $f(\mathbf{y}_t|\mathbf{Y}_{t-1};\boldsymbol{\varphi})$ is a conditional Gaussian density implied by the state space model (14.1) - (14.6). The vector of prediction errors $\mathbf{v}_t$ and prediction error variance matrices $\mathbf{F}_t$ are computed from the Kalman filter recursions.

A useful diagnostic is the estimated variance of the standardized prediction errors for a given value of $\boldsymbol{\varphi}$:

$$\hat{\sigma}^2(\boldsymbol{\varphi}) = \frac{1}{Nn}\sum_{t=1}^{n}\mathbf{v}_t'\mathbf{F}_t^{-1}\mathbf{v}_t \tag{14.45}$$

As mentioned by Koopman, Shephard and Doornik (1999), it is helpful to choose starting values for $\boldsymbol{\varphi}$ such that $\hat{\sigma}^2(\boldsymbol{\varphi}_{start}) \approx 1$. For well specified models, $\hat{\sigma}^2(\hat{\boldsymbol{\varphi}}_{\text{mle}})$ should be very close to unity.

Concentrated Log-likelihood

In some models, e.g. ARMA models, it is possible to solve explicitly for one scale factor and concentrate it out of the log-likelihood function (14.44). The resulting log-likelihood function is called the *concentrated log-likelihood* or *profile log-likelihood* and is denoted $\ln L^c(\boldsymbol{\varphi}|\mathbf{Y}_n)$. Following Koopman, Shephard and Doornik (1999), let σ denote such a scale factor, and let

$$\mathbf{y}_t = \boldsymbol{\theta}_t + \mathbf{G}_t^c\boldsymbol{\varepsilon}_t^c$$

with $\boldsymbol{\varepsilon}_t^c \sim$iid $N(0, \sigma^2\mathbf{I})$ denote the scaled version of the measurement equation (14.3). The state space form (14.1) - (14.3) applies but with $\mathbf{G}_t = \sigma\mathbf{G}_t^c$ and $\mathbf{H}_t = \sigma\mathbf{H}_t^c$. This formulation implies that one non-zero element of $\sigma\mathbf{G}_t^c$ or $\sigma\mathbf{H}_t^c$ is kept fixed, usually at unity, which reduces the dimension of the parameter vector $\boldsymbol{\varphi}$ by one. The solution for σ^2 from (14.44) is given by

$$\tilde{\sigma}^2(\boldsymbol{\varphi}) = \frac{1}{Nn}\sum_{t=1}^{n} \mathbf{v}_t' \left(\mathbf{F}_t^c\right)^{-1} \mathbf{v}_t$$

and the resulting concentrated log-likelihood function is

$$\ln L^c(\boldsymbol{\varphi}|\mathbf{Y}_n) = -\frac{nN}{2}\ln(2\pi) - \frac{nN}{2}\ln\left(\sigma^2(\boldsymbol{\varphi}) + 1\right) - \frac{1}{2}\sum_{t=1}^{n}\ln|\mathbf{F}_t^c| \quad (14.46)$$

14.4.2 Fitting State Space Models Using the S+FinMetrics/SsfPack Function SsfFit

The `S+FinMetrics/SsfPack` function `SsfFit` may be used to compute MLEs of the unknown parameters in the state space model (14.1)-(14.6) from the prediction error decomposition of the log-likelihood function (14.44). The arguments expected by `SsfFit` are

```
> args(SsfFit)
function(parm, data, FUN, conc = F, scale = 1, gradient =
NULL, hessian = NULL, lower =  - Inf, upper = Inf,
trace = T, control = NULL, ...)
```

where `parm` is a vector containing the starting values of the unknown parameters $\boldsymbol{\varphi}$, `data` is a rectangular object containing the response variables $\mathbf{y}_t$, and `FUN` is a character string giving the name of the function which takes `parm` together with the optional arguments in `...` and produces an "`ssf`" object representing the state space form. The remaining arguments control aspects of the `S-PLUS` optimization algorithm `nlminb`. An advantage of using `nlminb` is that box constraints may be imposed on the parameters of the log-likelihood function using the optional arguments `lower` and `upper`. See the online help for `nlminb` for details. A disadvantage of using `nlminb` is that the value of the Hessian evaluated at the MLEs is returned only if an analytic formula is supplied to compute the Hessian. The use of `SsfFit` is illustrated with the following examples.

Example 102 *Exact maximum likelihood estimation of AR(1) model*

Consider estimating by exact maximum likelihood the AR(1) model discussed earlier. First, $n = 250$ observations are simulated from the model

```
> ssf.ar1 = GetSsfArma(ar=0.75,sigma=.5)
> set.seed(598)
> sim.ssf.ar1 = SsfSim(ssf.ar1,n=250)
> y.ar1 = sim.ssf.ar1[,"response"]
```

Least squares estimation of the AR(1) model, which is equivalent to MLE conditional on the first observation, gives

```
> OLS(y.ar1~tslag(y.ar1)-1)

Call:
OLS(formula = y.ar1 ~tslag(y.ar1) - 1)

Coefficients:
 tslag(y.ar1)
 0.7739

Degrees of freedom: 249 total; 248 residual
Residual standard error: 0.4921
```

The S+FinMetrics/SsfPack function SsfFit requires as input a function which takes the unknown parameters $\boldsymbol{\varphi} = (\phi, \sigma^2)'$ and produces the state space form for the AR(1). One such function is

```
ar1.mod = function(parm) {
    phi = parm[1]
    sigma2 = parm[2]
    ssf.mod = GetSsfArma(ar=phi,sigma=sqrt(sigma2))
    CheckSsf(ssf.mod)
}
```

In addition, starting values for $\boldsymbol{\varphi}$ are required. Somewhat arbitrary starting values are

```
> ar1.start = c(0.5,1)
> names(ar1.start) = c("phi","sigma2")
```

The prediction error decomposition of the log-likelihood function evaluated at the starting values $\boldsymbol{\varphi} = (0.5, 1)'$ may be computed using the S+FinMetrics/SsfPack function KalmanFil with task="KFLIK"

```
> KalmanFil(y.ar1,ar1.mod(ar1.start),task="KFLIK")

Call:
KalmanFil(mY = y.ar1, ssf = ar1.mod(ar1.start), task =
"KFLIK")

Log-likelihood:  -265.5222
Prediction error variance:  0.2851
Sample observations: 250
```

```
Standardized Innovations:
    Mean Std.Error
 -0.0238  0.5345
```

Notice that the standardized prediction error variance (14.45) is 0.285, far below unity, which indicates that the starting values are not very good.

The MLEs for $\boldsymbol{\varphi} = (\phi, \sigma^2)'$ using `SsfFit` are computed as

```
> ar1.mle = SsfFit(ar1.start,y.ar1,"ar1.mod",
+                   lower=c(-.999,0), upper=c(0.999,Inf))
Iteration  0 : objective =  265.5
Iteration  1 : objective =  282.9
...
Iteration  18 : objective =  176.9
RELATIVE FUNCTION CONVERGENCE
```

In the call to `SsfFit`, the stationarity condition $-1 < \phi < 1$ and the positive variance condition $\sigma^2 > 0$ is imposed in the estimation. The result of `SsfFit` is a an object of class "`SsfFit`" with components

```
> names(ar1.mle)
[1]  "parameters" "objective" "message" "grad.norm" "iterations"
[6]  "f.evals"    "g.evals"   "hessian" "scale"     "aux"
[11] "call"       "vcov"
```

The exact MLEs for $\boldsymbol{\varphi} = (\phi, \sigma^2)'$ are

```
> ar1.mle
Log-likelihood:  -176.9
250 observations
Parameter Estimates:
     phi    sigma2
 0.77081 0.2402688
```

and the MLE for σ is

```
> sqrt(ar1.mle$parameters["sigma2"])
 sigma2
 0.4902
```

These values are very close to the least squares estimates. Standard errors and t-statistics are displayed using

```
> summary(ar1.mle)
Log-likelihood:  -176.936
250 observations
Parameters:
        Value Std. Error t value
   phi 0.7708    0.03977   19.38
sigma2 0.2403    0.02149   11.18
```

```
Convergence:  RELATIVE FUNCTION CONVERGENCE
```

A summary of the log-likelihood evaluated at the MLEs is

```
> KalmanFil(y.ar1,ar1.mod(ar1.mle$parameters),
+           task = "KFLIK")

Call:
KalmanFil(mY = y.ar1, ssf = ar1.mod(ar1.mle$parameters),
task = "KFLIK")

Log-likelihood:  -176.9359
Prediction error variance:  1
Sample observations: 250

Standardized Innovations:
    Mean Std.Error
 -0.0213  1.0018
```

Notice that the estimated variance of the standardized prediction errors is equal to 1.

Example 103 *Exact maximum likelihood estimation of AR(1) model using re-parameterization*

An alternative function to compute the state space form of the AR(1) is

```
ar1.mod2 = function(parm) {
     phi = exp(parm[1])/(1 + exp(parm[1]))
     sigma2 = exp(parm[2])
     ssf.mod = GetSsfArma(ar=phi,sigma=sqrt(sigma2))
     CheckSsf(ssf.mod)
}
```

In the above model, a stationary value for ϕ between 0 and 1 and a positive value for σ^2 is guaranteed by parameterizing the log-likelihood in terms of $\gamma_0 = \ln(\phi/(1-\phi))$ and $\gamma_1 = \ln(\sigma^2)$ instead of ϕ and σ^2. By the invariance property of maximum likelihood estimation, $\hat{\phi}_{\text{mle}} = \exp(\hat{\gamma}_{0,\text{mle}})/(1+\exp(\hat{\gamma}_{0,\text{mle}}))$ and $\hat{\sigma}^2_{\text{mle}} = \exp(\hat{\gamma}_{1,\text{mle}})$ are the MLEs for ϕ and σ^2. The MLEs for $\boldsymbol{\varphi} = (\gamma_0, \gamma_1)'$ computed using `SsfFit` are

```
> ar1.start = c(0,0)
> names(ar1.start) = c("ln(phi/(1-phi))","ln(sigma2)")
> ar1.mle = SsfFit(ar1.start,y.ar1,"ar1.mod2")
Iteration  0 : objective =  265.5222
...
Iteration  10 : objective =  176.9359
```

```
RELATIVE FUNCTION CONVERGENCE
> summary(ar1.mle)
Log-likelihood:  -176.936
250 observations
Parameters:
                  Value Std. Error t value
ln(phi/(1-phi))  1.213    0.22510   5.388
     ln(sigma2) -1.426    0.08945 -15.940

Convergence:  RELATIVE FUNCTION CONVERGENCE
```

The MLEs for ϕ and σ^2, and estimated standard errors computed using the delta method, are[5]

```
> ar1.mle2 = ar1.mle
> tmp = coef(ar1.mle)
> ar1.mle2$parameters[1] = exp(tmp[1])/(1 + exp(tmp[1]))
> ar1.mle2$parameters[2] = exp(tmp[2])
> dg1 = exp(-tmp[1])/(1 + exp(-tmp[1]))^2
> dg2 = exp(tmp[2])
> dg = diag(c(dg1,dg2))
> ar1.mle2$vcov = dg%*%ar1.mle2$vcov%*%dg
> summary(ar1.mle2)
Log-likelihood:  -176.936
250 observations
Parameters:
                  Value Std. Error t value
ln(phi/(1-phi)) 0.7708    0.03977   19.38
     ln(sigma2) 0.2403    0.02149   11.18

Convergence:  RELATIVE FUNCTION CONVERGENCE
```

and exactly match the previous MLEs.

Example 104 *Exact maximum likelihood estimation of AR(1) model using concentrated log-likelihood*

In the AR(1) model, the variance parameter σ^2 can be analytically concentrated out of the log-likelihood. The advantages of concentrating the log-likelihood are to reduce the number of parameters to estimate, and to improve the numerical stability of the optimization. A function to compute the state space form for the AR(1) model, as a function of ϕ only, is

```
ar1.modc = function(parm) {
```

[5]Recall, if $\sqrt{n}(\hat{\theta}-\theta) \xrightarrow{d} N(\theta, V)$ and g is a continuous function then $\sqrt{n}(g(\hat{\theta})-g(\theta)) \xrightarrow{d} N(0, \frac{\partial g}{\partial \theta'} V \frac{\partial g}{\partial \theta'}')$.

```
     phi = parm[1]
     ssf.mod = GetSsfArma(ar=phi)
     CheckSsf(ssf.mod)
}
```

By default, the function `GetSsfArma` sets $\sigma^2 = 1$ which is required for the computation of the concentrated log-likelihood function from (14.46). To maximize the concentrated log-likelihood function (14.46) for the AR(1) model, use `SsfFit` with `ar1.modc` and set the optional argument `conc=T` :

```
> ar1.start = c(0.7)
> names(ar1.start) = c("phi")
> ar1.cmle = SsfFit(ar1.start,y.ar1,"ar1.modc",conc=T,
+                    lower=0,upper=0.999)
Iteration  0 : objective =  178.506
...
Iteration  4 : objective =  176.9359
RELATIVE FUNCTION CONVERGENCE
> summary(ar1.cmle)
Log-likelihood:  -176.936
250 observations
Parameters:
     Value Std. Error t value
phi 0.7708    0.03977   19.38

Convergence:  RELATIVE FUNCTION CONVERGENCE
```

Notice that with the concentrated log-likelihood, the optimizer converges in only four interations. The values of the log-likelihood and the MLE for ϕ are the same as found previously. The MLE for σ^2 may be recovered by running the Kalman filter and computing the variance of the prediction errors:

```
> ar1.KF = KalmanFil(y.ar1,ar1.mod(ar1.cmle$parameters),
+                     task="KFLIK")
> ar1.KF$dVar
[1] 0.2403
```

One disadvantage of using the concentrated log-likelihood is the lack of an estimated standard error for $\hat{\sigma}^2$.

Example 105 *Maximum likelihood estimation of CAPM with time varying parameters*

Consider estimating the CAPM with time varying coefficients (14.17) - (14.19) using monthly data on Microsoft and the S&P 500 index over the period February, 1990 through December, 2000 contained in the `S+FinMetrics` "`timeSeries`" `excessReturns.ts`. The parameters of the model are the variances of the innovations to the transition and measurement equations; $\sigma^2_\xi, \sigma^2_\varsigma$ and σ^2_ν. Since these variances must be positive the log-likelihood is parameterized using $\boldsymbol{\varphi} = (\ln(\sigma^2_\xi), \ln(\sigma^2_\varsigma), \ln(\sigma^2_\nu))'$. Since the state space form for the CAPM with time varying coefficients requires a data matrix $\mathbf{X}$ containing the excess returns on the S&P 500 index, the function `SsfFit` requires as input a function which takes both $\boldsymbol{\varphi}$ and $\mathbf{X}$ and returns the appropriate state space form. One such function is

```
tvp.mod = function(parm,mX=NULL) {
   parm = exp(parm)   # 3 x 1 vector containing log variances
   ssf.tvp = GetSsfReg(mX=mX)
   diag(ssf.tvp$mOmega) = parm
   CheckSsf(ssf.tvp)
}
```

Starting values for $\boldsymbol{\varphi}$ are specified as

```
> tvp.start = c(0,0,0)
> names(tvp.start) = c("ln(s2.alpha)","ln(s2.beta)","ln(s2.y)")
```

The maximum likelihood estimates for $\boldsymbol{\varphi}$ based on `SsfFit` are computed using

```
> tvp.mle = SsfFit(tvp.start,msft.ret,"tvp.mod",mX=X.mat)
Iteration  0 : objective =  183.2
...
Iteration  22 : objective =  -123
RELATIVE FUNCTION CONVERGENCE
```

The MLEs for $\boldsymbol{\varphi} = (\ln(\sigma^2_\xi), \ln(\sigma^2_\varsigma), \ln(\sigma^2_\nu))'$ are

```
> summary(tvp.mle)
Log-likelihood:  122.979
131 observations
Parameters:
               Value Std. Error t value
ln(s2.alpha) -12.480     2.8030  -4.452
 ln(s2.beta)  -5.900     3.0890  -1.910
    ln(s2.y)  -4.817     0.1285 -37.480

Convergence:  RELATIVE FUNCTION CONVERGENCE
```

The estimates for the standard deviations σ_ξ, σ_ς and σ_ν as well as estimated standard errors, from the delta method, are:

```
> tvp2.mle = tvp.mle
> tvp2.mle$parameters = exp(tvp.mle$parameters/2)
> names(tvp2.mle$parameters) = c("s.alpha","s.beta","s.y")
> dg = diag(tvp2.mle$parameters/2)
> tvp2.mle$vcov = dg%*%tvp.mle$vcov%*%dg
> summary(tvp2.mle)
Log-likelihood:  122.979
131 observations
Parameters:
           Value Std. Error t value
s.alpha 0.001951   0.002734  0.7135
 s.beta 0.052350   0.080860  0.6474
    s.y 0.089970   0.005781 15.5600

Convergence:  RELATIVE FUNCTION CONVERGENCE
```

It appears that the estimated standard deviations for the time varying parameter CAPM are close to zero, suggesting a constant parameter model.

The smoothed estimates of the time varying parameters α_t and β_t as well as the expected returns may be extracted and plotted using

```
> smoothedEst.tvp = SsfCondDens(y.capm,
+ tvp.mod(tvp.mle$parameters,mX=X.mat),
+ task="STSMO")
> plot(smoothedEst.tvp,strip.text=c("alpha(t)",
+      "beta(t)","Expected returns"),main="Smoothed Estimates",
+      scales="free")
```

These estimates are illustrated in Figure 14.9. Notice the increase in $\hat{\beta}_t$ and decrease in $\hat{\alpha}_t$ over the sample.

14.4.3 Quasi-Maximum Likelihood Estimation

The `S+FinMetrics` function `SsfFitFast` is a fast version of `SsfFit` that also computes the sandwhich covariance matrix estimate which is appropriate for QMLE. This matrix is given in the `r.vcov` component of the "`SsfFit`" object returned by `SsfFitFast`.

Example 106 *Quasi-maximum likelihood estimation of a stochastic volatility model*

Let r_t denote the continuously compounded return on an asset between times $t-1$ and t. Following Harvey, Ruiz, and Shephard (1994), hereafter HRS, a simple discrete-time stochastic volatility (SV) model has the form

$$\begin{aligned} r_t &= \sigma_t \varepsilon_t, \qquad \varepsilon_t \sim \text{iid } N(0,1) \qquad (14.47) \\ h_t &= \ln \sigma_t^2 = \gamma + \phi h_{t-1} + \eta_t, \qquad \eta_t \sim \text{iid } N(0, \sigma_\eta^2) \\ E[\varepsilon_t \eta_t] &= 0 \end{aligned}$$

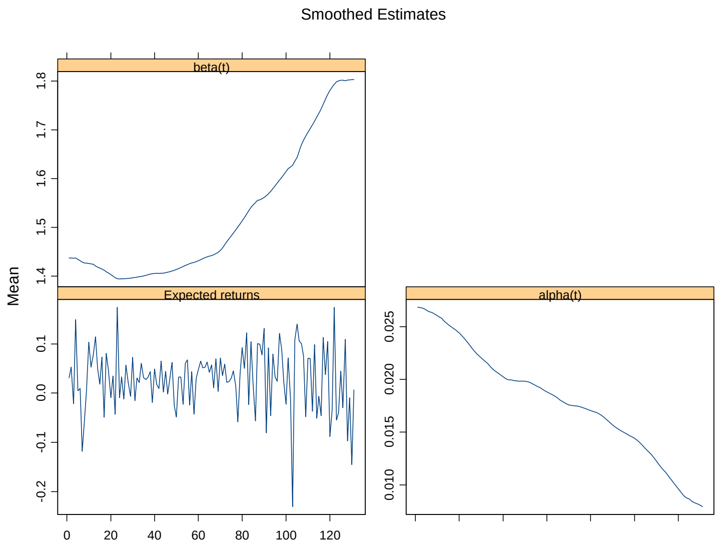


FIGURE 14.9. Smoothed estimates of α_t and β_t from CAPM with time varying parameter fit to the monthly excess returns on Microsoft.

Defining $y_t = \ln r_t^2$, and noting that $E[\ln \varepsilon_t^2] = -1.27$ and $\text{var}(\ln \varepsilon_t^2) = \pi^2/2$ an unobserved components state space representation for y_t has the form

$$\begin{aligned} y_t &= -1.27 + h_t + \xi_t, && \xi_t \sim \text{iid } (0, \pi^2/2) \\ h_t &= \gamma + \phi h_{t-1} + \eta_t, && \eta_t \sim \text{iid } N(0, \sigma_\eta^2) \\ E[\xi_t \eta_t] &= 0 \end{aligned}$$

If ξ_t were iid Gaussian then the parameters $\boldsymbol{\varphi} = (\gamma, \sigma_\eta^2, \phi)'$ of the SV model could be efficiently estimated by maximizing the prediction error decomposition of the log-likelihood function constructed from the Kalman filter recursions. However, since $\xi_t = \ln \varepsilon_t^2$ is not normally distributed the Kalman filter only provides minimum mean squared error linear estimators of the state and future observations. Nonetheless, HRS point out that even though the exact log-likelihood cannot be computed from the prediction error decomposition based on the Kalman filter, consistent estimates of $\boldsymbol{\varphi} = (\gamma, \sigma_\eta^2, \phi)'$ can still be obtained by treating ξ_t as though it were iid $N(0, \pi^2/2)$ and maximizing the quasi log-likelihood function constructed from the prediction error decomposition.

The state space representation of the SV model has system matrices

$$\boldsymbol{\delta} = \begin{pmatrix} \gamma \\ -1.27 \end{pmatrix}, \qquad \boldsymbol{\Phi} = \begin{pmatrix} \phi \\ 1 \end{pmatrix}, \qquad \boldsymbol{\Omega} = \begin{pmatrix} \sigma_\eta^2 & 0 \\ 0 & \pi^2/2 \end{pmatrix}$$

Assuming that $|\phi| < 1$, the initial value matrix has the form

$$\mathbf{\Sigma} = \begin{pmatrix} \sigma_\eta^2/(1-\phi^2) \\ \gamma/(1-\phi) \end{pmatrix}$$

If $\phi = 1$ then use

$$\mathbf{\Sigma} = \begin{pmatrix} -1 \\ 0 \end{pmatrix}$$

A function to compute the state space form of the SV model given a vector of parameters, assuming $|\phi| < 1$, is

```
sv.mod = function(parm) {
     g = parm[1]
     sigma2.n = exp(parm[2])
     phi = parm[3]
     ssf.mod = list(mDelta=c(g,-1.27),
     mPhi=as.matrix(c(phi,1)),
     mOmega=matrix(c(sigma2.n,0,0,0.5*pi^2),2,2),
     mSigma=as.matrix(c((sigma2.n/(1-phi^2)),g/(1-phi))))
     CheckSsf(ssf.mod)
}
```

Notice that an exponential transformation is utilized to ensure a positive value for σ_η^2. A sample of $T = 1000$ observations are simulated from the SV model using the parameters $\gamma = -0.3556$, $\sigma_\eta^2 = 0.0312$ and $\phi = 0.9646$:

```
> parm.hrs = c(-0.3556,log(0.0312),0.9646)
> nobs = 1000
> set.seed(179)
> e = rnorm(nobs)
> xi = log(e^2)+1.27
> eta = rnorm(nobs,sd=sqrt(0.0312))
> sv.sim = SsfSim(sv.mod(parm.hrs),
+ mRan=cbind(eta,xi),a1=(-0.3556/(1-0.9646)))
```

The first 250 simulated squared returns, r_t^2, and latent squared volatilities, σ_t^2, are shown in Figure 14.10.

Starting values for the estimation of $\boldsymbol{\varphi} = (\gamma, \ln\sigma_\eta^2, \phi)'$ are values close to the true values:

```
> sv.start = c(-0.3,log(0.03),0.9)
> names(sv.start) = c("g","ln.sigma2","phi")
```

Using `SsfFitFast`, the quasi-maximum likelihood (QML) estimates are

```
> low.vals = c(-Inf,-Inf,-0.999)
> up.vals = c(Inf,Inf,0.999)
> sv.mle = SsfFitFast(sv.start,sv.sim[,2],"sv.mod",
```

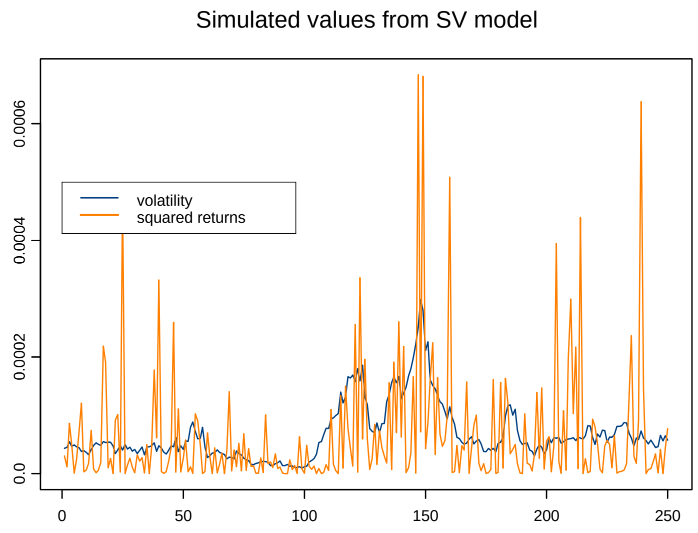


FIGURE 14.10. Simulated values from SV model.

```
+                      lower=low.vals,upper=up.vals)
Iteration  0 : objective =  5.147579
...
Iteration  15 : objective =  2.21826
RELATIVE FUNCTION CONVERGENCE
```

To show the estimates with the QMLE standard errors use `summary` with the optional argument `method="qmle"`

```
> summary(sv.mle,method="qmle")
Log-likelihood:  -2218.26
1000 observations
Parameters:
            Value QMLE Std. Error t value
        g -0.4810         0.18190  -2.644
ln.sigma2 -3.5630         0.53020  -6.721
      phi  0.9509         0.01838  51.740

Convergence:  RELATIVE FUNCTION CONVERGENCE
```

Using the delta method, the QMLE and estimated standard error for σ_η^2 are 0.02834 and 0.01503, respectively.

The filtered and smoothed estimates of log-volatility are computed using

```
> ssf.sv = sv.mod(sv.mle2$parameters)
```

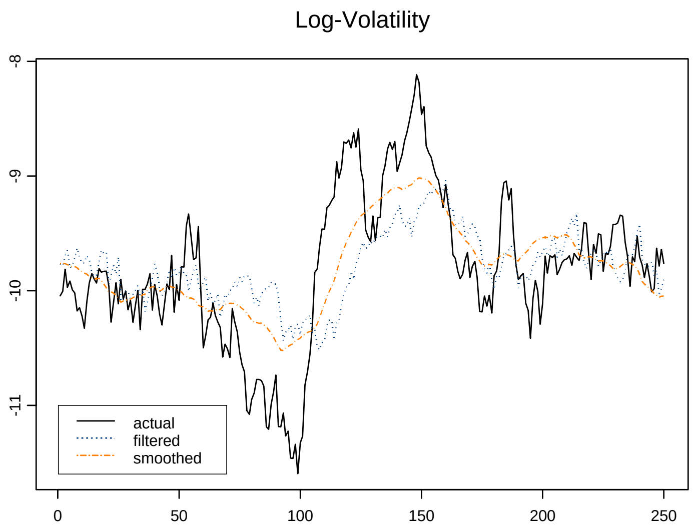


FIGURE 14.11. Log-volatility along with filtered and smoothed estimates from SV model.

```
> filteredEst.sv = SsfMomentEst(sv.sim[,2],ssf.sv,task="STFIL")
> smoothedEst.sv = SsfCondDens(sv.sim[,2],ssf.sv,task="STSMO")
```

The first 250 estimates along with actual log-volatility are illustrated in Figure 14.11

14.5 Simulation Smoothing

The simulation of state and response vectors $\boldsymbol{\alpha}_t$ and $\mathbf{y}_t$ or disturbance vectors $\boldsymbol{\eta}_t$ and $\boldsymbol{\varepsilon}_t$ conditional on the observations $\mathbf{Y}_n$ is called *simulation smoothing*. Simulation smoothing is useful for evaluating the appropriateness of a proposed state space model and for the Bayesian analysis of state space models using Markov chain Monte Carlo (MCMC) techniques. The `S+FinMetrics/SsfPack` function `SimSmoDraw` generates random draws from the distributions of the state and response variables or from the distributions of the state and response disturbances. The arguments expected by `SimSmoDraw` are

```
> args(SimSmoDraw)
function(kf, ssf, task = "DSSIM", mRan = NULL, a1 = NULL)
```

where `kf` is a “`KalmanFil`” object, `ssf` is a list which either contains the minimal necessary components for a state space form or is a valid “`ssf`” object and `task` determines whether the state smoothing (“`STSIM`”) or disturbance smoothing (“`DSSIM`”) is performed.

Example 107 *Simulation smoothing from the local level model*

Simulated state and response values from the local level model may be generated using

```
> KalmanFil.ll = KalmanFil(y.ll,ssf.ll,task="STSIM")
> ll.state.sim = SimSmoDraw(KalmanFil.ll,ssf.ll,
+ task="STSIM")
> class(ll.state.sim)
[1] "SimSmoDraw"
> names(ll.state.sim)
[1] "state"    "response" "task"
```

The resulting simulated values may be visualized using

```
> plot(ll.state.sim,layout=c(1,2))
```

To simulate disturbances from the state and response equations, set `task="DSSIM"` in the calls to `KalmanFil` and `SimSmoDraw`.

14.6 References

Bomhoff, E. J. (1994). *Financial Forecasting for Business and Economics.* Academic Press, San Diego.

Carmona, R. (2004). *Statistical Analysis of Financial Data, with S-PLUS.* Springer-Verlag, New York.

Chan, N.H. (2002). *Time Series: Applicatios to Finance.* John Wiley & Sons, New York.

Durbin, J. and S.J. Koopman (2001). *Time Series Analysis by State Space Methods.* Oxford University Press, Oxford.

Duan, J.-C. and J.-G. Simonato (1999). “Estimating Exponential-Affine Term Structure Models by Kalman Filter,” Review of Quantitative Finance and Accounting, 13, 111-135.

Hamilton, J.D. (1994). *Time Series Analysis.* Princeton University Press, Princeton, NJ.

Harvey, A. C. (1989). *Forecasting, Structural Time Series Models and the Kalman Filter*. Cambridge University Press, Cambridge.

Harvey, A.C. (1993). *Time Series Models*, 2nd edition. MIT Press, Cambridge.

Harvey, A.C., E. Ruiz and N. Shephard (1994). “Multivariate Stochastic Variance Models,” *Review of Economic Studies*, 61, 247-264.

Kim, C.-J., and C.R. Nelson (1999). *State-Space Models with Regime Switching*. MIT Press, Cambridge.

Koopman, S.J., N. Shephard, and J.A. Doornik (1999). “Statistical Algorithms for State Space Models Using SsfPack 2.2,” *Econometrics Journal*, 2, 113-166.

Koopman, S.J., N. Shephard, and J.A. Doornik (2001). “SsfPack 3.0beta: Statistical Algorithms for Models in State Space,” unpublished manuscript, Free University, Amsterdam.

Engle, R.F. and M.W. Watson (1987). “The Kalman Filter: Applications to Forecasting and Rational Expectations Models,” in T.F. Bewley (ed.) *Advances in Econometrics: Fifth World Congress, Volume I.* Cambridge University Press, Cambridge.

West, M. and J. Harrison (1997). *Bayesian Forecasting and Dynamic Models*, 2nd edition. Springer-Verlag, New York.

Zivot, E., Wang, J. and S.J. Koopman (2004). “State Space Models in Economics and Finance Using SsfPack in S+FinMetrics,” in A. Harvey, S.J. Koopman, and N. Shephard (eds.), *Unobserved Components Models.* Cambridge University Press, Cambridge.

15
Factor Models for Asset Returns

15.1 Introduction

Multifactor models can be used to predict returns, generate estimates of abnormal return, and estimate the variability and covariability of returns. This chapter focuses on the use of multifactor models to describe the covariance structure of returns[1]. Asset return covariance matrices are key inputs to portfolio optimization routines used for asset allocation and active asset management. A factor model decomposes an asset's return into factors common to all assets and an asset specific factor. Often the common factors are interpreted as capturing fundamental risk components, and the factor model isolates an asset's sensitivities to these risk factors. The three main types of multifactor models for asset returns are: (1) *macroeconomic* factor models; (2) *fundamental* factor models; and (3) *statistical* factor models. Macroeconomic factor models use observable economic time series like interest rates and inflation as measures of pervasive or common factors in asset returns. Fundamental factor models use observable firm or asset specific attributes such as firm size, dividend yield, and industry classification to determine common factors in asset returns. Statistical factor models treat the common factors as unobservable or latent factors. Estimation of multifactor models is type-specific, and this chapter summarizes

[1]A recent review of factor models for this purpose is given in Chan, Karceski and Lakonishok (1998).

the econometric issues associated with estimating each type of factor model and gives illustrations using `S-PLUS`.

This chapter is organized as follows. Section two presents the general factor model specification. Section three describes the macroeconomic factor model. Examples using Sharpe's single index model as well as a general macroeconomic model are given. Section four surveys the fundamental factor model and provides illustrations of an industry factor model and a Fama-French type model. Statistical factor models estimated using factor analysis and principal components analysis are covered in Section five. Particular emphasis is given to techniques appropriate for the case in which the number of assets is greater than the number of time periods.

Connor (1995) gives an overview of three types of factor models for asset returns and compares their explanatory power. Campbell, Lo, and MacKinlay (1997) and Grinold and Kahn (2000) survey the econometric specification of these models. Johnson and Wichern (1998) provides an excellent treatment of statistical factor models. Good textbook discussions of statistical factor models with applications in finance are given in Alexander (2001) and Tsay (2001).

15.2 Factor Model Specification

Each of the three types of multifactor models for asset returns has the general form

$$\begin{aligned} R_{it} &= \alpha_i + \beta_{1i} f_{1t} + \beta_{2i} f_{2t} + \cdots + \beta_{Ki} f_{Kt} + \varepsilon_{it} \\ &= \alpha_i + \boldsymbol{\beta}_i' \mathbf{f}_t + \varepsilon_{it} \end{aligned} \tag{15.1}$$

where R_{it} is the return (real or in excess of the risk-free rate) on asset i ($i = 1, \ldots, N$) in time period t ($t = 1, \ldots, T$), α_i is the intercept, f_{kt} is the k^{th} *common factor* ($k = 1, \ldots, K$), β_{ki} is the *factor loading* or *factor beta* for asset i on the k^{th} factor, and ε_{it} is the asset *specific factor.* In the multifactor model, it is assumed that the factor realizations, $\mathbf{f}_t$, are $I(0)$ with unconditional moments

$$\begin{aligned} E[\mathbf{f}_t] &= \boldsymbol{\mu}_f \\ \mathrm{cov}(\mathbf{f}_t) &= E[(\mathbf{f}_t - \boldsymbol{\mu}_f)(\mathbf{f}_t - \boldsymbol{\mu}_f)'] = \boldsymbol{\Omega}_f \end{aligned}$$

and that the asset specific error terms, ε_{it}, are uncorrelated with each of the common factors, f_{kt}, so that

$$\mathrm{cov}(f_{kt}, \varepsilon_{it}) = 0, \text{ for all } k, \ i \text{ and } t.$$

It is also assumed that the error terms ε_{it} are serially uncorrelated and contemporaneously uncorrelated across assets

$$\begin{aligned} \mathrm{cov}(\varepsilon_{it}, \varepsilon_{js}) &= \sigma_i^2 \text{ for all } i = j \text{ and } t = s \\ &= 0, \text{ otherwise} \end{aligned}$$

In applications, it is often the case that the number of assets, N, is substantially larger than the number of time periods, T. In what follows a subscript t represents time and a subscript i represents asset so that $\mathbf{R}_t$ represents an $(N \times 1)$ vector of assets at time t and $\mathbf{R}_i$ represents a $(T \times 1)$ vector of returns on asset i.

The multifactor model (15.1) may be rewritten as a *cross-sectional* regression model at time t by stacking the equations for each asset to give

$$\begin{aligned} \underset{(N\times 1)}{\mathbf{R}_t} &= \underset{(N\times 1)}{\boldsymbol{\alpha}} + \underset{(N\times K)}{\mathbf{B}}\underset{(K\times 1)}{\mathbf{f}_t} + \underset{(N\times 1)}{\boldsymbol{\varepsilon}_t}, \ t = 1, \ldots, T \qquad (15.2) \\ E[\boldsymbol{\varepsilon}_t\boldsymbol{\varepsilon}_t'|\mathbf{f}_t] &= \mathbf{D} \end{aligned}$$

where $\mathbf{B}$ is the $(N \times K)$ matrix of factor betas, $\mathbf{f}_t$ is the $(K \times 1)$ vector of factor realizations for time period t, and $\boldsymbol{\varepsilon}_t$ is the $(N \times 1)$ vector of asset specific error terms with $(N \times N)$ diagonal covariance matrix $\mathbf{D}$. Given the assumption of the multifactor model, the $(N \times N)$ covariance matrix of asset returns has the form

$$\mathrm{cov}(\mathbf{R}_t) = \boldsymbol{\Omega} = \mathbf{B}\boldsymbol{\Omega}_f\mathbf{B}' + \mathbf{D}$$

The multifactor model (15.1) may also be rewritten as a *time-series* regression model for asset i by stacking observations for a given asset i to give

$$\begin{aligned} \underset{(T\times 1)}{\mathbf{R}_i} &= \underset{(T\times 1)}{\mathbf{1}_T}\underset{(1\times 1)}{\alpha_i} + \underset{(T\times K)}{\mathbf{F}}\underset{(K\times 1)}{\boldsymbol{\beta}_i} + \underset{(T\times 1)}{\boldsymbol{\varepsilon}_i}, \ i = 1, \ldots, N \qquad (15.3) \\ E[\boldsymbol{\varepsilon}_i\boldsymbol{\varepsilon}_i'] &= \sigma_i^2\mathbf{I}_T \end{aligned}$$

where $\mathbf{1}_T$ is a $(T \times 1)$ vector of ones, $\mathbf{F}$ is a $(T \times K)$ matrix of factor realizations (the tth row of $\mathbf{F}$ is $\mathbf{f}_t'$), $\boldsymbol{\beta}_i$ is a $(K \times 1)$ vector of factor loadings, and $\boldsymbol{\varepsilon}_i$ is a $(T \times 1)$ vector of error terms with covariance matrix $\sigma_i^2\mathbf{I}_T$.

Finally, collecting data from $t = 1, \ldots, T$ allows the model (15.2) to be expressed in matrix form as the multivariate regression

$$\underset{(N\times T)}{\mathbf{R}} = \underset{(N\times 1)}{\boldsymbol{\alpha}} + \underset{(N\times K)}{\mathbf{B}}\underset{(K\times T)}{\mathbf{F}} + \underset{(N\times T)}{\mathbf{E}} \qquad (15.4)$$

15.3 Macroeconomic Factor Models for Returns

In a *macroeconomic factor model*, the factor realizations $\mathbf{f}_t$ in (15.1) are observed macroeconomic variables that are assumed to be uncorrelated with the asset specific error terms ε_{it}. The two most common macroeconomic factor models are Sharpe's (1970) single factor model and Chen, Roll and Ross's (1986) multifactor model. Once the macroeconomic factors are specified and constructed the econometric problem is then to estimate the factor betas, β_{ki}, residual variances, σ_i^2, and factor covariance, $\boldsymbol{\Omega}_f$, using time series regression techniques.

15.3.1 Sharpe's Single Index Model

The most famous macroeconomic factor model is *Sharpe's single factor model* or market model

$$R_{it} = \alpha_i + \beta_i R_{Mt} + \varepsilon_{it}, \; i = 1, \ldots, N; t = 1, \ldots, T \tag{15.5}$$

where R_{Mt} denotes the return or excess return (relative to the risk-free rate) on a market index (typically a value weighted index like the S&P 500 index) in time period t. The market index is meant to capture economy-wide or market risk, and the error term captures non-market firm specific risk. The multifactor model (15.1) reduces to (15.5) if $f_{1t} = R_{Mt}$, $\beta_{ik} = 0$ $(i = 1, \ldots, N; k = 2, \ldots, K)$. The covariance matrix of assets from the single factor model is

$$\mathbf{\Omega} = \sigma_M^2 \boldsymbol{\beta}\boldsymbol{\beta}' + \mathbf{D} \tag{15.6}$$

where $\sigma_M^2 = \text{var}(R_{Mt})$, $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_N)'$ and $\mathbf{D}$ is a diagonal matrix with $\sigma_i^2 = \text{var}(\varepsilon_{it})$ along the diagonal.

Because R_{Mt} is observable, the parameters β_i and σ_i^2 of the single factor model (15.5) for each asset can be estimated using time series regression giving

$$\begin{aligned} \mathbf{R}_i &= \widehat{\alpha}_i \mathbf{1} + \mathbf{R}_M \widehat{\beta}_i + \widehat{\boldsymbol{\varepsilon}}_i, \; i = 1, \ldots, N \\ \widehat{\sigma}_i^2 &= \frac{1}{T-2} \widehat{\boldsymbol{\varepsilon}}_i' \widehat{\boldsymbol{\varepsilon}}_i \end{aligned}$$

The variance of the market index is estimated using the time series sample variance

$$\begin{aligned} \widehat{\sigma}_M^2 &= \frac{1}{T-1} \sum_{t=1}^{T} (R_{Mt} - \overline{R}_M)^2 \\ \overline{R}_M &= \frac{1}{T} \sum_{t=1}^{T} R_{Mt} \end{aligned}$$

The estimated single factor model covariance is then

$$\widehat{\mathbf{\Omega}} = \widehat{\sigma}_M^2 \widehat{\boldsymbol{\beta}} \widehat{\boldsymbol{\beta}}' + \widehat{\mathbf{D}},$$

where $\widehat{\mathbf{D}}$ has $\widehat{\sigma}_i^2$ along the diagonal.

Remarks

1. Computational efficiency may be obtained by using multivariate regression[2]. The coefficients α_i and β_i and the residual variances σ_i^2

[2]Since R_M is the regressor for each asset, multivariate OLS estimates are numerically equivalent to multivariate GLS estimates that take into account the across equation correlation between the errors in each equation.

may be computed in one step in the multivariate regression model

$$\mathbf{R}_T = \mathbf{X}\mathbf{\Gamma}' + \mathbf{E}_T$$

where $\mathbf{R}_T$ is a $(T \times N)$ matrix of asset returns, $\mathbf{X} = [\mathbf{1} : \mathbf{R}_M]$ is a $(T \times 2)$ matrix, $\mathbf{\Gamma}' = [\boldsymbol{\alpha} : \boldsymbol{\beta}]'$ is a $(2 \times N)$ matrix of coefficients and $\mathbf{E}_T$ is a $(T \times N)$ matrix of errors. The multivariate OLS estimator of Γ' is

$$\widehat{\mathbf{\Gamma}}' = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{R}_T.$$

The estimate of the residual covariance matrix is

$$\widehat{\mathbf{\Sigma}} = \frac{1}{T-2}\widehat{\mathbf{E}}_T'\widehat{\mathbf{E}}_T$$

where $\hat{\mathbf{E}}_T = \mathbf{R}_T - \mathbf{X}\hat{\mathbf{\Gamma}}'$ is the multivariate least squares residual matrix. The diagonal elements of $\widehat{\mathbf{\Sigma}}$ are the diagonal elements of $\widehat{\mathbf{D}}$.

2. The R^2 from the time series regression is a measure of the proportion of "market" risk, and $1 - R^2$ is a measure of asset specific risk. Additionally, $\widehat{\sigma}_i$ is a measure of the typical size of asset specific risk.

3. Robust regression techniques can be used to estimate β_i and σ_i^2. Also, a robust estimate of σ_M^2 could be computed.

4. In practice, the estimated value of β_i is often adjusted toward unity. Adjusted β_i values are discussed in chapter seven of Elton and Gruber (1997).

5. The single factor covariance matrix (15.6) is constant over time. This may not be a good assumption. There are several ways to allow (15.6) to vary over time. For example, assume that β_i is constant and that σ_i^2 and σ_M^2 are conditionally time varying. That is, $\sigma_i^2 = \sigma_{it}^2$ and $\sigma_M^2 = \sigma_{Mt}^2$. To capture conditional heteroskedasticity, GARCH models may be used for σ_{it}^2 and σ_{Mt}^2. One may also use exponential weights in computing the sample variances of σ_i^2 and σ_M^2. Alternatively, one may assume that β_i is not constant over time.

Example 108 *Estimation of Sharpe's single factor model using `S-PLUS`*

The single factor model parameters and the return covariance matrix (15.6) may be efficiently estimated using the matrix functions in `S-PLUS`. To illustrate, consider estimating the single factor model using the monthly return data over the period January 1978 through December 1987 in the "`timeSeries`" `berndt.dat`. The variables in `berndt.dat` are

```
> colIds(berndt.dat)
 [1] "CITCRP" "CONED"  "CONTIL" "DATGEN" "DEC"    "DELTA"
 [7] "GENMIL" "GERBER" "IBM"    "MARKET" "MOBIL"  "PANAM"
[13] "PSNH"   "TANDY"  "TEXACO" "WEYER"  "RKFREE"
```

See the online help for `berndt.dat` for a description of these variables. The return data on the assets and the market index are extracted using:

```
> returns = as.matrix(seriesData(berndt.dat[, c(-10, -17)]))
> market = as.vector(seriesData(berndt.dat)[,10])
```

The single factor model parameters may be estimated by multivariate regression using

```
> n.obs = nrow(returns)
> X.mat = cbind(rep(1,n.obs),market)
> G.hat = solve(X.mat,returns)
> beta.hat = G.hat[2,]
> E.hat = returns - X.mat%*%G.hat
> diagD.hat = diag(crossprod(E.hat)/(n.obs-2))
> names(diagD.hat) = colIds(G.hat)
> r.sq = 1-(n.obs-2)*diagD.hat/diag(var(returns,SumSquares=T))
```

The second row of `G.hat` contains the estimated β_i values, and the vector `diagD.hat` contains the estimated residual variances σ_i^2:

```
> t(rbind(beta.hat,sqrt(diagD.hat),r.sq))
       beta.hat          r.square
CITCRP 0.667776 0.067163 0.317769
 CONED 0.091021 0.050096 0.015316
CONTIL 0.738357 0.142597 0.112158
DATGEN 1.028160 0.106880 0.303631
   DEC 0.843053 0.081018 0.337829
 DELTA 0.489461 0.090289 0.121627
GENMIL 0.267765 0.062676 0.079188
GERBER 0.624807 0.076966 0.236938
   IBM 0.453024 0.050461 0.275235
 MOBIL 0.713515 0.064072 0.368818
 PANAM 0.730140 0.122507 0.143372
  PSNH 0.212632 0.108961 0.017627
 TANDY 1.055494 0.105649 0.319860
TEXACO 0.613277 0.068076 0.276615
 WEYER 0.816867 0.064448 0.430829
```

The β_i and R^2 values are illustrated graphically in Figure 15.1. The assets most sensitive to the market factor (those with highest β_i values) are the technology and forest sector stocks `DATGEN`, `DEC`, `TANDY` and `WEYER`. Those least sensitive are the utility stocks `CONED` and `PSNH`. These stocks also have the lowest R^2 values.

The single factor covariance matrix (15.6) and corresponding correlation matrix are computed using

```
> cov.si = var(market)*(beta.hat%o%beta.hat) +
```

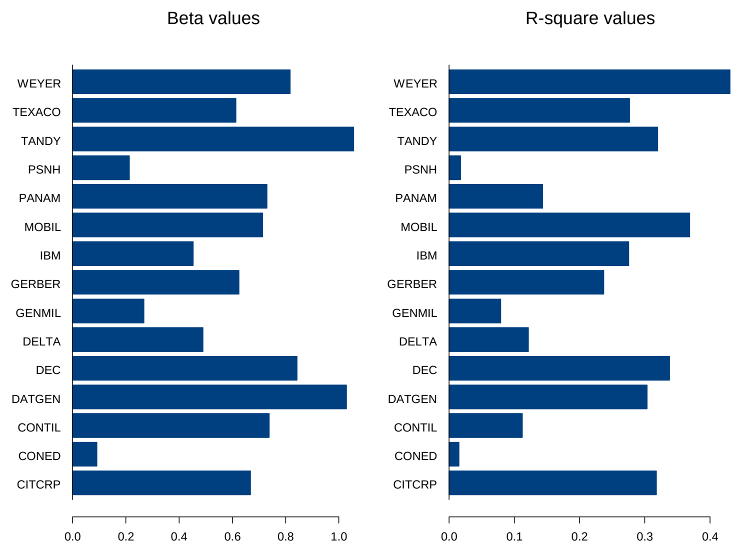


FIGURE 15.1. Estimated β_i and R^2 values from single index model for Berndt data.

```
+ diag(diagD.hat)
> sd = sqrt(diag(cov.si))
> cor.si = cov.si/outer(sd,sd)
```

Since all estimated β_i values are positive, all of the values in the single factor covariance (15.6) will be positive. To illustrate, some of the single factor correlations are displayed below

```
> print(cor.si,digits=1,width=2)
       CITCRP CONED CONTIL DATGEN  DEC DELTA GENMIL GERBER
CITCRP   1.00  0.07   0.19   0.31 0.33  0.20   0.16   0.27
 CONED   0.07  1.00   0.04   0.07 0.07  0.04   0.03   0.06
CONTIL   0.19  0.04   1.00   0.18 0.19  0.12   0.09   0.16
DATGEN   0.31  0.07   0.18   1.00 0.32  0.19   0.15   0.27
   DEC   0.33  0.07   0.19   0.32 1.00  0.20   0.16   0.28
 DELTA   0.20  0.04   0.12   0.19 0.20  1.00   0.10   0.17
GENMIL   0.16  0.03   0.09   0.15 0.16  0.10   1.00   0.14
GERBER   0.27  0.06   0.16   0.27 0.28  0.17   0.14   1.00
   IBM   0.29  0.06   0.17   0.29 0.30  0.18   0.15   0.25
 MOBIL   0.34  0.07   0.20   0.33 0.35  0.21   0.17   0.29
 PANAM   0.21  0.05   0.13   0.21 0.22  0.13   0.11   0.18
  PSNH   0.07  0.02   0.04   0.07 0.08  0.05   0.04   0.06
 TANDY   0.32  0.07   0.19   0.31 0.33  0.20   0.16   0.27
```

```
TEXACO   0.29  0.06   0.17   0.29 0.30  0.18   0.15   0.25
 WEYER   0.37  0.08   0.22   0.36 0.38  0.23   0.18   0.32
...
```

These correlations may be compared with the sample correlations

```
> print(cor(returns),digits=1,width=2)
       CITCRP  CONED CONTIL DATGEN  DEC DELTA GENMIL
CITCRP    1.0  0.269    0.5   0.53 0.49  0.40  0.473
 CONED    0.3  1.000    0.1   0.10 0.11  0.09  0.329
CONTIL    0.5  0.105    1.0   0.26 0.23  0.17  0.206
DATGEN    0.5  0.096    0.3   1.00 0.58  0.33  0.280
   DEC    0.5  0.108    0.2   0.58 1.00  0.43  0.212
 DELTA    0.4  0.092    0.2   0.33 0.43  1.00  0.373
GENMIL    0.5  0.329    0.2   0.28 0.21  0.37  1.000
GERBER    0.4  0.171    0.4   0.14 0.16  0.19  0.350
   IBM    0.4  0.091    0.3   0.49 0.44  0.34  0.170
 MOBIL    0.3  0.003    0.2   0.32 0.41  0.13  0.047
 PANAM    0.3  0.163    0.1   0.29 0.27  0.39  0.207
  PSNH    0.1  0.112    0.1   0.08 0.04  0.03  0.059
 TANDY    0.5  0.102    0.2   0.51 0.49  0.46  0.400
TEXACO    0.3 -0.106    0.2   0.32 0.25  0.13  0.002
 WEYER    0.5  0.158    0.2   0.48 0.59  0.49  0.357
...
```

Another way to compare the single index covariance matrix to the sample covariance is to compute the global minimum variance portfolio. The global minimum variance portfolio is the portfolio w that solves

$$\min_{w}\ \sigma^2_{p,w} = \mathbf{w}'\mathbf{\Omega}\mathbf{w} \text{ s.t. } \mathbf{w}'\mathbf{1} = 1$$

and is given by

$$\mathbf{w} = \frac{\mathbf{\Omega}^{-1}\mathbf{1}}{\mathbf{1}'\mathbf{\Omega}^{-1}\mathbf{1}}$$

The global minimum variance portfolios based on the single index covariance and the sample covariance are

```
> w.gmin.si = solve(cov.si)%*%rep(1,nrow(cov.si))
> w.gmin.si = w.gmin.si/sum(w.gmin.si)
> t(w.gmin.si)
numeric matrix: 1 rows, 15 columns.
      CITCRP  CONED   CONTIL   DATGEN       DEC  DELTA
[1,] 0.04379 0.3757 0.005229 -0.02348 -0.004413 0.0525

     GENMIL  GERBER    IBM   MOBIL    PANAM    PSNH
[1,] 0.1819 0.04272 0.1866 0.03372 0.007792 0.06618
```

```
        TANDY  TEXACO    WEYER
[1,] -0.02719 0.05782 0.001173

> w.gmin.sample = solve(var(returns))%*%rep(1,nrow(cov.si))
> w.gmin.sample = w.gmin.sample/sum(w.gmin.sample)
> t(w.gmin.sample)
numeric matrix: 1 rows, 15 columns.
       CITCRP  CONED    CONTIL   DATGEN     DEC   DELTA
[1,] -0.06035 0.3763 -0.002152 -0.06558 0.03626 0.03155

     GENMIL   GERBER    IBM   MOBIL   PANAM    PSNH
[1,] 0.1977 -0.02966 0.2846 0.02257 0.01071 0.07517

        TANDY TEXACO    WEYER
[1,] -0.01868 0.1996 -0.05804
```

15.3.2 The General Multifactor Model

The general macroeconomic multifactor model specifies K observable macro-variables as the factor realizations $\mathbf{f}_t$ in (15.1). The paper by Chen, Roll and Ross (1986) provides a description of the most commonly used macroeconomic factors. Typically, the macroeconomic factors are standardized to have mean zero and a common scale. The factors must also be transformed to be *stationary* (not trending). Sometimes the factors are made orthogonal but this in not necessary.

The general form of the covariance matrix for the macroeconomic factor model is

$$\mathbf{\Omega} = \mathbf{B}\mathbf{\Omega}_f\mathbf{B}' + \mathbf{D}$$

where $\mathbf{B} = [\boldsymbol{\beta}_1, \boldsymbol{\beta}_2, \cdots, \boldsymbol{\beta}_N]'$, $\mathbf{\Omega}_f = E[(\mathbf{f}_t - \boldsymbol{\mu}_f)(\mathbf{f}_t - \boldsymbol{\mu}_f)']$ is the covariance matrix of the observed factors and $\mathbf{D}$ is a diagonal matrix with $\sigma_i^2 = \text{var}(\varepsilon_{it})$ along the diagonal.

Because the factor realizations are observable, the parameter matrices $\mathbf{B}$ and $\mathbf{D}$ of the model may be estimated using time series regression giving

$$\begin{aligned}\mathbf{R}_i &= \widehat{\alpha}_i\mathbf{1} + \mathbf{F}\widehat{\boldsymbol{\beta}}_i + \widehat{\varepsilon}_i, \ i = 1, \ldots, N \\ \widehat{\sigma}_i^2 &= \frac{1}{T-K-1}\widehat{\varepsilon}_i'\widehat{\varepsilon}_i\end{aligned}$$

The covariance matrix of the factor realizations may be estimated using the time series sample covariance matrix

$$\begin{aligned}\widehat{\mathbf{\Omega}}_f &= \frac{1}{T-1}\sum_{t=1}^{T}(\mathbf{f}_t - \overline{\mathbf{f}})(\mathbf{f}_t - \overline{\mathbf{f}})' \\ \overline{\mathbf{f}} &= \frac{1}{T}\sum_{t=1}^{T}\mathbf{f}_t\end{aligned}$$

The estimated multifactor model covariance matrix is then

$$\widehat{\mathbf{\Omega}} = \widehat{\mathbf{B}}\widehat{\mathbf{\Omega}}_f\widehat{\mathbf{B}}' + \widehat{\mathbf{D}} \tag{15.7}$$

Remarks

1. As with the single factor model, robust regression may be used to compute $\boldsymbol{\beta}_i$ and σ_i^2. A robust covariance matrix estimator may also be used to compute and estimate of $\mathbf{\Omega}_f$.

Example 109 *Estimating a general macroeconomic factor model using S-PLUS*

As explained in Chen, Roll and Ross (1986), the macroeconomic factors should be constructed as surprise variables so that the returns on assets will respond to unexpected changes in economy-wide factors. To illustrate the construction of a macroeconomic factor model with macroeconomic surprise variables, consider a simple factor model for the monthly returns in the "`timeSeries`" `returns`, constructed earlier, using as factors the surprise to inflation and the surprise to industrial production growth. Monthly observations on inflation and industrial production growth are constructed from the `S+FinMetrics` "`timeSeries`" `CPI.dat` and `IP.dat` as follows

```
> infl = getReturns(CPI.dat)
> ipg = getReturns(IP.dat)
```

In general, to compute surprise variables, one must first explain the expected behavior and then define the surprise to be the difference between what is observed and what is expected. A common way to compute the expected behavior is to use a VAR model. For simplicity, consider a VAR(6) model for inflation and industrial production growth fit over the period July 1977 through December 1987

```
> factor.ts = seriesMerge(ipg,infl)
> var6.fit = VAR(cbind(CPI,IP)~ar(6),data=factor.ts,
> start="July 1977",end="Jan 1988",in.format="%m %Y")
```

The start date of July 1977 allows for six initial values so that the first fitted value is for January 1978. The factor surprises are constructed as the residuals from the VAR(6) fit:

```
> factor.surprise = residuals(var6.fit)
```

The factor betas and R^2 values for the fifteen assets in the "`timeSeries`" `returns` are computed using

```
> factor.surprise = as.matrix(seriesData(factor.surprise))
> n.obs = nrow(returns)
> X.mat = cbind(rep(1,n.obs),factor.surprise)
```

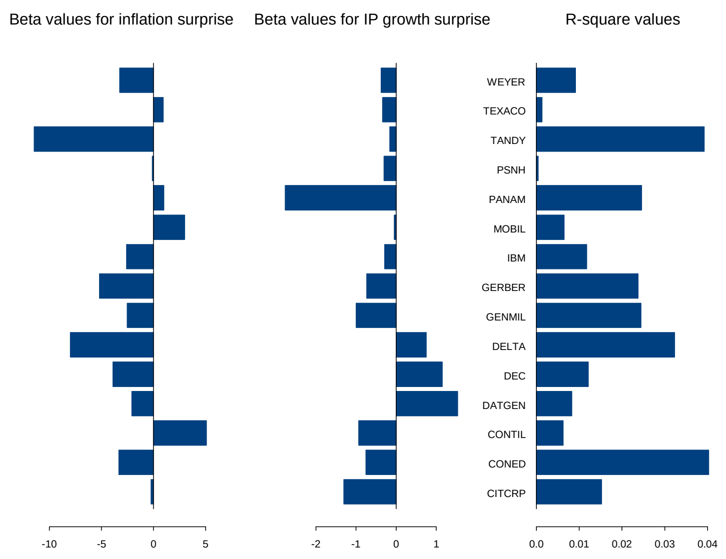


FIGURE 15.2. Estimated macroeconomic factor model for Berndt data.

```
> G.hat = solve(X.mat,returns)
> beta.hat = t(G.hat[2:3,])
> E.hat = returns - X.mat%*%G.hat
> diagD.hat = diag(crossprod(E.hat)/(n.obs-3))
> names(diagD.hat) = colIds(G.hat)
> r.sq = 1-(n.obs-3)*diagD.hat/diag(var(returns,SumSquares=T))
```

These results are illustrated graphically in Figure 15.2 created by

```
> par(mfrow=c(1,3))
> barplot(beta.hat[,1],names=names(beta.hat),horiz=T,
+ main="Beta values for inflation surprise")
> barplot(beta.hat[,2],names=names(beta.hat),horiz=T,
+ main="Beta values for IP growth surprise")
> barplot(r.sq,names=names(r.sq),horiz=T,
+ main="R-square values")
```

Most stocks have negative loadings on the inflation surprise factor. Notice the very low R^2 values indicating that the factor surprises do not explain much of the variability in the monthly asset returns.

The estimated factor model covariance using (15.7) is

```
> cov.macro = beta.hat%*%var(factor.surprise)%*%t(beta.hat) +
+ diag(diagD.hat)
```

and the corresponding correlation matrix is

```
> sd = sqrt(diag(cov.macro))
> cor.macro = cov.macro/outer(sd,sd)
> print(cor.macro,digits=1,width=2)
       CITCRP   CONED  CONTIL DATGEN    DEC   DELTA GENMIL  GERBER
CITCRP  1.000  0.0181  0.0035 -0.010 -0.008 -0.0019  0.017  0.0115
 CONED  0.018  1.0000 -0.0056 -0.007  0.002  0.0214  0.030  0.0300
CONTIL  0.004 -0.0056  1.0000 -0.005 -0.008 -0.0134 -0.002 -0.0062
DATGEN -0.010 -0.0069 -0.0052  1.000  0.009  0.0081 -0.008 -0.0030
   DEC -0.008  0.0017 -0.0083  0.009  1.000  0.0164 -0.002  0.0042
 DELTA -0.002  0.0214 -0.0134  0.008  0.016  1.0000  0.011  0.0200
GENMIL  0.017  0.0301 -0.0017 -0.008 -0.002  0.0114  1.000  0.0218
GERBER  0.011  0.0300 -0.0062 -0.003  0.004  0.0200  0.022  1.0000
   IBM  0.007  0.0208 -0.0049 -0.001  0.004  0.0150  0.015  0.0164
 MOBIL -0.002 -0.0128  0.0053 -0.002 -0.006 -0.0137 -0.008 -0.0109
 PANAM  0.019  0.0195  0.0061 -0.013 -0.012 -0.0066  0.019  0.0115
  PSNH  0.003  0.0033  0.0005 -0.002 -0.001 -0.0001  0.003  0.0021
 TANDY  0.007  0.0335 -0.0121  0.003  0.013  0.0325  0.022  0.0280
TEXACO  0.003  0.0002  0.0027 -0.003 -0.004 -0.0051  0.001 -0.0008
 WEYER  0.007  0.0183 -0.0042 -0.001  0.003  0.0131  0.013  0.014
...
```

The macroeconomic factor model global minimum variance portfolio is

```
> w.gmin.macro = solve(cov.macro)%*%rep(1,nrow(cov.macro))
> w.gmin.macro = w.gmin.macro/sum(w.gmin.macro)
> t(w.gmin.macro)
numeric matrix: 1 rows, 15 columns.
      CITCRP  CONED  CONTIL  DATGEN     DEC   DELTA GENMIL
[1,] 0.06958 0.1776 0.02309 0.03196 0.04976 0.04766 0.1049

      GERBER    IBM   MOBIL   PANAM    PSNH   TANDY
[1,] 0.05463 0.1318 0.08186 0.02469 0.04019 0.02282

      TEXACO   WEYER
[1,] 0.07759 0.06185
```

15.4 Fundamental Factor Model

Fundamental factor models use observable asset specific characteristics (fundamentals) like industry classification, market capitalization, style classification (value, growth) etc. to determine the common risk factors. In practice, fundamental factor models are estimated in two ways. The first way was pioneered by Bar Rosenberg, founder of BARRA Inc., and is discussed at length in Grinold and Kahn (2000). In this approach, hereafter referred to as the "BARRA" approach, the observable asset specific fundamentals (or some transformation of them) are treated as the factor betas, $\boldsymbol{\beta}_i$, which are time invariant[3]. The factor realizations at time t, $\mathbf{f}_t$, however, are unobservable. The econometric problem is then to estimate the factor realizations at time t given the factor betas. This is done by running T cross-section regressions. The second way was introduced by Eugene Fama and Kenneth French (1992) and is referred to as the "Fama-French" approach. For a given observed asset specific characteristic, e.g. size, they determined factor realizations using a two step process. First they sorted the cross-section of assets based on the values of the asset specific characteristic. Then they formed a hedge portfolio which is long in the top quintile of the sorted assets and short in the bottom quintile of the sorted assets. The observed return on this hedge portfolio at time t is the observed factor realization for the asset specific characteristic. This process is repeated for each asset specific characteristic. Then, given the observed factor realizations for $t = 1, \ldots, T$ the factor betas for each asset are estimated using N time series regressions.

15.4.1 BARRA-type Single Factor Model

Consider a single factor model in the form of a cross-sectional regression at time t

$$\underset{(N\times 1)}{\mathbf{R}_t} = \underset{(N\times 1)}{\boldsymbol{\beta}}\underset{(1\times 1)}{f_t} + \underset{(N\times 1)}{\boldsymbol{\varepsilon}_t}, t = 1, \ldots, T$$

where $\boldsymbol{\beta}$ is a vector of observed values of an asset specific attribute (e.g., market capitalization, industry classification, style classification) and f_t is an unobserved factor realization. It is assumed that

$$\begin{aligned} \text{var}(f_t) &= \sigma_f^2 \\ \text{cov}(f_t, \varepsilon_{it}) &= 0, \text{ for all } i, t \\ \text{var}(\varepsilon_{it}) &= \sigma_i^2, i = 1, \ldots, N. \end{aligned}$$

In the above model the factor realization f_t is the parameter to be estimated for each time period $t = 1, \ldots, T$. Since the error term $\boldsymbol{\varepsilon}_t$ is heteroskedastic, efficient estimation of f_t is done by weighted least squares (WLS) (assuming the asset specific variances σ_i^2 are known)

$$\hat{f}_{t,\text{wls}} = (\boldsymbol{\beta}'\mathbf{D}^{-1}\boldsymbol{\beta})^{-1}\boldsymbol{\beta}'\mathbf{D}^{-1}\mathbf{R}_t,\ t = 1, \ldots, T \tag{15.8}$$

where $\mathbf{D}$ is a diagonal matrix with σ_i^2 along the diagonal. The above WLS estimate of f_t is infeasible since σ_i^2 is not known. However, σ_i^2 may be

[3]See Sheikh (1995) for a description of the BARRA fundamental factor model for U.S. equities.

consistently estimated and a feasible WLS estimate may be computed. How σ_i^2 may be consistently estimated and how a feasible WLS estimate may be computed is illustrated below.

The WLS estimate of f_t in (15.8) has an interesting interpretation as the return on a portfolio $\mathbf{h} = (h_1, \ldots, h_N)'$ that solves

$$\min_{\mathbf{h}} \frac{1}{2}\mathbf{h}'\mathbf{D}\mathbf{h} \text{ subject to } \mathbf{h}'\boldsymbol{\beta} = 1$$

The portfolio $\mathbf{h}$ minimizes asset return residual variance subject to having unit exposure to the attribute β and is given by

$$\mathbf{h}' = (\boldsymbol{\beta}'\mathbf{D}^{-1}\boldsymbol{\beta})^{-1}\boldsymbol{\beta}'\mathbf{D}^{-1}$$

The estimated factor realization is then the portfolio return

$$\hat{f}_{t,\text{wls}} = \mathbf{h}'\mathbf{R}_t$$

When the portfolio $\mathbf{h}$ is normalized such that $\sum_i^N h_i = 1$, it is referred to as a *factor mimicking portfolio.*

15.4.2 BARRA-type Industry Factor Model

As an example of a fundamental factor model with K factors, consider a stylized BARRA-type industry factor model with K mutually exclusive industries. The factor sensitivities β_{ik} in (15.1) for each asset are time invariant and of the form

$$\begin{aligned} \beta_{ik} &= 1 \text{ if asset } i \text{ is in industry } k \\ &= 0, \text{ otherwise} \end{aligned}$$

and f_{kt} represents the factor realization for the k^{th} industry in time period t. Notice that factor betas are simply dummy variables indicating whether a given asset is in a particular industry. Hence, the industry factor betas do not have to be estimated from the data. The factor realizations, however, are not initially observable. As will become apparent, the estimated value of f_{kt} will be equal to the weighted average excess return in time period t of the firms operating in industry k. This weighted average excess return at time t can be easily estimated using a cross-section regression over all asset returns at time t.

The industry factor model with K industries is summarized as

$$\begin{aligned} R_{it} &= \beta_{i1}f_{1t} + \cdots + \beta_{iK}f_{Kt} + \varepsilon_{it},\ i = 1, \ldots, N; t = 1, \ldots, T \\ \text{var}(\varepsilon_{it}) &= \sigma_i^2,\ i = 1, \ldots, N \\ \text{cov}(\varepsilon_{it}, f_{jt}) &= 0,\ j = 1, \ldots, K;\ i = 1, \ldots, N \\ \text{cov}(f_{it}, f_{jt}) &= \sigma_{ij}^f,\ i, j = 1, \ldots, K \end{aligned}$$

where $\beta_{ik} = 1$ if asset i is in industry k $(k = 1, \ldots, K)$ and is zero otherwise[4]. It is assumed that there are N_k firms in the kth industry such $\sum_{k=1}^{K} N_k = N$.

Least Squares Estimation of the Factor Realizations

The factor realizations $f_{1t}, \ldots, f_{Kt}$ for $t = 1, \ldots, T$, can be estimated from the observed cross-section of asset returns at time period t as follows. Consider the cross-section regression at time t

$$\begin{aligned} \mathbf{R}_t &= \boldsymbol{\beta}_1 f_{1t} + \cdots + \boldsymbol{\beta}_K f_{Kt} + \boldsymbol{\varepsilon}_t, \\ &= \mathbf{B}\mathbf{f}_t + \boldsymbol{\varepsilon}_t \\ E[\boldsymbol{\varepsilon}_t \boldsymbol{\varepsilon}_t'] &= \mathbf{D},\ \mathrm{cov}(\mathbf{f}_t) = \boldsymbol{\Omega}_f \end{aligned} \tag{15.9}$$

where $\mathbf{R}_t$ is an $(N \times 1)$ vector of returns, $\mathbf{B} = [\boldsymbol{\beta}_1, \ldots, \boldsymbol{\beta}_K]$ is a $(N \times K)$ matrix of zeros and ones reflecting the industry factor sensitivities for each asset, $\mathbf{f}_t = (f_{1t}, \ldots, f_{Kt})'$ is a $(K \times 1)$ vector of unobserved factor realizations, $\boldsymbol{\varepsilon}_t$ is an $(N \times 1)$ error term, and $\mathbf{D}$ is a diagonal matrix with σ_i^2 along the diagonal. Note that the error term is heteroskedastic across assets. Since the industries are mutually exclusive it follows that

$$\boldsymbol{\beta}_j' \boldsymbol{\beta}_k = N_k \text{ for } j = k,\ 0 \text{ otherwise} \tag{15.10}$$

An unbiased but inefficient estimate of the factor realizations $\mathbf{f}_t$ can be obtained by OLS giving

$$\widehat{\mathbf{f}}_{t,\mathrm{OLS}} = (\mathbf{B}'\mathbf{B})^{-1} \mathbf{B}' \mathbf{R}_t \tag{15.11}$$

or

$$\begin{pmatrix} \widehat{f}_{1t,\mathrm{OLS}} \\ \vdots \\ \widehat{f}_{Kt,\mathrm{OLS}} \end{pmatrix} = \begin{pmatrix} \frac{1}{N_1} \sum_{i=1}^{N_1} R_{it}^1 \\ \vdots \\ \frac{1}{N_K} \sum_{i=1}^{N_K} R_{it}^K \end{pmatrix}$$

using (15.10) where R_{it}^k denotes the return on asset i if it is in industry k. Here, the estimated factor realizations $\widehat{f}_{kt}$ have nice interpretations. They represent an equally weighted average return in time period t on the industry k assets. Of course, this is expected given the nature of the binary industry factor beta values.

To get the time series of factor realizations, the cross-section regression (15.9) needs to be estimated for each $t = 1, \ldots, T$ giving the estimated factor realizations $(\widehat{\mathbf{f}}_{1,\mathrm{OLS}}, \ldots, \widehat{\mathbf{f}}_{T,\mathrm{OLS}})$.

[4]Notice that there is no intercept in the industry factor model. With K mutually exclusive industries, the intercept will be collinear with the factor betas and not identifiable.

Estimation of Factor Realization Covariance Matrix

Given the time series of factor realizations, the covariance matrix of the industry factors may be computed as the time series sample covariance

$$\begin{aligned}
\widehat{\mathbf{\Omega}}^F_{\text{OLS}} &= \frac{1}{T-1}\sum_{t=1}^{T}(\widehat{\mathbf{f}}_{t,\text{OLS}} - \overline{\mathbf{f}}_{\text{OLS}})(\widehat{\mathbf{f}}_{t,\text{OLS}} - \overline{\mathbf{f}}_{\text{OLS}})', \qquad (15.12)\\
\overline{\mathbf{f}}_{\text{OLS}} &= \frac{1}{T}\sum_{t=1}^{T}\widehat{\mathbf{f}}_{t,\text{OLS}}
\end{aligned}$$

Estimation of Residual Variances

The residual variances, $\text{var}(\varepsilon_{it}) = \sigma_i^2$, can be estimated from the time series of residuals from the T cross-section regressions given in (15.9) as follows. Let $\widehat{\boldsymbol{\varepsilon}}_{t,\text{OLS}}$, $t = 1, \ldots, T$, denote the $(N \times 1)$ vector of OLS residuals from (15.9), and let $\widehat{\varepsilon}_{it,\text{OLS}}$ denote the i^{th} row of $\widehat{\boldsymbol{\varepsilon}}_{t,\text{OLS}}$. Then σ_i^2 may be estimated using

$$\begin{aligned}
\widehat{\sigma}^2_{i,\text{OLS}} &= \frac{1}{T-1}\sum_{t=1}^{T}(\widehat{\varepsilon}_{it,\text{OLS}} - \overline{\varepsilon}_{i,\text{OLS}})^2,\ i = 1, \ldots, N \qquad (15.13)\\
\overline{\varepsilon}_{i,\text{OLS}} &= \frac{1}{T}\sum_{t=1}^{T}\widehat{\varepsilon}_{it,\text{OLS}}
\end{aligned}$$

Estimation of Industry Factor Model Asset Return Covariance Matrix

The covariance matrix of the N assets is then estimated by

$$\widehat{\mathbf{\Omega}}_{\text{OLS}} = \mathbf{B}\widehat{\mathbf{\Omega}}^F_{\text{OLS}}\mathbf{B}' + \widehat{\mathbf{D}}_{\text{OLS}}$$

where $\widehat{\mathbf{D}}_{\text{OLS}}$ is a diagonal matrix with $\widehat{\sigma}^2_{i,\text{OLS}}$ along the diagonal.

Remarks

1. Multivariate regression may be used to compute all of the factor returns in one step. The multivariate regression model is

$$\mathbf{R} = \mathbf{BF} + \mathbf{E},$$

where $\mathbf{R}$ is a $(N \times T)$ matrix of cross-sectionally demeaned asset returns, $\mathbf{F}$ is a $(K \times T)$ matrix of parameters to be estimated (factor returns) and $\mathbf{E}$ is a $(N \times T)$ matrix of errors such that $E[\mathbf{EE}'] = \mathbf{D}$.

2. Robust regression techniques can be used to estimate $\mathbf{f}_t$, and a robust covariance matrix estimate of $\mathbf{\Omega}_f$ can be computed.

3. The industry factor model may be extended to cover cases where an asset may be classified into several industry categories.

4. Given the estimated factor realizations, a time series regression may be run to assess the constructed model. The estimated factor loading may be compared to the imposed values and the proportion of asset variance attributable to all of the factors may be computed.

Weighted Least Squares Estimation

The OLS estimation of the factor realizations $\mathbf{f}_t$ is inefficient due to the cross-sectional heteroskedasticity in the asset returns. The estimates of the residual variances from (15.13) may be used as weights for weighted least squares (feasible GLS) estimation:

$$\widehat{\mathbf{f}}_{t,\mathrm{GLS}} = (\mathbf{B}'\widehat{\mathbf{D}}_{\mathrm{OLS}}^{-1}\mathbf{B})^{-1}\mathbf{B}'\widehat{\mathbf{D}}_{\mathrm{OLS}}^{-1}(\mathbf{R}_t - \overline{R}_t\mathbf{1}),\ t = 1, \ldots, T \tag{15.14}$$

Given the time series of factor realizations, $(\widehat{\mathbf{f}}_{1,\mathrm{GLS}}, \ldots, \widehat{\mathbf{f}}_{T,\mathrm{GLS}})$, the covariance matrix of the industry factors may be computed as the time series sample covariance

$$\begin{aligned}\widehat{\mathbf{\Omega}}_{\mathrm{GLS}}^{F} &= \frac{1}{T-1}\sum_{t=1}^{T}(\widehat{\mathbf{f}}_{t,\mathrm{GLS}} - \overline{\mathbf{f}}_{\mathrm{GLS}})(\widehat{\mathbf{f}}_{t,\mathrm{GLS}} - \overline{\mathbf{f}}_{\mathrm{GLS}})', \qquad (15.15)\\ \overline{\mathbf{f}}_{\mathrm{GLS}} &= \frac{1}{T}\sum_{t=1}^{T}\widehat{\mathbf{f}}_{t,\mathrm{GLS}}\end{aligned}$$

The residual variances, $\mathrm{var}(\varepsilon_{it}) = \sigma_i^2$, can be re-estimated from the time series of residuals from the T cross-section GLS regressions as follows. Let $\widehat{\boldsymbol{\varepsilon}}_{t,\mathrm{GLS}}$, $t = 1, \ldots, T$, denote the $(N \times 1)$ vector of GLS residuals from the industry factor model (15.9) and let $\widehat{\varepsilon}_{it,\mathrm{GLS}}$ denote the i^{th} row of $\widehat{\boldsymbol{\varepsilon}}_{t,\mathrm{GLS}}$. Then σ_i^2 may be estimated using

$$\begin{aligned}\widehat{\sigma}_{i,\mathrm{GLS}}^2 &= \frac{1}{T-1}\sum_{t=1}^{T}(\widehat{\varepsilon}_{it,\mathrm{GLS}} - \overline{\varepsilon}_{i,\mathrm{GLS}})^2,\ i = 1, \ldots, N \qquad (15.16)\\ \overline{\varepsilon}_{i,\mathrm{GLS}} &= \frac{1}{T}\sum_{t=1}^{T}\widehat{\varepsilon}_{it,\mathrm{GLS}}\end{aligned}$$

The covariance matrix of the N assets is then estimated by

$$\widehat{\mathbf{\Omega}}_{\mathrm{GLS}} = \mathbf{B}\widehat{\mathbf{\Omega}}_{\mathrm{GLS}}^{F}\mathbf{B}' + \widehat{\mathbf{D}}_{\mathrm{GLS}}$$

where $\widehat{\mathbf{D}}_{\mathrm{GLS}}$ is a diagonal matrix with $\widehat{\sigma}_{i,\mathrm{GLS}}^2$ along the diagonal.

Remarks

1. Since $\mathbf{B}$ and $\widehat{\mathbf{D}}_{\mathrm{OLS}}$ are time invariant, $(\mathbf{B}'\widehat{\mathbf{D}}_{\mathrm{OLS}}^{-1}\mathbf{B})^{-1}\mathbf{B}'\widehat{\mathbf{D}}_{\mathrm{OLS}}^{-1}$ only needs to be computed once, and this greatly speeds up the computation of $\widehat{\mathbf{f}}_{t,\mathrm{GLS}}$ $(t = 1, \ldots, T)$.

2. In principle, the GLS estimator may be iterated. Iteration does not improve the asymptotic efficiency of the estimator, and it may perform better or worse than the non-iterated estimator.
3. Weighted robust regression techniques can be used to estimate $\mathbf{f}_t$, and a robust covariance matrix estimate of $\mathbf{\Omega}_f$ can be computed.

Factor Mimicking Portfolios

The GLS estimates of the factor realizations (15.14) are just linear combinations of the observed returns in each industry. Further, these linear combinations sum to unity so that they can be interpreted as *factor mimicking portfolios*. Notice that they are simply weighted averages of the returns in each industry where the weights on each asset are based on the size of the residual variance. The $(N \times 1)$ vector of weights for the ith factor mimicking portfolio is given by

$$\mathbf{w}_i = \mathbf{H}_i = \left((\mathbf{B}'\widehat{\mathbf{D}}_{\text{OLS}}^{-1}\mathbf{B})^{-1}\mathbf{B}'\widehat{\mathbf{D}}_{\text{OLS},}^{-1} \right)_i, \; i = 1, \ldots, K$$

where $\mathbf{H}_i$ denotes the ith row of $\mathbf{H}$.

Seemingly Unrelated Regression Formulation of Industry Factor Model

The industry factor model may be expressed as a *seemingly unrelated regression* (SUR) model. The cross section regression models (15.9) can be stacked to form the giant regression

$$\begin{pmatrix} \mathbf{R}_1 \\ \vdots \\ \mathbf{R}_T \end{pmatrix} = \begin{pmatrix} \mathbf{B} & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \mathbf{B} \end{pmatrix} \begin{pmatrix} \mathbf{f}_1 \\ \vdots \\ \mathbf{f}_T \end{pmatrix} + \begin{pmatrix} \boldsymbol{\varepsilon}_1 \\ \vdots \\ \boldsymbol{\varepsilon}_T \end{pmatrix}.$$

The giant regression may be compactly expressed using Kronecker products as

$$\begin{aligned} \text{vec}(\mathbf{R}) &= (\mathbf{I}_T \otimes \mathbf{B})\mathbf{f} + \boldsymbol{\varepsilon} \\ E[\boldsymbol{\varepsilon}\boldsymbol{\varepsilon}'] &= \mathbf{I}_T \otimes \mathbf{D} \end{aligned}$$

where vec($\mathbf{R}$) is a $(NT \times 1)$ vector of returns, $\mathbf{f}$ is a $(TK \times 1)$ vector of factor realizations, and $\boldsymbol{\varepsilon}$ is a $(NT \times 1)$ vector of errors. The GLS estimator of $\mathbf{f}$ is

$$\begin{aligned} \widehat{\mathbf{f}}_{\text{GLS}} &= \left[(\mathbf{I}_T \otimes \mathbf{B})'(\mathbf{I}_T \otimes \mathbf{D})^{-1}(\mathbf{I}_T \otimes \mathbf{B}) \right]^{-1} (\mathbf{I}_T \otimes \mathbf{B})'(\mathbf{I}_T \otimes \mathbf{D})^{-1}\mathbf{R} \\ &= \left[\mathbf{I}_T \otimes (\mathbf{B}'\mathbf{D}^{-1}\mathbf{B})^{-1}\mathbf{B}'\mathbf{D}^{-1} \right] \mathbf{R} \end{aligned}$$

or

$$\begin{pmatrix} \widehat{\mathbf{f}}_{1,\text{GLS}} \\ \vdots \\ \widehat{\mathbf{f}}_{T,\text{GLS}} \end{pmatrix} = \begin{pmatrix} (\mathbf{B}'\mathbf{D}^{-1}\mathbf{B})^{-1}\mathbf{B}'\mathbf{D}^{-1}\mathbf{R}_1 \\ \vdots \\ (\mathbf{B}'\mathbf{D}^{-1}\mathbf{B})^{-1}\mathbf{B}'\mathbf{D}^{-1}\mathbf{R}_T \end{pmatrix}$$

which is just weighted least squares on each of the cross section regressions (15.9). Hence, equation by equation GLS estimation of (15.9) is efficient.

Of course, the above GLS estimator is not feasible because it requires knowledge of the firm specific variances in **D**. However, using the techniques described above to estimate σ_i^2, feasible GLS estimation is possible.

Example 110 *Estimating an industry factor model using* `S-PLUS`

Consider creating a three industry factor model for the fifteen assets taken from the `S+FinMetrics` "`timeSeries`" `berndt.dat`. The three industries are defined to be "technology", "oil" and "other". The 15×3 matrix **B** of industry factor loadings are created using

```
> n.stocks = numCols(returns)
> tech.dum = oil.dum = other.dum = matrix(0,n.stocks,1)
> tech.dum[c(4,5,9,13),] = 1
> oil.dum[c(3,6,10,11,14),] = 1
> other.dum = 1 - tech.dum - oil.dum
> B = cbind(tech.dum,oil.dum,other.dum)
> dimnames(B) = list(colIds(returns),c("TECH","OIL","OTHER"))
> B
integer matrix: 15 rows, 3 columns.
       TECH OIL OTHER
CITCRP    0   0     1
 CONED    0   0     1
CONTIL    0   1     0
DATGEN    1   0     0
   DEC    1   0     0
 DELTA    0   1     0
GENMIL    0   0     1
GERBER    0   0     1
   IBM    1   0     0
 MOBIL    0   1     0
 PANAM    0   1     0
  PSNH    0   0     1
 TANDY    1   0     0
TEXACO    0   1     0
 WEYER    0   0     1
```

The multivariate least squares estimates of the factor realizations are

```
> returns = t(returns)
> F.hat = solve(crossprod(B))%*%t(B)%*%returns
```

The multivariate GLS estimates are computed using

```
> E.hat = returns - B%*%F.hat
> diagD.hat = rowVars(E.hat)
> Dinv.hat = diag(diagD.hat^(-1))
> H = solve(t(B)%*%Dinv.hat%*%B)%*%t(B)%*%Dinv.hat
> F.hat = H%*%returns
> F.hat = t(F.hat)
```

The rows of the matrix `H` contain the weights for the factor mimicking portfolios:

```
> t(H)
numeric matrix: 15 rows, 3 columns.
         TECH     OIL   OTHER
 [1,] 0.0000 0.00000 0.19918
 [2,] 0.0000 0.00000 0.22024
 [3,] 0.0000 0.09611 0.00000
 [4,] 0.2197 0.00000 0.00000
 [5,] 0.3188 0.00000 0.00000
 [6,] 0.0000 0.22326 0.00000
 [7,] 0.0000 0.00000 0.22967
 [8,] 0.0000 0.00000 0.12697
 [9,] 0.2810 0.00000 0.00000
[10,] 0.0000 0.28645 0.00000
[11,] 0.0000 0.11857 0.00000
[12,] 0.0000 0.00000 0.06683
[13,] 0.1806 0.00000 0.00000
[14,] 0.0000 0.27561 0.00000
[15,] 0.0000 0.00000 0.15711
```

Notice that the weights sum to unity

```
> rowSums(H)
 TECH OIL OTHER
    1   1     1
```

The factor realizations are illustrated in Figure 15.3.

The industry factor model covariance and correlation matrices are computed using

```
> cov.ind = B%*%var(F.hat)%*%t(B) + diag(diagD.hat)
> sd = sqrt(diag(cov.ind))
> cor.ind = cov.ind/outer(sd,sd)
> print(cor.ind,digits=1,width=2)
       CITCRP CONED CONTIL DATGEN DEC DELTA GENMIL GERBER
CITCRP    1.0   0.4   0.10    0.2 0.2   0.1    0.3    0.2
 CONED    0.4   1.0   0.14    0.2 0.3   0.2    0.4    0.3
CONTIL    0.1   0.1   1.00    0.1 0.1   0.2    0.1    0.1
DATGEN    0.2   0.2   0.12    1.0 0.3   0.2    0.2    0.1
   DEC    0.2   0.3   0.14    0.3 1.0   0.2    0.2    0.2
 DELTA    0.1   0.2   0.20    0.2 0.2   1.0    0.2    0.1
```

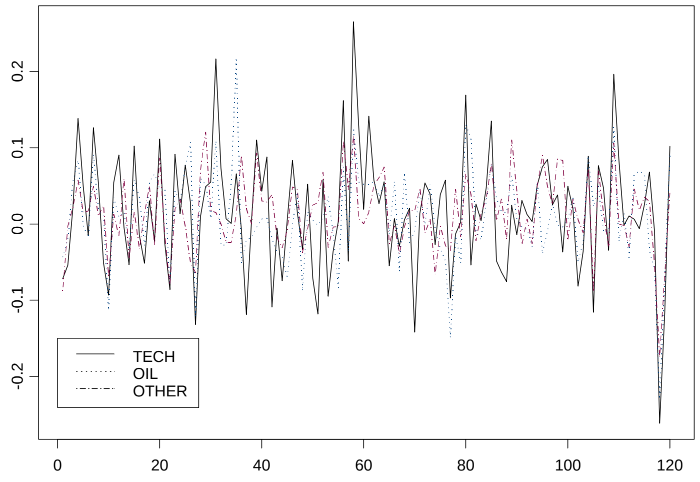


FIGURE 15.3. Estimated industry factor realizations from Berndt data.

```
GENMIL    0.3   0.4   0.12    0.2 0.2   0.2    1.0    0.3
GERBER    0.2   0.3   0.10    0.1 0.2   0.1    0.3    1.0
   IBM    0.2   0.3   0.18    0.4 0.5   0.3    0.3    0.2
 MOBIL    0.2   0.2   0.22    0.2 0.2   0.3    0.2    0.2
 PANAM    0.1   0.2   0.15    0.1 0.2   0.2    0.1    0.1
  PSNH    0.2   0.3   0.08    0.1 0.1   0.1    0.2    0.2
 TANDY    0.2   0.2   0.12    0.3 0.3   0.2    0.2    0.1
TEXACO    0.2   0.2   0.22    0.2 0.2   0.3    0.2    0.2
 WEYER    0.3   0.3   0.10    0.2 0.2   0.1    0.3    0.2
```

The industry factor model global minimum variance portfolio is

```
> w.gmin.ind = solve(cov.ind)%*%rep(1,nrow(cov.ind))
> w.gmin.ind = w.gmin.ind/sum(w.gmin.ind)
> t(w.gmin.ind)
numeric matrix: 1 rows, 15 columns.
     CITCRP  CONED  CONTIL   DATGEN     DEC   DELTA GENMIL
[1,] 0.0905 0.2409 0.02232 0.006256 0.01039 0.05656 0.1416

      GERBER     IBM   MOBIL   PANAM    PSNH    TANDY
[1,] 0.07775 0.02931 0.07861 0.02972 0.04878 0.006455

     TEXACO   WEYER
[1,] 0.0794 0.08149
```

15.5 Statistical Factor Models for Returns

In statistical factor models, the factor realizations $\mathbf{f}_t$ in (15.1) are not directly observable and must be extracted from the observable returns $\mathbf{R}_t$ using statistical methods. The primary methods are *factor analysis* and *principal components analysis*. Traditional factor analysis and principal component analysis are usually applied to extract the factor realizations if the number of time series observations, T, is greater than the number of assets, N. If $N > T$, then the sample covariance matrix of returns becomes singular which complicates traditional factor and principal components analysis. In this case, the method of *asymptotic principal component* analysis due to Connor and Korajczyk (1988) is more appropriate.

Traditional factor and principal component analysis is based on the $(N \times N)$ sample covariance matrix[5]

$$\widehat{\boldsymbol{\Omega}}_N = \frac{1}{T}\mathbf{R}\mathbf{R}'$$

where $\mathbf{R}$ is the $(N \times T)$ matrix of observed returns. Asymptotic principal component analysis is based on the $(T \times T)$ covariance matrix

$$\widehat{\boldsymbol{\Omega}}_T = \frac{1}{N}\mathbf{R}'\mathbf{R}.$$

15.5.1 Factor Analysis

Traditional factor analysis assumes a time invariant *orthogonal factor structure*[6]

$$\begin{aligned}
\underset{(N\times 1)}{\mathbf{R}_t} &= \underset{(N\times 1)}{\boldsymbol{\mu}} + \underset{(N\times K)}{\mathbf{B}}\underset{(K\times 1)}{\mathbf{f}_t} + \underset{(N\times 1)}{\boldsymbol{\varepsilon}_t} \qquad (15.17)\\
\operatorname{cov}(\mathbf{f}_t, \boldsymbol{\varepsilon}_s) &= 0, \text{ for all } t, s\\
E[\mathbf{f}_t] &= E[\boldsymbol{\varepsilon}_t] = 0\\
\operatorname{var}(\mathbf{f}_t) &= \mathbf{I}_K\\
\operatorname{var}(\boldsymbol{\varepsilon}_t) &= \mathbf{D}
\end{aligned}$$

where $\mathbf{D}$ is a diagonal matrix with σ_i^2 along the diagonal. Then, the return covariance matrix, $\boldsymbol{\Omega}$, may be decomposed as

$$\boldsymbol{\Omega} = \mathbf{B}\mathbf{B}' + \mathbf{D}$$

[5]The matrix of returns is assumed to be in deviations about the mean form. In some applications, a mean correction is not used because the means are small.

[6]An excellent overview of factor analysis is given in Johnson and Wichern (1998). Factor analysis using `S-PLUS` is described in the *S-PLUS 6 Guide to Statistics Vol. 2*, chapter 21.

Hence, the K common factors $\mathbf{f}_t$ account for all of the cross covariances of asset returns.

For a given asset i, the return variance variance may be expressed as

$$\operatorname{var}(R_{it}) = \sum_{j=1}^{K} \beta_{ij}^2 + \sigma_i^2$$

The variance portion due to the common factors, $\sum_{j=1}^{K} \beta_{ij}^2$, is called the *communality*, and the variance portion due to specific factors, σ_i^2, is called the *uniqueness*.

The orthogonal factor model (15.17) does not uniquely identify the common factors $\mathbf{f}_t$ and factor loadings $\mathbf{B}$ since for any orthogonal matrix $\mathbf{H}$ such that $\mathbf{H}' = \mathbf{H}^{-1}$

$$\begin{aligned} \mathbf{R}_t &= \boldsymbol{\mu} + \mathbf{BHH}'\mathbf{f}_t + \boldsymbol{\varepsilon}_t \\ &= \mu + \mathbf{B}^*\mathbf{f}_t^* + \varepsilon_t \end{aligned}$$

where $\mathbf{B}^* = \mathbf{BH}$, $\mathbf{f}_t^* = \mathbf{H}'\mathbf{f}_t$ and $\operatorname{var}(\mathbf{f}_t^*) = \mathbf{I}_K$. Because the factors and factor loadings are only identified up to an orthogonal transformation (rotation of coordinates), the interpretation of the factors may not be apparent until suitable rotation is chosen.

Estimation

Estimation using factor analysis consists of three steps:

- Estimation of the factor loading matrix $\mathbf{B}$ and the residual covariance matrix $\mathbf{D}$.
- Construction of the factor realizations $\mathbf{f}_t$.
- Rotation of coordinate system to enhance interpretation

Traditional factor analysis provides maximum likelihood estimates of $\mathbf{B}$ and $\mathbf{D}$ under the assumption that returns are jointly normally distributed and temporally *iid*. Given estimates $\widehat{\mathbf{B}}$ and $\widehat{\mathbf{D}}$, an empirical version of the factor model (15.2) may be constructed as

$$\mathbf{R}_t - \widehat{\boldsymbol{\mu}} = \widehat{\mathbf{B}}\mathbf{f}_t + \widehat{\varepsilon}_t \tag{15.18}$$

where $\widehat{\boldsymbol{\mu}}$ is the sample mean vector of $\mathbf{R}_t$. The error terms in (15.18) are heteroskedastic so that OLS estimation is inefficient. Using (15.18), the factor realizations in a given time period t, $\mathbf{f}_t$, can be estimated using the cross-sectional generalized least squares (GLS) regression

$$\widehat{\mathbf{f}}_t = (\widehat{\mathbf{B}}'\widehat{\mathbf{D}}^{-1}\widehat{\mathbf{B}})^{-1}\widehat{\mathbf{B}}'\widehat{\mathbf{D}}^{-1}(\mathbf{R}_t - \widehat{\boldsymbol{\mu}}) \tag{15.19}$$

Performing this regression for $t = 1, \ldots, T$ times gives the time series of factor realizations $(\widehat{\mathbf{f}}_1, \ldots, \widehat{\mathbf{f}}_T)$.

The factor model estimated covariance matrix is given by

$$\widehat{\boldsymbol{\Omega}}^F = \widehat{\mathbf{B}}\widehat{\mathbf{B}}' + \widehat{\mathbf{D}}$$

Remarks:

- Traditional factor analysis starts with a $\sqrt{T}$- consistent and asymptotically normal estimator of $\boldsymbol{\Omega}$, usually the sample covariance matrix $\widehat{\boldsymbol{\Omega}}$, and makes inference on K based on $\widehat{\boldsymbol{\Omega}}$. A likelihood ratio test is often used to select K under the assumption that ε_{it} is normally distributed (see below). However, when $N \to \infty$ consistent estimation of $\boldsymbol{\Omega}$, an $N \times N$ matrix, is not a well defined problem. Hence, if N is large relative to T, then traditional factor analysis may run into problems. Additionally, typical algorithms for factor analysis are not efficient for very large problems.

- Traditional factor analysis is only appropriate if ε_{it} is cross-sectionally uncorrelated, serially uncorrelated, and serially homoskedastic.

Factor Mimicking Portfolios

From (15.19), we see that the estimated factor realizations $\widehat{\mathbf{f}}_t$ are simply linear combinations of the observed returns $\mathbf{R}_t$. As such, it is possible to normalize the linear combination so that the weights sum to unity. The resulting re-scaled factors are the factor mimicking portfolios and are perfectly correlated with the factor realizations.

Tests for the Number of Factors

Using the maximum likelihood estimates of $\mathbf{B}$ and $\mathbf{D}$ based on a $K-$factor model and the sample covariance matrix $\widehat{\boldsymbol{\Omega}}$, a likelihood ratio test (modified for improved small sample performance) of the adequacy of K factors is of the form

$$\mathrm{LR}(K) = -(T - 1 - \frac{1}{6}(2N+5) - \frac{2}{3}K) \cdot \left(\ln|\widehat{\boldsymbol{\Omega}}| \; - \ln|\widehat{\mathbf{B}}\widehat{\mathbf{B}}' + \widehat{\mathbf{D}}|\right).$$

$\mathrm{LR}(K)$ is asymptotically chi-square with $\frac{1}{2}\left((N-K)^2 - N - K\right)$ degrees of freedom.

Example 111 *Estimating a statistical factor model by factor analysis using `S-PLUS`*

Factor analysis in `S-PLUS` is performed using the function `factanal`, which performs estimation of $\mathbf{B}$ and $\mathbf{D}$ using either the *principal factor method* or the *maximum likelihood method*, and it takes as input either raw data or an estimated covariance or correlation matrix. A robust version of factor analysis can be computed if the inputted covariance matrix is a robust covariance matrix (MCD, MVE or M-estimate). If the maximum likelihood method is used, then the LR test for the adequacy of the K factor model is computed.

A factor model with $k = 2$ factors for the fifteen returns from `berndt.dat` computed using maximum likelihood method is

```
> factor.fit = factanal(returns,factors=2,method="mle")
> class(factor.fit)
[1] "factanal"
> factor.fit
Sums of squares of loadings:
 Factor1 Factor2
   3.319   2.471

The number of variables is 15 and the number of observations
is 120

Test of the hypothesis that 2 factors are sufficient
versus the alternative that more are required:
The chi square statistic is 118.25 on 76 degrees of freedom.
The p-value is 0.00138

Component names:

 "loadings" "uniquenesses" "correlation" "criteria"


 "factors" "dof" "method" "center" "scale" "n.obs" "scores"


 "call"

Call:
factanal(x = returns, factors = 2, method = "mle")
```

The likelihood ratio test for determining the number of factors indicates that two factors is not enough to adequately explain the sample return covariance. A factor model with $k = 3$ factor appears to be adequate

```
> factor.fit = factanal(returns,factors=3,method="mle")
> factor.fit
Sums of squares of loadings:
 Factor1 Factor2 Factor3
   3.137   1.765   1.719
```

```
The number of variables is 15 and the number of observations
 is 120

Test of the hypothesis that 3 factors are sufficient
versus the alternative that more are required:
The chi square statistic is 71.6 on 63 degrees of freedom.
The p-value is 0.214
...
```

A summary of the three factor model is

```
> summary(factor.fit)
Importance of factors:
               Factor1 Factor2 Factor3
   SS loadings  3.1370  1.7651  1.7185
Proportion Var  0.2091  0.1177  0.1146
Cumulative Var  0.2091  0.3268  0.4414

The degrees of freedom for the model is 63.

Uniquenesses:
 CITCRP  CONED CONTIL DATGEN    DEC  DELTA GENMIL GERBER
 0.3125 0.8506 0.7052 0.4863 0.3794 0.6257  0.592 0.5175

    IBM  MOBIL  PANAM   PSNH  TANDY TEXACO  WEYER
 0.6463 0.2161 0.7643 0.9628 0.5442 0.3584 0.4182

Loadings:
       Factor1 Factor2 Factor3
CITCRP  0.518   0.217   0.610
 CONED  0.116           0.356
CONTIL  0.173   0.195   0.476
DATGEN  0.668   0.206   0.160
   DEC  0.749   0.236
 DELTA  0.563           0.239
GENMIL  0.306           0.556
GERBER          0.346   0.600
   IBM  0.515   0.224   0.197
 MOBIL  0.257   0.847
 PANAM  0.427           0.219
  PSNH  0.115           0.133
 TANDY  0.614           0.264
TEXACO  0.140   0.787
 WEYER  0.694   0.200   0.247
```

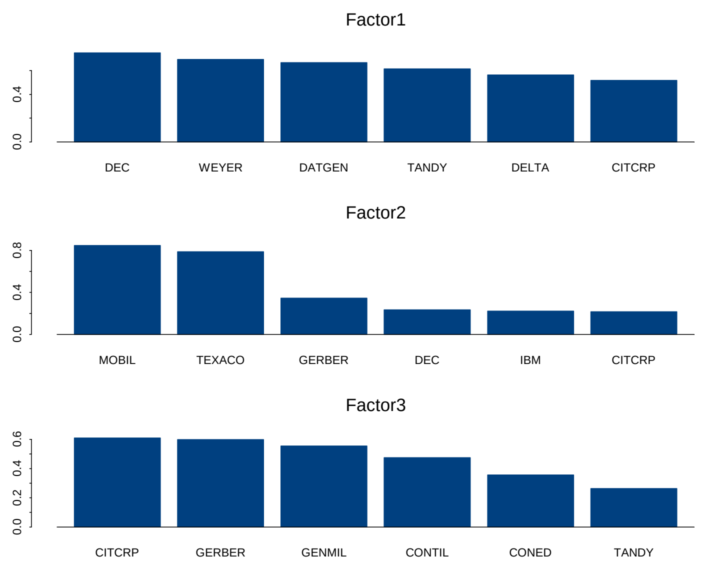


FIGURE 15.4. Estimated loadings from three factor model fit to Berndt data by factor analysis.

The three factors explain about forty four percent of the total variance of returns. The reported uniqueness for each asset is standardized such that the sum of the uniqueness and the communality is unity. Therefore, assets with uniqueness values close to zero are well explained by the factor model.

The factor loadings may be extracted using the generic `loadings` function. The extracted loadings have class “`loadings`” and may be visualized with `plot`

```
> plot(loadings(factor.fit))
```

Figure 15.4 gives the resulting plot. The first factor appears to be market-wide factor, and the second factor is concentrated on oil stocks. Since the factors are only defined up to an orthogonal rotation, the factor may be rotated to aid interpretation. The generic function `rotate` performs such rotation. For example, to rotate the factor using the `quartimax` rotation and view the rotated loadings use

```
> factor.fit2 = rotate(factor.fit,rotation="quartimax")
> loadings(factor.fit2)
       Factor1 Factor2 Factor3
CITCRP  0.722   0.108   0.393
 CONED  0.233  -0.153   0.268
CONTIL  0.351   0.113   0.398
```

```
DATGEN  0.693   0.168
   DEC  0.734   0.212  -0.193
 DELTA  0.610
GENMIL  0.485  -0.164   0.382
GERBER  0.299   0.243   0.578
   IBM  0.566   0.181
 MOBIL  0.307   0.829
 PANAM  0.472  -0.112
  PSNH  0.150  -0.101
 TANDY  0.673
TEXACO  0.205   0.767   0.103
 WEYER  0.748   0.148
```

See the online help for `rotate` for a description of the supported rotation methods.

The factor realizations (15.19) may be computed using the generic `predict` function:

```
> factor.ret = predict(factor.fit,type="weighted.ls")
```

The estimated factor model correlation matrix may be extracted using

```
> fitted(factor.fit)
numeric matrix: 15 rows, 15 columns.
       CITCRP    CONED  CONTIL  DATGEN     DEC   DELTA
CITCRP 1.0000  0.25701 0.42245 0.48833 0.47656 0.43723
 CONED 0.2570  1.00000 0.17131 0.11533 0.08655 0.15095
CONTIL 0.4224  0.17131 1.00000 0.23205 0.20474 0.21099
DATGEN 0.4883  0.11533 0.23205 1.00000 0.55841 0.41386
   DEC 0.4766  0.08655 0.20474 0.55841 1.00000 0.43597
 DELTA 0.4372  0.15095 0.21099 0.41386 0.43597 1.00000
GENMIL 0.4825  0.24043 0.30402 0.27919 0.24681 0.30570
GERBER 0.4693  0.18728 0.36267 0.20404 0.15936 0.17371
   IBM 0.4354  0.10893 0.22663 0.42116 0.45030 0.33649
 MOBIL 0.3278 -0.04399 0.21832 0.34861 0.39336 0.14722
 PANAM 0.3396  0.13446 0.16438 0.30585 0.31682 0.29320
  PSNH 0.1237  0.06825 0.06786 0.08206 0.07588 0.09689
 TANDY 0.5000  0.15650 0.25080 0.47188 0.49859 0.40868
TEXACO 0.2712 -0.04208 0.19960 0.26285 0.29353 0.08824
 WEYER 0.5531  0.14958 0.27646 0.54368 0.58181 0.44913
...
```

To obtain the estimated factor model covariance matrix, the estimated loadings and uniqueness values must be re-scaled. One way to do this is

```
> S.hat = diag(factor.fit$scale)
> D.hat = S.hat%*%diag(factor.fit$uniqueness)%*%S.hat
> D.hat.inv = diag(1/diag(D.hat))
> B.hat = S.hat%*%loadings(factor.fit)
> cov.factor = B.hat%*%t(B.hat)+D.hat
> dimnames(cov.fa) = list(colIds(returns),colIds(returns))
```

The factor analysis global minimum variance portfolio is then

```
> w.gmin.fa = solve(cov.fa)%*%rep(1,nrow(cov.fa))
> w.gmin.fa = w.gmin.fa/sum(w.gmin.fa)
> t(w.gmin.fa)
numeric matrix: 1 rows, 15 columns.
      CITCRP  CONED   CONTIL   DATGEN       DEC   DELTA
[1,] -0.0791 0.3985 -0.02537 -0.04279 -0.002584 0.04107

     GENMIL  GERBER    IBM  MOBIL   PANAM    PSNH    TANDY
[1,] 0.1889 0.01321 0.2171 0.1027 0.01757 0.07533 -0.03255

     TEXACO    WEYER
[1,] 0.1188 0.009147
```

15.5.2 Principal Components

Principal component analysis (PCA) is a dimension reduction technique used to explain the majority of the information in the sample covariance matrix of returns. With N assets there are N principal components, and these principal components are just linear combinations of the returns. The principal components are constructed and ordered so that the first principal component explains the largest portion of the sample covariance matrix of returns, the second principal component explains the next largest portion, and so on. The principal components are constructed to be orthogonal to each other and to be normalized to have unit length. In terms of a multifactor model, the K most important principal components are the factor realizations. The factor loadings on these observed factors can then be estimated using regression techniques.

Let $\widehat{\mathbf{\Omega}}$ denote the sample covariance matrix of returns. The first sample principal component is $\mathbf{x}_1^{*\prime}\mathbf{R}_t$ where the $(N \times 1)$ vector $\mathbf{x}_1^*$ solves

$$\max_{x_1} \ \mathbf{x}_1'\widehat{\mathbf{\Omega}}\mathbf{x}_1 \text{ s.t. } \mathbf{x}_1'\mathbf{x}_1 = 1.$$

The solution $\mathbf{x}_1^*$ is the eigenvector associated with the largest eigenvalue of $\widehat{\mathbf{\Omega}}$. The second principal component is $\mathbf{x}_2^{*\prime}\mathbf{R}_t$ where the $(N \times 1)$ vector $\mathbf{x}_2^*$ solves

$$\max_{x_2} \ \mathbf{x}_2'\widehat{\mathbf{\Omega}}\mathbf{x}_2 \text{ s.t. } \mathbf{x}_2'\mathbf{x}_2 = 1 \text{ and } \mathbf{x}_1^{*\prime}\mathbf{x}_2 = 0$$

The solution $\mathbf{x}_2^*$ is the eigenvector associated with the second largest eigenvalue of $\widehat{\mathbf{\Omega}}$. This process is repeated until K principal components are computed.

The estimated factor realizations are simply the first K principal components

$$\widehat{f}_{kt} = \mathbf{x}_k^{*\prime}\mathbf{R}_t,\ k = 1, \ldots, K. \tag{15.20}$$

The factor loadings for each asset, $\boldsymbol{\beta}_i$, and the residual variances, $\text{var}(\varepsilon_{it}) = \sigma_i^2$ can be estimated via OLS[7] from the time series regression

$$R_{it} = \alpha_i + \boldsymbol{\beta}_i'\widehat{\mathbf{f}}_t + \varepsilon_{it},\ t = 1, \ldots, T \tag{15.21}$$

giving $\widehat{\boldsymbol{\beta}}_i$ and $\widehat{\sigma}_i^2$ for $i = 1, \ldots, N$. The factor model covariance matrix of returns is then

$$\widehat{\mathbf{\Omega}} = \widehat{\mathbf{B}}\widehat{\mathbf{\Omega}}^F\widehat{\mathbf{B}}' + \widehat{\mathbf{D}} \tag{15.22}$$

where

$$\widehat{\mathbf{B}} = \begin{pmatrix} \widehat{\boldsymbol{\beta}}_1' \\ \vdots \\ \widehat{\boldsymbol{\beta}}_N' \end{pmatrix},\ \widehat{\mathbf{D}} = \begin{pmatrix} \widehat{\sigma}_1^2 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & \cdots & \widehat{\sigma}_N^2 \end{pmatrix},$$

and

$$\begin{aligned} \widehat{\mathbf{\Omega}}^F &= \frac{1}{T-1}\sum_{t=1}^{T}(\widehat{\mathbf{f}}_t - \overline{\mathbf{f}})(\widehat{\mathbf{f}}_t - \overline{\mathbf{f}})', \\ \overline{\mathbf{f}} &= \frac{1}{T}\sum_{t=1}^{T}\widehat{\mathbf{f}}_t. \end{aligned}$$

Usually $\widehat{\mathbf{\Omega}}^F = \mathbf{I}_K$ because the principal components are orthonormal.

Factor Mimicking Portfolios

Since the principal components (factors) $\mathbf{x}_i^*$ are just linear combinations of the returns, it is possible to construct portfolios that are perfectly correlated with the principal components by re-normalizing the weights in the $\mathbf{x}_i^*$ vectors so that they sum to unity. Hence, the weights in the factor mimicking portfolios have the form

$$\mathbf{w}_i = \left(\frac{1}{\mathbf{1}'\mathbf{x}_i^*}\right) \cdot \mathbf{x}_i^*,\ i = 1, \ldots, K \tag{15.23}$$

where $\mathbf{1}$ is a $(N \times 1)$ vector of ones.

[7]OLS estimation is efficient even though assets are contemporaneously correlated because the time series regression for each asset has the same regressors.

Variance Decomposition

It can be shown that

$$\sum_{i=1}^{k} \text{var}(R_{it}) = \sum_{i=1}^{k} \text{var}(f_{it}) = \sum_{i=1}^{k} \lambda_i$$

where λ_i are the ordered eigenvalues of $\text{var}(\mathbf{R}_i) = \mathbf{\Omega}$. Therefore, the ratio

$$\frac{\lambda_i}{\sum_{i=1}^{N} \lambda_i}$$

gives the proportion of the total variance $\sum_{i=1}^{N} \text{var}(R_{it})$ attributed to the ith principal component factor return, and the ratio

$$\frac{\sum_{i=1}^{K} \lambda_i}{\sum_{i=1}^{N} \lambda_i}$$

gives the cumulative variance explained. Examination of these ratios help in determining the number of factors to use to explain the covariance structure of returns.

Example 112 *Estimating a statistical factor model by principal components using* `S-PLUS`

Principal component analysis in `S-PLUS` is performed using the function `princomp`. The `S+FinMetrics` function `mfactor` simplifies the process of estimating a statistical factor model for asset returns using principal components. To illustrate, consider estimating a statistical factor model for the assets in the `S+FinMetrics` “`timeSeries`” `berndt.dat` excluding market portfolio and the thirty-day T-bill

```
> returns.ts = berndt.dat[,c(-10,-17)]
```

To estimate a statistical factor model with the default of one factor use

```
> pc.mfactor = mfactor(returns.ts)
> class(pc.mfactor)
[1] "mfactor"
```

The result of the function `mfactor` is an object of class “`mfactor`”, for which there are `print` and `plot` methods and extractor functions `factors`, `loadings`, `residuals` and `vcov`. The components of an “`mfactor`” object are

```
> names(pc.mfactor)
[1] "factors"      "loadings"     "k"
[4] "alpha"        "Omega"        "r2"
[7] "eigen"        "call"         "sum.loadings"
```

where `factors` contains the estimated factor returns (15.20), `loadings` contains the asset specific factor loadings $\hat{\beta}_i$ estimated from (15.21), `alpha` contains the estimated intercepts α_i from (15.21), `r2` contains the regression R^2 values from (15.21), `k` is the number of factors and `eigen` contains the eigenvalues from the sample covariance matrix.

The `print` method gives a brief summary of the PCA fit

```
> pc.mfactor

Call:
mfactor(x = returns.ts)

Factor Model:
 Factors Variables Periods
       1        15     120

Factor Loadings:
      Min. 1st Qu. Median  Mean 3rd Qu.  Max.
F.1 0.0444   0.139   0.25 0.231   0.308 0.417

Regression R-squared:
  Min. 1st Qu. Median  Mean 3rd Qu.  Max.
 0.032 0.223   0.329  0.344 0.516   0.604
```

Notice that all of the estimated loadings on the first factor are positive, and the median R^2 is around thirty percent. These results are very similar to those found for the single index model. The factor loadings and factor regression R^2 values may be extracted using the functions `loadings` and `mfactor.r2`

```
> loadings(pc.mfactor)
numeric matrix: 1 rows, 15 columns.
    CITCRP   CONED CONTIL DATGEN    DEC  DELTA GENMIL
F.1 0.2727 0.04441 0.3769 0.4172 0.3049 0.2502 0.1326

    GERBER    IBM  MOBIL  PANAM    PSNH  TANDY TEXACO
F.1 0.1672 0.1464 0.1552 0.3107 0.08407 0.4119 0.1323

     WEYER
F.1 0.2649
> mfactor.r2(pc.mfactor)
 CITCRP  CONED CONTIL DATGEN   DEC  DELTA GENMIL GERBER
 0.6041 0.1563 0.3285 0.5633 0.516 0.3665 0.2662 0.2181

    IBM  MOBIL  PANAM    PSNH  TANDY TEXACO  WEYER
 0.3408 0.2277 0.2922 0.03241 0.5643 0.1635 0.5153
```

The factor returns (15.20) and the residuals from the regression (15.21) may be extracted using the functions `factors` and `residuals`, respectively.

The function `vcov` extracts the PCA covariance matrix (15.22). The corresponding correlation matrix may computed using

```
> cov.pca = vcov(pc.mfactor)
> sd = sqrt(diag(cov.pca))
> cor.pca = cov.pca/outer(sd,sd)
> print(cor.pca,digits=1,width=2)
       CITCRP CONED CONTIL DATGEN DEC DELTA GENMIL GERBER
CITCRP    1.0  0.16    0.4    0.6 0.5   0.5   0.36   0.34
 CONED    0.2  1.00    0.1    0.2 0.1   0.1   0.09   0.09
CONTIL    0.4  0.12    1.0    0.4 0.4   0.3   0.27   0.25
DATGEN    0.6  0.15    0.4    1.0 0.5   0.4   0.35   0.33
   DEC    0.5  0.14    0.4    0.5 1.0   0.4   0.33   0.31
 DELTA    0.5  0.12    0.3    0.4 0.4   1.0   0.28   0.26
GENMIL    0.4  0.09    0.3    0.3 0.3   0.3   1.00   0.20
GERBER    0.3  0.09    0.2    0.3 0.3   0.3   0.20   1.00
   IBM    0.4  0.11    0.3    0.4 0.4   0.3   0.26   0.25
 MOBIL    0.3  0.09    0.3    0.3 0.3   0.3   0.21   0.19
 PANAM    0.4  0.11    0.3    0.4 0.4   0.3   0.25   0.23
  PSNH    0.1  0.04    0.1    0.1 0.1   0.1   0.08   0.08
 TANDY    0.6  0.15    0.4    0.6 0.5   0.4   0.34   0.32
TEXACO    0.3  0.08    0.2    0.3 0.3   0.2   0.18   0.16
 WEYER    0.5  0.14    0.4    0.5 0.5   0.4   0.33   0.31
```

The PCA global minimum variance portfolio is

```
> w.gmin.pca = solve(cov.pca)%*%rep(1,nrow(cov.pca))
> w.gmin.pca = w.gmin.pca/sum(w.gmin.pca)
> t(w.gmin.pca)
numeric matrix: 1 rows, 15 columns.
      CITCRP  CONED CONTIL   DATGEN      DEC  DELTA GENMIL
[1,] 0.02236 0.3675 -0.021 -0.06549 -0.01173 0.0239 0.1722

      GERBER    IBM   MOBIL     PANAM    PSNH    TANDY
[1,] 0.07121 0.2202 0.09399 -0.006415 0.06427 -0.06079

     TEXACO   WEYER
[1,]  0.105 0.02472
```

The `plot` method allows a graphical investigation of the PCA fit

```
> plot(pc.mfactor)

Make a plot selection (or 0 to exit):
```

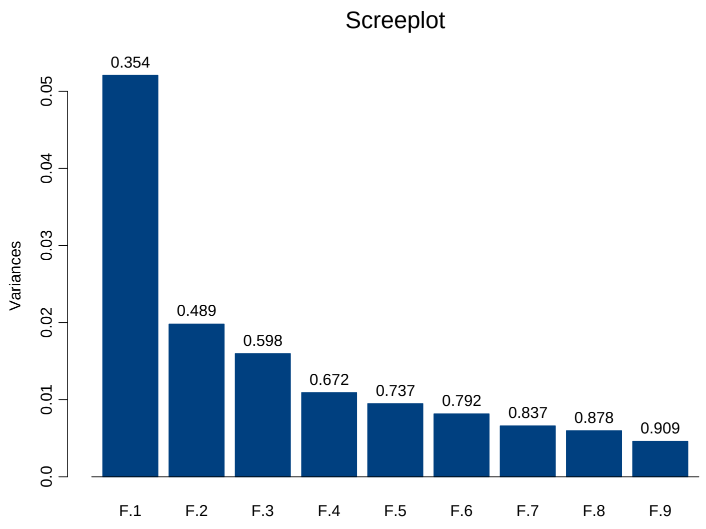


FIGURE 15.5. Screeplot of eigenvalues from PCA of Berndt returns.

```
1: plot: All
2: plot: Screeplot of Eigenvalues
3: plot: Factor Returns
Selection:
```

The Screeplot of Eigenvalues is illustrated in Figure 15.5. The first principal component explains about thirty five percent of the total variance, and the first two components explain about half of the total variance. It appears that two or three factors may be sufficient to explain most of the variability of the assets. The screeplot may also be computed directly using the `S+FinMetrics` function `screeplot.mfactor`.

The PCA factor model is re-estimated using two factors with

```
> pc2.mfactor = mfactor(returns.ts,k=2)
> pc2.mfactor

Call:
mfactor(x = returns.ts, k = 2)

Factor Model:
 Factors Variables Periods
       2        15     120

Factor Loadings:
```

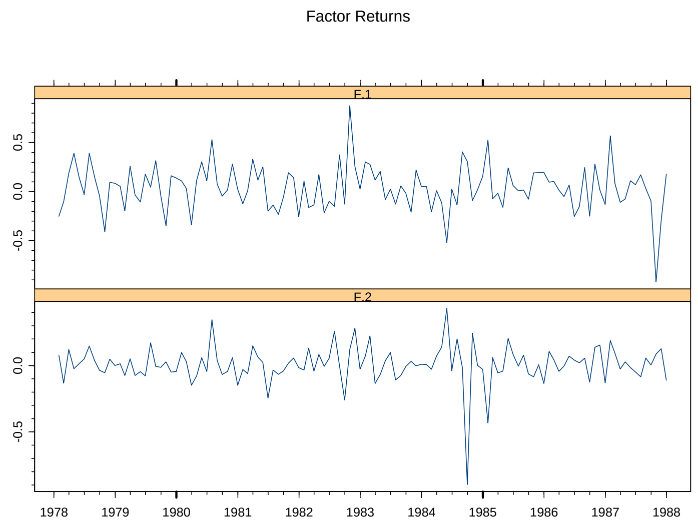


FIGURE 15.6. Estimated factors from PCA of Berndt data.

```
        Min.  1st Qu. Median      Mean  3rd Qu.  Max.
F.1   0.0444   0.1395 0.2502  0.23143    0.308 0.417
F.2 -0.8236  -0.0671 0.0124 -0.00245    0.142 0.365

Regression R-squared:
  Min. 1st Qu. Median  Mean 3rd Qu.  Max.
 0.033 0.253   0.435  0.419 0.577   0.925
```

The first factor is the same as before and has all positive loadings. The second factor has both positive and negative loadings. The median regression R^2 has increased to about forty four percent. The factor returns are illustrated in Figure 15.6, created by selecting option 3 from the plot menu.

The factor return plot may also be computed directly by first extracting the factors and then using the `S+FinMetrics` function `fplot`:

```
> fplot(factors(pc2.mfactor))
```

The factor loadings are shown in Figure 15.7, created by

```
> pc2.betas = loadings(pc2.mfactor)
> par(mfrow=c(1,2))
> barplot(pc2.betas[1,],names=colIds(pc2.betas),horiz=T,
+ main="Beta values for first PCA factor")
> barplot(pc2.betas[2,],names=colIds(pc2.betas),horiz=T,
+ main="Beta values for second PCA factor")
```

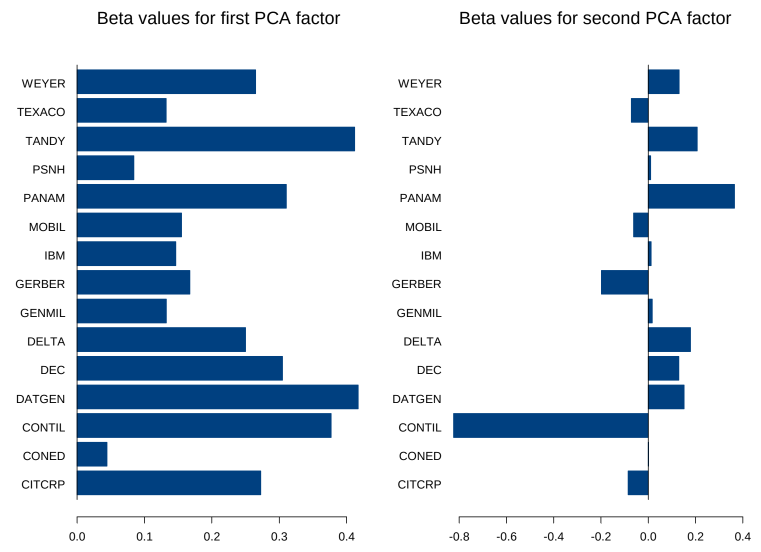


FIGURE 15.7. Estimated loadings on PCA factors for Berndt data.

The factor mimicking portfolios (15.23) may be computed using the S+FinMetrics function `mimic`

```
> pc2.mimic = mimic(pc2.mfactor)
> class(pc2.mimic)
[1] "mimic"
> pc2.mimic
           F.1      F.2
CITCRP 0.07856   2.3217
 CONED 0.01279  -0.0324
...
 WEYER 0.07630  -3.5637
attr(, "class"):
[1] "mimic"
```

These weights in these portfolios may be summarized using

```
> pc2.mimic.sum = summary(pc2.mimic,n.top=3)
> pc2.mimic.sum

Factor  1
  Top.Long.Name Top.Long.Weight Top.Short.Name Top.Short.Weight
1        DATGEN             12%          CONED             1.3%
2         TANDY             12%           PSNH             2.4%
```

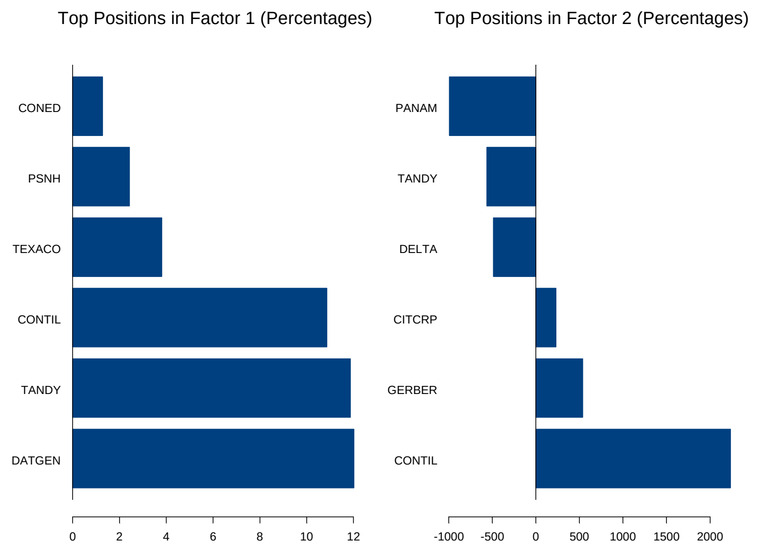


FIGURE 15.8. Weights in factor mimicking portfolios from PCA fit to Berndt data.

```
3         CONTIL              11%          TEXACO              3.8%

Factor  2
  Top.Long.Name Top.Long.Weight Top.Short.Name Top.Short.Weight
1         CONTIL           2200%           PANAM            -990%
2         GERBER            540%           TANDY            -560%
3         CITCRP            230%           DELTA            -490%
```

The optional argument `n.top=3` specifies that the three assets with the largest and smallest weights in each factor mimicking portfolio are displayed. For the first factor, the assets `DATGEN`, `TANDY` and `CONTIL` have the highest weights and the assets `CONED`, `PSNH` and `TEXACO` have the lowest weights. Examination of the weights helps to interpret the factor mimicking portfolio. For the first portfolio, the weights are all positive (long positions in all assets) and are roughly equal suggesting the interpretation of a market-wide factor. The second factor has both positive and negative weights (long and short positions in the assets), and it is not clear how to interpret the weights. The weights may also be examined graphically using

```
> par(mfrow=c(1,2))
> plot(pc2.mimic.sum)
```

which produces the plot in Figure 15.8.

15.5.3 Asymptotic Principal Components

Asymptotic principal component analysis (APCA), proposed and developed in Conner and Korajczyk (1986) and based on the analysis in Chamberlain and Rothschild (1983), is similar to traditional PCA except that it relies on asymptotic results as the number of cross-sections N (assets) grows large. APCA is based on eigenvector analysis of the $T \times T$ matrix $\widehat{\mathbf{\Omega}}_T$. Conner and Korajczyk proved that as N grows large, eigenvector analysis of $\widehat{\mathbf{\Omega}}_T$ is asymptotically equivalent to traditional factor analysis. That is, the APCA estimates of the factors $\mathbf{f}_t$ are the first K eigenvectors of $\widehat{\mathbf{\Omega}}_T$. Specifically, let $\widehat{\mathbf{F}}$ denote the orthornormal $K \times T$ matrix consisting of the first K eigenvectors of $\widehat{\mathbf{\Omega}}_T$. Then $\widehat{\mathbf{f}}_t$ is the t^{th} column of $\widehat{\mathbf{F}}$.

The main advantages of the APCA approach are:

- It works in situations where the number of assets, N, is much greater than the number of time periods, T. Eigenvectors of the smaller $T \times T$ matrix $\widehat{\mathbf{\Omega}}_T$ only need to be computed, whereas with traditional principal component analysis eigenvalues of the larger $N \times N$ matrix $\widehat{\mathbf{\Omega}}_N$ need to be computed.

- The method allows for an approximate factor structure of returns. In an approximate factor structure, the asset specific error terms ε_{it} are allowed to be contemporaneously correlated, but this correlation is not allowed to be too large across the cross section of returns. Allowing an approximate factor structure guards against picking up local factors, e.g. industry factors, as global common factors.

Refinement

Connor and Korajczyk (1988) offered a refinement of the APCA procedure that may improve the efficiency of the procedure.

1. Estimate the factors $\mathbf{f}_t$ $(t = 1, \ldots, T)$ by computing the first K eigenvalues of $\widehat{\mathbf{\Omega}}_T$.

2. For each asset, estimate the time series regression (factor model) by OLS

$$R_{it} = \alpha_i + \boldsymbol{\beta}_i'\widehat{\mathbf{f}}_t + \varepsilon_{it},\ t = 1, \ldots, T$$

   and compute the residual variances $\widehat{\sigma}_i^2$. Use these variance estimates to compute the residual covariance matrix

$$\widehat{\mathbf{D}} = \begin{pmatrix} \widehat{\sigma}_1^2 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & \cdots & \widehat{\sigma}_N^2 \end{pmatrix}$$

3. Form the $N \times T$ matrix of re-scaled returns

$$\mathbf{R}^{*}=\widehat{\mathbf{D}}^{-1/2}\mathbf{R}$$

and recompute the $T \times T$ covariance matrix

$$\widehat{\mathbf{\Omega}}_{T}^{*} = \frac{1}{N}\mathbf{R}^{*\prime}\mathbf{R}^{*}$$

4. Re-estimate the factors $\mathbf{f}_t$ by computing the first K eigenvalues of $\widehat{\mathbf{\Omega}}_{T}^{*}$.

Example 113 *Estimation of a statistical factor model by asymptotic principal component analysis using* `S-PLUS`

The `S+FinMetrics` function `mfactor` estimates a statistical factor model by asymptotic principal components whenever the number of assets, N, is greater than the number of time periods, T. To illustrate, consider fitting a statistical factor model using the `S+FinMetrics` "`timeSeries`" `folio.dat`, which contains weekly data on 1618 stocks over the period January 8, 1997 to June 28, 2000. For this data, $N = 1618$ and $T = 182$. To compute the APCA fit with $k = 15$ factors use

```
> folio.mf = mfactor(folio.dat,k=15)
> folio.mf

Call:
mfactor(x = folio.dat, k = 15)

Factor Model:
 Factors Variables Periods
      15      1618     182

Factor Loadings:
       Min. 1st Qu.    Median     Mean 3rd Qu.  Max.
 F.1 -0.977 -0.4261 -0.314658 -0.33377 -0.2168 0.160
 F.2 -0.420 -0.1041 -0.014446  0.06519  0.1628 1.110
 F.3 -0.463 -0.0784 -0.011839 -0.00311  0.0392 0.998
 F.4 -0.556 -0.0588  0.004821  0.00866  0.0771 0.495
 F.5 -1.621 -0.0622  0.015520  0.01373  0.0858 0.467
 F.6 -0.835 -0.0635 -0.001544  0.00307  0.0665 0.468
 F.7 -0.758 -0.0633 -0.006376 -0.01183  0.0509 2.090
 F.8 -0.831 -0.0685 -0.012736 -0.01413  0.0479 0.517
 F.9 -0.464 -0.0466  0.006447  0.01200  0.0640 1.095
F.10 -0.640 -0.0659 -0.008760 -0.01050  0.0482 0.687
F.11 -1.515 -0.0540 -0.001114 -0.00457  0.0539 0.371
F.12 -1.682 -0.0637 -0.005902 -0.01068  0.0451 0.515
```

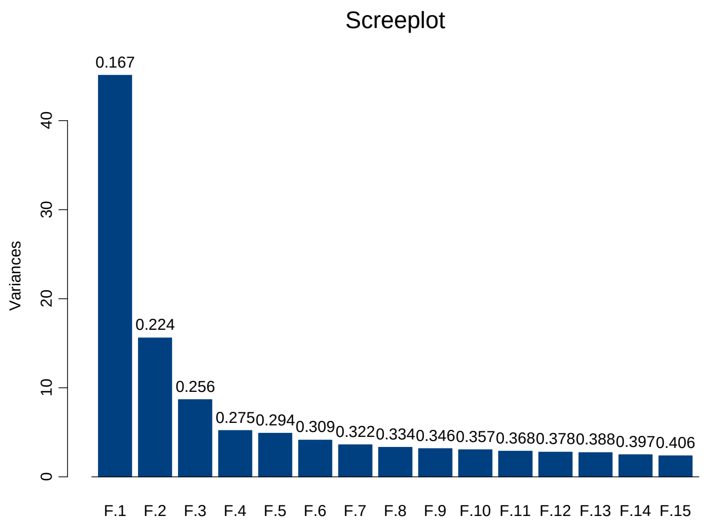


FIGURE 15.9. Screeplot of eigenvalues from APCA fit to 1618 assets.

```
F.13 -0.462 -0.0480  0.001901  0.00164  0.0516 0.685
F.14 -0.912 -0.0523 -0.001072 -0.00443  0.0472 0.436
F.15 -0.681 -0.0505 -0.000977 -0.00366  0.0473 0.548

Regression R-squared:
  Min. 1st Qu. Median  Mean 3rd Qu.  Max.
 0.066 0.265   0.354  0.372 0.459   0.944
```

By default, the APCA fit uses the Connor-Korajczyk refinement. To compute the APCA fit without the refinement, set the optional argument `refine=F` in the call to `mfactor`. The factor loadings appear to be reasonably scaled and skewed toward negative values. The loadings for the first factor appear to be almost all negative. Multiplying the first factor by negative one would make it more interpretable. The median regression R^2 is about thirty five percent, which is a bit higher than what one would expect from the single index model.

Figures 15.9 and 15.10 show the screeplot of eigenvalues and factor returns for the APCA fit, computed using

```
> screeplot.mfactor(folio.mf)
> fplot(factors(folio.mf)
```

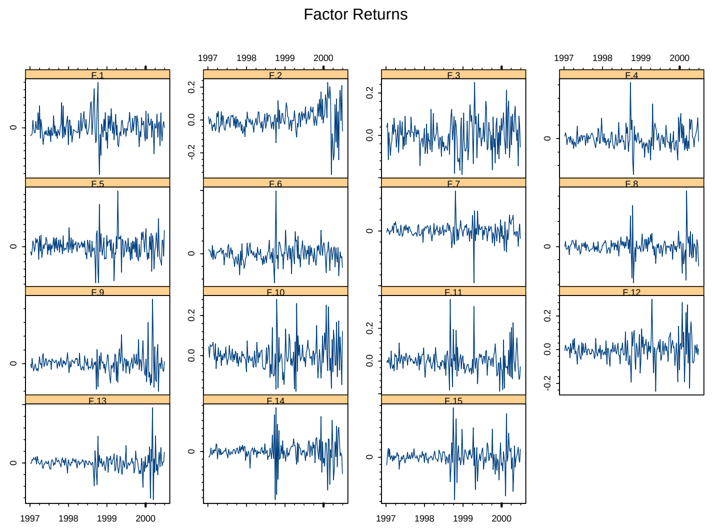


FIGURE 15.10. Estimated factors returns from APCA fit to 1618 assets.

The first two factors clearly have the largest explanatory power, and the fifteen factors together explain roughly forty one percent of the total variance.

The factor mimicking portfolios are computed using

```
> folio.m = mimic(folio.mf)
```

which is an object of class "`mimic`" of dimension 1618 × 15. It is difficult to concisely summarize the factor mimicking portfolios when the number of assets is large. This is why the `summary` method for "`mimic`" objects has an option for displaying only the largest and smallest `n.top` weights for each factor mimicking portfolio. To view the top five largest and smallest weights for the fifteen factors use

```
> folio.ms = summary(folio.m,n.top=5)
> folio.ms
```

```
Factor  1
  Top.Long.Name Top.Long.Weight Top.Short.Name Top.Short.Weight
1           OWC           0.23%            BBY           -0.15%
2           FNV           0.22%           SCON           -0.14%
3            MT           0.22%           PUMA           -0.12%
4           BAC           0.21%           THDO           -0.11%
5          CACC           0.21%           AVTC           -0.11%
```

```
...
Factor  15
  Top.Long.Name Top.Long.Weight Top.Short.Name Top.Short.Weight
1          ALXN             79%           SCTC             -85%
2          AVID             59%           MCLL             -82%
3          LTXX             53%           WLNK             -65%
4           IDX             52%           LRCX             -63%
5          SEAC             51%           TSCC             -63%
```

The summary information may be visualized using the generic `plot` function

```
> plot(folio.ms)
```

which generates a fifteen page graph sheet, with one page for each factor.

The correlations of the assets giving the largest and smallest weights for a given factor may be visualized using an image plot. To do this, first compute the correlation matrix for all of the assets

```
> folio.cov = vcov(folio.mf)
> sd = sqrt(diag(folio.cov))
> folio.cor = folio.cov/outer(sd,sd)
```

Extract the names of the assets in the summary for the first factor

```
> top.names = c(as.character(folio.m[[1]][,1]),
+ rev(as.character(folio.ms[[1]][,3])))
```

and call the `S+FinMetrics` function `image.plot`

```
> image.plot(folio.cor[top.names, top.names],
+ sub="Risk factor 1", main="Correlations of top positions")
```

The resulting plot is shown in Figure 15.11.

15.5.4 Determining the Number of Factors

The statistical methods described above are based on knowing the number of common factors. In practice, the number of factors is unknown and must be determined from the data. If traditional factor analysis is used, then there is a likelihood ratio test for the number of factors. However, this test will not work if $N > T$. Connor and Korajczyk (1993) described a procedure for determining the number of factors in an approximate factor model that is valid for $N > T$ and Connor (1995) applied this method to a variety of factor models. Recently Bai and Ng (2002) have proposed an alternative method.

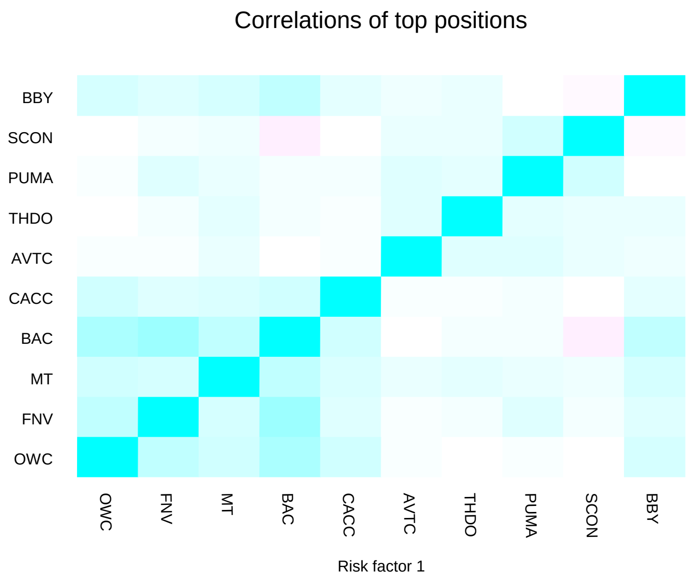


FIGURE 15.11. Image plot correlations between assets with top five largest and smallest weights in first factor mimicking portfolio.

Connor and Korajczyk Method

The intuition behind this method is that if K is the correct number of common factors then there should be no significant decrease in the cross-sectional variance of the asset specific error, ε_{it}, in moving from K to $K+1$ factors. The procedure is implemented as follows:

1. Given observed returns on asset i and a time series of $K+1$ factors, estimate the time series regression models

$$\begin{aligned} R_{it} &= \alpha_i + \boldsymbol{\beta}_i'\widehat{\mathbf{f}}_t + \varepsilon_{it} \\ R_{it} &= \alpha_i + \boldsymbol{\beta}_i'\widehat{\mathbf{f}}_t + \beta_{K+1,i} f_{K+1,t} + \varepsilon_{it}^* \end{aligned}$$

   giving residuals $\widehat{\varepsilon}_{it}$ and $\widehat{\varepsilon}_{it}^*$.

2. Calculate degrees-of-freedom adjusted squared residuals

$$\begin{aligned} \widehat{\sigma}_{it} &= \frac{\widehat{\varepsilon}_{it}^2}{1-(K+1)/T-K/N} \\ \widehat{\sigma}_{it}^* &= \frac{\widehat{\varepsilon}_{it}^{*2}}{1-(K+3)/T-(K+1)/N} \end{aligned}$$

3. Calculate the cross-sectional difference in squared errors based on odd and even time periods

$$\begin{aligned}\widehat{\Delta}_s &= \widehat{\mu}_{2s-1} - \widehat{\mu}^*_{2s},\ s = 1, \ldots, T/2 \\ \widehat{\mu}_t &= \frac{1}{N}\sum_{i=1}^{N} \widehat{\sigma}_{it} \\ \widehat{\mu}^*_t &= \frac{1}{N}\sum_{i=1}^{N} \widehat{\sigma}^*_{it}\end{aligned}$$

and compute the $T/2 \times 1$ vector of differences

$$\widehat{\boldsymbol{\Delta}} = \left(\widehat{\Delta}_1, \widehat{\Delta}_2, \ldots, \widehat{\Delta}_{T/2}\right)'$$

4. Compute the time series sample mean and variance of the differences

$$\begin{aligned}\overline{\Delta} &= \frac{T}{2}\sum_{s=1}^{T/2} \widehat{\Delta}_s \\ \widehat{\sigma}^2_\Delta &= \frac{2}{T-2}\sum_{s=1}^{T/2}\left(\widehat{\Delta}_s - \overline{\Delta}\right)^2\end{aligned}$$

5. Compute the t-statistic

$$t = \frac{\overline{\Delta}}{\widehat{\sigma}_\Delta}$$

and use it to test for a positive mean value.

Bai and Ng Method

Bai and Ng (2002) proposed some panel C_p (Mallows-type) information criteria for choosing the number of factors. Their criteria are based on the observation that eigenvector analysis on $\widehat{\boldsymbol{\Omega}}$ or $\widehat{\boldsymbol{\Omega}}_N$ solves the least squares problem

$$\min_{\boldsymbol{\beta}_i, \mathbf{f}_t} (NT)^{-1}\sum_{i=1}^{N}\sum_{t=1}^{T}(R_{it} - \alpha_i - \boldsymbol{\beta}_i'\mathbf{f}_t)^2$$

Bai and Ng's model selection or information criteria are of the form

$$\mathrm{IC}(K) = \widehat{\sigma}^2(K) + K \cdot g(N, T)$$

where

$$\widehat{\sigma}^2(K) = \frac{1}{N}\sum_{i=1}^{N} \widehat{\sigma}^2_i$$

is the cross-sectional average of the estimated residual variances for each asset based on a model with K factors and $g(N, T)$ is a penalty function depending only on N and T. The preferred model is the one which minimizes the information criteria $IC(K)$ over all values of $K < K_{\max}$. Bai and Ng consider several penalty functions and the preferred criteria are

$$\begin{aligned}
\mathrm{PC}_{p1}(K) &= \widehat{\sigma}^2(K) + K \cdot \widehat{\sigma}^2(K_{\max}) \left(\frac{N+T}{NT} \right) \cdot \ln \left(\frac{NT}{N+T} \right), \\
\mathrm{PC}_{p2}(K) &= \widehat{\sigma}^2(K) + K \cdot \widehat{\sigma}^2(K_{\max}) \left(\frac{N+T}{NT} \right) \cdot \ln \left(C_{NT}^2 \right),
\end{aligned}$$

where $C_{NT} = \min(\sqrt{N}, \sqrt{T})$.

The implementation of the Bai and Ng strategy for determining the number of factors is a follows. First, select a number $K_{\max}$ indicating the maximum number of factors to be considered. Then for each value of $K < K_{\max}$, do the following:

1. Extract realized factors $\widehat{\mathbf{f}}_t$ using the method of APCA.
2. For each asset i, estimate the factor model

   $$R_{it} = \alpha_i + \boldsymbol{\beta}_i' \widehat{\mathbf{f}}_t^K + \varepsilon_{it},$$

   where the superscript K indicates that the regression has K factors, using time series regression and compute the residual variances

   $$\widehat{\sigma}_i^2(K) = \frac{1}{T-K-1} \sum_{t=1}^{T} \widehat{\varepsilon}_{it}^2.$$

3. Compute the cross-sectional average of the estimated residual variances for each asset based on a model with K factors

   $$\widehat{\sigma}^2(K) = \frac{1}{N} \sum_{i=1}^{N} \widehat{\sigma}_i^2(K)$$

4. Compute the cross-sectional average of the estimated residual variances for each asset based on a model with $K_{\max}$ factors, $\widehat{\sigma}^2(K_{\max})$.
5. Compute the information criteria $\mathrm{PC}_{p1}(K)$ and $\mathrm{PC}_{p2}(K)$.
6. Select the value of K that minimized either $\mathrm{PC}_{p1}(K)$ or $\mathrm{PC}_{p2}(K)$.

Bai and Ng performed an extensive simulation study and found that the selection criteria PC_{p1} and PC_{p2} yield high precision when $\min(N, T) > 40$.

Example 114 *Determining the number of factors for a statistical factor model estimated by asymptotic principal components*

To determine the number of factors in the "timeSeries" folio.dat using the Connor-Korajczyk method with a maximum number of factors equal to ten and a significance level equal to five percent use[8]

```
> folio.mf.ck = mfactor(folio.dat,k="ck",max.k=10,sig=0.05)
> folio.mf.ck

Call:
mfactor(x = folio.dat, k = "ck", max.k = 10, sig = 0.05)

Factor Model:
 Factors Variables Periods
       2      1618     182

Factor Loadings:
      Min. 1st Qu.   Median   Mean 3rd Qu. Max.
F.1 -0.177  0.2181  0.31721 0.3317   0.419 0.95
F.2 -0.411 -0.0958 -0.00531 0.0777   0.181 1.12

Regression R-squared:
  Min. 1st Qu. Median  Mean 3rd Qu.  Max.
 0.000 0.124   0.188  0.206 0.268   0.837
```

Two factors are selected by the Connor-Korajczyk method. Notice that most of the loadings on the first factor are positive.

Similarly, to determine the number of factors using the Bai-Ng method use

```
> folio.mf.bn = mfactor(folio.dat,k="bn",max.k=10,sig=0.05)
> folio.mf.bn$k
[1] 2
```

Again, two factors are determined.

15.6 References

Alexander, C. (2001). *Market Models: A Guide to Financial Data Analysis*, John Wiley & Sons, Chichester, UK.

Bai, J. and Ng, S., (2002). "Determining the Number of Factors in Approximate Factor Models," *Econometrica*, 70, 191-221.

[8]For a data set with a large number of assets, the Connor-Korajczyk and Bai-Ng methods may take a while.

Chamberlain, G. and Rothschild, M. (1983). “Arbitrage, Factor Structure and Mean-Variance Analysis in Large Asset Markets,” *Econometrica*, 51, 1305-1324.

Chan, L.K., Karceski, J. and Lakonishok, J. (1998). ”The Risk and Return from Factors,” *Journal of Financial and Quantitative Analysis*, 33(2), 159-188.

Chan, L.K., Karceski, J. and Lakonishok, J. (1999). “On Portfolio Optimization: Forecasting Covariances and Choosing the Risk Model,” *Review of Financial Studies*, 5, 937-974.

Chen, N.F., Roll, R., and Ross, S.A. (1986). “Economic Forces and the Stock Market,” *The Journal of Business*, 59(3), 383-404.

Campbell, J.Y., Lo, A.W., and MacKinlay, A.C. (1997). *The Econometrics of Financial Markets.* Princeton University Press, Princeton, NJ.

Connor, G. (1995). “The Three Types of Factor Models: A Comparison of Their Explanatory Power,” *Financial Analysts Journal*, 42-46.

Connor, G., and Korajczyk, R.A. (1986). “Performance Measurement with the Arbitrage Pricing Theory: A New Framework for Analysis,” *Journal of Financial Economics*, 15, 373-394.

Connor, G., and Korajczyk, R.A. (1988). “Risk and Return in an Equilibrium APT: Application of a New Test Methodology,” *Journal of Financial Economics*, 21, 255-289.

Connor, G. and Korajczyk, R.A. (1993). “A Test for the Number of Factors in an Approximate Factor Model,” *The Journal of Finance*, vol. 48(4), 1263-92.

Elton, E. and M.J. Gruber (1997). *Modern Portfolio Theory and Investment Analysis, 5th Edition.* John Wiley & Sons, New York.

Fama, E. and K.R. French (1992). “The Cross-Section of Expected Stock Returns”, *Journal of Finance*, 47, 427-465.

Grinold, R.C. and Kahn, R.N. (2000). *Active Portfolio Management: A Quantitative Approach for Producing Superior Returns and Controlling Risk,* Second Edition. McGraw-Hill, New York.

Johnson and Wichern (1998). *Multivariate Statistical Analysis.* Prentice-Hall, Englewood Cliffs, New Jersey.

Sharpe, W.F. (1970). *Portfolio Theory and Capital Markets*. McGraw-Hill, New York.

Sheikh, A. (1995). “BARRA’s Risk Models,” mimeo, BARRA.

16
Term Structure of Interest Rates

16.1 Introduction

In financial markets, the term structure of interest rates is crucial to pricing of fixed income securities and derivatives. The last thirty years have seen great advances in the financial economics of term structure of interest rates. This chapter will focus on interpolating the term structure of interest rates from discrete bond yields. Refer to Campbell, Lo, and MacKinlay (1997) for basic concepts in fixed income calculations and Hull (1997) for an introduction to theoretical term structure modeling.

Section 16.2 first defines different rates, such as spot or zero coupon interest rate, forward rate, and discount rate, and documents how one rate can be converted to another. Section 16.3 shows how to interpolate term structure data using quadratic or cubic spline. Section 16.4 illustrates how to use smoothing splines to fit term structure data. Section 16.5 introduces the parametric Nelson-Siegel function and its extension and shows how it can be used to interpolate term structure data. Bliss (1997) and Ferguson and Raymar (1998) compared the performance of these different methods. Section 16.6 concludes this chapter.

16.2 Discount, Spot and Forward Rates

16.2.1 Definitions and Rate Conversion

Although many theoretical models in financial economics hinge on an abstract interest rate, in reality there are many different interest rates. For example, the rates of a three month U.S. Treasury bill are different from those of a six month U.S. Treasury bill. The relationship between these different rates of different maturity is known as the term structure of interest rates. The term structure of interest rates can be described in terms of spot rate, discount rate or forward rate.

The *discount function*, $d(m)$, gives the present value of \$1.00 which is repaid in m years. The corresponding *yield to maturity* of the investment, $y(m)$, or *spot interest rate*, or *zero coupon rate*, must satisfy the following equation under continuous compounding:

$$d(m)e^{y(m)\cdot m} = 1$$

or

$$d(m) = e^{-y(m)\cdot m} \tag{16.1}$$

Obviously, the discount function is an exponentially decaying function of the maturity, and must satisfy the constraint $d(0) = 1$.

The above equation easily shows that under continuous compounding

$$y(m) = -\frac{\log d(m)}{m}.$$

If discrete compounding is used instead, one can similarly show that

$$y(m) = p[d(m)^{-\frac{1}{p\cdot m}} - 1]$$

where p is the number of compounding periods in a year.

The spot interest rate is the single rate of return applied over the maturity of m years starting from today. It is also useful to think of it as the average of a series of future spot interest rates, or *forward rates*, with different maturities starting from a point in the future, and thus:

$$e^{y(m)\cdot m} = e^{\int_0^m f(x)dx}$$

from which one can easily obtain:

$$y(m) = \frac{1}{m}\int_0^m f(x)dx \tag{16.2}$$

with $f(m)$ denoting the forward rate curve as a function of the maturity m.

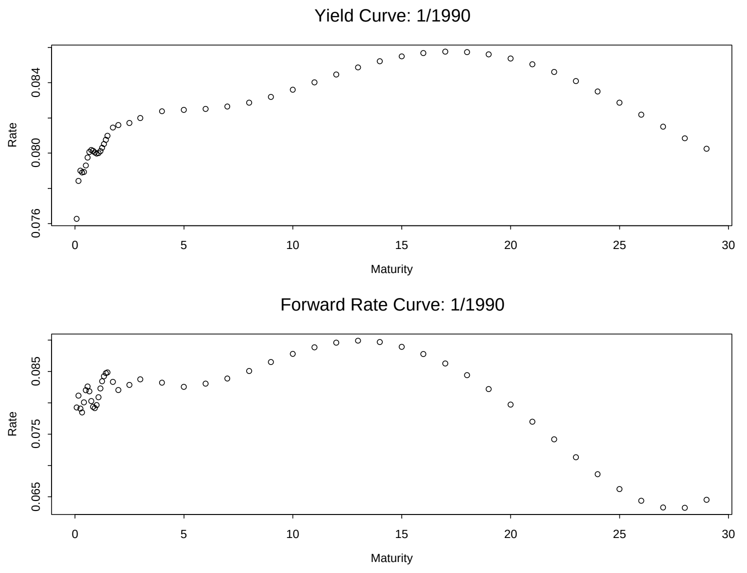


FIGURE 16.1. Yield curve and forward rate curve for January 1990.

From (16.1) and (16.2), the relationship between the discount function and forward rate can be derived:

$$d(m) = \exp\{-\int_0^m f(x)dx\}$$

or

$$f(m) = -\frac{d'(m)}{d(m)}.$$

Hence the forward rate gives the rate of decay of the discount function as a function of the maturity m. The relationship between these different rates under discrete compounding can be similarly obtained.

16.2.2 Rate Conversion in `S+FinMetrics`

To facilitate the interpolation of term structure from any of discount rate, spot rate, or forward rate, `S+FinMetrics` provides a group of functions for converting one rate into another rate. These functions will be illustrated using the `mk.zero2` and `mk.fwd2` data sets in `S+FinMetrics`, which contains the U.S. zero coupon rates and forward rates, respectively, as computed by McCulloch and Kwon (1993).

Both `mk.zero2` and `mk.fwd2` are “`timeSeries`” objects with 55 columns, with each column representing the rate with the corresponding maturity in the 55×1 vector `mk.maturity`. For example, the first element of the vector `mk.maturity` is 0.083, so the first columns of `mk.zero2` and `mk.fwd2` correspond to the rates with maturity of one month. Use the following code to plot the yield curve and forward rate curve for January 1990, and the graph is shown in Figure 16.1:

```
> par(mfrow=c(2,1))
> plot(mk.maturity,mk.zero2[54,],xlab="Maturity",ylab="Rate")
> title(paste("Yield Curve:", positions(mk.zero2[54,])))
> plot(mk.maturity,mk.fwd2[54,],xlab="Maturity",ylab="Rate")
> title(paste("Forward Rate Curve:",positions(mk.fwd2[54,])))
> par(mfrow=c(1,1))
```

To convert the spot interest rate or forward rate into the discount rate, use the `S+FinMetrics` function `bond.discount`. For example, to convert the first 48 spot rates in Figure 16.1 to discount rates, use the following command:

```
> disc.rate = bond.discount(mk.zero2[54, 1:48],
+ mk.maturity[1:48], input="spot", compounding=2)
```

The `bond.discount` function takes two required arguments: the first is a vector of rates, and the second is a vector of the corresponding maturity. Note that the optional argument `input` is used to specify the type of the input rates, and `compounding` to specify the number of compounding periods in each year. So `compounding=2` corresponds to semi-annual compounding.[1] If the input rates are forward rates, simply set `input="forward"`.

The functions `bond.spot` and `bond.forward` can be called in a similar fashion to compute the spot interest rate and forward rate, respectively, from different input rates. For all those three functions, the rates should be expressed as decimal numbers, and the maturity should be expressed in units of years. For example, to convert `disc.rate` back into the spot rates, use the following command:

```
> spot.rate = bond.spot(disc.rate, mk.maturity[1:48],
+ input="discount", compounding=2)
```

It can be easily checked that `spot.rate` is the same as `mk.zero2[54, 1:48]`.

16.3 Quadratic and Cubic Spline Interpolation

The interest rates are observed with discrete maturities. In fixed income analysis, the rate for a maturity which is not observed can sometimes be

[1] To use continuous compounding, specify `compounding=0`.

used. Those unobserved rates can usually be obtained by interpolating the observed term structure.

Since the discount rate should be a monotonically decreasing function of maturity and the price of bonds can be expressed as a linear combination of discount rates, McCulloch (1971, 1975) suggested that a spline method could be used to interpolate the discount function, or the bond prices directly. In particular, use k continuously differentiable functions $s_j(m)$ to approximate the discount rates:

$$d(m) = a_0 + \sum_{j=1}^{k} a_j s_j(m) \tag{16.3}$$

where $s_j(m)$ are known functions of maturity m, and a_j are the unknown coefficients to be determined from the data. Since the discount rate must satisfy the constraint $d(0) = 1$, set $a_0 = 1$ and $s_j(0) = 0$ for $j = 1, \cdots, k$. Note that once the functional form of $s_j(m)$ is determined, the coefficients a_j can be easily estimated by linear regression. Thus the discount rate, or forward rate, or spot rate, associated with an unobserved maturity can be easily interpolated using the above functional form, as long as the maturity is smaller than the largest maturity used in the estimation.

Figure 16.1 shows that there are usually more points in the short end of the term structure, and less points in the long end of the term structure. To obtain a reliable interpolation using the spline method, the functional form of $s_j(m)$ should be chosen so that it adapts to the density of maturity m. McCulloch (1971) gave a functional form of $s_j(m)$ using quadratic spline, which is based on piecewise quadratic polynomials, while McCulloch (1975) gave a functional form of $s_j(m)$ using cubic spline, which is based on piecewise cubic polynomials.

Term structure interpolation using quadratic or cubic spline methods can be performed by calling the `term.struct` function in `S+FinMetrics`. The arguments taken by `term.struct` are:

```
> args(term.struct)
function(rate, maturity, method = "cubic", input.type = "spot",
        na.rm = F, plot = T, compounding.frequency = 0,
        k = NULL, cv = F, penalty = 2, spar = 0, ...)
NULL
```

Similar to `bond.spot`, `bond.discount` and `bond.forward` functions, the first argument `rate` should be a vector of interest rates, while the second argument `maturity` specifies the corresponding maturity in units of years. The type of the input interest rate should be specified through the optional argument `input.type`. Note that the quadratic or cubic spline methods operate on discount rates. If the input interest rates are not discount rates, the optional argument `compounding.frequency` should also be set for proper conversion, which is set to zero for continuous compounding

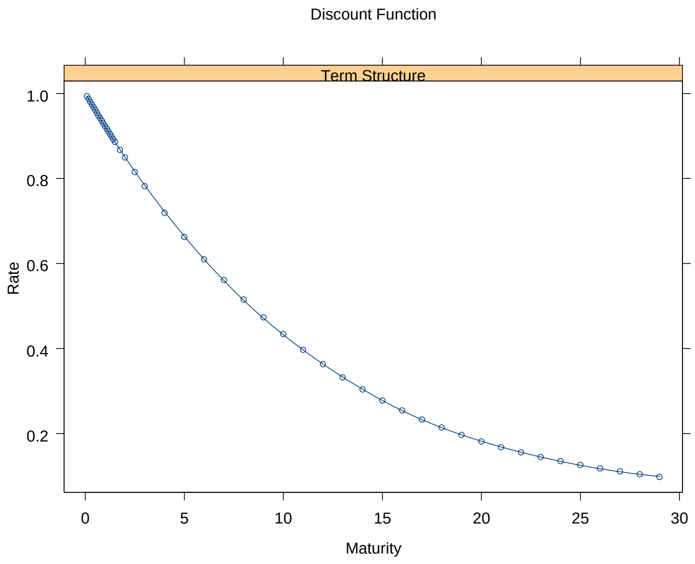


FIGURE 16.2. U.S. discount function for January 1990: quadratic spline.

by default. The optional argument `k` determines the number of functions in (16.3), also known as *knot points*. By default, follow McCulloch (1971, 1975) and set $k = [\sqrt{n}]$ where n is the length of the input rates. Other optional arguments will be discussed in later sections.

To illustrate the usage of the spline methods, in order to interpolate the term structure corresponding to January 1990, using `mk.zero2`, use the following command:

```
> disc.rate = term.struct(mk.zero2[54,], mk.maturity,
+ method="quadratic", input="spot", na.rm=T)
```

Note that `na.rm=T` is set to remove the missing values at the long end of the term structure. By default, the interpolated discount rate is plotted automatically, which is shown in Figure 16.2. The points in the figure represent the original discount rates, while the line represents the spline interpolation.

The returned object `disc.rate` is of class "`term.struct`". As usual, typing the name of the object at the command line invokes its `print` method:

```
> class(disc.rate)
[1] "term.struct"
> disc.rate

Call:
```

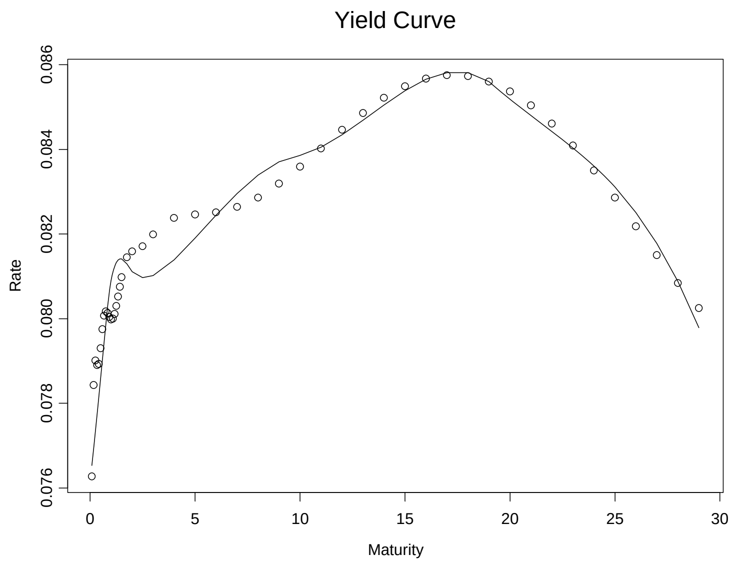


FIGURE 16.3. U.S. yield curve for January 1990: quadratic spline.

```
term.struct(rate = mk.zero2[54,  ], maturity = mk.maturity,
      method = "quadratic", input.type = "spot", na.rm = T)

Coefficients:
      a1      a2      a3      a4      a5      a6
 -0.0761 -0.0791 -0.0688 -0.0373 -0.0146 -0.0045

Degrees of freedom: 48 total; 42 residual
Residual standard error: 0.001067688
```

Since the unknown coefficients a_j of the spline are estimated by linear regression, the output looks very much similar to linear regression output. Since there are 48 spot rates available for January 1990, the number of knot points is chosen to be 6 by default.

The plot generated in Figure 16.2 shows the interpolated discount function because the quadratic or cubic spline methods are designed to operate on discount function. This plot can be later regenerated by calling the generic `plot` function on a "`term.struct`" object. However the yield curve or forward rate curve is usually of more interest. These can also be easily plotted using the components of a "`term.struct`" object. For example, use the `S-PLUS` `names` function to find out the components of `disc.rate`:

```
> names(disc.rate)
 [1] "coefficients" "residuals"  "fitted.values" "effects"
 [5] "R"             "rank"       "assign"        "df.residual"
 [9] "contrasts"     "terms"      "call"          "fitted"
[13] "knots"         "method"     "maturity"      "rate"
```

The first 10 components are inherited from an "`lm`" object, because the `S-PLUS` `lm` function is used for the linear regression. The `fitted` (instead of the `fitted.values`) component represents the estimated discount rates associated with the `maturity` component. To plot the interpolated yield curve or forward rate curve, simply convert the estimated discount rates into the rates you want. For example, use the following code to plot the interpolated yield curve:

```
> spot.rate = bond.spot(disc.rate$fitted, disc.rate$maturity,
+ input="discount", compounding=0)
> plot(mk.maturity[1:48], mk.zero2[54,1:48],
+ xlab="Maturity", ylab="Rate", main="Yield Curve")
> lines(disc.rate$maturity, spot.rate)
```

and the plot is shown in Figure 16.3. Note that in the plot the points represent the original zero coupon rates, while the line represents the quadratic spline interpolation.

16.4 Smoothing Spline Interpolation

The previous section demonstrated that the polynomial spline methods proposed by McCulloch (1971, 1975) can fit the discount rate and yield curve very well. However, since the methods operate on (linear combinations of) discount functions, the implied forward rate curve usually has some undesirable features. For example, use the following code to generate the implied forward rate curve from the object `disc.rate` fitted in the previous section:

```
> fwd.rate = bond.forward(disc.rate$fitted, disc.rate$maturity,
+ input="discount", compounding=0)
> plot(disc.rate$maturity, fwd.rate, type="l",
+ xlab="Maturity", ylab="Rate", main="Forward Rate")
> points(mk.maturity[1:48], mk.fwd2[54, 1:48])
```

The plot is shown in Figure 16.4. The implied forward rate is way off at the long end of the term structure.

In addition to the undesirable behavior of implied forward rate, the choice of knot points for polynomial splines is rather *ad hoc*. For a large number of securities, the rule can imply a large number of knot points, or coefficients a_j. To avoid these problems with polynomial spline methods, Fisher, Nychka and Zervos (1995) proposed to use *smoothing splines* for interpolating the term structure of interest rates.

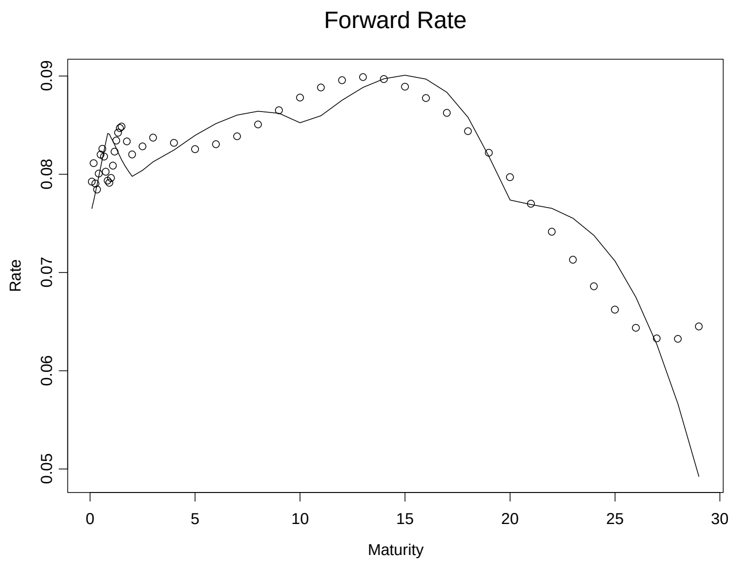


FIGURE 16.4. U.S. forward rate for January 1990: quadratic spline.

In general, for an explanatory variable x_i and a response variable y_i, the smoothing spline tries to find a smooth function $f(\cdot)$ to minimize the penalized residual sum of squares (PRSS):

$$\text{PRSS} = \sum_{i=1}^{n}[y_i - f(x_i)]^2 + \lambda \int [f''(t)]^2 dt \tag{16.4}$$

where the first term is the residual sum of squares (RSS), and the second term is the penalty term, and the parameter λ controls the trade-off between goodness-of-fit and parsimony. By using the penalty term, the spline function can be over-parameterized, while using λ to reduce the effective number of parameters.

Let S denote the $n \times n$ implicit smoother matrix such that $f(x_i) = \sum_{j=1}^{n} S(x_i, x_j)y_j$. Fisher, Nychka and Zervos (1995) suggested using generalized cross validation (GCV) to choose λ. That is, λ is chosen to minimize

$$\text{GCV} = \frac{\text{RSS}}{n - \theta \cdot \text{tr}(S)}$$

where θ is called the *cost*, and $\text{tr}(S)$ denotes the trace of the implicit smoother matrix and is usually used as the measure of effective number of parameters.

Interpolation of term structure using smoothing spline can also be performed using the `term.struct` function by setting the optional argument

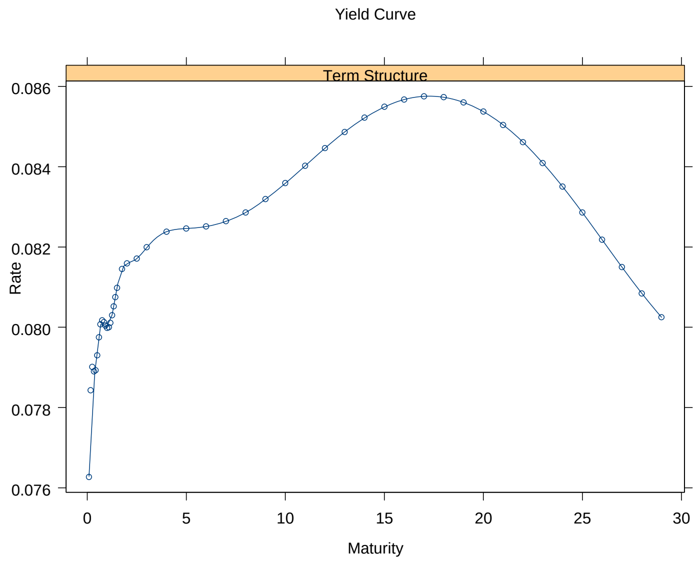


FIGURE 16.5. U.S. yield curve for January 1990: smoothing spline.

`method="smooth"`. The procedure uses the `S-PLUS` `smooth.spline` function as the workhorse.[2] In particular, for all the arguments taken by the function `term.struct`, `cv`, `penalty` and `spar` are specifically used for smoothing spline methods and passed to the `smooth.spline` function. By default, use GCV by setting `cv=F` and thus `spar`, which specifies the value of λ, is ignored.[3] The optional argument `penalty` is used to specify the value for θ. Following Fisher, Nychka, and Zervos (1995), set $\theta = 2$ by default.

For example, use the following command to interpolate the yield curve for January 1990, with the smoothing spline method:

```
> fnz.fit = term.struct(mk.zero2[54,], mk.maturity,
+ method="smooth", input="spot", na.rm=T)
```

Again, the interpolated yield curve is plotted automatically, as shown in Figure 16.5. Although the returned object `fnz.fit` is of class "`term.struct`", its components are different from the `disc.rate` object fitted in the previous section, because now the `smooth.spline` function is used as the workhorse:

[2]Refer to Hastie (1993) and *S-PLUS Guide to Statistics* for the description of `smooth.spline` function.

[3]For further details regarding these arguments, see the on-line help file for `smooth.spline` function.

```
> class(fnz.fit)
[1] "term.struct"
> names(fnz.fit)
 [1] "x"        "y"      "w"      "yin"    "lev"    "cv.crit"
 [7] "pen.crit" "df"     "spar"   "fit"    "call"   "method"
[13] "maturity" "rate"
```

The first 10 components are inherited from a “smooth.spline” object, while the last four components are generated by the term.struct function. For the same reason, the print function now shows different information:

```
> fnz.fit
Call:
term.struct(rate = mk.zero2[54,  ], maturity = mk.maturity,
        method = "smooth", input.type = "spot", na.rm = T)

Smoothing Parameter (Spar): 4.767984e-11
Equivalent Degrees of Freedom (Df): 47.57122
Penalized Criterion: 4.129338e-10
GCV: 3.605842e-14
```

which shows the optimal smoothing parameter λ, and its associated GCV, penalized criterion, and equivalent degrees of freedom.

For “term.struct” objects, S+FinMetrics also implements a predict method, which can be used to obtain the interpolated rate associated with an arbitrary vector of maturity. For example, to recover the fitted spot rates from fnz.fit, use the predict method as follows:

```
> fnz.spot = predict(fnz.fit, fnz.fit$maturity)
```

From the fitted spot rates, one can compute the implied forward rates for the smoothing spline:

```
> fnz.forward = bond.forward(fnz.spot, fnz.fit$maturity,
+ input="spot", compounding=0)
> plot(mk.maturity[1:48], mk.fwd2[54,1:48],
+ xlab="Maturity", ylab="Rate", main="Forward Rate")
> lines(fnz.fit$maturity, fnz.forward)
```

The “real” forward rates and the smoothing spline interpolations are shown together in Figure 16.6. The interpolations agree very well with the “real” forward rates. The slight difference is partly caused by the fact that mk.zero2[54,] and the spot rates implied by mk.fwd2[54,] are slightly different.

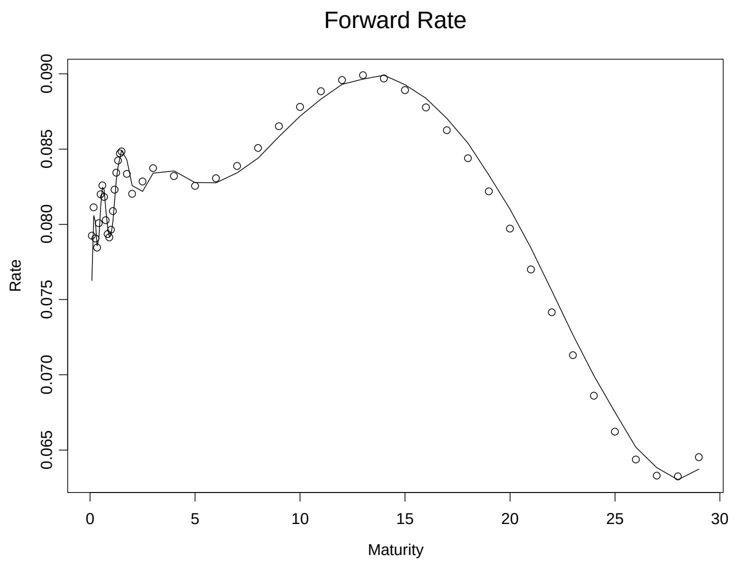


FIGURE 16.6. U.S. forward rate for January 1990: smoothing spline.

16.5 Nelson-Siegel Function

The previous sections have shown that both the polynomial and smoothing spline methods can fit the term structure very well, except that the implied forward rates from polynomial spline methods have some undesirable features at the long end of the term structure. However, the non-parametric spline based methods usually do not generate good out-of-sample forecasts. There is substantial evidence showing that a parametric function suggested by Nelson and Siegel (1987) has better out-of-sample forecasting performance.

Using a heuristic argument based on the expectation theory of the term structure of interest rates, Nelson and Siegel (1987) proposed the following parsimonious model for the forward rate:

$$f(m) = \beta_0 + \beta_1 \cdot e^{-m/\tau} + \beta_2 \cdot m/\tau \cdot e^{-m/\tau}.$$

They suggested that the model may also be viewed as a constant plus a Laguerre function, and thus can be generalized to higher-order models. Based on the above equation, the corresponding yield curve can be derived as follows:

$$y(m) = \beta_0 + \beta_1 \frac{1 - e^{-m/\tau}}{m/\tau} + \beta_2 \left[\frac{1 - e^{-m/\tau}}{m/\tau} - e^{-m/\tau} \right]. \qquad (16.5)$$

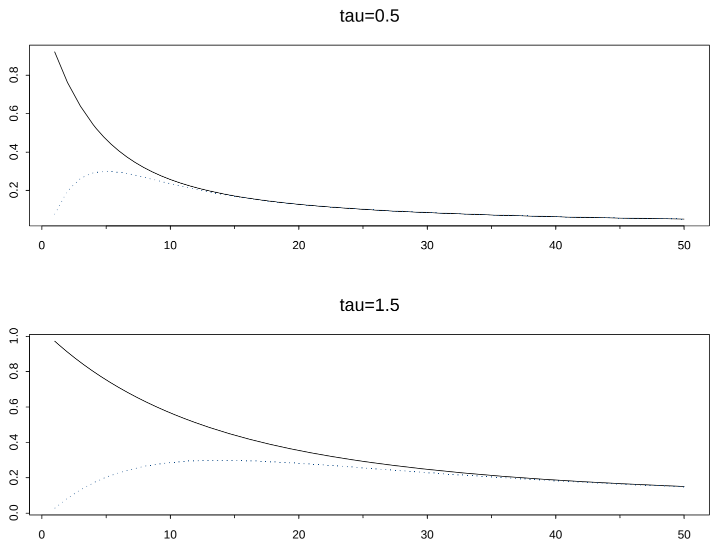


FIGURE 16.7. Short term and medium term components of Nelson-Siegel function.

For a given constant τ, both the forward rate curve and the yield curve are linear functions of the coefficients β_0, β_1 and β_2. Nelson and Siegel (1987) showed that, depending on the values of β_1 and β_2, the yield curve can assume the common shapes of observed yield curves, such as upward sloping, downward sloping, humped, or inverted humped. In addition, consistent with stylized facts of the yield curve, the three components in (16.5) can be interpreted as the long term, short term and medium term component, or the level, slope, and curvature component of the yield curve.[4]

Example 115 *Interpretation of Nelson-Siegel function*

The function `term.struct.nsx` in `S+FinMetrics` can be used to generate the regressors in (16.5) given a vector of maturity and a value for τ. Use the following code to visualize these components for different values of τ:

```
> ns.maturity = seq(1/12, 10, length=50)
> ns05 = term.struct.nsx(ns.maturity, 0.5)
> ns15 = term.struct.nsx(ns.maturity, 1.5)
> par(mfrow=c(2,1))
> tsplot(ns05[,2:3], main="tau=0.5")
> tsplot(ns15[,2:3], main="tau=1.5")
```

[4]Refer to Diebold and Li (2002) for a detailed explanation.

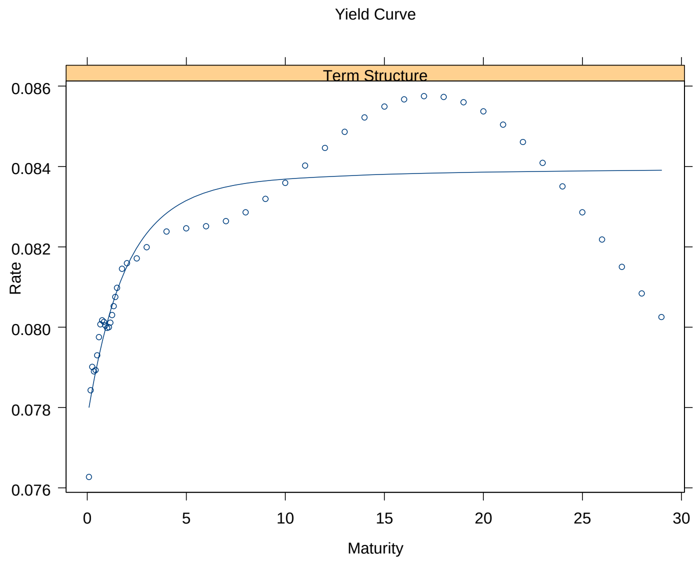


FIGURE 16.8. U.S. yield curve for January 1990: Nelson-Siegel function.

```
> par(mfrow=c(1,1))
```

A vector of maturity was created from one month to ten years. The regressor matrix has three columns, and only the last two columns were plotted because the first column is always one, and the plot is shown in Figure 16.7. The parameter τ controls the rate of decay of those components. When τ is smaller, the short and medium term components decay to zero at a faster rate. Asymptotically, both the short and medium term components approach zero, and thus β_0 can be interpreted as the long term component, or the level of the yield curve.

To interpolate yield curves using the Nelson-Siegel function, choose the value of τ which gives the best fit for equation (16.5). The `term.struct` function employs this procedure if the optional argument `method` is set to `"ns"`. For example, use the following command to interpolate the yield curve for January 1990:

```
> ns.fit = term.struct(mk.zero2[54,], mk.maturity,
+ method="ns", input="spot", na.rm=T)
> ns.fit

Call:
term.struct(rate = mk.zero2[54,  ], maturity = mk.maturity,
      method = "ns", input.type = "spot", na.rm = T)
```

```
Coefficients:
      b0      b1      b2
  0.0840 -0.0063  0.0044

Degrees of freedom: 48 total; 45 residual
Residual standard error: 0.001203026
Tau estimate: 1.7603
```

Again, the fit is plotted by default as shown in Figure 16.8. The graph shows that although the Nelson-Siegel generally captures the shape of the yield curve, the in-sample fit is usually not as good as the non-parametric spline methods because it only uses three coefficients. The output shows the estimates of those coefficients, along with the estimate of τ.

Since the Nelson-Siegel function does not fit the data very well when the yield curve has a rich structure as in the above example, Svensson (1994) proposed to extend the Nelson-Siegel forward function as follows:

$$f(m) = \beta_0 + \beta_1 e^{-m/\tau_1} + \beta_2 \cdot m/\tau_1 \cdot e^{-m/\tau_1} + \beta_3 \cdot m/\tau_2 \cdot e^{-m/\tau_2}$$

which adds another term to the Nelson-Siegel function to allow for a second hump. The corresponding yield function can be shown to be:

$$\begin{aligned} y(m) = \beta_0 + \beta_1 \frac{1 - e^{-m/\tau_1}}{m/\tau_1} + \beta_2 \left[\frac{1 - e^{-m/\tau_1}}{m/\tau_1} - e^{-m/\tau_1} \right] \\ + \beta_3 \left[\frac{1 - e^{-m/\tau_2}}{m/\tau_2} - e^{-m/\tau_2} \right]. \end{aligned} \quad (16.6)$$

To use the above function for interpolating yield curve, simply call the function `term.struct` with `method="nss"`:

```
> nss.fit = term.struct(mk.zero2[54,], mk.maturity,
+ method="nss", input="spot", na.rm=T)
> nss.fit

Call:
term.struct(rate = mk.zero2[54,  ], maturity = mk.maturity,
      method = "nss", input.type = "spot", na.rm = T)

Coefficients:
     b0     b1     b2     b3
 0.0000 0.0761 0.1351 0.0104

Degrees of freedom: 48 total; 44 residual
Residual standard error: 0.0005997949
Tau estimate: 21.8128 0.5315
```

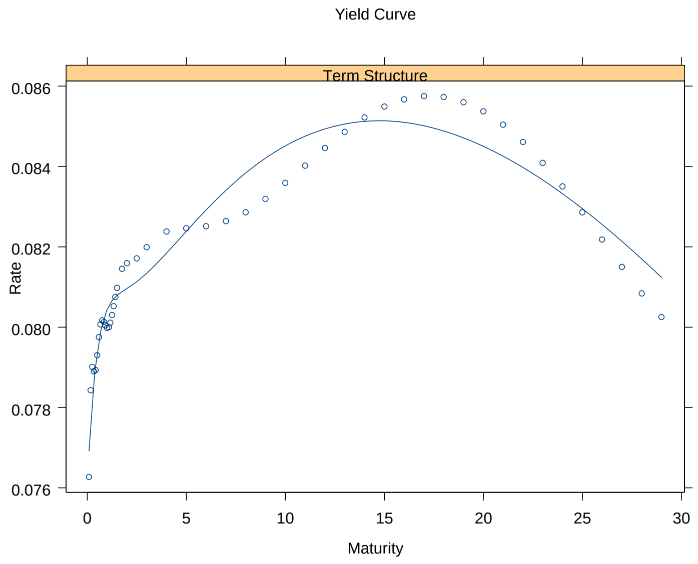


FIGURE 16.9. U.S. yield curve for January 1990: Svensson function.

The output now shows two estimates for τ and one more coefficient for the additional term. The plot of the interpolated yield curve is shown in Figure 16.9.

16.6 Conclusion

For all the term structure interpolation methods discussed in this chapter, they all work with the yield curve for a given time, and thus do not consider the time series aspect of the yield curve. Recently Diebold and Li (2002) considered estimating the three components β_0, β_1 and β_2 of the Nelson-Siegel function for each available time, and building a time series model (in particular, an AR(1)-GARCH(1,1) model) for the estimated β_0, β_1 and β_2. By employing the times series forecasts of β_0, β_1 and β_2, they are able to generate reliable forecasts of yield curve. However, in this approach, the coefficients β_0, β_1 and β_2 are still estimated ignoring the time series aspect.

In recent years, many researchers have proposed to use state space models and Kalman filter to estimate the term structure of interest rates using a panel data, for example, see Duan and Simonato (1999), Geyer and Pichler (1999), Babbs and Nowman (1999), de Jong and Santa-Clara (1999) and de Jong (2000). Most of these models are special cases of the affine term structure model proposed by Duffie and Kan (1996), which can be readily expressed in a state space model by discretizing the continuous-time models. These models can be easily implemented using the state space modeling functions in `S+FinMetrics` as illustrated in Zivot, Wang and Koopman (2004).

16.7 References

Babbs, S. H., and Nowman, K. B. (1999). "Kalman Filtering of Generalized Vasicek Term Structure Models," *Journal of Financial and Quantitative Analysis*, 34(1), 115-129.

Bliss, R. R. (1997). "Testing Term Structure Estimation Methods", in P. Boyle, G. Pennacchi, and P. Ritchken (eds.), *Advances in Futures and Options Research*, Volume 9. Elsevier, Amsterdam, pp. 197-231.

Campbell, J. Y., Lo, A. W., and MacKinlay, A. C. (1997). *The Econometrics of Financial Markets*. Princeton University Press, Princeton, NJ.

de Jong, F. (2000). "Time Series and Cross Section Information in Affine Term-Structure Models," *Journal of Business and Economic Statistics*, 18(3), 300-314.

de Jong, F., and Santa-Clara, P. (1999). "The Dynamics of the Forward Interest Rate Curve: a Formulation with State Variables," *Journal of Financial and Quantitative Analysis*, 34(1), 131-157.

Diebold, F. X., and Li, C. (2003). "Forecasting the Term Structure of Government Bond Yields," NBER Working Paper No. 10048.

Duan, J.-C., and Simonato, J. (1999). "Estimating and Testing Exponential-Affine Term Structure Models by Kalman Filter," *Review of Quantitative Finance and Accounting*, 13, 111-135.

Duffie, D., and Kan, R. (1996). "A Yield-Factor Model of Interest Rates," *Mathematical Finance*, 6(4), 379-406.

Ferguson, R., and Raymar, S. (1998). "A Comparative Analysis of Several Popular Term Structure Estimation Models," *Journal of Fixed Income*, March 1998, 17-33.

Fisher, M., Nychka, D., and Zervos, D. (1995). "Fitting the Term Structure of Interest Rates with Smoothing Splines," Finance and Economics Discussion Series #1995-1, Board of Governors of the Federal Reserve System.

Geyer, A. L. J., and Pichler, S. (1999). "A State-Space Approach to Estimate and Test Multifactor Cox-Ingersoll-Ross Models of the Term Structure," *Journal of Financial Research*, 22(1), 107-130.

Hastie, T. J. (1993). "Generalized Additive Models," in J. M. Chambers and T. J. Hastie (eds.), *Statistical Models in S*. Chapman & Hall.

Hull, J. C. (1997). *Options, Futures, and Other Derivatives*, Prentice Hall, New York.

McCulloch, J. H. (1971). "Measuring the Term Structure of Interest Rates," *Journal of Business*, 44, 19-31.

McCulloch, J. H. (1975). "The Tax-Adjusted Yield Curve", *Journal of Finance*, 30(3), 811-830.

McCulloch, J. H., and Kwon, H.-C. (1993). "U.S. Term Structure Data: 1947-1991," Department of Economics, Working Paper #93-6, Ohio State University.

Nelson, C. R., and Siegel, A. F. (1987). "Parsimonious Modeling of Yield Curves," *Journal of Business*, 60(4), 473-489.

Svensson, L. E. O. (1994). "Estimating and Interpreting Forward Interest Rates: Sweden 1992-1994", NBER Working Paper No. 4871.

Zivot, E., Wang, J. and S.J. Koopman (2004). "State Space Models in Economics and Finance Using SsfPack in S+FinMetrics," in A. Harvey, S.J. Koopman, and N. Shephard (eds.), *Unobserved Components Models.* Cambridge University Press, Cambridge.

17
Robust Change Detection

17.1 Introduction

In time series analysis, autoregressive integrated moving average (ARIMA) models have found extensive use since the publication of Box and Jenkins (1976). For an introduction to the standard ARIMA modeling in `S-PLUS`, see *S-PLUS Guide to Statistics*. Regression models are also frequently used in finance and econometrics research and applications. For example, as "factor" models for empirical asset pricing research and for parsimonious covariance matrix estimation in portfolio risk models. Often ARIMA models and regression models are combined by using an ARIMA model to account for serially correlated residuals in a regression model, resulting in REGARIMA models.

In reality, most time series data are rarely completely well behaved and often contain outliers and level shifts, which is especially true for economic and financial time series. The classical maximum likelihood estimators of both ordinary regression model parameters and ARIMA model parameters are not robust in that they can be highly influenced by the presence of even a small fraction of outliers and/or level shifts in a time series. It is therefore not suprising that classical maximum likelihood estimators of REGARIMA models also lack robustness toward outliers and/or level shifts.

`S+FinMetrics` provides functions that compute robust alternatives to the classical non-robust MLE's for robust fitting and diagnostics of REGARIMA models. In particular, the robust procedure `arima.rob` allows reliable model fitting when the data contain outliers and/or level shifts. In addition, it also detects the types and locations of the outliers in the time series and thus can be used to perform robust change detection.

This chapter is organized as follows: Section 17.2 gives a brief introduction to REGARIMA models, and Section 17.3 shows how to fit a robust REGARIMA model using functions in `S+FinMetrics`. Section 17.4 shows how to predict from a robustly fitted REGARIMA model, while Section 17.5 illustrates more options which can be used to control the robust fitting of REGARIMA models. Finally in Section 17.6, some technical details are given about how REGARIMA model parameters are estimated robustly in the procedure `arima.rob`.

17.2 REGARIMA Models

The REGARIMA model considered in this chapter takes the following form:

$$y_t = \mathbf{x}_t'\boldsymbol{\beta} + \epsilon_t, \text{ for } t = 1, 2, \cdots, T \tag{17.1}$$

where $\mathbf{x}_t$ is a $k \times 1$ vector of predictor variables, and $\boldsymbol{\beta}$ is a $k \times 1$ vector of regression coefficients. The error term ϵ_t follows a seasonal ARIMA process:

$$\Phi(L)(1-L)^d(1-L^s)^D\epsilon_t = (1-\theta^* L^s)\Theta(L)u_t \tag{17.2}$$

where L is the lag (or backshift) operator, d the number of regular differences, D the number of seasonal differences, s the seasonality frequency, $\Phi(L) = 1 - \phi_1 L - \cdots - \phi_p L^p$ a stationary autoregressive operator of order p, $\Theta(L) = 1 - \theta_1 L - \cdots - \theta_q L^q$ a moving average operator of order q and θ^* a seasonal moving average parameter. Note that currently only one seasonal moving average term is allowed in the discussions in this chapter. The innovations u_t are assumed to be i.i.d. random variables with distribution F.

In practice, observed time series data are rarely well behaved as assumed in the REGARIMA model (17.1) and (17.2). An observed time series y_t^* is usually some kind of variant of y_t in equation (17.1). When the observed time series y_t^* might be influenced by some outliers, the classical maximum likelihood estimates as implemented in the `S-PLUS` function `arima.mle` are not robust. In contrast, the `S+FinMetrics` function `arima.rob` allows the robust estimation of the model parameters $(\boldsymbol{\beta}, \boldsymbol{\lambda})$, where $\boldsymbol{\lambda} = (\boldsymbol{\phi}, \boldsymbol{\theta}, \theta^*)$, $\boldsymbol{\phi}$ is a vector of the autoregressive parameters and $\boldsymbol{\theta}$ is a vector of the moving average parameters. Furthermore, it will detect three kinds of outliers in the original data y_t^*:

Additive outliers (AO): An additive outlier occurs at time t_0 if $y_{t_0}^* = y_{t_0} + c$, where c is a constant. The effect of this type of outlier is restricted to the time period t_0.

Innovation outliers (IO): An innovation outlier occurs at time t_0 if $u_{t_0} = v_{t_0} + c$, where v_{t_0} is generated by the distribution F. Usually it is assumed that F is the normal distribution $N(0, \sigma^2)$. Note that the effect of an innovation outlier is not restricted to time t_0 because of the structure of an ARIMA model. It also has influence on the subsequent observations.

Level shifts (LS): If one level shift occurs at time t_0, the observed series is $y_t^* = y_t + c$ for all $t \geq t_0$, with c being a constant. Note that if the series y_t^* has a level shift at t_0, the differenced series $y_t^* - y_{t-1}^*$ has an additive outlier at t_0.

In all those three cases c is the size of the outlier or level shift. Without any potential confusion, the general term "outlier" may refer to any of the three types of behavior.

17.3 Robust Fitting of REGARIMA Models

The `S+FinMetrics` function `arima.rob` computes the so-called "filtered τ-estimates" of the parameters $(\boldsymbol{\beta}, \boldsymbol{\lambda}, \sigma)$ of REGARIMA model (17.1)-(17.2) when a time series is influenced by outliers. The technical details of this type of estimation can be found in Section 17.6.

`S+FinMetrics` comes with a "`timeSeries`" data `frip.dat`, which represents monthly industrial production of France from January 1960 to December 1989. This data set will be used to illustrate the usage of `arima.rob` function. First, a plot of the data will show the general properties of the time series:

```
> plot(frip.dat)
```

A few characteristics of the time series can be seen from Figure 17.1: (i) there are three big outliers around 1963 and 1968; (ii) it appears that a level shift happened around 1975; (iii) there is an obvious trend in the time series, and the trend looks like a exponential one, especially in the last five years. For diagnostic purpose, a robust ARIMA(2,1,0) model can be tried on the logarithm of `frip.dat`, due to the exponential-looking trend:

```
> frip.rr = arima.rob(log(frip.dat)~1, p=2, d=1)
```

Note that the `arima.rob` function has only one required argument: a formula specifying the regression model. The optional argument `p` specifies the autoregressive order, and `d` specifies the order of difference. In this case, the only predictor variable is the intercept term.

Caveat: The interpretation of the intercept term in `arima.rob` is different from that for other formulas in `S-PLUS`. When both `d`

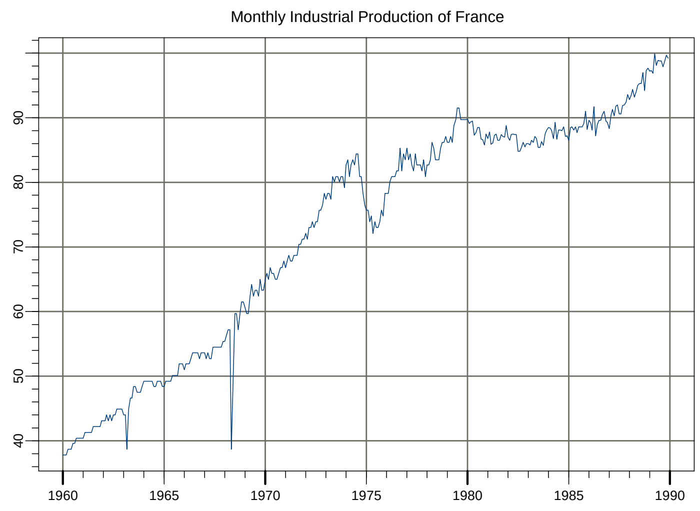


FIGURE 17.1. Monthly industrial production of France.

and `sd` (seasonal difference) are zero (which is the default), the intercept is the constant term as usual. However, when either `d` or `sd` is positive, the intercept is the coefficient of the lowest order time trend that can be identified. For instance, in the above example, the intercept corresponds to the coefficient of the trend term t. One can easily verify this using the following command:

```
> frip.t = 1:length(frip.dat)
> tmp = arima.rob(log(frip.dat)~frip.t-1, p=2, d=1)
```

which should give the same fit as `frip.rr`. The reason for this modification is obvious: some coefficients are not identifiable when differencing is involved.

The object returned by the function `arima.rob` is of class "`arima.rob`", which has `print` and `summary` methods, just like most modeling objects. For "`arima.rob`" objects, there is one additional advantage of using the `summary` method instead of the `print` method: if the data object is of class "`timeSeries`", the outliers will be lined up in a table with the time stamps of the observations, the types of the outliers, the impacts of the outliers, and the t-statistics. For example,

```
> summary(frip.rr)
```

```
Call:
arima.rob(formula = log(frip.dat) ~ 1, p = 2, d = 1)

Regression model:
 log(frip.dat) ~ 1

ARIMA model:
Ordinary differences: 1 ; AR order: 2 ; MA order: 0

Regression Coefficients:
             Value Std. Error t value Pr(>|t|)
(Intercept) 0.0024 0.0005     4.6558  0.0000

AR Coefficients:
        Value Std. Error t value Pr(>|t|)
AR(1) -0.3099  0.0537    -5.7742  0.0000
AR(2) -0.0929  0.0537    -1.7310  0.0843

Degrees of freedom: 360 total; 356 residual

Innovations standard deviation: 0.01311

 Number of outliers detected:  9

Outliers detected:

       |Time    |Type   |Impact  |t-value|
-------+--------+-------+--------+-------+
1      |Mar 1963|AO     |-0.1457 |13.76  |
-------+--------+-------+--------+-------+
2      |May 1968|AO     |-0.3978 |38.1   |
-------+--------+-------+--------+-------+
3      |Jun 1968|AO     |-0.1541 |14.55  |
-------+--------+-------+--------+-------+
4      |Sep 1968|AO     |-0.04516| 4.41  |
-------+--------+-------+--------+-------+
5      |Apr 1969|LS     | 0.04511| 3.814 |
-------+--------+-------+--------+-------+
6      |Sep 1974|LS     |-0.04351| 3.767 |
-------+--------+-------+--------+-------+
7      |Nov 1974|LS     |-0.04844| 4.092 |
-------+--------+-------+--------+-------+
8      |Sep 1976|AO     | 0.0382 | 3.829 |
-------+--------+-------+--------+-------+
9      |Apr 1986|AO     | 0.03935| 3.932 |
-------+--------+-------+--------+-------+

Innovation scale estimate before correcting outliers:
  0.01311

Innovation scale estimate after correcting outliers:
  0.01215
```

The output generated by the `summary` method actually has two sections. The first section contains the parameter estimates in the REGARIMA model. In this section, one can see that the intercept (which, again, is actually the slope of the first order time trend) and the first autoregressive coefficient are very significant (that is, they have very small p-values), while the second autoregressive coefficient is not very significant.

The second section contains a summary of the outliers automatically detected by the `arima.rob` function. In this case, nine outliers are found: the first four and the last two are additive outliers, while the middle three are level shifts. The three additive outliers shown in Figure 17.1 are all detected with very large t-statistics.

A picture is always better than a thousand words. A visual diagnostic of the model fit `frip.rr` can be obtained by using the generic `plot` function:

```
> plot(frip.rr)

Make a plot selection (or 0 to exit):

1: plot: all
2: plot: Robust ACF of Innov.
3: plot: Robust PACF of Innov.
4: plot: Normal QQ-Plot of Innov.
5: plot: Original and Cleaned Series
6: plot: Detected Outliers
Selection:
```

Selections 2 and 3 will plot the robustly estimated autocorrelations and partial autocorrelations of the innovations u_t, respectively. Selection 4 produces the normal qq-plot of the innovations, as shown in Figure 17.2, from which one can see that the three additive outliers are far away from the bulk of the data. Selection 5 plots the original response time series together with the series obtained by cleaning the original series of additive outliers using a robust filter, which is shown in Figure 17.3. Finally, Selection 6 plots the detected outliers, as shown in Figure 17.4.

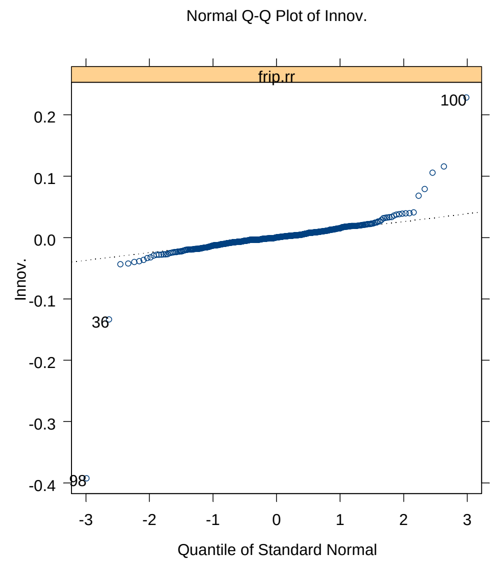


FIGURE 17.2. Normal qq-plot of robust innovations.

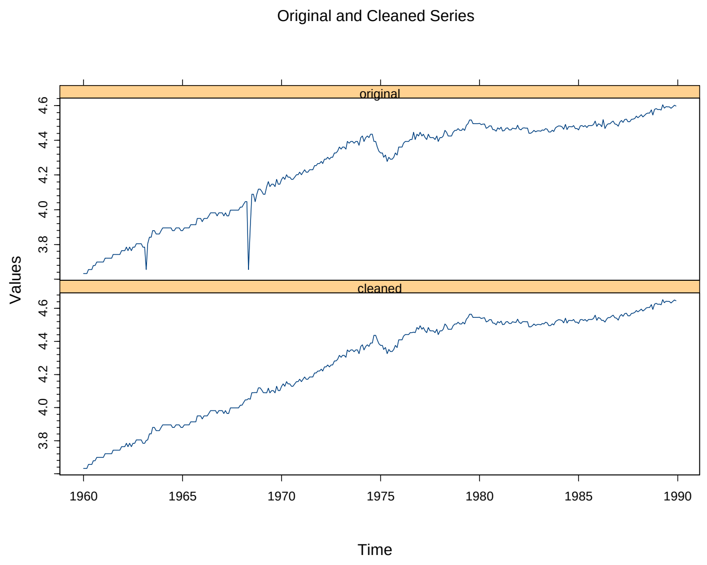


FIGURE 17.3. Original and cleaned series of `frip.dat`.

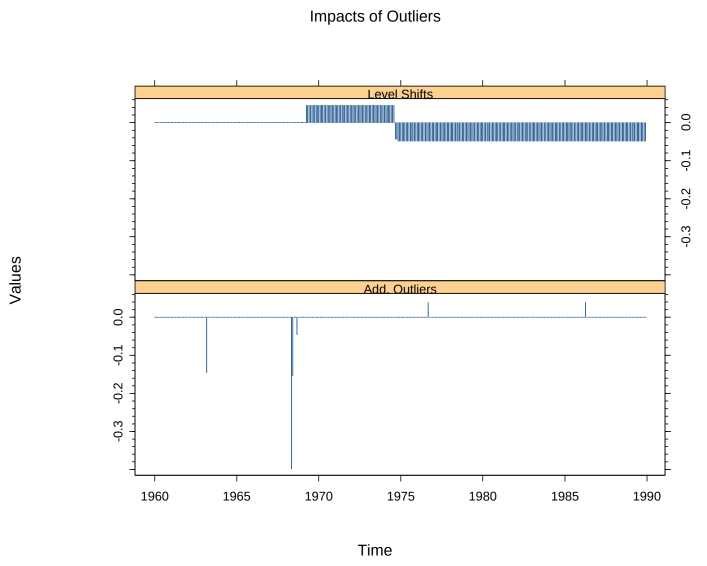


FIGURE 17.4. Detected outliers in `frip.dat`.

17.4 Prediction Using REGARIMA Models

One of the main applications of a REGARIMA model is to predict future values of response variable y_t based on past values of y_t and the corresponding future values of x_t. If future predictions are intended, then the call to the `arima.rob` function should specify the optional argument `n.predict`. This argument should be set to a number equal or greater than the number of predictions, the default of which is set to 20.

Prediction from REGARIMA models will be illustrated using the data set `import.dat` in `S+FinMetrics`, which contains two monthly time series from January 1983 to December 1990. The first series `taxes` corresponds to Argentinian import taxes and the second `import` to Argentinian imports. Another data frame `newtaxes.dat` contains the values of the variable `taxes` from January 1992 to October 1992. First fit a REGARIMA model with ARIMA(2,1,0) errors:

```
> import.rr = arima.rob(import~taxes-1, data=import.dat,
+ p=2, d=1)
```

Now with the new data of the predictor variable `taxes` in `newtaxes.dat`, one can predict `import` from January 1992 to October 1992 as follows:

```
> import.hat = predict(import.rr,10,newdata=newtaxes.dat,se=T)
> class(import.hat)
```

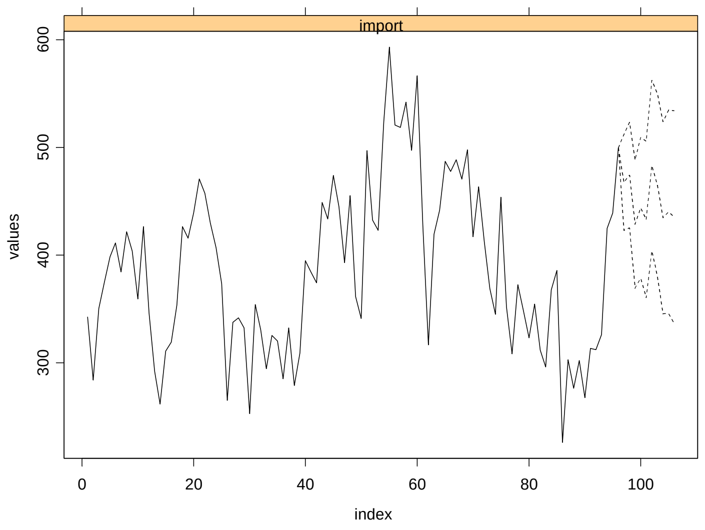


FIGURE 17.5. Fitted values ± 2· standard deviation.

```
[1] "forecast"
> names(import.hat)
[1] "values"  "std.err"
```

The optional argument `se=T` to the `predict` method tells the procedure to return the standard errors of the forecasts. The returned object `import.hat` is a "`forecast`" object with two components: `values` are the predicted values of `import`, and `std.err` are the standard errors of the prediction. Since `import.hat` is a "`forecast`" object, as we have seen from earlier chapters, the predictions can be easily plotted together with the original data:

```
> plot(import.hat, import.dat[, "import"])
```

The plot is shown in Figure 17.5.

17.5 Controlling Robust Fitting of REGARIMA Models

17.5.1 Adding Seasonal Effects

The `arima.rob` function allows for two kinds of seasonal effects options: the order of seasonal difference and the inclusion of a seasonal moving average term, controlled by the optional arguments `sd` and `sma`, respectively. For example, `frip.dat` is a monthly series, and you might expect that there are some seasonal effects in the series. Toward this end, you can add a seasonal moving average term by specifying the optional argument `sma`:

```
> frip.srr = arima.rob(log(frip.dat)~1, p=2, d=1, sfreq=12,
+ sma=T)
> summary(frip.srr)

Call:
arima.rob(formula = log(frip.dat) ~ 1, p = 2, d = 1,
    sfreq = 12, sma = T)

Regression model:
 log(frip.dat) ~ 1

ARIMA model:
Ordinary differences: 1 ; AR order: 2 ; MA order: 0
Seasonal differences: 0 ; Seasonal period: 12 ; Seasonal MA: 1

Regression Coefficients:
             Value Std. Error t value Pr(>|t|)
(Intercept) 0.0024 0.0004     5.3946  0.0000

AR Coefficients:
        Value Std. Error t value Pr(>|t|)
AR(1) -0.3135  0.0518    -6.0494  0.0000
AR(2) -0.1124  0.0518    -2.1697  0.0307

Seasonal MA Coefficient:
      Value Std. Error t value Pr(>|t|)
[1,] 0.0945 0.0519     1.8208  0.0695

Degrees of freedom: 360 total; 355 residual

Innovations standard deviation: 0.01304

 Number of outliers detected:  10

Outliers detected:

       |Time    |Type   |Impact  |t-value|
-------+--------+-------+--------+-------+
1      |Mar 1963|AO     |-0.1438 |13.58  |
-------+--------+-------+--------+-------+
2      |May 1963|LS     | 0.03988| 3.545 |
-------+--------+-------+--------+-------+
3      |May 1968|AO     |-0.3952 |38.67  |
-------+--------+-------+--------+-------+
4      |Jun 1968|AO     |-0.1519 |14.27  |
-------+--------+-------+--------+-------+
5      |Sep 1968|AO     |-0.04653| 4.615 |
-------+--------+-------+--------+-------+
6      |Apr 1969|LS     | 0.04602| 4.005 |
-------+--------+-------+--------+-------+
7      |Sep 1974|LS     |-0.04247| 3.739 |
-------+--------+-------+--------+-------+
8      |Nov 1974|LS     |-0.04914| 4.24  |
-------+--------+-------+--------+-------+
9      |Sep 1976|AO     | 0.038  | 3.891 |
-------+--------+-------+--------+-------+
10     |Apr 1986|AO     | 0.03792| 3.946 |
-------+--------+-------+--------+-------+

Innovation scale estimate before correcting outliers:
  0.01304

Innovation scale estimate after correcting outliers:
  0.01199
```

From the first section of the summary, one can see that the seasonal moving average term is relatively significant, and the estimates of other parameters are not altered very much. However, in the second section of the summary, one more level shift is detected, which corresponds to May 1963.

17.5.2 Controlling Outlier Detection

The outlier detection procedure used in `arima.rob` is similar to those proposed by Chang, Tiao and Chen (1988) and Tsay (1988) for ARIMA models, and the one used in the X12-REGARIMA program of U.S. Census Bureau. The main difference with those procedures is that `arima.rob` uses innovation residuals based on the filtered τ-estimates of $\boldsymbol{\beta}$ and $\boldsymbol{\lambda}$, instead of the classical maximum likelihood estimates.

To detect the presence of an outlier at a given time t_0, the outlier detection procedure in `arima.rob` computes:

$$\mathcal{T} = \max_{t_0} \max\{T_{t_0,\mathrm{AO}},\ T_{t_0,\mathrm{LS}},\ T_{t_0,\mathrm{IO}}\},$$

where $T_{t_0,\mathrm{AO}}$, $T_{t_0,\mathrm{LS}}$ and $T_{t_0,\mathrm{IO}}$ are the statistics corresponding to an AO, LS and IO at time t_0 respectively. The test statistic is defined as follows:

$$T = \frac{|\hat{\omega}|}{\hat{V}(\hat{\omega})^{1/2}},$$

where $\hat{\omega}$ is an estimate of ω, the size of the outlier, based on the residuals of the filtered τ-estimates and $\hat{V}(\hat{\omega})$ an estimate of its variance. If $\mathcal{T} > \xi$, where ξ is a conveniently chosen critical value, one declares that there is an outlier. The time t_0 where the outlier occurs and the type of the outlier are those where the double maximum is attained.

The critical value ξ is similar to the constant used by Chang, Tiao, and Chen (1988). They recommend using $\xi = 3$ for high sensitivity in outlier detection, $\xi = 3.5$ for medium sensitivity and $\xi = 4$ for low sensitivity, when the length of the series is less than 200. For `arima.rob` the critical value ξ is specified by the optional argument `critv`. The default value of `critv` is set as follows:

$$\xi = \begin{cases} 3 & \text{if} \quad T \leq 200, \\ 3.5 & \text{if} \quad 200 < T \leq 500, \\ 4 & \text{if} \quad T > 500. \end{cases}$$

More details of this procedure can be found in Bianco, Garcia Ben, Martinez, and Yohai (1996, 2001).

So far none of the outliers detected is an innovation outlier. This is not a coincidence. By default, the outlier detection procedure in `arima.rob` does not consider innovation outliers. To allow for innovation outliers, use the optional argument `innov.outlier`:

```
> frip.nrr = arima.rob(log(frip.dat)~1, p=2, d=1, sma=T,
+ sfreq=12, innov.outlier=T)
```

`S+FinMetrics` also provides a function `outliers` to extract the information of the detected outliers from an “`arima.rob`” object. The object returned by `outliers` is of class “`outliers`”. The methods `print` and `summary` are available for an “`outliers`” object. For example,

```
> summary(outliers(frip.nrr))

 Number of outliers detected:  10

Outliers detected:

       |Time    |Type   |Impact  |t-value|
-------+--------+-------+--------+-------+
1      |Mar 1963|AO     |-0.1438 |13.58  |
-------+--------+-------+--------+-------+
2      |May 1963|LS     | 0.03988| 3.545 |
-------+--------+-------+--------+-------+
3      |May 1968|AO     |-0.3952 |38.67  |
-------+--------+-------+--------+-------+
4      |Jun 1968|AO     |-0.1519 |14.27  |
-------+--------+-------+--------+-------+
5      |Sep 1968|AO     |-0.04653| 4.615 |
-------+--------+-------+--------+-------+
6      |Apr 1969|LS     | 0.04602| 4.005 |
-------+--------+-------+--------+-------+
7      |Sep 1974|LS     |-0.04247| 3.739 |
-------+--------+-------+--------+-------+
8      |Nov 1974|LS     |-0.04914| 4.24  |
-------+--------+-------+--------+-------+
9      |Sep 1976|AO     | 0.038  | 3.891 |
-------+--------+-------+--------+-------+
10     |Apr 1986|AO     | 0.03792| 3.946 |
-------+--------+-------+--------+-------+

Innovation scale estimate before correcting outliers:
  0.01304

Innovation scale estimate after correcting outliers:
  0.01199
```

In this case, still no innovation outlier is detected even though we allowed for innovation outliers.

17.5.3 Iterating the Procedure

After the outlier detection, one can clean the original series of additive outliers and level shifts. If all the outliers in the data have been detected, and `arima.rob` is called on the cleaned data again, one should not find any new outliers. By this line of argument, the process of robust estimation and outlier detection can be iterated to obtain a more thorough detection of outliers. Before illustrating how this can be done using `arima.rob` function, we want to warn that this procedure is *ad hoc*, and sometimes the results may not be easily interpretable.

To carry out the iteration process, simply set the optional argument `iter=T` when calling `arima.rob` function. For example,

```
> frip.irr = arima.rob(log(frip.dat)~1, p=2, d=1, iter=T)
> summary(frip.irr)

Call:
arima.rob(formula = log(frip.dat) ~ 1, p = 2, d = 1, iter = T)
```

```
Regression model:
 log(frip.dat) ~ 1

ARIMA model:
Ordinary differences: 1 ; AR order: 2 ; MA order: 0

Regression Coefficients:
             Value Std. Error t value Pr(>|t|)
(Intercept) 0.0023 0.0005     4.6027  0.0000

AR Coefficients:
        Value Std. Error t value Pr(>|t|)
AR(1) -0.2861  0.0577    -4.9542  0.0000
AR(2) -0.0728  0.0577    -1.2608  0.2082

Degrees of freedom: 360 total; 356 residual

Innovations standard deviation: 0.01178

 Number of outliers detected:  10

Outliers detected:

       |Time    |Type   |Impact  |t-value|
-------+--------+-------+--------+-------+
1      |Mar 1963|AO     |-0.1457 |13.76  |
-------+--------+-------+--------+-------+
2      |May 1968|AO     |-0.3978 |38.1   |
-------+--------+-------+--------+-------+
3      |Jun 1968|AO     |-0.1541 |14.55  |
-------+--------+-------+--------+-------+
4      |Sep 1968|AO     |-0.04516| 4.41  |
-------+--------+-------+--------+-------+
5      |Apr 1969|LS     | 0.04511| 3.814 |
-------+--------+-------+--------+-------+
6      |Sep 1974|LS     |-0.04351| 3.767 |
-------+--------+-------+--------+-------+
7      |Sep 1974|LS     | 0.04162| 3.598 |
-------+--------+-------+--------+-------+
8      |Nov 1974|LS     |-0.04844| 4.092 |
-------+--------+-------+--------+-------+
9      |Dec 1975|LS     | 0.04037| 3.534 |
-------+--------+-------+--------+-------+
10     |Dec 1975|LS     | 0.0414 | 3.619 |
-------+--------+-------+--------+-------+
11     |Sep 1976|AO     | 0.0382 | 3.829 |
-------+--------+-------+--------+-------+
12     |Apr 1986|AO     | 0.03935| 3.932 |
-------+--------+-------+--------+-------+

Innovation scale estimate before correcting outliers:
  0.01311

Innovation scale estimate after correcting outliers:
  0.01176
```

In the first section of the output, the parameter estimates are from the last iterated model. In the second section of the output, it is stated that 10 outliers have been detected altogether, though there are 12 outliers listed in the table. The difference comes from the fact that some outliers are detected repeatedly during the iteration process. For example, two level shifts have been detected corresponding to September 1974.

To obtain a summary of the outliers detected for each iteration, one can use the `outliers` function with an `iter` argument. For example,

```
> summary(outliers(frip.irr, iter=2))

  Number of outliers detected:  2

Outliers detected:

       |Time    |Type   |Impact |t-value|
-------+--------+-------+-------+-------+
1      |Sep 1974|LS     |0.04162|3.598  |
-------+--------+-------+-------+-------+
2      |Dec 1975|LS     |0.04037|3.534  |
-------+--------+-------+-------+-------+

Innovation scale estimate before correcting outliers:
  0.01202

Innovation scale estimate after correcting outliers:
  0.01176
```

which summarizes the outliers detected in the second iteration.

17.6 Algorithms of Filtered τ-Estimation

This section briefly introduces filtered τ-estimates for REGARIMA models. The technical details can be found in the references cited in this chapter.

17.6.1 Classical Maximum Likelihood Estimates

For the REGARIMA model in equations (17.1) and (17.2), the model parameters are usually estimated by maximum likelihood estimation (MLE). The MLE can be computed using prediction error decomposition, for example, see Chapter 14.

First, let $d_0 = d + sD$. Note that we will lose the first d_0 observations because of the ARIMA differencing and/or seasonal differencing. For the moment, consider only equation (17.2). Let

$$\hat{\epsilon}_{t|t-1}(\boldsymbol{\lambda}) = \mathrm{E}_{\boldsymbol{\lambda}}[\epsilon_t|\epsilon_1, \dots, \epsilon_{t-1}], \text{ for } t \geq d_0$$

be the one-step-ahead predictor of ϵ_t given the knowledge of historic values of ϵ_t. Then

$$\hat{u}_t(\boldsymbol{\lambda}) = \epsilon_t - \hat{\epsilon}_{t|t-1}(\boldsymbol{\lambda}) \tag{17.3}$$

will be the one-step-ahead prediction error, and the variance of $\hat{u}_t(\lambda)$ is of the form

$$\sigma_t^2(\lambda) = \mathrm{E}_\lambda[\epsilon_t - \hat{\epsilon}_{t|t-1}(\lambda)]^2 = a_t^2(\lambda)\sigma_u^2,$$

where $\lim_{t\to\infty} a_t(\lambda) = 1$.

Second, for the REGARIMA model considered, the prediction error $\hat{u}_t(\boldsymbol{\beta}, \boldsymbol{\lambda})$ can be obtained similarly as in equation (17.3), replacing ϵ_t with $\epsilon_t(\boldsymbol{\beta}) = y_t - \mathbf{x}_t'\boldsymbol{\beta}$.

Now, let $L(\boldsymbol{\beta}, \boldsymbol{\lambda}, \sigma^2)$ be the conditional likelihood function of the sample observations, and let

$$Q(\boldsymbol{\beta}, \boldsymbol{\lambda}) = -2 \operatorname*{argmax}_{\sigma^2} \log L(\boldsymbol{\beta}, \boldsymbol{\lambda}),$$

which is -2 times the log-likelihood concentrated with respect to σ^2. Using prediction error decomposition, it can be easily shown that

$$Q(\boldsymbol{\beta}, \boldsymbol{\lambda}) = \sum_{t=d_0+1}^{T} \log a_t^2(\boldsymbol{\lambda}) + (T - d_0)s^2\left(\frac{\hat{u}_{d_0+1}(\boldsymbol{\beta}, \boldsymbol{\lambda})}{a_{d_0+1}(\boldsymbol{\lambda})}, \dots, \frac{\hat{u}_T(\boldsymbol{\beta}, \boldsymbol{\lambda})}{a_T(\boldsymbol{\lambda})}\right), \tag{17.4}$$

up to a constant, where

$$s^2(u_1, \dots, u_T) = \frac{1}{n}\sum_{t=1}^{n} u_t^2 \tag{17.5}$$

is the square of the scale estimate.

The classical maximum likelihood estimates of $\boldsymbol{\beta}$ and $\boldsymbol{\lambda}$ are obtained by minimizing $Q(\boldsymbol{\beta}, \boldsymbol{\lambda})$, that is,

$$(\hat{\boldsymbol{\beta}}, \hat{\boldsymbol{\lambda}}) = \operatorname*{argmin}_{\boldsymbol{\beta}, \boldsymbol{\lambda}} Q(\boldsymbol{\beta}, \boldsymbol{\lambda}),$$

and the maximum likelihood estimate of σ^2 is given by

$$\hat{\sigma}^2 = s^2(\hat{\boldsymbol{\beta}}, \hat{\boldsymbol{\lambda}}).$$

17.6.2 *Filtered τ-Estimates*

It is well known that the classical maximum likelihood estimates in the previous section are not robust and can produce poor estimates when the data contain outliers. Bianco, Garcia Ben, Martinez, and Yohai (1996) proposed a class of robust estimates for REGARIMA model called *filtered τ-estimates*. See also Martin and Yohai (1996). These estimates are based on a robustification of the log-likelihood function. The robustification is accomplished through two steps: (1) use the filtered prediction error instead of the usual prediction error; (2) use a robust τ-estimate of the scale in equation (17.4).

The filtered τ-estimation uses a robust filter proposed by Masreliesz (1975) which eliminates the influence of previous outliers or bad observations. That is, the robust prediction error $\tilde{\epsilon}_{t|t-1}$ is computed based on cleaned series $\tilde{\epsilon}_{t|t}$ instead of the contaminated series ϵ_t. For an AR(1) model, the two series $\tilde{\epsilon}_{t|t-1}$ and $\tilde{\epsilon}_{t|t}$ are obtained simultaneously by a recursion procedure as follow:

$$\tilde{\epsilon}_{t|t} = w_t\epsilon_t + (1 - w_t)\tilde{\epsilon}_{t|t-1},$$

where

$$w_t = w(\frac{|\epsilon_t - \tilde{\epsilon}_{t|t-1}|}{m\hat{\sigma}_t})$$

and $w(\cdot)$ is an even and non-increasing weight function, m is a tuning constant, and $\hat{\sigma}_t^2$ is an estimate of the prediction variance σ_t^2. For the general case the robust filtering procedure is based on the state space representation of the ARIMA model. The details can be found in Martin, Samarov, and Vandaele (1983).

The filtered τ-estimation replaces the statistic s^2 in equation (17.5) with a robust τ-estimate of scale. For details of how τ-estimates of scale can be computed, see Yohai and Zamar (1983).

In summary, the filtered τ-estimates are defined by

$$(\hat{\boldsymbol{\beta}}, \hat{\boldsymbol{\lambda}}) = \underset{\boldsymbol{\beta},\boldsymbol{\lambda}}{\operatorname{argmin}}\, Q^*(\boldsymbol{\beta}, \boldsymbol{\lambda}),$$

where

$$Q^*(\boldsymbol{\beta}, \boldsymbol{\lambda}) = \sum_{t=d_0+1}^{T} \log a_t^2(\boldsymbol{\lambda}) + (T - d_0)\tau^2(\frac{\tilde{u}_{d_0+1}(\boldsymbol{\beta}, \boldsymbol{\lambda})}{a_{d_0+1}(\boldsymbol{\lambda})}, \dots, \frac{\tilde{u}_T(\boldsymbol{\beta}, \boldsymbol{\lambda})}{a_T(\boldsymbol{\lambda})}),$$

with $\tilde{u}_t = \epsilon_t - \tilde{\epsilon}_{t|t-1}$, and $\tau^2(\cdot)$ is the square of the τ-estimate of scale.

17.7 References

Bianco, A., Garcia Ben, M., Martinez, E., and Yohai, V. (1996). "Robust Procedure for Regression Models with ARIMA Errors," in A.

Prat (ed.) *COMPSTAT 96 Proceedings Computational Statistics*. Physica-Verlag, Heidelberg.

Bianco, A., Garcia Ben, M., Martinez, E., and Yohai, V. (2001). "Outlier Detection in Regression Rodels with ARIMA Errors Using Robust Estimates," *Journal of Forecasting*, 20, 565-579.

Box, G., and Jenkins, G. (1976). *Time Series Analysis: Forecasting and Control*. Holden-Day, San Francisco.

Chang, I., Tiao, G. C., and Chen. C (1988). "Estimation of Time Series Parameters in the Presence of Outliers," *Technometrics*, 30, 193-204.

Martin, R. D., Samarov, A., and Vandaele, W. (1983). "Robust Methods for ARIMA Models," in A. Zellner (ed.) *Applied Time Series Analysis of Economic Data*. U.S. Census Bureau, Government Printing Office.

Martin, R.D., and V. J. Yohai (1996). "Highly Robust Estimation of Autoregressive Integrated Time Series Models," Publicaciones Previas No. 89, Facultad de Ciencias Exactas y Naturales, Universidad de Buenos Aires.

Masreliesz, C. J. (1975). "Approximate non-Gaussian Filtering with Linear State and Observation Relations," *IEEE Transactions on Automatic Control*, AC-20, 107-110.

Tsay, R. S. (1988). "Outliers, Level Shifts and Variance Changes in Time Series," *Journal of Forecasting*, 7, 1-20.

Yohai, V. J., and Zamar, R. H. (1988). "High Breakdown-Point Estimates of Regression by Means of the Minimization of an Efficient Scale," *Journal of the American Statistical Association*, 83, 406-413.

18
Nonlinear Time Series Models

18.1 Introduction

Most of the time series models discussed in the previous chapters are linear time series models. Although they remain at the forefront of academic and applied research, it has often been found that simple linear time series models usually leave certain aspects of economic and financial data unexplained. Since economic and financial systems are known to go through both structural and behavioral changes, it is reasonable to assume that different time series models may be required to explain the empirical data at different times. This chapter introduces some popular nonlinear time series models that have been found to be effective at modeling nonlinear behavior in economic and financial time series data.

To model nonlinear behavior in economic and financial time series, it seems natural to allow for the existence of different *states of the world* or *regimes* and to allow the dynamics to be different in different regimes. This chapter focuses on models that assume in each regime that the dynamic behavior of the time series is determined by an autoregressive (AR) model, such as threshold AR, self-exciting threshold AR, and smooth transition AR models. This is because simple AR models are arguably the most popular time series model and are easily estimated using regression methods. By extending AR models to allow for nonlinear behavior, the resulting nonlinear models are easy to understand and interpret. In addition, this chapter also covers more general Markov switching models using state space representations. The types of model that can be cast into this form are enormous.

However, there are many other types of nonlinear time series model that are not covered in this chapter, such as bilinear models, k nearest neighbor methods, and neural network models.[1] Book length treatment of nonlinear time series models can be found in Tong (1990), Granger and Teräsvirta (1993), and Franses and van Dijk (2000). Kim and Nelson (1999) provides a comprehensive account of different Markov switching models that have been used in economic and financial research.

Given the wide range of nonlinear time series models available and the inherent flexibility of these models, the possibility of getting a spuriously good fit to any time series data set is very high. Therefore, it is usually recommended to perform a test of linearity against nonlinearity before building a possibly complex nonlinear model. Section 18.2 first introduces a popular test for nonlinearity, the BDS test, which has been found to have power against a wide range of nonlinear time series models. There are many other types of nonlinearity test that are developed to test against specific nonlinear models. Some of these tests will be introduced together with the nonlinear models in later sections. For example, Section 18.3 introduces threshold AR models and two tests for threshold nonlinearity, and Section 18.4 introduces smooth transition AR (STAR) models and a test for STAR nonlinearity. Finally, Section 18.5 describes the Markov switching state space models and Section 18.6 gives an extended example of how to estimate Markov switching models in `S+FinMetrics`.

18.2 BDS Test for Nonlinearity

The BDS test developed by Brock, Dechert, and Scheinkman (1987) (and later published as Brock, Dechert, Scheinkman, and LeBaron, 1996) is arguably the most popular test for nonlinearity. It was originally designed to test for the null hypothesis of independent and identical distribution (iid) for the purpose of detecting non random chaotic dynamics.[2] However, many studies have shown that the BDS test has power against a wide range of linear and nonlinear alternatives; for example, see Brock, Hsieh, and LeBaron (1991) and Barnett, Gallant, Hinich, Jungeilges, Kaplan, and Jensen (1997). In addition, it can also be used as a portmanteau test or misspecification test when applied to the residuals from a fitted model. In particular, when applied to the residuals from a fitted *linear* time series model, the BDS test can be used to detect remaining dependence and the presence of an omitted nonlinear structure. If the null hypothesis cannot be rejected, then the original linear model cannot be rejected; if the null

[1]A function to estimate single-hidden-layer neural network models is in the `nnet` library provided with S-PLUS.

[2]Loosely speaking, a time series is said to be “chaotic” if it follows a nonlinear deterministic process but looks random.

hypothesis is rejected, the fitted linear model is misspecified, and in this sense, it can also be treated as a test for nonlinearity.

18.2.1 BDS Test Statistic

The main concept behind the BDS test is the *correlation integral*, which is a measure of the frequency with which temporal patterns are repeated in the data. Consider a time series x_t for $t = 1, 2, \ldots, T$ and define its m-history as $x_t^m = (x_t, x_{t-1}, \ldots, x_{t-m+1})$. The correlation integral at *embedding dimension* m can be estimated by

$$C_{m,\epsilon} = \frac{2}{T_m(T_m - 1)} \sum\sum_{m \le s < t \le T} I(x_t^m, x_s^m; \epsilon) \tag{18.1}$$

where $T_m = T - m + 1$ and $I(x_t^m, x_s^m; \epsilon)$ is an indicator function that is equal to one if $|x_{t-i} - x_{s-i}| < \epsilon$ for $i = 0, 1, \ldots, m-1$ and zero otherwise. Intuitively, the correlation integral estimates the probability that any two m-dimensional points are within a distance of ϵ of each other; that is, it estimates the joint probability:

$$\Pr(|x_t - x_s| < \epsilon,\ |x_{t-1} - x_{s-1}| < \epsilon,\ \ldots,\ |x_{t-m+1} < x_{s-m+1}| < \epsilon)$$

If x_t are iid, this probability should be equal to the following in the limiting case:

$$C_{1,\epsilon}^m = \Pr(|x_t - x_s| < \epsilon)^m$$

Brock, Dechert, Scheinkman, and LeBaron (1996) define the *BDS statistic* as follows:

$$V_{m,\epsilon} = \sqrt{T}\frac{C_{m,\epsilon} - C_{1,\epsilon}^m}{s_{m,\epsilon}} \tag{18.2}$$

where $s_{m,\epsilon}$ is the standard deviation of $\sqrt{T}(C_{m,\epsilon} - C_{1,\epsilon}^m)$ and can be estimated consistently as documented by Brock, Dechert, Scheinkman, and LeBaron (1997). Under fairly moderate regularity conditions, the BDS statistic converges in distribution to $N(0, 1)$

$$V_{m,\epsilon} \xrightarrow{d} N(0, 1) \tag{18.3}$$

so the null hypothesis of iid is rejected at the 5% significance level whenever $|V_{m,\epsilon}| > 1.96$.

18.2.2 Size of BDS Test

`S+FinMetrics` provides the `BDSTest` function for performing the BDS test.[3] The arguments expected by `BDSTest` function are:

[3]The `BDSTest` function is implemented using the C source file provided by LeBaron (1997). The same test can also be performed by calling `nonlinearTest` function with the optional argument `method` set to `"BDS"`.

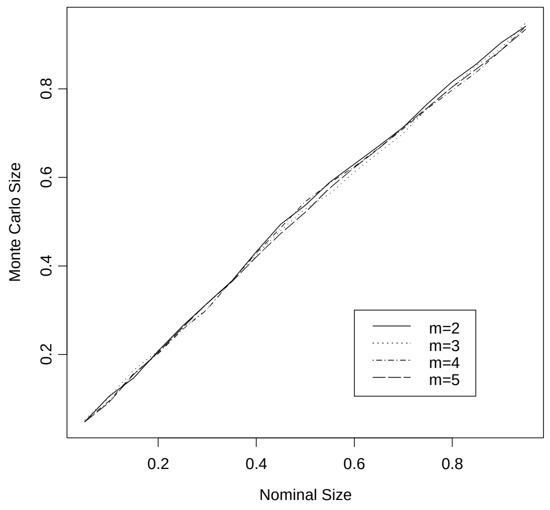


FIGURE 18.1. Size of BDS test statistics using t distribution.

```
> args(BDSTest)
function(x, m = 3, eps = NULL, variable.removal = T)
```

where `x` specifies the time series to be tested, `m` instructs the test to use the embedding dimensions from `2` to `m`, and `eps` specifies in units of sample standard deviations the distance threshold ϵ in (18.1). By default, `BDSTest` computes the BDS statistics with ϵ set to 0.5, 1, 1.5, and 2 standard deviations of the data set. When the optional argument `variable.removal` is set to `TRUE`, different numbers of points in the sample are removed for different values of m such that the test is always computed using all of the sample observations available; if it is set to `FALSE`, the same points are removed for different values of m such that the test is always computed using the same sample observations.

Example 116 *Size of BDS test*

The following script illustrates how to use the `BDSTest` function in a Monte Carlo experiment to investigate the size of the BDS test:

```
set.seed(10)
size.mat = matrix(0, 1000, 4)
for (i in 1:1000) {
  if (i %% 100 == 0) {
    cat("i =", i, "\n")
  }
  test.dat = rt(500, df=8)
  size.mat[i,] = BDSTest(test.dat, m=5, eps=1)$stat[,1]
}
```

One advantage of the BDS test is that it is a statistic that requires no distributional assumption on the data to be tested. In fact, in the above Monte Carlo experiment, the data are simulated from a t distribution with 8 degrees of freedom. Each simulated sample has 500 observations, which is usually thought to be the minimal sample size for the BDS test to have reliable performance. The data are simulated 1000 times and BDS statistics using embedding dimensions from 2 to 5 are computed by setting ϵ to one standard deviation of the sample observations. The following commands plot the size of the "one-sided" test against its nominal value[4]:

```
> size.p = seq(0.05, 0.95, by=0.05)
> size.q = qnorm(size.p)
> size.bds = apply(size.mat, 2,
+      function(x) colMeans(outer(x, size.q, FUN="<=")))
> par(fty="s")
> matplot(matrix(size.p, nrow=length(size.p), ncol=4),
+         size.bds, type="l",
+         xlab="Nominal Size", ylab="Monte Carlo Size")
> legend(0.6, 0.3, paste("m=",2:5,sep=""), type="l", lty=1:4)
```

and the result is shown in Figure 18.1. Considering that the Monte Carlo experiment is conducted using only 1000 replications, the plot shows that the test has very good size behavior for all of the chosen embedding dimensions.

18.2.3 *BDS Test as a Nonlinearity Test and a Misspecification Test*

Another advantage of the BDS test is that when applied to model residuals, the first-order asymptotic distribution of BDS statistic given in (18.3) is independent of estimation errors under certain sufficient conditions. In general, de Lima (1996) shows that for linear additive models, or models that can be transformed into that format, the BDS test is nuisance-parameter-free and does not require any adjustment when applied to fitted model residuals. Thus, the BDS test can be used as a test for nonlinearity or as a test for model misspecification.

Example 117 *Nonlinearity in weekly returns of Dutch Guilder foreign exchange rates*

[4]The BDS test is actually a two-sided test. However, for the purpose of illustrating distributional properties of BDS statistics, the plots are generated using the "incorrect" one-sided test.

The "timeSeries" data set `DFX.ts` in `S+FinMetrics` contains weekly returns on the Dutch Guilder spot exchange rate from January 1980 to December 1998. To test for the existence of nonlinearity in this data set, use the following command:

```
> BDSTest(DFX.ts, m=5)

BDS Test for Independence and Identical Distribution

Null Hypothesis: DFX.ts is independently and identically
distributed.

Embedding dimension =  2 3 4 5

Epsilon for close points =  0.0073 0.0146 0.0219 0.0291

Test Statistics =
      [ 0.01 ] [ 0.01 ] [ 0.02 ] [ 0.03 ]
[ 2 ]   1.0802   1.5908   1.9991   2.6097
[ 3 ]   3.1661   3.0984   3.5817   4.1536
[ 4 ]   4.0523   3.9006   4.4871   5.1613
[ 5 ]   5.2798   4.7189   5.3238   5.9882

p-value =
      [ 0.01 ] [ 0.01 ] [ 0.02 ] [ 0.03 ]
[ 2 ]   0.2801   0.1117   0.0456   0.0091
[ 3 ]   0.0015   0.0019   0.0003   0.0000
[ 4 ]   0.0001   0.0001   0.0000   0.0000
[ 5 ]   0.0000   0.0000   0.0000   0.0000
```

In the above output, the default values of $\epsilon = (0.5, 1.0, 1.5, 2.0)$ used in the test are converted back to the units of the original data, and the null hypothesis that the data are iid is rejected for most combinations of m and ϵ at conventional significance levels. Since there is almost no discernible linear structure in the levels of `DFX.ts`, the results from the BDS test suggest that there may be nonlinear structure in the data.

One possibility to model the nonlinear structure in `DFX.ts` is to use a GARCH(1,1) model:

```
> DFX.garch = garch(DFX.ts~1, ~garch(1,1), trace=F)
> summary(DFX.garch)$coef
                Value     Std.Error    t value      Pr(>|t|)
       C 0.00021084425 3.939145e-004  0.5352539 5.925817e-001
       A 0.00001942582 5.508377e-006  3.5265964 4.381551e-004
 ARCH(1) 0.10297320531 2.096693e-002  4.9112210 1.041116e-006
GARCH(1) 0.80686268689 3.798031e-002 21.2442379 0.000000e+000
```

All of the estimated parameters in DFX.garch are highly significant except for the conditional mean parameter C. To evaluate if the GARCH(1,1) model adequately captures the nonlinear structure in DFX.ts, the BDS test can be used again on the standardized residuals of DFX.garch as a misspecification test. There are two ways to apply the BDS test to GARCH standardized residuals: One is to apply the BDS test directly to the standardized residuals:

```
> BDSTest(residuals(DFX.garch, standard=T), m=5,
+ eps=c(0.5, 1, 1.5))

BDS Test for Independence and Identical Distribution

Null Hypothesis: residuals(DFX.garch, standard = T) is
independently and identically distributed.

Embedding dimension =  2 3 4 5

Epsilon for close points =  0.5002 1.0004 1.5006

Test Statistics =
      [ 0.5 ]   [ 1 ] [ 1.5 ]
[ 2 ] -1.9487 -1.5430 -1.6035
[ 3 ] -1.4581 -1.1172 -1.2687
[ 4 ] -1.2832 -0.9735 -1.1355
[ 5 ] -0.8634 -0.6079 -0.8305

p-value =
      [ 0.5 ]  [ 1 ] [ 1.5 ]
[ 2 ]  0.0513 0.1228  0.1088
[ 3 ]  0.1448 0.2639  0.2045
[ 4 ]  0.1994 0.3303  0.2561
[ 5 ]  0.3879 0.5432  0.4062
```

and the other is to apply it to the logarithms of squared standardized residuals[5]:

```
> BDSTest(log(residuals(DFX.garch, standard=T)^2),
+ m=5, eps=c(0.5, 1, 1.5))

BDS Test for Independence and Identical Distribution
```

[5]When the BDSTest function is applied to a fitted model object, it is currently always applied to the residuals of the fittd model, instead of standardized residuals or logarithms of squared standardized residuals.

```
Null Hypothesis: log(residuals(DFX.garch, standard = T)^2)
is independently and identically distributed.

Embedding dimension =  2 3 4 5

Epsilon for close points =  1.1218 2.2435 3.3653

Test Statistics =
      [ 1.12 ] [ 2.24 ] [ 3.37 ]
[ 2 ]  -0.6461  -0.5538  -0.5463
[ 3 ]  -0.8508  -0.9030  -0.9175
[ 4 ]  -0.7540  -0.9977  -1.0821
[ 5 ]  -0.9397  -0.8581  -1.0252

p-value =
      [ 1.12 ] [ 2.24 ] [ 3.37 ]
[ 2 ]   0.5182   0.5797   0.5849
[ 3 ]   0.3949   0.3665   0.3589
[ 4 ]   0.4509   0.3184   0.2792
[ 5 ]   0.3474   0.3909   0.3052
```

Here, both ways of applying the BDS test suggest that the GARCH(1,1) model provides an adequate fit to the original data and successfully removes the nonlinearity in the data. In general, when applied to standardized residuals from a fitted GARCH model, earlier studies (e.g., see Brock, Hsieh, and LeBaron, 1991) suggested that the BDS statistic needs to be adjusted to have the right size and Monte Carlo simulations are usually relied upon to derive the adjustment factor for specific GARCH models. However, following suggestions in Brock and Potter (1993) and de Lima (1996), recent studies (e.g., see Caporale, Ntantamis, Pantelidis, and Pittis, 2004 and Fernandes and Preumont, 2002) showed that if applied to the logarithms of squared standardized residuals from a fitted GARCH model, the BDS test actually has correct size, because the logarithmic transformation casts the GARCH model into a linear additive model that satisfies the conditions in de Lima (1996) for the BDS test to be nuisance-parameter-free.[6]

Example 118 *Size of BDS misspecification test for GARCH models*

The following script performs a Monte Carlo experiment to illustrate the different size behaviors of the BDS test when applied to standardized residuals and logarithms of squared standardized residuals for the GARCH(1,1) model. The data sets are simulated using the GARCH fit in `DFX.garch` with

[6]Since GARCH models with leverage effects cannot be transformed into a linear additive model, BDS test may not have good size behavior for those models.

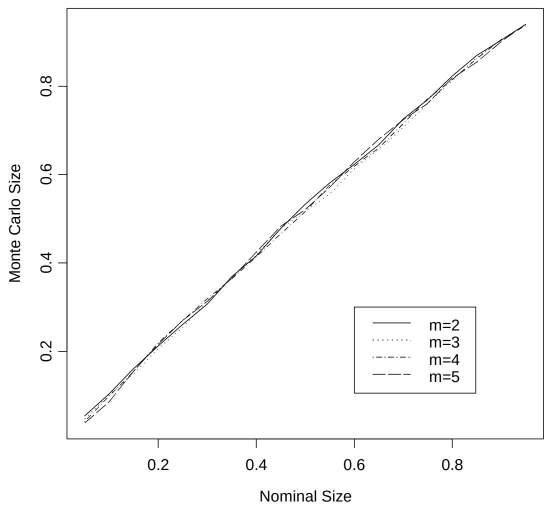


FIGURE 18.2. Size of the BDS test when applied to logarithms of squared standardized GARCH residuals.

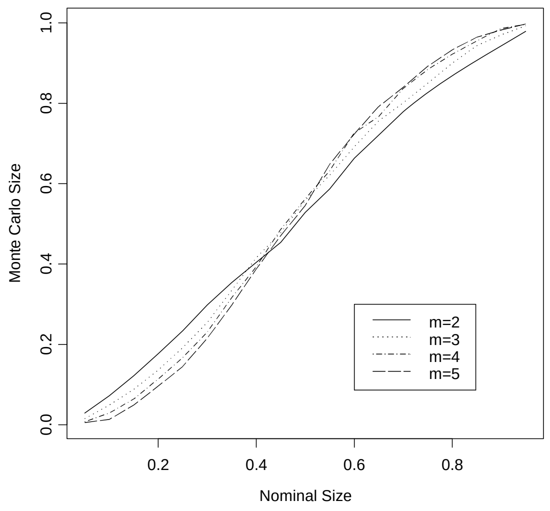


FIGURE 18.3. Size of the BDS test when applied to standardized GARCH residuals.

1000 observations. The GARCH estimation and BDS test are repeated 1000 times.

```
set.seed(10)
sim.garch.dat = simulate(DFX.garch, sigma=F, n.start=500,
    n=1000, n.rep=1000)
size.garch.res = matrix(0, 1000, 4)
size.garch.log = matrix(0, 1000, 4)
for (i in 1:1000) {
  tmp = garch(sim.garch.dat[,i]~1, ~garch(1,1), trace=F)
  if (i %% 10 == 0)
    cat("Simulation No.", i, "\n")
  tmp.res = residuals(tmp, standardized=T)
  size.garch.res[i,] = BDSTest(tmp.res, m=5, eps=1)$stat[,1]
  size.garch.log[i,] = BDSTest(log(tmp.res^2), m=5,
                              eps=1)$stat[,1]
}

size.p = seq(0.05, 0.95, by=0.05)
size.q = qnorm(size.p)
size.garch.res = apply(size.garch.res, 2,
    function(x) colMeans(outer(x, size.q, FUN="<=")))
size.garch.log = apply(size.garch.log, 2,
    function(x) colMeans(outer(x, size.q, FUN="<=")))
```

As in Example 116, the sizes of the “one-sided” test applied to the standardized residuals and the logarithms of squared standardized residuals are plotted against the nominal sizes in Figure 18.3 and Figure 18.2, respectively. Obviously the sizes of the BDS test computed using standardized residuals are off and become more conservative for larger values of m, but those using logarithms of squared standardized residuals are reliable.

18.3 Threshold Autoregressive Models

As discussed in the previous subsection, when there is no prior knowledge about the type of nonlinearity a time series may have, the BDS test can be used to test for the existence of nonlinearity in either the time series itself or the residuals from a fitted linear time series model. However, sometimes economic or financial theory, or even stylized empirical facts, may suggest a specific form of nonlinearity for a time series. In these cases, it is usually preferred to perform the test for the specific form of nonlinearity and build a nonlinear time series model for the form of nonlinearity detected.

One popular class of nonlinear time series models is the *threshold autoregressive* (TAR) models, which was probably first proposed by Tong (1978) and discussed in detail in Tong (1990). The TAR models are simple and easy to understand, but rich enough to generate complex nonlinear dynamics. For example, it can be shown that the TAR models can have limit cycles and thus be used to model periodic time series or produce asymmetries and jump phenomena that cannot be captured by a linear time series model.

In spite of the simplicity of the TAR model form, there are many free parameters to estimate and variables to choose when building a TAR model, and this has hindered its early use. Recently, however, much progress has been made with regard to specification and estimation of TAR models. The next subsection introduces the general form of TAR models and a special class called SETAR models and then illustrates how to perform tests for threshold nonlinearity and estimate unknown parameters in TAR models using ready-to-use functions in `S+FinMetrics`.

18.3.1 TAR and SETAR Models

Consider a simple AR(p) model for a time series[7] y_t:

$$y_t = \mu + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \cdots + \phi_p y_{t-p} + \sigma\epsilon_t \tag{18.4}$$

where ϕ_i $(i = 1, 2, \ldots, p)$ are the AR coefficients, $\epsilon_t \sim WN(0, 1)$, and $\sigma > 0$ is the standard deviation of disturbance term. The model parameters $\boldsymbol{\phi} = (\mu, \phi_1, \phi_2, \ldots, \phi_p)$ and σ are independent of time t and remain constant. To capture nonlinear dynamics, TAR models allow the model parameters to change according to the value of a weakly exogenous *threshold variable* z_t:

$$y_t = \mathbf{X}_t \boldsymbol{\phi}^{(j)} + \sigma^{(j)}\epsilon_t \quad \text{if } r_{j-1} < z_t \leq r_j \tag{18.5}$$

where $\mathbf{X}_t = (1, y_{t-1}, y_{t-2}, \ldots, y_{t-p})$, $j = 1, 2, \cdots, k$, and $-\infty = r_0 < r_1 < \ldots < r_k = \infty$. In essence, the $k-1$ nontrivial *thresholds* $(r_1, r_2, \ldots, r_{k-1})$ divide the domain of the threshold variable z_t into k different regimes. In each different regime, the time series y_t follows a different AR(p) model.[8]

When the threshold variable $z_t = y_{t-d}$, with the *delay parameter d* being a positive integer, the dynamics or regime of y_t is determined by its own lagged value y_{t-d} and the TAR model is called a *self-exciting* TAR, or SETAR model. For the ease of notation, let SETAR(1) denote the one-regime linear AR model with $k = 1$, SETAR(2) denote the two-regime TAR model with $k = 2$, etc. For the one-regime SETAR(1) model, $-\infty = r_0 < r_1 = \infty$ and the unknown parameters are $\boldsymbol{\Theta} = (\boldsymbol{\phi}^{(1)}, \sigma^{(1)})$; for the

[7]See Chapter 3 and the references therein for basic concepts in linear time series analysis.

[8]Although the AR order p is assumed to be the same in different regimes throughout this chapter and in the related `S+FinMetrics` functions for the ease of illustration and programming, in theory the AR order can be different for different regimes.

two-regime SETAR(2) model, the unknown parameters include the single threshold $-\infty < r_1 < \infty$ and $\mathbf{\Theta} = (\boldsymbol{\phi}^{(1)}, \boldsymbol{\phi}^{(2)}, \sigma^{(1)}, \sigma^{(2)})$.

The next subsection introduces two approaches for testing threshold nonlinearity and estimating the unknown parameters in the associated SETAR models, following Tsay (1989) and Hansen (1997), respectively. Although the illustrations and examples focus on SETAR models, the theory and procedures can also be applied to TAR models in general. Finally, note that if only the intercept terms $\mu^{(j)}$ are different in different regimes, SETAR models can be used to capture *level shifts* in y_t; if only the variance terms $\sigma^{(j)}$ are different in different regimes, SETAR models can be used to capture *additive outliers* or *innovation outliers* in y_t. Chapter 17 provides a more comprehensive approach for analyzing time series models that are robust to level shifts and outliers.

18.3.2 Tsay's Approach

Before developing a SETAR model, it is preferred to test for the existence of threshold-type nonlinearity in the time series first. The null hypothesis is that usually the time series y_t follows the SETAR(1) model, whereas the alternative hypothesis is that y_t follows a SETAR(j) model with $j > 1$. One complicating issue in testing for threshold nonlinearity is that the thresholds r_i for $i = 1, 2, \cdots, k-1$ are only identified under the alternative hypothesis. To avoid dealing with the thresholds directly, Tsay (1989) proposed a conventional F test based on an auxiliary regression.

Arranged Autoregression and Tsay's F Test

Tsay's approach centers on the use of an *arranged autoregression* with recursive least squares (RLS) estimation. Consider the SETAR model in (18.5) with $z_t = y_{t-d}$. Since the threshold values r_i are usually unknown, Tsay suggested arranging the equations in (18.5) for $t = \max(d, p) + 1, \ldots, n$, where n is the sample size, such that the equations are sorted according to the threshold variable y_{t-d}, which may take any value in $\mathbf{Y}_d = (y_h, \ldots, y_{n-d})$ with $h = \max(1, p + 1 - d)$:

$$y_{\pi_i} = \mathbf{X}_{\pi_i} \hat{\boldsymbol{\phi}} + \hat{\sigma}\epsilon_{\pi_i} \tag{18.6}$$

where $i = 1, 2, \ldots, n'$, $n' = n - d - h + 1$ is the effective sample size for the above arranged autoregression, and π_i corresponds to the index in the original sample such that $y_{\pi_i - d}$ is the i-th smallest value in $\mathbf{Y}_d$. For example, if y_{10} is the smallest value in $\mathbf{Y}_d$, then $\pi_1 = 10 + d$; if y_{20} is the second smallest value in $\mathbf{Y}_d$, then $\pi_2 = 20 + d$, etc. So if the original time series is generated by a SETAR(2) model and there are $m < n$ values in $\mathbf{Y}_d$ that are smaller than the threshold r_1, then the first m equations in (18.6) correspond to the first regime and the remaining equations correspond to the second regime.

To test for the existence of threshold-type nonlinearity, Tsay suggested computing RLS estimates of $\hat{\boldsymbol{\phi}}$ in (18.6). If there is no threshold nonlinearity, the standardized predictive residuals $\hat{e}_{\pi_i}$ from the RLS of (18.6) should be white noise asymptotically and orthogonal to $\mathbf{X}_{\pi_i}$. However, if y_t is a SETAR(j) process with $j > 1$, the RLS estimates of $\hat{\boldsymbol{\phi}}$ are biased, and $\hat{\boldsymbol{\Psi}}$ in the following auxiliary regression will be statistically significant:

$$\hat{e}_{\pi_i} = \mathbf{X}'_{\pi_i}\boldsymbol{\Psi} + u_{\pi_i} \tag{18.7}$$

Thus, the conventional F statistic for testing $\boldsymbol{\Psi} = \mathbf{0}$ the above regression can be used as a test for threshold nonlinearity.

Example 119 *SETAR nonlinearity in NASDAQ realized volatility*

To illustrate the usage of Tsay's F test for threshold nonlinearity, consider the weekly realized volatility of the NASDAQ 100 index constructed as follows from the `S+FinMetrics` data set `ndx.dat`:

```
> ndx.ret2 = getReturns(ndx.dat[,"Close"])^2
> ndx.rvol = sqrt(aggregate(ndx.ret2, FUN=sum, by="weeks",
+ week.align=1))
> colIds(ndx.rvol) = "RVOL"
> par(mfrow=c(2,2))
> plot(ndx.rvol, reference.grid=F, main="RVOL")
> plot(log(ndx.rvol), reference.grid=F, main="Log RVOL")
```

The levels and the logarithms of the weekly realized volatility series are shown in the top half of Figure 18.4. The time series plots suggest that the volatility may have switched to a different regime after the first quarter of 2000. Before testing for threshold nonlinearity, the ACF and PACF plots can be used to help identify the AR order to use:

```
> ndx.acf = acf(log(ndx.rvol))
> ndx.pacf = acf(log(ndx.rvol), type="partial")
```

The resulting plots are shown in the bottom half of Figure 18.4. The ACF function decays very slowly and remains significant even after 30 lags, whereas the PACF function is significant for the first 6 lags. This suggests that an AR model with order from 2 to 6 may be considered as a starting point for modeling the logarithms of realized volatility `log(ndx.rvol)`.[9]

The `S+FinMetrics` function `nonlinearTest` can now be used to test for threshold nonlinearity:

[9]Hereafter, the logarithms of `ndx.rvol` are used because usually the logarithms of realized volatility tend to be normally distributed. See Andersen, Bollerslev, Diebold, and Ebens (2001) for a detailed analysis of properties of realized volatility for stock returns.

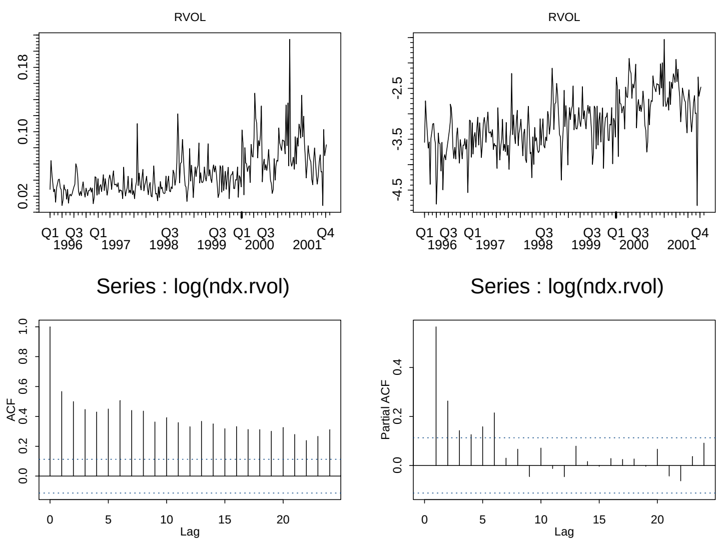


FIGURE 18.4. Weekly realized volatility of the NASDAQ 100 index.

```
> nonlinearTest(log(ndx.rvol), method="threshold", p=6, d=1:6)

Nonlinearity Test: Threshold Nonlinearity

Null Hypothesis: no threshold nonlinearity

    F-stat     dof  P-val
d=1 1.2568 (7,253) 0.2724
d=2 1.4203 (7,253) 0.1974
d=3 1.2586 (7,253) 0.2714
d=4 0.5104 (7,253) 0.8264
d=5 0.5224 (7,253) 0.8173
d=6 0.1179 (7,253) 0.9971
```

Note that the optional argument `p` specifies the AR order to use in the arranged autoregression, and the optional argument `d` is used to select the delay parameters from 1 to 6. The output gives the F statistics and their corresponding p-values for all chosen values of delay parameter d, and it shows that the evidence for threshold nonlinearity is not strong with the AR(6) specification. Since a high-order AR model may actually approximate nonlinear dynamics relatively well, a lower-order AR(2) specification may also be tried:

```
> nonlinearTest(log(ndx.rvol), method="threshold", p=2, d=1:2)
```

```
Nonlinearity Test: Threshold Nonlinearity

Null Hypothesis: no threshold nonlinearity

    F-stat     dof  P-val
d=1 4.4468 (3,265) 0.0046
d=2 4.0010 (3,265) 0.0082
```

Now the null hypothesis of no threshold nonlinearity is actually rejected for both $d = 1$ and $d = 2$ with an AR(2) specification!

Choice of Delay Parameter and Thresholds

After rejecting the null hypothesis of no threshold nonlinearity, one proceeds to the next stage of estimating a SETAR model. Tsay (1989) suggested identifying the delay parameter d and the thresholds r_i for $i = 1, \ldots, k-1$ first, and then used least squares (LS) to estimate the unknown parameters $\boldsymbol{\Theta}$ in (18.5) with given values of d and thresholds. As long as there are enough observations in each regime, the LS estimates are consistent.

For a given AR order p, Tsay suggested choosing the delay parameter d such that

$$d = \underset{v \in S}{\operatorname{argmax}}\, F(p, v)$$

where $F(p, v)$ is the F statistic of the auxiliary regression (18.7) with AR order p and the delay parameter equal to v, and S is a set of values of d to consider. For the NASDAQ realized volatility series, d can be set to 1 according to the nonlinearity test output using this rule.

Tsay (1989) also proposed using two graphical tools for identifying the threshold values: (1) the scatter plot of standardized predictive residuals $\hat{e}_{\pi_i}$ from the arranged autoregression versus the ordered threshold variable; (2) the scatter plot of the t-statistics of the RLS estimates of $\hat{\boldsymbol{\phi}}$ from the arranged autoregression versus the ordered threshold variable. Both plots may exhibit structural breaks at the threshold values. To produce such plots for the nonlinearity test, set the optional argument `save.RLS` to `TRUE` when calling `nonlinearTest`:

```
> ndx.test = nonlinearTest(log(ndx.rvol), method="threshold",
+ p=2, d=1, save.RLS=T)
> names(ndx.test)
[1] "stat"      "df"        "threshold" "residuals"
[4] "tRatios"   "yd"        "method"
```

The returned object `ndx.test` includes the following components: `yd` is the ordered threshold variable, `residuals` is the standardized predictive residuals, and `tRatios` is the t-statistics of RLS estimates of the AR

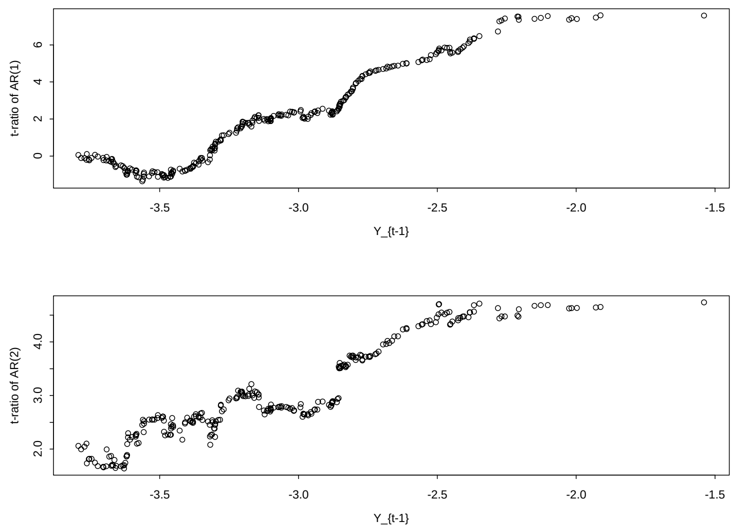


FIGURE 18.5. Scatter plot of t-statistics of RLS estimates of AR coefficients versus ordered threshold variable.

coefficients. To produce the scatter plot of t-statistics versus the ordered threshold variable, for example, use the following commands:

```
> par(mfrow=c(2,1))
> plot(ndx.test$yd, ndx.test$tRatio[,1], xlab="Y_{t-1}",
+      ylab="t-ratio of AR(1)")
> plot(ndx.test$yd, ndx.test$tRatio[,2], xlab="Y_{t-1}",
+      ylab="t-ratio of AR(2)")
```

The plots in Figure 18.5 show that both estimates are significant, with t-statistics greater than 2 in absolute values in most cases. In addition, the trend in the t-statistics seems to have two breaks: One occurs when the threshold variable is around -2.8 and the other occurs when the threshold variable is around -2.4. This suggests a SETAR(3) model with two non-trivial threshold values: $r_1 = -2.8$ and $r_2 = -2.4$.

LS Estimates of SETAR Model

After choosing the delay parameter d and the thresholds, other unknown parameters in $\boldsymbol{\Theta}$ of the SETAR model may be simply estimated by LS using the `S+FinMetrics` function `SETAR`, which takes the following arguments:

```
> args(SETAR)
function(x, threshold, p = 1, d = NULL)
```

where the first argument specifies the data to be used, the second argument gives the vector of threshold values, and the optional arguments `p` and `d` specify the AR order and delay parameter, respectively. To estimate the SETAR(3) model with thresholds $(-2.8, -2.4)$, use the following command:

```
> ndx.setar = SETAR(log(ndx.rvol), c(-2.8, -2.4), p=2, d=1)
> summary(ndx.setar)

Call:
SETAR(x = log(ndx.rvol), threshold = c(-2.8, -2.4), p = 2,
d = 1)

Coefficients:
          regime.1 regime.2 regime.3
Intercept -1.5043  -2.4463  -3.2661
(std.err)  0.2778   1.1323   0.8676
 (t.stat) -5.4157  -2.1605  -3.7643

     lag1  0.2866  -0.0373  -0.6283
(std.err)  0.0776   0.4400   0.3795
 (t.stat)  3.6942  -0.0848  -1.6555

     lag2  0.2573   0.1381   0.2191
(std.err)  0.0687   0.1305   0.1279
 (t.stat)  3.7449   1.0577   1.7138


Std. Errors of Residuals:
 regime.1 regime.2 regime.3
 0.4291   0.3794   0.3583

Information Criteria:
      logL       AIC       BIC        HQ
 -157.5830  333.1659  366.5000  346.5063

                   total regime.1 regime.2 regime.3
Degree of freedom:   300      228       44       19
Time period: from 01/15/1996 to 10/08/2001
```

Note that the AR coefficients for the first regime are estimated to be $(0.29, 0.26)$, which appear to be significant, whereas the AR coefficients for the second and third regimes are estimated to be $(-0.03, 0.14)$ and $(-0.63, 0.22)$, respectively, and are not very significant. The estimated regime indices can be plotted as follows:

```
> plot(timeSeries(ndx.setar$regime,
```

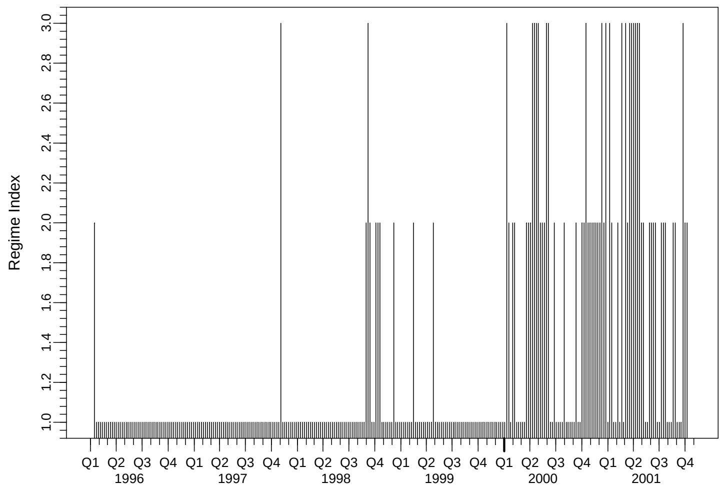


FIGURE 18.6. Estimated regime indices of `ndx.setar`.

```
+ pos=positions(ndx.rvol)[-(1:2)]), reference.grid=F,
+ ylab="Regime Index", plot.args=list(type="h"))
```

and the plot is shown in Figure 18.6. It can be seen that most of the observations prior to 2000 fall into the first regime, and the third regime observations usually follow the second regime observations.

Predictions from SETAR Models

After estimating a SETAR model, sometimes a more important task is to generate forecasts of future values of the time series that is of interest. Predictions from SETAR models can be easily computed using Monte Carlo simulations, by following the same principle used for vector autoregressive forecasting (see Section 11.3 for details). For example, to generate 1-step-ahead to 100-step-ahead forecasts from the fitted model `ndx.setar`, use the following command:

```
> class(ndx.setar)
[1] "SETAR"
> ndx.pred = predict(ndx.setar, n.predict=100, CI.alpha=0.6,
+ n.sim=10000)
```

Note that the fitted object `ndx.setar` has class `"SETAR"`. By calling the generic `predict` function on `"SETAR"` objects, the simulation-based forecasting method implemented in `predict.SETAR` is automatically applied

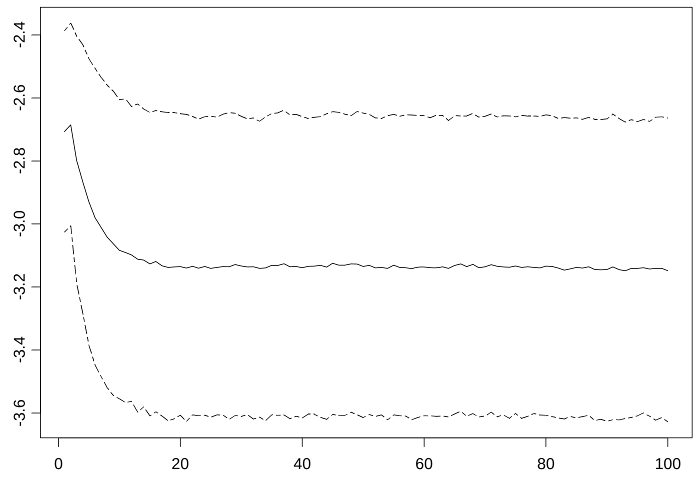


FIGURE 18.7. Predicted realized volatility (in logarithm scale) from `ndx.setar`.

on the `"SETAR"` objects. The optional argument `n.predict` is used to specify the number of forecasts to obtain in the future, the argument `CI.alpha` is used to specify 60% pointwise confidence intervals for the forecasts based on Monte Carlo simulations, and the argument `n.sim` is used to specify the number of simulations to be used for computing the forecasts. The forecasts and their pointwise confidence intervals can be plotted as follows:

```
> tsplot(cbind(ndx.pred$values, ndx.pred$CI), lty=c(1,6,6))
```

and the plot is shown Figure 18.7. After less than 20 steps, the forecasts settle down to the asymptotic mean of the SETAR process.

18.3.3 Hansen's Approach

Although the procedure introduced in the above subsection for identifying and estimating SETAR models is easy to perform, it requires some human decisions, especially for choosing the threshold values. This subsection introduces another test for threshold nonlinearity and another procedure for estimating SETAR models as proposed by Hansen (1997). The advantage of this procedure is that the thresholds can be estimated together with other model parameters and valid confidence intervals can be constructed for the estimated thresholds. The disadvantage is that the current implementation only supports the two-regime SETAR model and thus only one threshold can be estimated.[10]

Hansen's sup-LR Test

Hansen (1997) considered the following two-regime variant of (18.5):

$$y_t = \mathbf{X}_t \boldsymbol{\phi}^{(1)}(1 - I(y_{t-d} > r_1)) + \mathbf{X}_t \boldsymbol{\phi}^{(2)} I(y_{t-d} > r_1) + \epsilon_t \tag{18.8}$$

where $I(A)$ is the indicator function that is equal to 1 if A is true and 0 otherwise, $\epsilon_t \sim$iid$(0, \sigma^2)$, and there is only one nontrivial threshold r_1. As discussed in the previous subsection, if d and r_1 are known, then the model parameters $\boldsymbol{\Theta} = (\boldsymbol{\phi}^{(1)}, \boldsymbol{\phi}^{(2)}, \sigma^2)$ can be estimated by least squares:

$$\hat{\boldsymbol{\Theta}} = \underset{\phi^{(1)},\phi^{(2)}}{\operatorname{argmin}} \ \hat{\sigma}^2(r_1) = \underset{\phi^{(1)},\phi^{(2)}}{\operatorname{argmin}} \ \frac{1}{n'} \sum_{t=h}^{n} \hat{\epsilon}_t^2 \tag{18.9}$$

where $h = \max(1, p + 1 - d)$ and $n' = n - d - h + 1$ is the effective sample size after adjusting for starting values and the delay parameter.

To test the null hypothesis of SETAR(1) against the alternative hypothesis of SETAR(2), the likelihood ratio test assuming normally distributed errors can be used:

$$F(r_1) = \frac{\text{RSS}_0 - \text{RSS}_1}{\hat{\sigma}_1^2(r_1)} = n' \frac{\hat{\sigma}_0^2 - \hat{\sigma}_1^2(r_1)}{\hat{\sigma}_1^2(r_1)} \tag{18.10}$$

where RSS_0 is the residual sum of squares from SETAR(1), RSS_1 is the residual sum of squares from SETAR(2) given the threshold r_1, and $\hat{\sigma}_0^2$ is the residual variance of SETAR(1). The above test is also the standard F test since (18.8) is a linear regression. However, since the threshold r_1 is usually unknown, Hansen (1997) suggested computing the following *sup-LR test*:

$$F_s = \sup_{r_1 \in \mathbf{Y}_d} F(r_1) \tag{18.11}$$

by searching over all of the possible values of the threshold variable y_{t-d}. In practice, to ensure that each regime has a nontrivial proportion of observations, a certain percentage of $\mathbf{Y}_d$ at both ends are usually trimmed and not used.

The sup-LR test has near-optimal power as long as the error term ϵ_t is iid. If ϵ_t is not iid, the F test needs to be replaced by the heteroskedasticity-consistent Wald or Lagrange multiplier test. One complicating issue is that since r_1 is only identified under the alternative, the asymptotic distribution of F_s is not χ^2 and nonstandard. Hansen (1996) showed that the asymptotic

[10]Hansen (1999) has generalized this procedure to SETAR models with more than two regimes.

distribution may be approximated by a bootstrap procedure in general, and Hansen (1997) gave the analytic form of the asymptotic distribution for testing against SETAR(2) models.

The `nonlinearTest` function in `S+FinMetrics` can also be used to produce Hansen's sup-LR test, simply by setting the optional argument `method` to `"sup-LR"`. For example, to test for threshold nonlinearity in weekly realized volatility of the NASDAQ 100 index using the same AR(2) specification and choosing the threshold variable to be $z_t = y_{t-1}$ as in Tsay's F test, use the following command[11]:

```
> nonlinearTest(log(ndx.rvol), method="sup-LR", p=2, d=1,
+ trim.pct=0.1, n.boot=1000)

Nonlinearity Test: Hansen sup-LR Nonlinearity

Null Hypothesis: no threshold with the specified threshold
variable
Under Maintained Assumption of Homoskedastic Errors --

Number of Bootstrap Replications  1000
Trimming percentage               0.1
Threshold Estimate                -2.8768
F-test for no threshold           22.9687
Bootstrap P-Value                 0
```

Note that the optional argument `trim.pct` is used to trim 10% observations at both ends of $\mathbf{Y}_d$, and `n.boot` is used to set the number of bootstrap simulations for computing the p-value of the test. Again, the null hypothesis of no threshold nonlinearity is strongly rejected. To produce the test robust to heteroskedastic errors, simply set the optional argument `hetero` to `TRUE`:

```
> nonlinearTest(log(ndx.rvol), method="sup-LR", p=2, d=1,
+ trim.pct=0.1, n.boot=1000, hetero=T)

Nonlinearity Test: Hansen sup-LR Nonlinearity

Null Hypothesis: no threshold with the specified threshold
variable
Allowing Heteroskedastic Errors using White Correction --

Number of Bootstrap Replications  1000
Trimming percentage               0.1
```

[11]General TAR alternatives with arbitrary threshold variable can also be tested by using setting the optional argument `q` instead of `d`.

```
Threshold Estimate                 -2.8768
F-test for no threshold            18.7357
Bootstrap P-Value                  0
```

Sequential Estimation of SETAR Models

After confirming the existence of threshold nonlinearity, Hansen (1997) suggested estimating the threshold value r_1 together with $\boldsymbol{\phi}$ using LS methods:

$$\hat{r}_1 = \underset{r_1 \in \mathbf{Y}_d}{\operatorname{argmin}}\, \hat{\sigma}^2(r_1, d) \tag{18.12}$$

where $\hat{\sigma}^2(r_1, d)$ is the residual variance of the LS estimate of (18.8) given the threshold r_1 and the delay parameter d. So the threshold value r_1 can be estimated sequentially by searching over the possible values of r_1. If the delay parameter is not known, it can be estimated similarly by expanding the search to another dimension:

$$(\hat{r}_1, \hat{d}) = \underset{r_1, d}{\operatorname{argmin}}\, \hat{\sigma}^2(r_1, d) \tag{18.13}$$

One thing to note is that for the asymptotic inference on SETAR models to work correctly, each regime must have a nontrivial proportion of observations in the limit. Therefore, just as in computing Hansen's sup-LR test, a certain percentage of $\mathbf{Y}_d$ at both ends are usually trimmed and not used when searching for the value of r_1.

The `TAR` function in `S+FinMetrics` implements the above sequential estimation approach.[12] For example, to estimate a two-regime SETAR model with $d = 1$ and AR(2) components, use the following command:

```
> ndx.setar.r = TAR(log(ndx.rvol), p=2, d=1, trim.pct=0.1)
> ndx.setar.r

Call:
TAR(x = log(ndx.rvol), p = 2, d = 1, trim.pct = 0.1)

Coefficients:
          regime.1 regime.2
intercept -2.0356  -1.4614
     lag1  0.1903   0.2183
     lag2  0.2056   0.2435
```

[12] As its name suggests, the `TAR` function actually supports general TAR models, in addition to SETAR models. A general threshold variable can be used by specifying the optional argument `q`. In addition, the `TAR` function also allows for the use of some popular functions of a variable as the threshold variable. See the online help file for `TAR` for details.

```
Std. Errors of Residuals:
 regime.1 regime.2
 0.4233   0.3828

Information Criteria:
      logL       AIC       BIC        HQ
 -155.7369  323.4739  345.6966  332.3674

                   total regime.1 regime.2
Degree of freedom:   300      207       87
Time period: from 01/15/1996 to 10/08/2001
```

Note that the optional argument `trim.pct` is used to set the trimming percentage for $\mathbf{Y}_d$ to 10%. Compared with the three-regime SETAR fit in the previous subsection, this two-regime SETAR model actually gives a better fit in terms of log-likelihood value and Bayesian information criterion (BIC), which is probably due to the fact the threshold value is also optimized in this fit. The estimated threshold value is given as a component in the returned object `ndx.setar.r`:

```
> ndx.setar.r$qhat
[1] -2.876807
```

which is quite close to the first threshold identified using Tsay's t-statistics plot in the previous subsection.

Confidence Interval for the Threshold

Using the generic `summary` function on the fitted model object `ndx.setar.r` displays more details of the model fit:

```
> summary(ndx.setar.r)

Call:
TAR(x = log(ndx.rvol), p = 2, d = 1, trim.pct = 0.1)

Minimized SSE for all threshold variable candidates:
 RVOL.lag1
  49.84288

Threshold estimate for the threshold variable chosen with
smallest minimized SSE:
  CI.lower     point  CI.upper
 -3.826435 -2.876807 -2.828314

Coefficients and standard errors:
          regime.1   (se) regime.2   (se)
```

```
intercept -2.036    0.325 -1.461    0.372
     lag1  0.190    0.103  0.218    0.150
     lag2  0.206    0.073  0.244    0.099

Coefficient confidence intervals:
          regime.1.lower regime.1.upper
intercept -2.700         -1.075
     lag1 -0.020          0.417
     lag2  0.055          0.412

          regime.2.lower regime.2.upper
intercept -2.435         -0.454
lag1      -0.093          0.600
lag2      -0.003          0.472

Std. Errors of Residuals:
 regime.1 regime.2
 0.423    0.383

Information Criteria:
     logL      AIC      BIC       HQ
 -155.737  323.474  345.697  332.367

                   total regime.1 regime.2
Degree of freedom:   300      207       87
Time period: from 01/15/1996 to 10/08/2001
```

Note that standard inference statistics as well as confidence intervals for both the coefficients and the threshold are given. In particular, as proposed by Hansen (1997), an asymptotically valid confidence interval for the threshold is constructed by inverting the likelihood ratio (LR) test for testing the null hypothesis that the threshold is equal to a given value r:

$$\mathrm{LR}(r) = n' \frac{\hat{\sigma}^2(r) - \hat{\sigma}^2(\hat{r}_1)}{\hat{\sigma}^2(\hat{r}_1)} \tag{18.14}$$

The $100\cdot\alpha\%$ confidence interval (CI) for the threshold r_1 is given by the set of values of r for which the above LR test cannot be rejected at significance level $1-\alpha$:

$$\mathrm{CI}(\alpha) = \{r : \ LR(r) \leq \mathrm{Z}_\alpha\} \tag{18.15}$$

where Z_α is the $100\cdot\alpha\%$ quantile of the asymptotic distribution of the LR statistic given in Hansen (1997). A graphical tool to help locate the confidence interval for the threshold is to plot the above LR statistics against different values of r, and choose the region of r close to r_1 where the LR statistics are smaller than the critical value Z_α. The necessary information

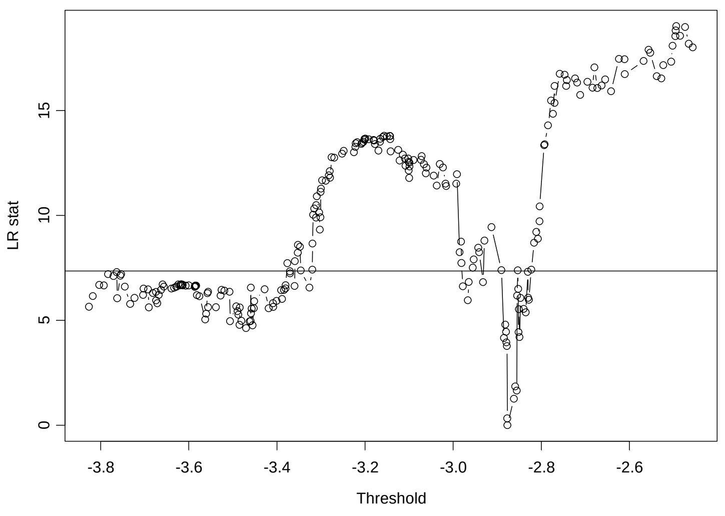


FIGURE 18.8. Confidence interval for threshold value by inverting likelihood ratio statistics.

to generate such a plot is contained in the `LR.q` component of the fitted model object. For example, to produce the plot using the fitted model object `ndx.setar.r`, use the following commands:

```
> names(ndx.setar.r$LR.q)
[1] "LR"        "Threshold" "Critical"
> plot(ndx.setar.r$LR.q$Threshold, ndx.setar.r$LR.q$LR,
+      type="b", xlab="Threshold", ylab="LR stat")
> abline(h=ndx.setar.r$LR.q$Critical)
```

and the plot is shown in Figure 18.8. This plot can also be generated directly and applying the generic `plot` function on the fitted model object `ndx.setar.r`.

Predictions from TAR Models

Just like with SETAR models, predictions from general TAR models can be computed using Monte Carlo simulations, as long as the future values of the threshold variable are known. In fact, the objects returned by the `TAR` function have class `"TAR"`, which inherits from the `"SETAR"` class. For example,

```
> class(ndx.setar.r)
[1] "TAR"
```

```
> inherits(ndx.setar.r, "SETAR")
[1] T
```

Thus, when the generic `predict` function is called on `"TAR"` objects, the simulation-based forecasting procedure in `predict.SETAR` is also used to produce the forecasts. For example, to generate forecasts from the fitted model object `ndx.setar.r`, use the following command:

```
> ndx.pred.2 = predict(ndx.setar.r, n.predict=100,
+ CI.alpha=0.6, n.sim=10000)
```

which are very similar to the forecasts produced earlier using a three-regime model.

18.4 Smooth Transition Autoregressive Models

In the TAR models introduced in the previous section, a regime switch happens when the threshold variable crosses a certain threshold. Although the model can capture many nonlinear features usually observed in economic and financial time series, sometimes it is counterintuitive to suggest that the regime switch is abrupt or discontinuous. Instead, in some cases it is reasonable to assume that the regime switch happens gradually in a smooth fashion. If the discontinuity of the thresholds is replaced by a smooth transition function, TAR models can be generalized to smooth transition autoregressive (STAR) models.

In this section, two main STAR models – logistic STAR and exponential STAR – are introduced. After illustrating how to test for STAR nonlinearity, examples will be given to show how to estimate STAR models in `S+FinMetrics`. A systematic modeling cycle approach for STAR models was proposed by Teräsvirta (1994), and van Dijk, Teräsvirta, and Franses (2002) provided a survey of recent development for STAR models.

18.4.1 Logistic and Exponential STAR Models

In the SETAR model (18.8) considered in the previous section, the observations y_t are generated either from the first regime when y_{t-d} is smaller than the threshold or from the second regime when y_{t-d} is greater than the threshold. If the binary indicator function is replaced by a smooth transition function $0 < G(z_t) < 1$, which depends on a *transition variable* z_t (like the threshold variable in TAR models), the model becomes a *STAR* model:

$$y_t = \mathbf{X}_t \boldsymbol{\phi}^{(1)}(1 - G(z_t)) + \mathbf{X}_t \boldsymbol{\phi}^{(2)} G(z_t) + \epsilon_t \tag{18.16}$$

Now, the observations y_t switch between two regimes smoothly in the sense that the dynamics of y_t may be determined by both regimes, with one

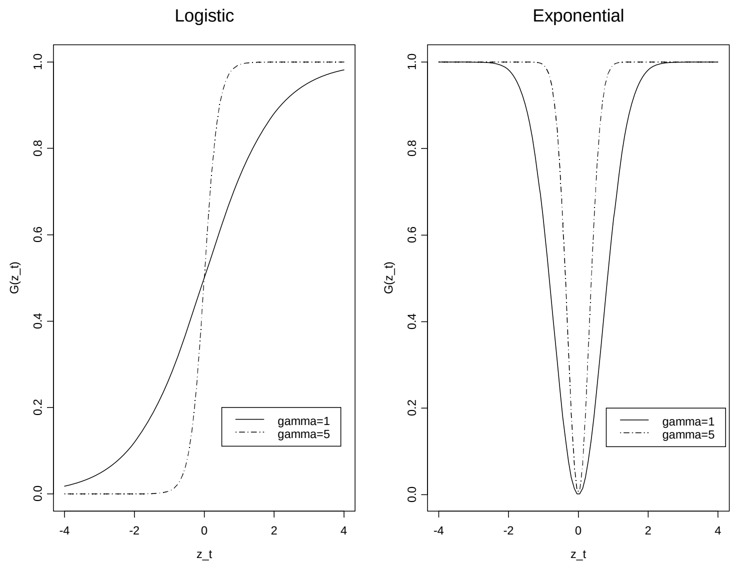


FIGURE 18.9. Logistic and exponential transition functions.

regime having more impacts in some times and the other regime having more impacts in other times. Another interpretation is that STAR models actually allow for a "continuum" of regimes, each associated with a different value of $G(z_t)$.

Two popular choices for the smooth transition function are the *logistic* function and the *exponential* function. Using the logistic function, the transition function can be specified as

$$G(z_t; \gamma, c) = \frac{1}{1 + e^{-\gamma(z_t - c)}}, \ \gamma > 0 \tag{18.17}$$

and the resulting model is referred to as *logistic STAR* or LSTAR model. The parameter c can be interpreted as the threshold, as in TAR models, and γ determines the speed and smoothness of transition. Using the exponential function, the transition function can be specified as

$$G(z_t; \gamma, c) = 1 - e^{-\gamma(z_t - c)^2}, \ \gamma > 0 \tag{18.18}$$

and the resulting model is referred to as *exponential STAR* or ESTAR model. As in LSTAR models, c can be interpreted as the threshold and γ determines the speed and smoothness of transition.

In spite of the similarity between LSTAR and ESTAR models, they actually allow for different types of transitional behavior. To illustrate this point, Figure 18.9 plots the logistic and exponential transition functions with $c = 0$ and $\gamma = 1$ and 5. The following properties can be readily observed:

1. If γ is small, both transition functions switch between 0 and 1 very smoothly and slowly; if γ is large, both transition functions switch between 0 and 1 more quickly.

2. As $\gamma \to \infty$, both transition functions become binary. However, the logistic function approaches the indicator function $I(z_t > c)$ and the LSTAR model reduces to a TAR model; the exponential function approaches the indicator function $I(z_t = c)$ and the model does not nest the TAR model as a special case.

3. The logistic function is monotonic and the LSTAR model switches between two regimes smoothly depending on how much the transition variable z_t is smaller than or greater than the threshold c. The exponential function is symmetrical and the ESTAR model switches between two regimes smoothly depending on how far the transition variable z_t is from the threshold c. For the LSTAR model, both the distance between z_t and c and its sign matter; for the ESTAR model, only the distance between z_t and c matters, but not the sign.

18.4.2 Test for STAR Nonlinearity

Testing for the existence of STAR-type nonlinearity is usually the first step toward building a STAR model. However, just like the test for threshold-type nonlinearity, tests for the null hypothesis of a simple AR model against the alternative of a STAR model have nonstandard asymptotic distributions, because some parameters in the STAR model are not identified under the null hypothesis, such as the AR coefficients $\boldsymbol{\phi}^{(2)}$ in the second regime, the transition parameter γ, and the threshold c.

STAR Nonlinearity Test with Homoskedastic Errors

To avoid complicated issues caused by the unidentified STAR model parameters under the null hypothesis of a linear AR model, Luukkonen, Saikkonen, and Teräsvirta (1988) proposed replacing the transition function $G(z_t; \gamma, c)$ by a suitable Taylor series approximation around $\gamma = 0$. It turns out that if the transition function $G(z_t; \gamma, c)$ in the LSTAR model is replaced by its third-order Taylor series approximation, the LSTAR model in (18.16) can be written as[13]

$$y_t = \mathbf{X}_t \boldsymbol{\beta}_0 + \mathbf{X}_t z_t \boldsymbol{\beta}_1 + \mathbf{X}_t z_t^2 \boldsymbol{\beta}_2 + \mathbf{X}_t z_t^3 \boldsymbol{\beta}_3 + e_t \tag{18.19}$$

[13]See Franses and van Dijk (2000) for details.

where the coefficient vectors $\boldsymbol{\beta}_i$ for $i = 0, 1, 2, 3, 4$ are functions of the original model parameter $\boldsymbol{\phi}$. Similarly, if the transition function $G(z_t; \gamma, c)$ in the ESTAR model is replaced by its second-order Taylor series approximation, the ESTAR model in (18.16) can be written as

$$y_t = \mathbf{X}_t\boldsymbol{\beta}_0 + \mathbf{X}_t z_t\boldsymbol{\beta}_1 + \mathbf{X}_t z_t^2\boldsymbol{\beta}_2 + \mathbf{X}_t z_t^3\boldsymbol{\beta}_3 + \mathbf{X}_t z_t^4\boldsymbol{\beta}_4 + e_t \tag{18.20}$$

Now, testing the null hypothesis of a linear AR model against a nonlinear STAR model is equivalent to testing the null hypothesis $H_0 : \boldsymbol{\beta}_j = 0$ for $j = 1, 2, 3, 4$ in the above auxiliary regressions, which is a conventional Lagrange multiplier (LM) test with an asymptotic χ^2 distribution.

In practice, it has been found that the LM test based on (18.19) for LSTAR models also has power against ESTAR alternatives. Thus, for reasons of parsimony, usually only the LM test based on (18.19) is computed for testing STAR-type nonlinearity in general. Also, instead of using the asymptotic χ^2 distribution, in small samples it is usually preferred to use the F version of the LM test, which tends to have better size and power properties. Finally, since TAR models are special cases of LSTAR models when the transition parameter $\gamma \to \infty$, it can be shown that the LM test also has power against threshold-type nonlinearity. Granger and Teräsvirta (1993) discuss these issues in more detail.

The LM test for STAR nonlinearity can be performed in `S+FinMetrics` using the `nonlinearTest` function, by setting the optional argument `method` to `"STAR-LM"`. For example, to test for STAR-type nonlinearity in NASDAQ realized volatility `ndx.rvol`, use the command

```
> nonlinearTest(log(ndx.rvol), method="STAR-LM", p=2, d=1:2)

Nonlinearity Test: STAR Nonlinearity

Null Hypothesis: no smooth threshold nonlinearity
Under Maintained Assumption of Homoskedastic Errors --

           ChiSq-stat ChiSq-dof ChiSq.pv-val
RVOL.lag1    21.3008         6       0.0016
RVOL.lag2    13.6974         6       0.0332

             F-stat   F-dof F.pv-val
RVOL.lag1    3.7068 (6,291)   0.0014
RVOL.lag2    2.3204 (6,291)  0.0333
```

In the above example, the transition variable is set to y_{t-d} by specifying the optional argument `d`.[14] More than one value of `d` can be specified and

[14]A weakly exogenous variable can also be used as the transition variable by setting the optional argument `q` instead of `d`. See the online help file for `nonlinearTest` for details.

`nonlinearTest` automatically computes the LM test for all of the given values of `d`. If the null hypothesis of a linear AR model is rejected, the test statistics based on different values of `d` can be used to choose the appropriate value of d in the final STAR model. In the output shown above, the null hypothesis of no STAR-type nonlinearity is rejected at the 5% significance level for both $d = 1$ and $d = 2$. In addition, the p-values from both the χ^2 test and F test prefer $d = 1$, which is consistent with the results of threshold-type nonlinearity tests presented in the previous section.

STAR Nonlinearity Test with Heteroskedastic Errors

The LM test presented above assumes that the error term in (18.16) has constant variance. However, economic and financial time series are often heteroskedastic, and neglected heteroskedasticity may lead to spurious rejection of the null hypothesis. Based on Davidson and MacKinnon (1985), Granger and Teräsvirta (1993) summarized the following LM test for nonlinearity, which is robust toward heteroskedastic errors:

1. Regress y_t on $\mathbf{X}_t$ to obtain the LS residuals $\hat{\epsilon}_t$.
2. Regress $\mathbf{X}_t z_t^j$ for $j = 1, 2, 3$ on $\mathbf{X}_t$ to obtain the residuals $\hat{\mathbf{R}}_t$.
3. Regress the unit vector on $\hat{\mathbf{R}}_t \hat{\epsilon}_t$ and compute the LM statistic as the explained sum of squares from this regression.

This test can be performed just as before by setting the optional argument `hetero` to `TRUE`:

```
> nonlinearTest(log(ndx.rvol), method="STAR-LM", p=2, d=1:2,
+ hetero=T)

Nonlinearity Test: STAR Nonlinearity

Null Hypothesis: no smooth threshold nonlinearity
Allowing Heteroskedastic Errors using White Correction --

          ChiSq-stat ChiSq-dof ChiSq.pv-val
RVOL.lag1    15.0731         6       0.0197
RVOL.lag2    10.8287         6       0.0938

             F-stat   F-dof F.pv-val
RVOL.lag1    2.5657 (6,291)   0.0195
RVOL.lag2    1.8162 (6,291)   0.0957
```

Now, the null hypothesis cannot be rejected at 5% significance level when $d = 2$, but it is still rejected at 5% level when $d = 1$. However, based on some simulation evidence, Lundbergh and Teräsvirta (1998) suggested that in some cases this robustification may not be desirable because it removes most of the power of the test.

18.4.3 Estimation of STAR Models

After confirming the existence of STAR-type nonlinearity in a time series, one can proceed to the next stage of building a STAR model. This usually involves choosing the transition variable and the form of transition function. As mentioned in the previous subsection, the test for STAR-type nonlinearity can be computed for a range of transition variables, and the p-values of the test statistics can be used to help choose the appropriate transition variable. The choice between the LSTAR model and the ESTAR model can usually be made by considering the specific transitional behavior under investigation or by comparing different information criteria. This subsection first shows how to estimate LSTAR models using the `STAR` function in `S+FinMetrics` and then it walks through an example of estimating an ESTAR model using the `S-PLUS` function `nlregb`.

LSTAR Model

Once the AR order and the transition variable have been chosen, LSTAR models can be estimated by *nonlinear least squares* (NLS):

$$\hat{\mathbf{\Theta}} = \underset{\gamma, c}{\operatorname{argmin}} \sum_t \hat{\epsilon}_t^2 \tag{18.21}$$

where

$$\hat{\epsilon}_t = y_t - \tilde{\mathbf{X}}_t \hat{\boldsymbol{\phi}}$$
$$\tilde{\mathbf{X}}_t = \begin{bmatrix} \mathbf{X}_t(1 - G(z_t; \gamma, c)) \\ \mathbf{X}_t G(z_t; \gamma, c) \end{bmatrix}$$
$$\hat{\boldsymbol{\phi}} = \begin{bmatrix} \hat{\boldsymbol{\phi}}^{(1)} \\ \hat{\boldsymbol{\phi}}^{(2)} \end{bmatrix} = \left[\sum_t (\tilde{\mathbf{X}}_t' \tilde{\mathbf{X}}_t) \right]^{-1} \left[\sum_t \tilde{\mathbf{X}}_t' y_t \right]$$

Note that the minimization of the NLS objective function is only performed over γ and c because $\hat{\boldsymbol{\phi}}^{(1)}$ and $\hat{\boldsymbol{\phi}}^{(2)}$ can be estimated by least squares once γ and c are known. Under the additional assumption that the errors are normally distributed, NLS is equivalent to the maximum likelihood estimation. Otherwise, the NLS estimates can be interpreted as quasi-maximum likelihood estimates.

Example 120 *LSTAR model for NASDAQ realized volatility*

The following command fits an LSTAR model to the logarithms of weekly realized volatility of the NASDAQ 100 index, with the same AR order and delay parameter used in the previous examples:

```
> ndx.lstar = STAR(log(ndx.rvol), p=2, d=1)
> summary(ndx.lstar)

Call:
STAR(X = log(ndx.rvol), p = 2, d = 1)

Parameter estimates:
          Values Std.Error Std.Error.white
    gamma  1.608  1.113     1.282
threshold -2.845  0.398     0.309


Coefficient estimates and standard errors:

Lower regime:
                 Values Std.Error Std.Error.white
intercept(lower) -3.729  1.832     2.696
     lag1(lower) -0.221  0.404     0.632
     lag2(lower)  0.205  0.092     0.092

Upper regime:
                 Values Std.Error Std.Error.white
intercept(upper) -2.668  1.904     1.497
     lag1(upper) -0.396  1.076     0.896
     lag2(upper)  0.216  0.134     0.131

Std. Errors of Residuals:
[1] 0.415

Information Criteria:
     logL      AIC      BIC       HQ
 -158.863  329.727  351.950  338.620

Degrees of freedom:
 total residuals
   300       294
Time period: from 01/15/1996 to 10/08/2001
```

Note that the threshold estimate -2.85 is very close to the SETAR estimate of -2.88 given by the TAR estimate `ndx.setar.r`. However, by allowing for a smooth transition between two regimes, the AR coefficients in both regimes are quite different from those estimated by `ndx.setar.r`.

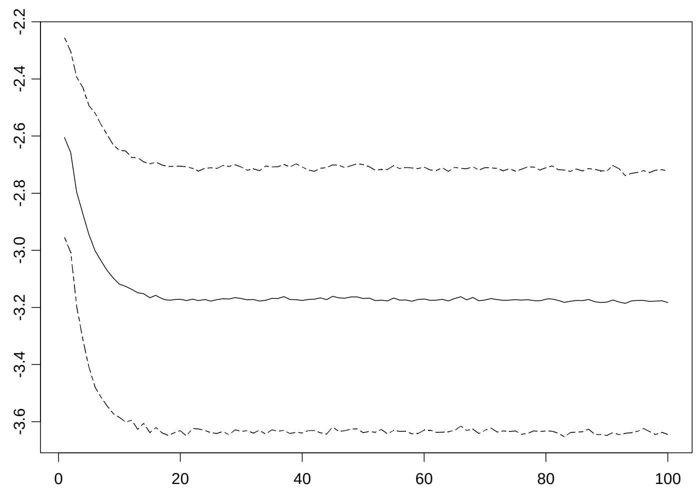


FIGURE 18.10. Predicted realized volatility (in logarithmic scale) from `ndx.lstar`.

Predictions from LSTAR Model

Simulation-based forecasts from the LSTAR model can be easily generated using the same principle for generating forecasts from VAR models and SETAR models. The fitted model objects returned by the `STAR` function have class `"STAR"`. By calling the generic `predict` function on fitted model objects, the method function `predict.STAR` is automatically invoked. For example, the following command generates 100-step-ahead forecasts from `ndx.lstar`:

```
> ndx.pred.3 = predict(ndx.lstar, n.predict=100,
+ CI.alpha=0.6, n.sim=10000)
> tsplot(cbind(ndx.pred.3$values, ndx.pred.3$CI),
+ lty=c(1,6,6))
```

and Figure 18.10 shows the forecasts with 60% pointwise confidence intervals. The forecasts are very similar to those generated by the SETAR model object `ndx.setar`, except they do not have the initial small peak exhibited by the SETAR forecasts.

ESTAR Model

Currently, the `STAR` function in `S+FinMetrics` only supports LSTAR models, not ESTAR models. However, the estimation of ESTAR models follows essentially the same procedure in (18.21) with the transition function given by (18.18). Here, an example is given to show how to estimate ESTAR models using the S-Plus function nlregb for nonlinear least squares estimation.

The arguments expected by nlregb are as follows:

```
> args(nlregb)
function(nres, start, residuals, jacobian=NULL, scale=NULL,
control = NULL, lower = -Inf, upper = Inf, ...)
```

where the first argument, nres, specifies the number of observations or residuals to be used, the second argument, start, specifies the starting values for the unknown parameters, and the third argument, residuals, is an S-PLUS function that takes the parameter values and computes the residual vector with length equal to nres. The optional arguments lower and upper can be used to specify lower and upper bounds on the unknown parameters.

One general issue in estimating STAR models is that the transition parameter γ can become large and cause numerical problems in the optimization procedure. To alleviate the potential numerical problems in estimating ESTAR models, it is usually preferred to estimate the following transition function instead of the original exponential function in (18.18):

$$G(z_t; \tilde{\gamma}, c) = 1 - \exp\left\{-e^{\tilde{\gamma}}\frac{(z_t - c)^2}{\sigma_z^2}\right\} \tag{18.22}$$

where σ_z^2 is the sample variance of the transition variable z_t. The new parameter $\tilde{\gamma}$ can be transformed to the original parameter γ as follows:

$$\gamma = \frac{e^{\tilde{\gamma}}}{\sigma_z^2} \tag{18.23}$$

This transformation has the following numerical properties:

1. The squared distance between z_t and the threshold c is now scaled by the variance of z_t, which makes it scale-free.

2. The original parameter γ lies in $(0, \infty)$, which requires a constrained optimization in terms of γ. The new parameter $\tilde{\gamma}$ lies in $(-\infty, \infty)$ and is unconstrained.

3. The new parameter $\tilde{\gamma}$ is a linear function of the logarithm of γ, which is more dampened than γ.

Using the new formulation in (18.22), the following S-PLUS function takes the unknown parameter values $(\tilde{\gamma}, c)$ and returns the residual vector:

```
ESTAR.res = function(theta, g.scale, x, y, q)
{
  k = ncol(x)
  G = 1 - exp( - exp(theta[1])/g.scale * (q - theta[2])^2)
  X = cbind(x * (1 - G), x * G)
  m = crossprod(t(backsolve(chol(crossprod(X)), diag(2 * k))))
  beta = m %*% t(X) %*% y
  y - X %*% beta
}
```

Now, to estimate an ESTAR model with an AR(2) specification and transition variable $z_t = y_{t-1}$ using the NASDAQ realized volatility series, use the following commands:

```
> ndx.LHS = log(ndx.rvol)[3:length(ndx.rvol)]@data
> ndx.RHS = cbind(1, tslag(log(ndx.rvol), 1:2, trim=T)@data)
> ndx.estar = nlregb(length(ndx.rvol)-2,
+    start=c(0,mean(ndx.RHS[,2])),
+    residuals=ESTAR.res,
+    lower=c(-Inf, min(ndx.RHS[,2])),
+    upper=c( Inf, max(ndx.RHS[,2])),
+    g.scale=var(ndx.RHS[,2]),
+    x=ndx.RHS, y=ndx.LHS, q=ndx.RHS[,2]))
```

Note that the regressors `ndx.RHS` include a constant term and two lagged values of y_t, and the transition variable y_{t-1} is given by the second column of `ndx.RHS`. In the call to the `nlregb` function, the starting values of $\tilde{\gamma}$ is set to zero, which corresponds to setting $\gamma = 1$, and the starting value of c is simply set to the mean of the transition variable y_{t-1}. The other arguments `g.scale`, `x`, `y`, and `q` to the residual function `ESTAR.res` are passed as optional arguments to `nlregb`. The NLS estimates of $(\tilde{\gamma}, c)$ are given by

```
> ndx.estar$parameters
[1] -1.239878 -2.774638
```

Note that the threshold estimate of -2.77 is close to the threshold estimates obtained in earlier examples. The transition parameter γ in the original exponential function can be obtained as follows:

```
> exp(ndx.estar$parameters[1])/var(ndx.RHS[,2])
[1] 1.013556
```

18.5 Markov Switching State Space Models

The nonlinear time series models introduced so far all allow for different regimes, with each regime represented by a simple AR model. For TAR and SETAR models, the regimes are solely determined by the magnitude of an *observable* weakly exogenous variable, whereas for STAR models, the regimes are allowed to switch smoothly according to the magnitude of a weakly exogenous variable relative to a threshold value. This section introduces another type of regime switching model – the Markov switching model – where the regimes are determined by an *unobserved* state or regime variable that follows a discrete state Markov process. Discrete state Markov processes, also called Markov chains, are very popular choices for modeling state-dependent behavior. Since Hamilton (1989) proposed using a simple Markov switching AR process to model the U.S. real Gross National Product (GNP), Markov switching time series models have seen extraordinary growth and become extremely popular for modeling economic and financial time series. They have been applied to model and forecast business cycles, the term structure of interest rates, volatility in economic and financial variables, foreign exchange rate dynamics, inflation rate dynamics, etc.

This section first introduces the discrete state Markov process used to model the hidden state variable and it then illustrates how the discrete state Markov process can be combined with an AR model to produce the Markov switching AR process. To allow for Markov switching dynamics in a much broader context, Markov switching state space models are then introduced and examples will be given to illustrate the estimation of these models using `S+FinMetrics` functions.

18.5.1 Discrete State Markov Process

Discrete state Markov processes are very popular choices for modeling state-dependent behavior in natural phenomena and are natural candidates for modeling the hidden state variables in Markov switching models. A discrete state Markov process classifies the state of the world S_t at any time t into a few discrete regimes. The state switches between different regimes according to its previous value and transition probabilities given by[15]

$$\Pr(S_t = j | S_{t-1} = i) = P_{ij} \geq 0 \tag{18.24}$$

where $i, j = 1, 2, \ldots, k$, with k different possible states or regimes, and

$$\sum_{j=1}^{k} \Pr(S_t = j | S_{t-1} = i) = 1 \tag{18.25}$$

[15]A discrete state Markov process that only depends on its most recent observation is called the first-order Markov process. Since higher-order Markov processes can always be rewritten as a first-order Markov process, it usually suffices to consider only the first-order Markov process.

It is usually convenient to collect the transition probabilities into a *transition matrix*:

$$\mathcal{P} = \begin{bmatrix} P_{11} & P_{12} & \cdots & P_{1k} \\ P_{21} & P_{22} & \cdots & P_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ P_{k1} & P_{k2} & \cdots & P_{kk} \end{bmatrix}$$

where each row sums up to 1. For example, at time t, the state of the economy S_t can be classified as either recessionary ($S_t = 1$) or expansionary ($S_t = 2$). Using quarterly observations of the U.S. real GNP from 1952 to 1984, Kim (1994) estimated the transition matrix to be

$$\mathcal{P} = \begin{bmatrix} 47\% & 53\% \\ 5\% & 95\% \end{bmatrix} \tag{18.26}$$

These transition probabilities imply that if the economy is in an expansion, it tends to stay in expansion with a very high probability of 95%; if the economy is in a recession, it has a 47% chance of staying in a recession and a 53% chance of getting out of a recession. These numbers also reflect the common observation that the transition from an expansion to a recession is usually very quick, whereas the recovery from a recession is relatively slow.

Suppose at time t, the probability of each state or regime is given by the vector $\boldsymbol{\pi}_t = (P_1, P_2, \ldots, P_k)$, then the probability of each state at time $t + 1$ is given by

$$\boldsymbol{\pi}_{t+1} = \mathcal{P}'\boldsymbol{\pi}_t \tag{18.27}$$

For a stationary discrete state Markov process, the *ergodic probability* vector $\boldsymbol{\pi}$ exists such that

$$\boldsymbol{\pi} = \mathcal{P}'\boldsymbol{\pi} \tag{18.28}$$

The ergodic probability vector can also be treated as the *steady state*, or the *unconditional* probability of each state of the world. `S+FinMetrics` provides a convenience function `mchain.p0` to compute the ergodic probability vector for a stationary Markov chain.[16] For example, the following command computes the ergodic probabilities for the state of the economy using the transition matrix in (18.26):

```
> mchain.p0(matrix(c(0.47, 0.05, 0.53, 0.95), 2, 2))
[1] 0.0862069 0.9137931
```

Thus the unconditional probability of the economy being in a recession is about 9% and the unconditional probability of the economy being in an expansion is about 91%.

[16] See Hamilton (1994) for the analytic formula for computing the ergodic probabilities for a stationary Markov chain.

The transition probabilities can also be used to infer the duration of each state or regime. For example, using the transition matrix in (18.26), the average duration of an economic expansion can be computed as[17]

$$\frac{1}{1-P_{22}} = 20 \text{ quarters} = 5 \text{ years}$$

and the average duration of an economic recession can be computed as

$$\frac{1}{1-P_{11}} = 2 \text{ quarters}$$

which is consistent with the fact that a recession is usually defined as a drop in real GDP for two consecutive quarters.

18.5.2 Markov Switching AR Process

If the model parameters in the simple AR(p) model in (18.4) are relaxed to be dependent on a *latent* or hidden state variable S_t, it becomes

$$y_t = \mu_{S_t} + \mathbf{X}_t \boldsymbol{\phi}_{S_t} + u_t \quad \text{for } t = 1, 2, \ldots, n \tag{18.29}$$

where $\mathbf{X}_t = (y_{t-1}, y_{t-2}, \ldots, y_{t-p})$, $\boldsymbol{\phi}_{S_t}$ is the $p \times 1$ vector of AR coefficients, $u_t \sim N(0, \sigma^2_{S_t})$, and the hidden state variable S_t follows a k-regime Markov chain given by (18.24) and (18.25). This is usually referred to as the *Markov switching* AR(p) *process*. The Markov switching AR(p) model has proved to be effective at modeling nonlinear dynamics usually observed in economic and financial time series. For example, Hamilton (1989) used a two-state Markov switching AR(4) model with constant σ^2 to capture the different dynamics observed in the U.S. real GNP during economic recessions and expansions.

In general, if the states $\mathbf{S} = (S_{p+1}, \ldots, S_n)$ are known, the unknown parameters $\boldsymbol{\Theta}$ of the Markov switching AR(p) model, which include the intercept terms, the AR coefficients, and the error variance in different regimes, can be estimated by maximizing the following log-likelihood function:

$$L(\boldsymbol{\Theta}|\mathbf{S}) = \sum_{t=p+1}^{n} \log f(y_t|\mathcal{Y}_{t-1}, S_t)$$

where $\mathcal{Y}_{t-1}$ denotes all the information available at time $t-1$ and includes all of the observations in $\mathbf{X}_j$ for $j \le t$, and

$$f(y_t|\mathcal{Y}_{t-1}, S_t) \propto \exp\left\{-\frac{1}{2}\log\sigma^2_{S_t} - \frac{(y_t - \mu_{S_t} - \mathbf{X}_t\boldsymbol{\phi}_{S_t})^2}{2\sigma^2_{S_t}}\right\} \tag{18.30}$$

[17]See Kim and Nelson (1999), for example, for the derivation of this result.

However, the states $\mathbf{S}$ are usually unobserved and must be inferred from the data. When the states $\mathbf{S}$ are unknown, the parameters of the Markov switching AR(p) model are expanded to include the transition probabilities $\mathcal{P}$. By applying the law of total probability, the log-likelihood function can now be written as

$$\begin{aligned} L(\mathbf{\Theta}) &= \sum_{t=p+1}^{n} \log f(y_t|\mathcal{Y}_{t-1}) \\ &= \sum_{t=p+1}^{n} \log \left\{ \sum_{j=1}^{k} f(y_t|\mathcal{Y}_{t-1}, S_t = j) \Pr(S_t = j|\mathcal{Y}_{t-1}) \right\} \end{aligned} \tag{18.31}$$

where $f(y_t|\mathcal{Y}_{t-1}, S_t = j)$ is given in (18.30), and by Bayes theorem the *predictive* probability $\Pr(S_t = j|\mathcal{Y}_{t-1})$ can be shown to be

$$\begin{aligned} \Pr(S_t = j|\mathcal{Y}_{t-1}) &= \sum_{i=1}^{k} \Pr(S_t = j|S_{t-1} = i, \mathcal{Y}_{t-1}) \Pr(S_{t-1} = i|\mathcal{Y}_{t-1}) \\ &= \sum_{i=1}^{k} P_{ij} \frac{f(y_{t-1}|\mathcal{Y}_{t-2}, S_{t-1} = i) \Pr(S_{t-1} = i|\mathcal{Y}_{t-2})}{\sum_{m=1}^{k} f(y_{t-1}|\mathcal{Y}_{t-2}, S_{t-1} = m) \Pr(S_{t-1} = m|\mathcal{Y}_{t-2})} \end{aligned} \tag{18.32}$$

Thus, given an estimate of the initial probability of each state as $\Pr(S_{p+1} = i|\mathcal{Y}_p)$ for $i = 1, 2, \ldots, k$, the log-likelihood function of the Markov switching AR(p) model can be computed iteratively using (18.31) and (18.32) and the unknown parameters $\mathbf{\Theta}$ can be estimated by maximum likelihood estimation (MLE).

The evaluation of the above log-likelihood function for the Markov switching AR(p) model can be easily programmed in `S-PLUS`. However, since it involves an iterative process that prevents the use of vectorized operations in `S-PLUS`, the optimization process of obtaining the MLE can be slow and computationally inefficient. In order to be able to estimate a broad range of Markov switching models using the same code, the following subsection introduces Markov switching state space models that includes the Markov switching AR(p) model as a special case. The Markov switching state space models utilize optimized C code for fast calculation.

18.5.3 Markov Switching State Space Models

As shown in Chapter 14, most linear time series regression models can be cast into a state space form, and the state space representation provides a convenient framework for obtaining filtered and smoothed estimates of the unobserved state variables. In this subsection, the state space representation in Chapter 14 is generalized to allow for Markov switching dynamics so that a vast number of Markov switching models can be easily estimated using the same framework.

Using the notation in Chapter 14, a state space model can be represented as follows:

$$\begin{aligned} \boldsymbol{\alpha}_{t+1} &= \mathbf{d}_t + \mathbf{T}_t \cdot \boldsymbol{\alpha}_t + \mathbf{H}_t \cdot \boldsymbol{\eta}_t & (18.33) \\ \mathbf{y}_t &= \mathbf{c}_t + \mathbf{Z}_t \cdot \boldsymbol{\alpha}_t + \mathbf{G}_t \cdot \boldsymbol{\varepsilon}_t & (18.34) \end{aligned}$$

where $\boldsymbol{\alpha}_{t+1}$ is the $m \times 1$ state vector, $\mathbf{y}_t$ is the $N \times 1$ vector of observed variables, $\boldsymbol{\eta}_t \sim$ iid $N(0, \mathbf{I}_r)$ is the $r \times 1$ vector of disturbance terms in the *transition equation* governing the dynamics of the state vector $\boldsymbol{\alpha}_{t+1}$, $\boldsymbol{\varepsilon}_t \sim$ iid $N(\mathbf{0}, \mathbf{I}_N)$ is the $N \times 1$ vector of disturbance terms in the *measurement equation* governing the dynamics of the observed variables $\mathbf{y}_t$, and $\mathbf{d}_t$, $\mathbf{T}_t$, $\mathbf{H}_t$, $\mathbf{c}_t$, $\mathbf{Z}_t$, and $\mathbf{G}_t$ are conformable *hyperparameter matrices* or *system matrices*. More compactly, the above representation can be rewritten as

$$\begin{pmatrix} \boldsymbol{\alpha}_{t+1} \\ \mathbf{y}_t \end{pmatrix} = \boldsymbol{\delta}_t + \boldsymbol{\Phi}_t \cdot \boldsymbol{\alpha}_t + \mathbf{u}_t, \tag{18.35}$$

where $\mathbf{u}_t \sim$ iid $N(\mathbf{0}, \boldsymbol{\Omega}_t)$ and

$$\boldsymbol{\delta}_t = \begin{pmatrix} \mathbf{d}_t \\ \mathbf{c}_t \end{pmatrix}, \ \boldsymbol{\Phi}_t = \begin{pmatrix} \mathbf{T}_t \\ \mathbf{Z}_t \end{pmatrix}, \ \mathbf{u}_t = \begin{pmatrix} \mathbf{H}_t \boldsymbol{\eta}_t \\ \mathbf{G}_t \boldsymbol{\varepsilon}_t \end{pmatrix}, \ \boldsymbol{\Omega}_t = \begin{pmatrix} \mathbf{H}_t \mathbf{H}_t' & \mathbf{0} \\ \mathbf{0} & \mathbf{G}_t \mathbf{G}_t' \end{pmatrix}$$

For *Markov switching state space models*, the hyperparameter matrices are assumed to be dependent on a *latent* or unobserved discrete state variable S_t:

$$\begin{aligned} \boldsymbol{\delta}_t &= \boldsymbol{\delta}_{S_t} \\ \boldsymbol{\Phi}_t &= \boldsymbol{\Phi}_{S_t} \\ \boldsymbol{\Omega}_t &= \boldsymbol{\Omega}_{S_t} \end{aligned}$$

and the discrete state variable S_t follows a k-regime Markov chain given in (18.24) and (18.25). For example, by setting the continuous state vector to $\boldsymbol{\alpha}_t = (y_t, y_{t-1})$, the Markov switching AR(2) model can be put into the above state space representation with

$$\boldsymbol{\delta}_t = \begin{bmatrix} \mu_{S_{t+1}} \\ 0 \\ 0 \end{bmatrix}, \ \boldsymbol{\Phi}_t = \begin{bmatrix} \phi'_{S_{t+1}} \\ \mathbf{I}_{2\times 2} \end{bmatrix}, \ \mathbf{I}_{2\times 2} = \begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix}$$

and $\boldsymbol{\Omega}_t$ is a 3×3 matrix with $\sigma^2_{S_{t+1}}$ being the $(1, 1)$ element and zero elsewhere.

Example 121 *State space representation of Markov switching AR(2) model*

S+FinMetrics uses a "list" object with some required components to represent a state space model in S-PLUS, and Chapter 14 has many examples showing how to create such objects for some popular time series regression models. In order for Markov switching state space models to be represented by an S-PLUS object, the "list" object is expanded to allow for the following components: mTrans, mDelta.other, mPhi.other and mOmega.other. The mTrans component is required for a Markov switching state space representation and specifies the transition matrix $\mathcal{P}$ for the underlying Markov chain, and at least one of mDelta.other, mPhi.other, and mOmega.other must be specified so that at least some hyperparameter of the model is Markov switching. The usual components mDelta, mPhi and mOmega specify the hyperparameter matrices for the first regime, and the new components mDelta.other, mPhi.other, and mOmega.other specify the hyperparameter matrices for other regimes if necessary. If there are $k > 2$ regimes for the discrete state variable S_t, the components mDelta.other, mPhi.other, and mOmega.other store the hyperparameter matrices for regimes 2 to k stacked columnwise.

For example, the unknown parameters of a two-regime Markov switching AR(2) model can be collected in the vector:

$$v = (\mu_1, \mu_2, \phi_{11}, \phi_{12}, \phi_{21}, \phi_{22}, \sigma_1, \sigma_2, P_{11}, P_{22}) \tag{18.36}$$

where μ_1, ϕ_{11}, ϕ_{12}, and σ_1 are the intercept term, the AR coefficients, and error standard deviation for the first regime, μ_2, ϕ_{21}, ϕ_{22}, and σ_2 are the counterparts for the second regime, P_{11} and P_{22} are the diagonal elements of the transition matrix $\mathcal{P}$. Note that since each row of $\mathcal{P}$ sums up to 1, only two transition probabilities are required to identify $\mathcal{P}$. The following S-PLUS function takes the vector (18.36) and returns a "list" object giving the state space representation of the two-regime Markov switching AR(2) model[18]:

```
GetSsfMSAR = function(parm)
{
  mDelta = mDelta.other = rep(0, 3)
  mDelta[1] = parm[1]
  mDelta.other[1] = parm[2]
#
  mPhi = mPhi.other = matrix(0, 3, 2)
  mPhi[1,] = c(parm[3], parm[4])
  mPhi.other[1,] = c(parm[5], parm[6])
  mPhi[2:3,1] = mPhi.other[2:3,1] = 1
#
  mOmega = mOmega.other = matrix(0, 3, 3)
  mOmega[1,1] = parm[7]
  mOmega.other[1,1] = parm[8]
#
  mSigma = matrix(0, 3, 2)
```

[18]One may also use the S+FinMetrics function GetSsf.MSAR.

```
  mSigma[1:2, 1:2] = diag(1e+6, 2)
#
  mTrans = matrix(0, 2, 2)
  mTrans[1,1] = parm[9]
  mTrans[1,2] = 1 - mTrans[1,1]
  mTrans[2,2] = parm[10]
  mTrans[2,1] = 1 - mTrans[2,2]
#
  list(mDelta=mDelta, mDelta.other=mDelta.other,
       mPhi=mPhi, mPhi.other=mPhi.other,
       mOmega=mOmega, mOmega.other=mOmega.other,
       mSigma=mSigma, mTrans=mTrans)
}
```

Note that a diffuse prior on the initial state vector is specified by setting the first 2×2 block of `mSigma` to a diagonal matrix with large values on the diagonal and setting the last row of `mSigma` to zero.

Approximate MLE of Markov Switching State Space Models

Since Markov switching state space models allow for nonlinear dynamics, the traditional Kalman filtering and smoothing algorithms for Gaussian linear state space models can no longer be applied to obtain valid inference on the unobserved state vector. In particular, given the initial estimate $\mathbf{a}_{t|t}^{(i)}$ and $\mathbf{P}_{t|t}^{(i)}$ for $S_t = i$ with $i = 1, \ldots, k$, the prediction equations for the Gaussian linear state space model in (14.39) and (14.40) now become

$$\mathbf{a}_{t+1|t}^{(i,j)} = \mathbf{T}_t \mathbf{a}_{t|t}^{(i)} \tag{18.37}$$

$$\mathbf{P}_{t+1|t}^{(i,j)} = \mathbf{T}_t \mathbf{P}_{t|t}^{(i)} \mathbf{T}_t' + \mathbf{H}_t \mathbf{H}_t' \tag{18.38}$$

where the superscript (i, j) denotes the case of $S_t = i$ and $S_{t+1} = j$ for $i, j = 1, \ldots, k$. The updating equations for the Gaussian linear state space model in (14.34) and (14.35) now become

$$\mathbf{a}_{t|t}^{(i,j)} = \mathbf{a}_{t|t-1}^{(i,j)} + \mathbf{K}_t^{(i,j)} \mathbf{v}_t^{(i,j)} \tag{18.39}$$

$$\mathbf{P}_{t|t}^{(i,j)} = \mathbf{P}_{t|t-1}^{(i,j)} - \mathbf{P}_{t|t-1}^{(i,j)} \mathbf{Z}_t' (\mathbf{K}_t^{(i,j)})' \tag{18.40}$$

where

$$\mathbf{v}_t^{(i,j)} = \mathbf{y}_t - \mathbf{c}_t - \mathbf{Z}_t \mathbf{a}_{t|t-1}^{(i,j)}$$

$$\mathbf{F}_t^{(i,j)} = \mathbf{Z}_t \mathbf{P}_{t|t-1}^{(i,j)} \mathbf{Z}_t' + \mathbf{G}_t \mathbf{G}_t'$$

$$\mathbf{K}_t^{(i,j)} = \mathbf{P}_{t|t-1}^{(i,j)} \mathbf{Z}_t' (\mathbf{F}_t^{(i,j)})^{-1}$$

Therefore, at each step, the set of statistics that needs to be computed and stored will increase by the order of k. Obviously, even for a relatively small sample, the Kalman filtering algorithm will become computationally infeasible.

To make the filtering algorithm manageable, Kim (1994) proposed collapsing the set of statistics in the updating equations (18.39) and (18.40) as follows:

$$\mathbf{a}_{t|t}^{(j)} = \frac{\sum_{i=1}^{k} \Pr(S_t = j, S_{t-1} = i|\mathcal{Y}_t)\mathbf{a}_{t|t}^{(i,j)}}{\Pr(S_t = j|\mathcal{Y}_t)} \tag{18.41}$$

$$\mathbf{P}_{t|t}^{(j)} = \frac{\sum_{i=1}^{k} \Pr(S_t = j, S_{t-1} = i|\mathcal{Y}_t)[\mathbf{P}_{t|t}^{(i,j)} + (\mathbf{a}_{t|t}^{(j)} - \mathbf{a}_{t|t}^{(i,j)})(\mathbf{a}_{t|t}^{(j)} - \mathbf{a}_{t|t}^{(i,j)})']}{\Pr(S_t = j|\mathcal{Y}_t)} \tag{18.42}$$

where the *filtered* probability $\Pr(S_t = j|\mathcal{Y}_t)$ can be updated similarly as in (18.32), given an initial estimate. Now at each step, only k sets of statistics need to be stored, which can be fed into the prediction equations (18.37) and (18.38) to complete the filtering algorithm. This algorithm is sometimes referred to as *Kim's filtering algorithm*.

Just like the Kalman filtering algorithm for Gaussian linear state space models, Kim's filtering algorithm can be used to provide the prediction error decomposition for computing the log-likelihood function of Markov switching state space models. However, the drawback of the above filtering algorithm is that the filtered estimates $\mathbf{a}_{t|t}^{(j)}$ now follow normal mixture distributions instead of normal distributions as in Gaussian linear state space models. As a result, the MLEs obtained using Kim's algorithm are only approximate and not optimal, but empirical evidence seems to suggest that approximate MLEs obtained using Kim's filtering algorithm are very reliable.[19]

The `SsfLoglikeMS` function in `S+FinMetrics` implements Kim's filtering algorithm to compute the log-likelihood function for arbitrary Markov switching state space models, and the `SsfFitMS` function uses it to obtain approximate MLEs of the unknown parameters in Markov switching state space models. However, Markov switching state space models can be difficult to fit due to various numerical issues. Here, a few guidelines are provided for using the `SsfFitMS` function for maximum likelihood estimation of Markov switching state space models:

1. Make sure that the model to be fitted is actually identified. It can be very easy to specify a Markov switching model that is not identified or poorly identified. Overidentification or poor identification can cause the optimization procedure to fail.

[19]In recent years, more computationally intensive Bayesian methods have also been developed to analyze Markov switching state space models or non-Gaussian state space models on a case-by-case basis. See Kim and Nelson (1998), Kim, Shephard, and Chib (1998), and Aguilar and West (2000) for some examples.

2. Start from a small model. If the estimation of the small model does not pose any problem, extend the model to allow for more features.

3. Provide good starting values to `SsfFitMS`. Good starting values can be found by calling `SsfLoglikeMS` with different sets of parameter values and choosing the one with largest log-likelihood value.

4. Although the `SsfFitMS` function allows lower and upper bound constraints on the parameters, sometimes better convergence can be obtained by transforming the parameters so that the parameters to be estimated are unconstrained.

Example 122 *Markov switching AR(2) model for NASDAQ realized volatility*

Earlier examples in this chapter show that the logarithms of weekly realized volatility of the NASDAQ 100 index can be modeled by a switching AR(2) process, with the switching determined by either a TAR model or a STAR model. It is interesting to see if the Markov switching AR(2) model can provide a better or equivalent characterization of the nonlinear dynamics observed in the data.

Instead of directly estimating the unknown parameters for the Markov switching AR(2) model as given in (18.36), it is usually better to transform these parameters so that they are unconstrained. For example, the following monotonic transformations are usually adopted:

1. If x lies within $(0, \infty)$, then $y = \log x$ is unconstrained and $x = e^y$.

2. If x lies within $(0, 1)$, then $y = \log[x/(1-x)]$ is unconstrained and $x = 1/(1 + e^{-y})$.

3. If x lies within $(-1, 1)$, then $y = \log[(1+x)/(1-x)]$ is unconstrained and $x = 2/(1 + e^{-y}) - 1$.

4. For the AR(2) process $y_t = \phi_1 y_{t-1} + \phi_2 y_{t-2} + u_t$ to be stationary, the roots z_1 and z_2 of the characteristic equation $z^2 - \phi_1 z - \phi_2 = 0$ must lie within the unit circle, with $z_1 + z_2 = \phi_1$ and $z_1 \cdot z_2 = -\phi_2$.

The following `S-PLUS` function modifies the `GetSsfMS` function given earlier in this subsection by employing the above transformations. It now takes an unconstrained parameter vector and returns the state space representation of Markov switching AR(2) model:

```
GetSsfMSAR2 = function(parm)
{
  parm = as.vector(parm)
#
  mDelta = mDelta.other = rep(0, 3)
  mDelta[1] = parm[1]
  mDelta.other[1] = parm[1] + exp(parm[2])
#
  AR11 = 2/(1+exp(-parm[3])) - 1
  AR12 = 2/(1+exp(-(parm[3]+exp(parm[4])))) - 1
  AR21 = 2/(1+exp(-parm[5])) - 1
  AR22 = 2/(1+exp(-(parm[5]+exp(parm[6])))) - 1
#
  mPhi = mPhi.other = matrix(0, 3, 2)
  mPhi[1,] = c(AR11+AR12, -AR11*AR12)
  mPhi.other[1,] = c(AR21+AR22, -AR21*AR22)
  mPhi[2:3,1] = mPhi.other[2:3,1] = 1
#
  mOmega = matrix(0, 3, 3)
  mOmega[1,1] = exp(parm[7])
#
  mSigma = matrix(0, 3, 2)
  mSigma[1:2, 1:2] = diag(1e+6, 2)
#
  mTrans = matrix(0, 2, 2)
  mTrans[1,2] = 1/(1+exp(-parm[8]))
  mTrans[1,1] = 1 - mTrans[1,2]
  mTrans[2,1] = 1/(1+exp(-parm[9]))
  mTrans[2,2] = 1 - mTrans[2,1]
#
  ssf = list(mDelta=mDelta, mDelta.other=mDelta.other,
        mPhi=mPhi, mPhi.other=mPhi.other, mOmega=mOmega,
        mTrans=mTrans, mSigma=mSigma)
  CheckSsf(ssf)
}
```

A few comments on the function GetSsfMSAR2 are as follows:

1. The second parameter parm[2] is actually $\log(\mu_2 - \mu_1)$. By employing this transformation, μ_2 is guaranteed to be greater than μ_1, and thus the first regime can be identified as the low-volatility regime and the second as the high-volatility regime.
2. The fourth and sixth parameters, parm[4] and parm[6], are actually the logarithmic difference between two characteristic roots of their respective AR(2) processes. By employing this transformation, the first roots are identified as the smaller roots and the second are identified as the larger ones.
3. Finally, it is usually preferred to call the CheckSsf function before returning the list with the state space representation, which makes sure that the returned list is a valid state space representation.

Now, to fit the Markov switching AR(2) model to `log(ndx.rvol)`, use the following commands[20]:

```
> ndx.start = c(-2, -0.7, -0.7, 0.7, -0.7, 0.7, -2, -2, -3)
> names(ndx.start) = c("mu1", "mu2", "phi11", "phi12",
+ "phi21", "phi22", "sigma", "p", "q")
> ndx.msar = SsfFitMS(ndx.start, log(ndx.rvol), GetSsfMSAR2,
+ l.start=11)
Iteration  0 : objective =  0.5575044
Iteration  1 : objective =  0.9047186
Iteration  2 : objective =  0.555338
...
Iteration  98 : objective =  0.5161791
RELATIVE FUNCTION CONVERGENCE
```

Note that the first argument to `SsfFitMS` specifies the starting values, the second argument specifies the data to be used, and the third argument specifies the `S-PLUS` function that takes a vector of model parameters and returns a valid state space representation of a Markov switching model. Since the filtering algorithm is started with diffuse priors on the state vector, the optional argument `l.start` is used to start the log-likelihood function evaluation from the 11th observation, which allows the effects of diffuse priors on the state vector to dissipate before log-likelihood values are computed.

The returned object is a `"SsfFit"` object, and applying the generic `summary` function returns the standard errors of the estimated parameters and associated t-statistics:

```
> class(ndx.msar)
[1] "SsfFit"
> summary(ndx.msar)
Log-likelihood:  -150.724
302 observations
Parameters:
        Value Std. Error  t value
  mu1 -1.8670    0.27600  -6.7640
  mu2 -0.9385    1.08100  -0.8684
phi11 -0.3336    0.23730  -1.4060
phi12  0.4073    0.32060   1.2710
phi21 -0.8366    0.25960  -3.2230
phi22  0.8109    0.22670   3.5760
```

[20]`S+FinMetrics` provides the function `MSAR` for estimating general Markov switching AR(p) processes. The `MSAR` function returns an `"MSAR"` object, and methods for many generic functions, such as `summary`, `plot`, `residuals`, `vcov`, and `simulate`, are provided for `"MSAR"` objects. See the online help file for `MSAR` for details.

```
sigma -1.8310    0.08313 -22.0300
    p -5.3150    1.00900  -5.2670
    q -8.4870    6.00100  -1.4140

Convergence:  RELATIVE FUNCTION CONVERGENCE
```

From the above output, most of the parameters are significant according to the t-statistics. To transform the estimated parameters into the parameters for the Markov switching AR(2) model, simply call `GetSsfMSAR2` on the ML estimates[21]:

```
> ndx.ssf = GetSsfMSAR2(ndx.msar$parameters)
> cbind(ndx.ssf$mDelta, ndx.ssf$mDelta.other)
         [,1]      [,2]
[1,] -1.86719 -1.475965
[2,]  0.00000  0.000000
[3,]  0.00000  0.000000
> ndx.ssf$mPhi
          [,1]       [,2]
[1,] 0.3606984 0.08693354
[2,] 1.0000000 0.00000000
[3,] 1.0000000 0.00000000
> ndx.ssf$mPhi.other
          [,1]      [,2]
[1,] 0.2130623 0.2406814
[2,] 1.0000000 0.0000000
[3,] 1.0000000 0.0000000
> ndx.ssf$mOmega
          [,1] [,2] [,3]
[1,] 0.1601773    0    0
[2,] 0.0000000    0    0
[3,] 0.0000000    0    0
> ndx.ssf$mTrans
             [,1]        [,2]
[1,] 0.9951049274 0.004895073
[2,] 0.0002061726 0.999793827
```

Note that the intercept terms in both regimes and the AR coefficients in the high-volatility regime are similar to those estimated by the SETAR model `ndx.setar.r` in Section 18.3. However, the AR coefficients in the low-volatility regime are somewhat different from those estimated by `ndx.setar.r`. In addition, both the transition probabilities P_{11} and P_{22} are estimated to be very close to 1, which suggests that once y_t is in a certain regime, it tends to stay in that regime.

[21]Standard errors for these parameters may be obtained using the delta method.

Filtered and Smoothed Estimates of Regime Probabilities

Once the unknown parameters of Markov switching models are estimated, it is usually of interest to obtain the *filtered* estimates of the latent discrete state or regime probability $\Pr(S_t = j|\mathcal{Y}_t)$. However, this quantity is already computed by Kim's filtering algorithm and, thus, is a side product of the log-likelihood function evaluation. In addition, it is also of interest to obtain the *smoothed* estimates of the latent discrete state probability $\Pr(S_t = j|\mathcal{Y}_n)$, which is useful for retrospective analysis. To obtain the smoothed estimates $\Pr(S_t = j|\mathcal{Y}_n)$, note that at time n

$$\begin{aligned}
\Pr(S_n = j, S_{n-1} = i|\mathcal{Y}_n) &= \Pr(S_n = j|I_n)\Pr(S_{n-1} = i|S_n = j, \mathcal{Y}_n) \\
&\approx \Pr(S_n = j|\mathcal{Y}_n)\Pr(S_{n-1} = i|S_n = j, \mathcal{Y}_{n-1}) \\
&= \frac{\Pr(S_n = j|\mathcal{Y}_n)\Pr(S_{n-1} = i, S_n = j|\mathcal{Y}_{n-1})}{\Pr(S_n = j|\mathcal{Y}_{n-1})} \\
&= \frac{\Pr(S_n = j|\mathcal{Y}_n)\Pr(S_{n-1} = i|\mathcal{Y}_{n-1})\Pr(S_n = j|S_{n-1} = i)}{\Pr(S_n = j|\mathcal{Y}_{n-1})}
\end{aligned}$$

and, thus, the smoothed estimate $\Pr(S_{n-1} = j|\mathcal{Y}_n)$ can be obtained as

$$\Pr(S_{n-1} = j|\mathcal{Y}_n) = \sum_{i=1}^{k} \Pr(S_n = j, S_{n-1} = i|\mathcal{Y}_n)$$

This procedure can be repeated iteratively backward from time $n-1$ to time 1 to obtain the smoothed estimates of regime probabilities.

In `S+FinMetrics`, the filtered and smoothed regime probabilities can be obtained using the `SsfLoglikeMS` function with the optional argument `save.regm` set to `TRUE`. For example, the following commands plot the filtered and smoothed estimates of regime probabilities based on the fit `ndx.msar`:

```
> ndx.f = SsfLoglikeMS(log(ndx.rvol), ndx.ssf, save.rgm=T)
> par(mfrow=c(2,1))
> plot(timeSeries(ndx.f$regimes[,1], pos=positions(ndx.rvol)),
+ reference.grid=F, main="Filtered Low Vol Regime Prob")
> plot(timeSeries(ndx.f$regimes[,3], pos=positions(ndx.rvol)),
+ reference.grid=F, main="Smoothed Low Vol Regime Prob")
```

and the plot is shown in Figure 18.11. The smoothed regime probabilities suggest that there is actually an abrupt switch around the first quarter of 2000.

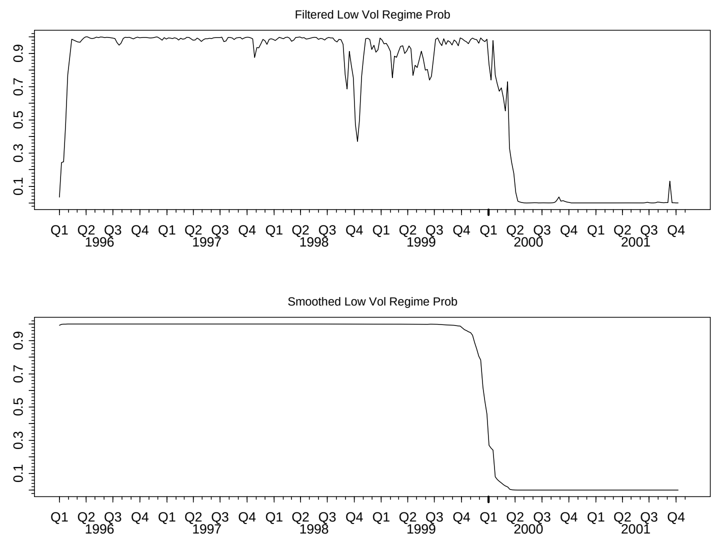


FIGURE 18.11. Filtered and smoothe regime probabilities of NASDAQ realized volatility.

18.6 An Extended Example: Markov Switching Coincident Index

The United States Department of Commerce periodically publishes the *Index of Coincident Economic Indicators* (CEI) based on four macroeconomic coincident variables, which provides a composite measure of the general state of the economy. The method used for the construction of the coincident index is *ad doc*, and the coincident index is subject to revisions after it is published. To provide a systematic probabilistic model for building an alternative coincident index, Stock and Watson (1991) have developed a dynamic factor model using a state space representation that models the coincident index as a common factor driving the four macroeconomic coincident variables: industrial production (IP), total personal income less transfer payments (PI), total manufacturing and trade sales (Sales), and employees on nonagricultural payrolls (Payroll). Stock and Watson (1991) show that their probabilistic coincident index matches very well with the Index of CEI compiled by the Department of Commerce.

Stock and Watson's dynamic factor model has been extended by Kim and Yoo (1995), Chauvet (1998), and Kim and Nelson (1998) to allow for Markov switching dynamics in the common factor that represents the coincident index. In addition to matching very well with the Index of CEI compiled by the Department of Commerce, the Markov switching coincident index is also shown to capture the economic expansions and recessions in the U.S. economy as classified by the National Bureau of Economic Research (NBER). Chauvet and Potter (2000) have developed coincident indicators for the U.S. stock market using the same methodology.

This section is provided to show how the Markov switching coincident index model can be represented as a Markov switching state space model and estimated using the functions in `S+FinMetrics`.

18.6.1 State Space Representation of Markov Switching Coincident Index Model

Since the levels of most macroeconomic variables are usually found to be nonstationary (e.g., see Nelson and Plosser, 1982), it is reasonable to assume that the coincident index representing the state of the economy is also nonstationary. Thus, in this example the growth rates of the four macroeconomic variables $\Delta\mathbf{y}_t$ are modeled and they are assumed to be driven by a common factor ΔC_t interpreted as the change in the coincident index:

$$\Delta\mathbf{y}_t = \boldsymbol{\beta} + \boldsymbol{\lambda}_1\Delta C_t + \boldsymbol{\lambda}_2\Delta C_{t-1} + \mathbf{e}_t \tag{18.43}$$

where $\Delta\mathbf{y}_t$, $\boldsymbol{\beta}$, $\boldsymbol{\lambda}_1$, $\boldsymbol{\lambda}_2$, and $\mathbf{e}_t$ are 4×1 vectors with

$$\boldsymbol{\lambda}_1 = \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \\ \lambda_{41} \end{bmatrix}, \quad \boldsymbol{\lambda}_2 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \lambda_{42} \end{bmatrix}$$

Thus, the four macroeconomic coincident variables are driven by the common factor ΔC_t and idiosyncratic components $\mathbf{e}_t$. Note that only the current value of ΔC_t affects the first three variables (IP, PI, and Sales) in $\Delta\mathbf{y}_t$, while both ΔC_t and ΔC_{t-1} affect the last variable (employees on nonagricultral payroll) because the employment data tend to lag other coincident variables.

The idiosyncratic components are assumed to be independent of each other and are assumed to follow simple AR(1) models:

$$\mathbf{e}_t = \boldsymbol{\Psi}\mathbf{e}_{t-1} + \boldsymbol{\epsilon}_t, \ \boldsymbol{\epsilon}_t \sim N(0, \boldsymbol{\sigma}^{\mathbf{2}}) \tag{18.44}$$

where $\boldsymbol{\Psi}$ is a diagonal matrix with $(\psi_1, \psi_2, \psi_3, \psi_4)$ on the diagonal and $\boldsymbol{\sigma}^2$ is a diagonal matrix with $(\sigma_1^2, \sigma_2^2, \sigma_3^2, \sigma_4^2)$ on the diagonal. The common factor ΔC_t is assumed to follow a Markov switching AR(2) process:

$$\Delta C_t = \delta_{S_t} + \phi_1\Delta C_{t-1} + \phi_2\Delta C_{t-2} + u_t, \ u_t \sim N(0, \sigma_C^2) \tag{18.45}$$

where the unobserved discrete state variable S_t follows a two-regime Markov chain, and only the intercept term δ_{S_t} is Markov switching. When the economy is in a recession ($S_t = 1$), the coincident index C_t grows at a slower rate δ_1; when the economy is in an expansion ($S_t = 2$), the coincident index C_t grows at a faster rate δ_2.

Note that in the above model, the intercept term $\boldsymbol{\beta}$ and δ_{S_t} are not separately identified and the variance term σ_C^2 cannot be separated from the coefficients $\boldsymbol{\lambda}_1$ and $\boldsymbol{\lambda}_2$. To make the model identifiable, the original data $\Delta\mathbf{y}_t$ are standardized to remove its mean and make it scale-free so that $\boldsymbol{\beta}$ can be set to 0. In addition, the error variance σ_C^2 for ΔC_t can be normalized to 1. Using $\boldsymbol{\alpha}_t = (\Delta C_t, \Delta C_{t-1}, \mathbf{e}_t, C_{t-1})$ as the continuous state vector, the Markov switching coincident index model in (18.43) and (18.45) can now be written in a state space form with the following representation:

$$\boldsymbol{\delta}_{S_t} = \begin{bmatrix} \delta_{S_{t+1}} \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad \boldsymbol{\Phi}_{S_t} = \begin{bmatrix} \phi_1 & \phi_2 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \psi_1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \psi_2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \psi_3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \psi_4 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ \gamma_1 & 0 & 1 & 0 & 0 & 0 & 0 \\ \gamma_2 & 0 & 0 & 1 & 0 & 0 & 0 \\ \gamma_3 & 0 & 0 & 0 & 1 & 0 & 0 \\ \gamma_{41} & \gamma_{42} & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

and

$$\boldsymbol{\Omega} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \sigma_1^2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \sigma_2^2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \sigma_3^2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \sigma_4^2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Note that $C_t = \Delta C_t + C_{t-1}$ is also included as one of the state variables, but does not enter the measurement equation for the observables $\Delta\mathbf{y}_t$. By including C_t as one of the state variables, filtered estimates of C_t can be readily obtained from Kim's filtering algorithm.

By collecting the unknown model parameters in the vector $\boldsymbol{\Theta} = (\delta_1, \delta_2, \phi_1, \phi_2, \psi_1, \psi_2, \psi_3, \psi_4, \lambda_1, \lambda_2, \lambda_3, \lambda_{41}, \lambda_{42}, \sigma_1^2, \sigma_2^2, \sigma_3^2, \sigma_4^2, P_{12}, P_{21})$, the following function takes such a vector and returns the state space representation of the model in S-PLUS:

```
GetSsfCoinIndex = function(parm) {
  parm = as.vector(parm)
  mDelta = mDelta.other = rep(0, 11)
  mDelta[1] = parm[1]
  mDelta.other[1] = parm[1] + exp(parm[2])
#
  AR.C1 = 2/(1+exp(-parm[3])) - 1
  AR.C2 = 2/(1+exp(-(parm[3]+exp(parm[4])))) - 1
#
  AR.e1 = 2/(1+exp(-parm[5])) - 1
  AR.e2 = 2/(1+exp(-parm[6])) - 1
  AR.e3 = 2/(1+exp(-parm[7])) - 1
  AR.e4 = 2/(1+exp(-parm[8])) - 1
#
  mPhi = matrix(0, 11, 7)
  mPhi[1,1:2] = c(AR.C1+AR.C2, -AR.C1*AR.C2)
  mPhi[2,1] = 1
  mPhi[3,3] = AR.e1
  mPhi[4,4] = AR.e2
  mPhi[5,5] = AR.e3
  mPhi[6,6] = AR.e4
  mPhi[7,1] = mPhi[7,7] = 1
#
  mPhi[8:10,1] = parm[9:11]
  mPhi[11,1:2] = parm[12:13]
  mPhi[8,3] = mPhi[9,4] = mPhi[10,5] = mPhi[11,6] = 1
#
  mOmega = matrix(0, 11, 11)
  mOmega[1,1] = 1
  mOmega[3,3] = exp(parm[14])
  mOmega[4,4] = exp(parm[15])
  mOmega[5,5] = exp(parm[16])
  mOmega[6,6] = exp(parm[17])
#
  mTrans = matrix(0, 2, 2)
  mTrans[1,2] = 1/(1+exp(-parm[18]))
  mTrans[1,1] = 1-mTrans[1,2]
  mTrans[2,1] = 1/(1+exp(-parm[19]))
  mTrans[2,2] = 1-mTrans[2,1]
#
  mSigma = matrix(0, 8, 7)
  mSigma[1:7, 1:7] = diag(1e+6, 7)
  ans = list(mDelta=mDelta, mDelta.other=mDelta.other,
             mSigma=mSigma, mOmega=mOmega,
             mPhi=mPhi, mTrans=mTrans)
  CheckSsf(ans)
}
```

A few comments on the function `GetSsfCoinIndex` are in order:

1. The second parameter `parm[2]` is actually $\log(\delta_2 - \delta_1)$. By employing this transformation, δ_2 is guaranteed to be greater than δ_1; thus, the first regime can be identified as the recessionary regime and the second as the expansionary regime.

2. Like in the Markov switching AR(2) model for the NASDAQ realized volatility, instead of directly estimating the AR(2) coefficients for ΔC_t, the two real characteristic roots are estimated and the first root is constrained to be the smaller one. By constraining the real characteristic roots to lie within the unit circle, the estimated AR(2) process is guaranteed to be stationary and aperiodic.

3. The AR(1) coefficients for the idiosyncratic components are transformed to guarantee that they lie within $(-1, 1)$, and the corresponding AR processes are stationary.

4. The logarithmic variances $\log \sigma_i^2$ $(i = 1, 2, 3, 4)$ are estimated because they are unbounded.

5. Like in the Markov switching AR(2) model for the NASDAQ realized volatility, the transition probabilities P_{12} and P_{21} are transformed to guarantee that they lie within $(0, 1)$.

6. Finally, diffuse priors on the state vector $\boldsymbol{\alpha}_t$ are employed by setting the top 7×7 block of `mSigma` to a diagonal matrix with large values on the diagonal and zero in the last row.

18.6.2 *Approximate MLE of Markov Switching Coincident Index*

To fit the above Markov switching model to the four coincident variables, the data are first standardized for model identification and better numerical convergence:

```
> DOC.dat = getReturns(DOC.ts[,1:4], percentage=T)
> DOC.dat@data = t(t(DOC.dat@data) - colMeans(DOC.dat@data))
> DOC.dat@data = t(t(DOC.dat@data) / colStdevs(DOC.dat@data))
```

then the `SsfFitMS` function can be used to fit the model with the following starting values:

```
> DOC.start = c(-1.5, 0.6, 0.3, 0.1, .1, .1, .1, .1, 0.3,
+      0.3, 0.3, 0.3, 0.1, -.5, -.5, -.5, -.5, -1.5, -3)
+ names(DOC.start) = c("mu1", "mu2", "phi1", "phi2", "psi1",
+      "psi2", "psi3", "psi4", "L1", "L2", "L3", "L41",
+      "L42", "s1", "s2", "s3", "s4", "p", "q")
```

```
> DOC.fit = SsfFitMS(DOC.start, DOC.dat, GetSsfCoinIndex,
+      l.start=13, trace=T)
> summary(DOC.fit)
Log-likelihood:  -1998.11
432 observations
Parameters:
       Value Std. Error t value
 mu1 -1.5650    0.30180  -5.187
 mu2  0.6053    0.16900   3.582
phi1 -0.8171    0.20610  -3.965
phi2  0.7124    0.17010   4.187
psi1  0.3711    0.14940   2.484
psi2 -0.6070    0.10590  -5.731
psi3 -0.5169    0.10930  -4.729
psi4 -0.7584    0.18340  -4.135
  L1  0.5059    0.03832  13.200
  L2  0.2977    0.03193   9.322
  L3  0.3480    0.03406  10.220
 L41  0.4443    0.04013  11.070
 L42  0.1966    0.03504   5.610
  s1 -1.1590    0.12180  -9.517
  s2 -0.2758    0.07225  -3.817
  s3 -0.4155    0.07624  -5.449
  s4 -1.3940    0.15220  -9.156
   p -1.9560    0.52340  -3.738
   q -3.7600    0.43460  -8.652

Convergence:  RELATIVE FUNCTION CONVERGENCE
```

Note that the optional argument `l.start` to `SsfFitMS` is used to start log-likelihood evaluation from the 13th observation. From the summary output, it can be seen that all the estimated model parameters are significantly different from zero.

To transform the parameters into the original model form, simply call the `GetSsfCoinIndex` function with the estimated parameters:

```
> DOC.ssf = GetSsfCoinIndex(DOC.fit$parameters)
> c(DOC.ssf$mDelta[1], DOC.ssf$mDelta.other[1])
[1] -1.565361  0.266435
> print(DOC.ssf$mPhi, digits=3)
      [,1]  [,2]  [,3]   [,4]   [,5]   [,6] [,7]
 [1,] 0.158 0.211 0.000  0.000  0.000  0.000    0
 [2,] 1.000 0.000 0.000  0.000  0.000  0.000    0
 [3,] 0.000 0.000 0.183  0.000  0.000  0.000    0
 [4,] 0.000 0.000 0.000 -0.295  0.000  0.000    0
 [5,] 0.000 0.000 0.000  0.000 -0.253  0.000    0
```

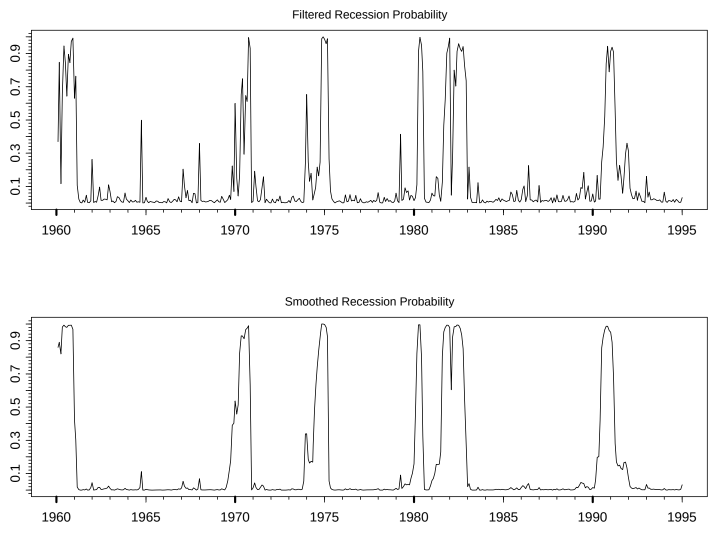


FIGURE 18.12. Filtered and smoothed recession probabilities of Markov switching coincident index.

```
 [6,] 0.000 0.000 0.000  0.000  0.000 -0.362    0
 [7,] 1.000 0.000 0.000  0.000  0.000  0.000    1
 [8,] 0.506 0.000 1.000  0.000  0.000  0.000    0
 [9,] 0.298 0.000 0.000  1.000  0.000  0.000    0
[10,] 0.348 0.000 0.000  0.000  1.000  0.000    0
[11,] 0.444 0.197 0.000  0.000  0.000  1.000    0
```

The growth rate of ΔC_t in a recession is estimated to be -1.57, and the growth rate in an expansion is estimated to be 0.27. Although the growth rates of the four macroeconomic variables are positively correlated with the Markov switching coincident index, only the idiosyncractic component of industrial production has a positive AR(1) coefficient and all other idiosyncratic components have a negative AR(1) coefficient.

To obtain the filtered and smoothed regime probabilities, simply call the `SsfLoglikeMS` function with the estimated state space representation and set the optional argument `save.rgm` to `TRUE`:

```
> DOC.f = SsfLoglikeMS(DOC.dat, DOC.ssf, save.rgm=T,
+ l.start=13)
> DOC.dates = positions(DOC.dat)[-(1:12)]
> filt.p = timeSeries(DOC.f$regimes[,1], pos=DOC.dates)
> smoo.p = timeSeries(DOC.f$regimes[,3], pos=DOC.dates)
> par(mfrow=c(2,1))
```

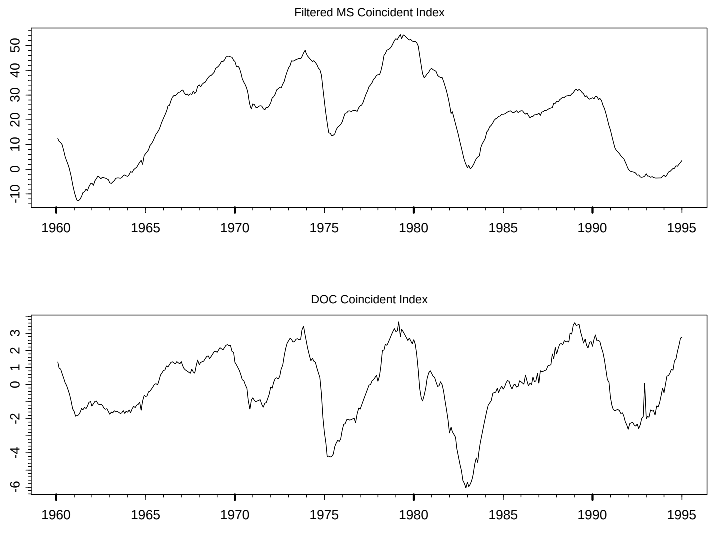


FIGURE 18.13. Filtered Markov switching coincident index and DOC coincident index.

```
> plot(filt.p, reference.grid=F,
+ main="Filtered Recession Probability")
> plot(smoo.p, reference.grid=F,
+ main="Smoothed Recession Probability")
```

and Figure 18.12 shows the filtered and smoothed probabilities for the recession regime.

To visualize the estimated Markov switching coincident index, note that the object `DOC.f` also has a `states` component:

```
> names(DOC.f)
[1] "loglike" "err"     "regimes" "states"
```

which contains the filtered estimates of the states $\boldsymbol{\alpha}_{t|t}^{(j)}$ for $j = 1, 2$. Since there are seven state variables in the model, the first seven columns correspond to $\boldsymbol{\alpha}_{t|t}^{(1)}$ and the next seven columns correspond to $\boldsymbol{\alpha}_{t|t}^{(2)}$. The following commands plot the weighted average of filtered estimates of C_t and compare it with the coincident index compiled by the U.S. Department of Commerce:

```
> DOC.index = lm(DOC.ts@data[,5]~I(1:433))$residuals[-(1:13)]
> filt.ci = rowSums(DOC.f$state[,c(7,14)]*DOC.f$regime[,1:2])
> filt.ci = timeSeries(filt.ci, pos=DOC.dates)
> plot(filt.ci, reference.grid=F,
+ main="Filtered MS Coincident Index")
> doc.ci = timeSeries(DOC.index, pos=DOC.dates)
> plot(doc.ci, reference.grid=F,
+ main="DOC Coincident Index")
```

and the plot is shown in Figure 18.13. Note that since the Markov switching coincident index is estimated with demeaned data, a time trend is also removed from the coincident index `DOC.ts[,5]` compiled by the U.S. Department of Commerce. In general, both series share the same pattern, although the Markov switching coincident index seems to be smoother.

18.7 References

Andersen, T., T. Bollerslev, F.X. Diebold and H. Ebens (2001). "The Distribution of Realized Stock Return Volatility," *Journal of Financial Economics*, 61, 43-76.

Aguilar, O. and M. West (2000). "Bayesian Dynamic Factor Models and Portfolio Allocation," *Journal of Business and Economic Statistics*, 18(3), 338-357.

Barnett, W.A., R.A. Gallant, M.J. Hinich, J.A. Jungeilges, D.T. Kaplan and M.J. Jensen (1997). "A Single-blind Controlled Competition Among Tests for Nonlinearity and Chaos," *Journal of Econometrics*, 82, 157-192.

Brock, W.A., W.D. Dechert and J.A. Scheinkman (1987). "A Test for Independence Based on the Correlation Dimension," unpublished manuscript, Department of Economics, University of Wisconsin, Madison.

Brock, W.A., W.D. Dechert, J.A. Scheinkman and B. LeBaron (1996). "A Test for Independence Based on the Correlation Dimension," *Econometric Reviews*, 15, 197-235.

Brock, W.A., D.A. Hsieh and B. LeBaron (1991). *Nonlinear Dynamics, Chaos, and Instability: Statistical Theory and Economic Evidence.* MIT Press.

Brock, W.A. and S. Potter (1993). "Nonlinear Time Series and Macroeconomics," in G.S. Maddala, C. R. Rao and H. D. Vinod (eds.), *Handbook of Statistics, Vol. II.* North-Holland, Amsterdam.

Caporale, G.M., C. Ntantamis, T. Pantelidis and N. Pittis (2004). "The BDS Test As a Test for the Adequacy of a GARCH(1,1) Specification: A Monte Carlo Study," Working Paper, Brunel University.

Chauvet, M. (1998). "An Econometric Characterization of Business Cycle Dynamics with Factor Structure and Regime Switching," *International Economic Review*, 39(4), 969-996.

Chauvet, M. and S. Potter (2000). "Coincident and Leading Indicators of the Stock Market," *Journal of Empirical Finance*, 7, 87-111.

Davidson, R. and J. G. MacKinnon (1985). "Heteroskedasticity-Robust Tests in Regressions Directions," *Annales de l'INSEE*, 59/60, 183-218.

de Lima, P.J.F. (1996). "Nuisance Parameter Free Properties of Correlation Integral Based Statistics," *Econometric Reviews*, 15, 237-259.

Fernandes, M. and P.-Y. Preumont (2002). "The Finite-Sample Size of the BDS Test for GARCH Standardized Residuals," unpublished manuscript, Department of Economics, Queen Mary, University of London.

Franses, P.H. and D. van Dijk (2000). *Non-Linear Time Series Models in Empirical Finance*. Cambridge University Press, Cambridge.

Granger, C.W.J. and T. Teräsvirta (1993). *Modelling Nonlinear Economic Relationships*. Oxford University Press, Oxford.

Hamilton, J.D. 1989. "A New Approach to the Economic Analysis of Nonstationary Time Series Subject to Changes in Regime," *Econometrica*, 57, 357-384.

Hamilton, J.D. (1994). *Time Series Analysis*. Princeton University Press, Princeton.

Hansen, B.E. (1996). "Inference When a Nuisance Parameter is Not Identified Under the Null Hypothesis," *Econometrica*, 64, 413-430.

Hansen, B.E. (1997). "Inference in TAR Models," *Studies in Nonlinear Dynamics and Econometrics*, 2, 1-14.

Hansen, B.E. (1999). "Testing for Linearity," *Journal of Economic Surveys*, 13(5), 551-576.

Kim, C.J. (1994). "Dynamic Linear Models with Markov-Switching," *Journal of Econometrics*, 60, 1-22.

Kim, C.-J. and C.R. Nelson (1998). "Business Cycle Turning Points, a New Coincident Index, and Tests of Duration Dependence Based on a Dynamic Factor Model with Regime-Switching," *Review of Economics and Statistics*, 80, 188-201.

KIM, C.-J. AND C.R. NELSON (1999). *State-Space Models with Regime-Switching: Classical and Gibbs-Sampling Approaches with Applications.* MIT Press, Cambridge, MA.

KIM, M.-J. AND J.-S. YOO (1995). "New Index of Coincident Indicators: A Multivariate Markov Switching Factor Model Approach," *Journal of Monetary Economics*, 36, 607-630.

KIM, S., N. SHEPHARD AND S. CHIB (1998). "Stochastic Volatility: Likelihood Inference and Comparison with ARCH Models," *Review of Economic Studies*, 65, 361-393.

LEBARON, B. (1997). "A Fast Algorithm for the BDS Statistic," *Studies in Nonlinear Dynamics and Econometrics*, 2, 53-59.

LUNDBERGH, S. AND T. TERÄSVIRTA (1998). "Modelling Economic High-Frequency Time Series with STAR-GARCH Models," Working Paper Series in Economics and Finance No. 291, Stockholm School of Economics.

LUNDBERGH, S. AND T. TERÄSVIRTA (2002). "Forecasting with Smooth Transition Autoregressive Models," in M. P. Clements and D. F. Hendry (eds.), *A Companion to Economic Forecasting.* Blackwell Publishers, London.

LUUKKONEN, R, P. SAIKKONEN AND T. TERÄSVIRTA (1988). "Testing Linearity Against Smooth Transition Autoregressive Models," *Biometrika* 75, 491–499.

NELSON, C.R. AND C.I. PLOSSER (1982). "Trends and Random Walks in Macroeconomic Time Series: Some Evidence and Implications," *Journal of Monetary Economics*, 10, 139-162.

TERÄSVIRTA, T. (1994). "Specification, Estimation, and Evaluation of Smooth Transition Autoregressive Models," *Journal of the American Statistical Association*, 89, 208–218.

STOCK, J.H. AND M.W. WATSON (1991). "A Probability Model of the Coincident Economic Indicators," in K. Lahiri and G.H. Moore (eds.), *Leading Economic Indicators: New Approaches and Forecasting Records.* Cambridge University Press, Cambridge.

TONG, H. (1978). "On a Threshold Model," in C.H. Chen (ed.), *Pattern Recognition and Signal Processing.* Sijhoff & Noordhoff, Amsterdam.

TONG, H. (1990). *Non-Linear Time Series: A Dynamical System Approach.* Oxford University Press, Oxford.

Tsay, R.S. (1989). "Testing and Modeling Threshold Autoregressive Processes," *Journal of the American Statistical Association*, 84(405), 231-240.

van Dijk, D., T. Teräsvirta and P.H. Franses (2002). "Smooth Transition Autoregressive Models – A Survey of Recent Developments," *Econometric Reviews*, 21 (1), 1-47.

19 Copulas

19.1 Introduction

Capturing comovement between financial asset returns with linear correlation has been the staple approach in modern finance since the birth of Harry Markowitz's portfolio theory. Linear correlation is the appropriate measure of dependence if asset returns follow a multivariate normal (or elliptical) distribution. However, the statistical analysis of the distribution of individual asset returns frequently finds fat tails, skewness, and other non-normal features. If the normal distribution is not adequate, then it is not clear how to appropriately measure the dependence between multiple asset returns. Fortunately, the theory of copulas provides a flexible methodology for the general modeling of multivariate dependence. As Cherubini, Luciano, and Vecchiato (2004) state the following in the introduction to their book: "the copula function methodology has become the most significant new technique to handle the co-movement between markets and risk factors in a flexible way."

This chapter gives an overview of the copula function methodology for modeling arbitrary bivariate distributions of asset returns. Attention is restricted to bivariate distributions because the mathematical and statistical theory for bivariate copulas is more complete than it is for multivariate copulas. The bivariate theory also lays the foundation for the general multivariate theory. Section 19.2 provides a motivating example to introduce the main issues and problems. Section 19.3 defines copulas and gives some basic properties. The parametric copula classes and families implemented in S+FinMetrics are reviewed in Section 19.4. Section 19.5 describes fitting copulas to data, and Section 19.6 discusses risk management using copulas.

The mathematical theory of copulas is covered in the monographs by Joe (1997) and Nelsen (1999). Excellent treatments of copula methods in finance are given in Carmona (2004) and Cherubini, Luciano, and Vecchiato (2004). The latter book contains numerous examples of using copulas for the analysis of credit risk and for the pricing of derivative securities.

The S+FinMetrics functions for analyzing bivariate copulas are based on the functions in the *EVANESCE* (Extreme Value ANalysis Employing Statistical Copula Estimation) library written by René Carmona and Julia Morrison and described in Carmona and Morrison (2001), Morrison (2001), and Carmona (2004).

19.2 Motivating Example

As a motivating example, consider the problem of modeling the unconditional bivariate distribution of the daily log returns on BMW and Siemens stock, contained in the S+FinMetrics "timeSeries" bmw and siemens, over the period January 2, 1973 through July 23, 1996.[1] To simplify the following analysis, these series are combined in the timeSeries "german.ts":

```
> german.ts = seriesMerge(bmw,siemens)
```

Time series plots of the returns are given in Figure 19.1. The two return series behave similarly over time, exhibit periods of high and low volatility, and sometimes take on extremely large and small values. For the analysis in this chapter, the time dependent nature of the return volatility will be ignored. Modeling time-dependent volatility is covered in Chapters 7, 13, and 22.

The two return series have a particular feature that complicates the analysis of their distributions. This feature is the joint presence of a large number of zero returns, which result from the prices being constant across certain days. The percentage of common zero returns is

```
> zero.idx = (bmw == 0 & siemens == 0)
> sum(zero.idx)/length(bmw)
[1] 0.05076473
```

The presence of these zero returns causes the empirical marginal and joint distribution functions to have large jumps at 0. Following Carmona (2004), before analyzing the distribution of returns, the returns are adjusted to eliminate these zero returns. In what follows, the nonzero returns are identified by the logical index variable nz.idx:

[1]Carmona (2004), Chapter 2, presents a similar analysis using the S+FinMetrics "timeSeries" columbia.coffee and brazil.coffee.

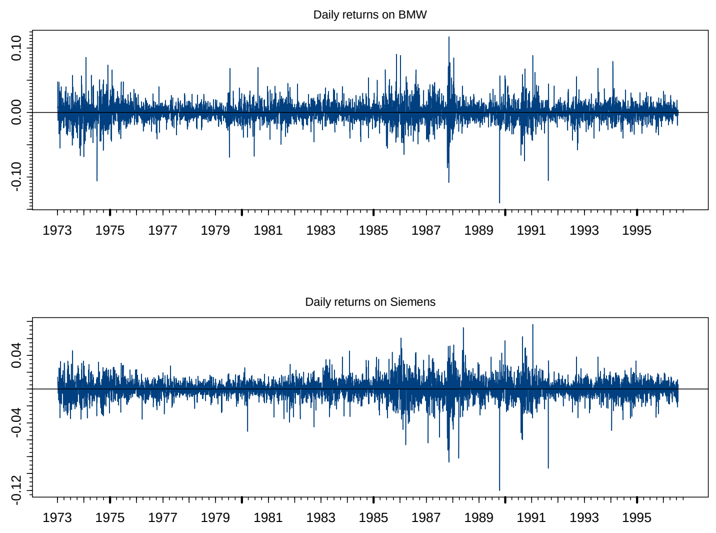


FIGURE 19.1. Daily log returns on BMW and Siemens stock.

```
> nz.idx = (seriesData(bmw) != 0 & seriesData(siemens) != 0)
```

Regarding the marginal distribution of returns, Figure 19.2 shows the qq plots against the normal distribution computed using the `S+FinMetrics` function `qqPlot`. The strong departure from linearity at the ends of the qq plots indicates non-normal fat-tailed behavior. To be sure, the Jarque-Bera test

```
> normalTest(german.ts[nz.idx,],method="jb")

Test for Normality: Jarque-Bera

Null Hypothesis: data is normally distributed

Test Statistics:
                bmw   siemens
Test Stat  8371.159 10433.032
  p.value     0.000     0.000

Dist. under Null: chi-square with 2 degrees of freedom
   Total Observ.: 5350
```

clearly rejects the null hypothesis that the individual returns are normally distributed.

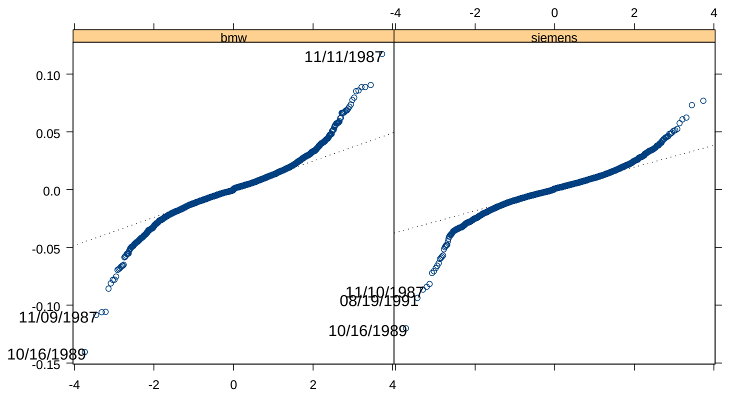


FIGURE 19.2. Normal qq plots of the daily log returns on BMW and Siemens stocks.

Given that the marginal distributions of the two return series are not normal, it would be surprising if the bivariate normal is a good characterization of the joint distribution. To see this, Figure 19.3 shows a scatter plot of the actual returns together with a scatter plot of simulated bivariate normal data calibrated to the actual data. The simulated data is created using

```
> mu.hat = colMeans(german.ts[nz.idx,])
> Sigma.hat = var(german.ts[nz.idx,])
> nobs = numRows(german.ts[nz.idx,])
> set.seed(0)
> german.sim = rmvnorm(nobs,mean=mu.hat,cov=Sigma.hat)
> colIds(german.sim) = colIds(german.ts)
```

The dependence captured by the bivariate normal distribution is completely described by the (Pearson) correlation coefficient

```
> cor(german.ts[nz.idx,])[1,2]
[1] 0.6517873
```

which indicates a moderately strong positive linear dependence. The simulated normal data matches the joint behavior of returns in the middle of the distribution fairly well, but does not capture the observed positive dependence in the tails of the distribution. Since the bivariate normal dis-

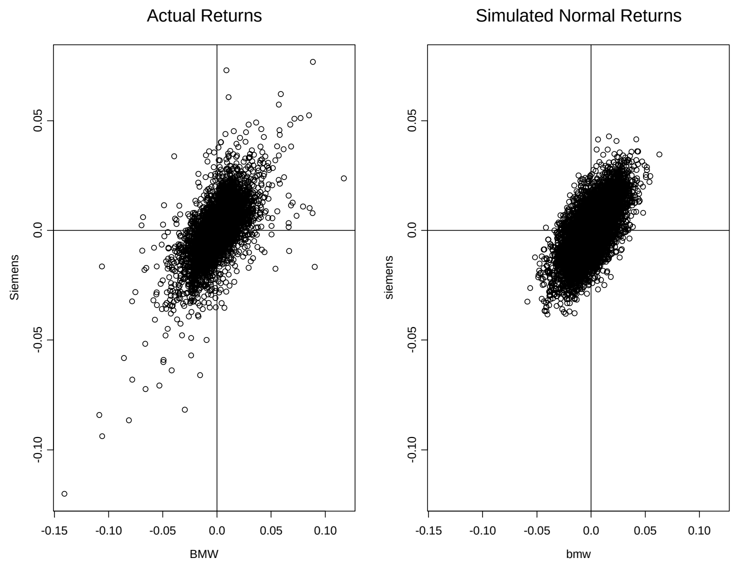


FIGURE 19.3. Scatter plots of actual returns and simulated bivariate normal returns for BMW and Siemens stocks.

tribution does not adequately describe the joint behavior of returns, the correlation coefficient *may not* be the proper measure of dependence.

As described in Chapter 5, the tails of the marginal distribution of asset returns can be successfully modeled using the generalized Pareto distribution (GPD). As demonstrated in Carmona (2004), standard nonparametric techniques based on the empirical distribution function can be used to model the center of the distribution. This modeling strategy leads to a *semiparametric model* (parametric in the tails and nonparametric in the center) for the marginal distribution. The `S+FinMetrics/ EVANESCE` function `gpd.tail` implements such a semiparametric model. It fits separate parametric GPDs to the lower and upper tails and uses the empirical CDF to fit the remaining part of the distribution.

Fitting semiparametric GPD models to the BMW and Siemens returns requires specification of the lower and upper thresholds. These thresholds may be inferred from sample mean excess plots and verified by inspecting plots showing how the GPD shape parameters vary as a function of the specified thresholds. Figures 19.4 and 19.5 give these plots for the BMW and Siemens returns. The plots suggest lower and upper threshold values of -0.015 and 0.015, respectively, for BMW and lower and upper threshold values of -0.01 and 0.01, respectively, for Siemens. Using these thresh-

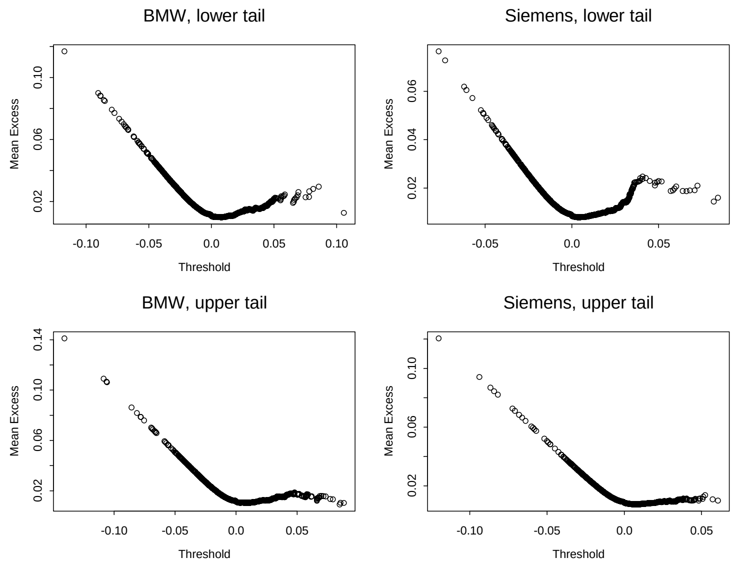


FIGURE 19.4. Mean excess plots for the lower and upper tails of daily log returns on BMW and Siemens stock.

old values, the semiparametric GPDs models for BMW and Siemens are estimated using[2]

```
> gpd.bmw.fit2 = gpd.tail(bmw[nz.idx], upper = 0.015,
+                         lower = -0.015)
> gpd.bmw.fit2
Generalized Pareto Distribution Fit --

Total of  5350  observations

Upper Tail Estimated with ml --
Upper Threshold at  0.015  or  12.93 % of the data
ML estimation converged.
Log-likelihood value:  2457


Parameter Estimates, Standard Errors and t-ratios:
      Value Std.Error t value
```

[2]The L-moment fitting method is used for the Siemens upper tail estimates because the default ML estimate did not converge. The function `gpd.tail` does not return estimated standard errors. Use the function `gpd` to get estimated standard errors.

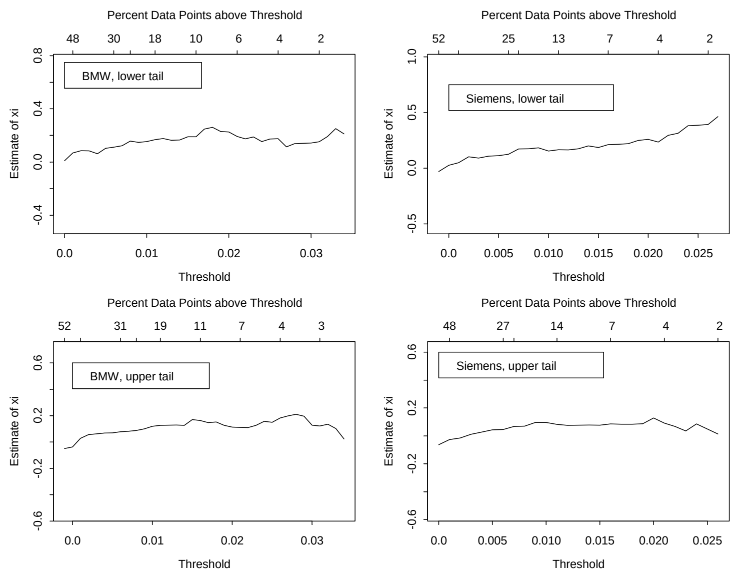


FIGURE 19.5. Estimated GPD shape parameter ξ, as a function of increasing thresholds, for the lower and upper tails of daily log returns on BMW and Siemens stocks.

```
  xi 0.1701     NA        NA
beta 0.0089     NA        NA

Lower Tail Estimated with ml --
Lower Threshold at  -0.015  or  11.5 % of the data
ML estimation converged.
Log-likelihood value:  2189


Parameter Estimates, Standard Errors and t-ratios:
      Value Std.Error t value
  xi 0.1901     NA        NA
beta 0.0087     NA        NA

> gpd.siemens.fit2 = gpd.tail(siemens[nz.idx], upper = 0.01,
+                    lower = -0.01, upper.method="lmom")
> gpd.siemens.fit2
Generalized Pareto Distribution Fit --

Total of  5350  observations
```

```
Upper Tail Estimated with lmom --
Upper Threshold at  0.01  or  16.19 % of the data
Log-likelihood value:  NA


Parameter Estimates, Standard Errors and t-ratios:
      Value Std.Error t value
  xi 0.1153     NA        NA
beta 0.0066     NA        NA

Lower Tail Estimated with ml --
Lower Threshold at  -0.01  or  14.45 % of the data
ML estimation converged.
Log-likelihood value:  2923


Parameter Estimates, Standard Errors and t-ratios:
      Value Std.Error t value
  xi 0.1543     NA        NA
beta 0.0072     NA        NA
```

To evaluate the fit of the GPD model in the tails of the distribution, the function `gpd.tail` automatically creates qq plots of excesses over the specified lower and upper thresholds against the quantiles of the fitted GPD model. The linearity of the left parts of the plots (not shown) indicates a good fit to the tails of the distributions. The GPD fit may also be evaluated graphically by examining the plots of the tails of the distributions along with the fitted GPD tails. These plots, produced using the `S+FinMetrics/EVANESCE` function `tailplot`, are given in Figure 19.6 and indicate good fits to the tails of the BMW and Siemens return distributions.

As a further reality check on the overall fit of the semiparametric GPD models to the entire distribution of the BMW and Siemens returns, simulated returns from the fitted models are generated using

```
> nobs = numRows(bmw[nz.idx])
> set.seed(123)
> bmw.gpd.sim = gpd.2q(runif(nobs),gpd.bmw.fit2)
> siemens.gpd.sim = gpd.2q(runif(nobs),gpd.siemens.fit2)
```

In the above code, the `S+FinMetrics/EVANESCE` function `gpd.2q` is used to compute quantiles from a fitted semiparametric GPD model. Figure 19.7 shows qq plots of the simulated returns against the actual returns created using

```
> par(mfrow=c(1,2))
>    qqplot(seriesData(bmw[nz.idx]),bmw.gpd.sim,
+    xlab="Actual returns", ylab="Simulated returns")
```

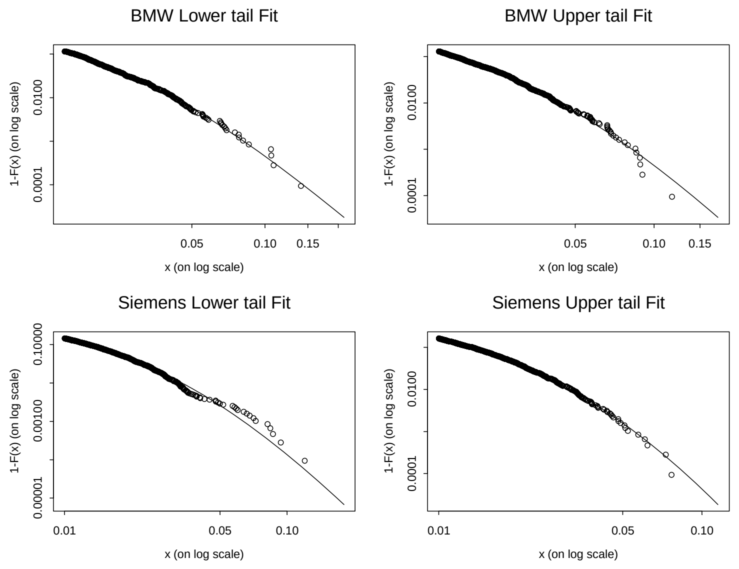


FIGURE 19.6. Estimated tails from GPD models fit to lower and upper tails of the daily log returns on BMW and Siemens stock.

```
>    abline(0,1)
>    title("BMW")
>    qqplot(seriesData(siemens[nz.idx]),siemens.gpd.sim,
+    xlab="Actual returns", ylab="Simulated returns")
>    abline(0,1)
>    title("Siemens")
> par(mfrow=c(1,1))
```

The approximate linearity of these plots verifies that the fitted semiparametric GPD models adequately describe the marginal distributions of the BMW and Siemens returns. The simulated returns from the fitted semiparametric GPD models capture the marginal behavior of actual returns, but not the joint behavior since the uniform variables used to simulate each return series are independent. To see this, Figure 19.8 gives the scatter plot of the simulated returns. The occurrence of extreme values observed in the actual returns from Figure 19.3 are reproduced for the marginal distributions, but the dependence of the extreme values is not produced for the bivariate distribution.

The modeling problem introduced in this example involves modeling the individual marginal distributions of returns to capture fat-tailed behavior, and modeling the dependence structure of the joint behavior to capture the observed dependence – especially in the tails of the joint distribution. The

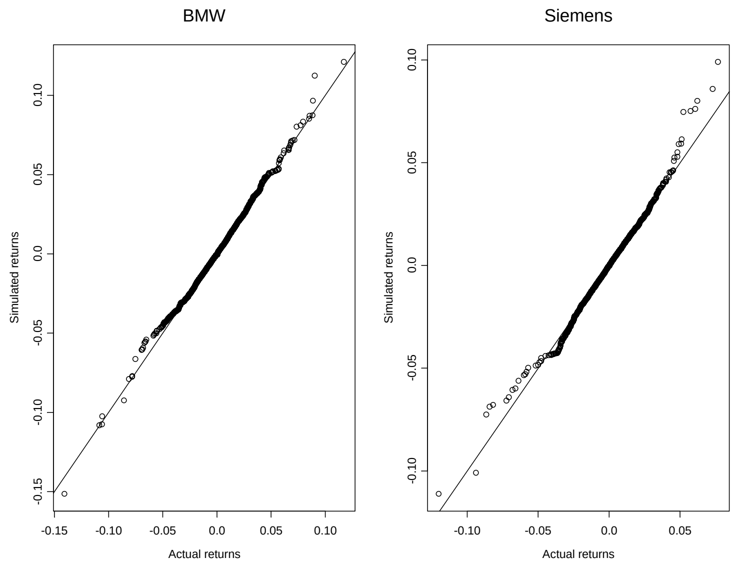


FIGURE 19.7. qq plots of actual returns versus simulated returns from fitted semiparametric GPD models.

marginal distributions are shown to be adequately modeled using semiparametric GPD models. However, it is not clear how to model the joint dependence given models for the marginal distributions. As the following sections will show, a flexible way to successfully model the joint behavior of financial returns, after modeling the marginal distributions, is with copulas.

19.3 Definitions and Basic Properties of Copulas

This section lays out the basic mathematical and statistical theory for bivariate copulas.

19.3.1 Properties of Distributions

Let X be a random variable with distribution function (df) $F_X(x) = \Pr(X \leq x)$. For simplicity and ease of exposition, F_X is assumed to be continuous and differentiable. The density function $f_X(x)$ is defined by

$$F_X(x) = \int_{-\infty}^{x} f_X(z)\, dz$$

so that $f_X = F'_X$.

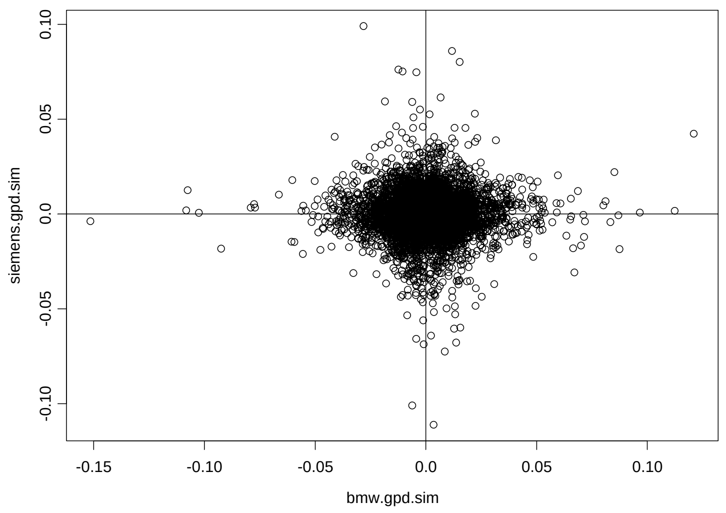


FIGURE 19.8. Scatter plot of simulated bivariate returns from fitted semiparametric GPD models.

Let F_X^{-1} denote the quantile function

$$F_X^{-1}(\alpha) = \inf\{x \mid F_X(x) \geq \alpha\}$$

for $\alpha \in (0,1)$. The following are useful results from probability theory:

- $F_X(x) \sim U(0,1)$, where $U(0,1)$ denotes a uniformly distributed random variable on $(0,1)$.
- If $U \sim U(0,1)$ then $F_X^{-1}(U) \sim F_X$.

The latter result gives a simple way to simulate observations from F_X provided F_X^{-1} is easy to calculate.

Let X and Y be random variables with marginal dfs (margins) F_X and F_Y, respectively, and joint df

$$F_{XY}(x,y) = \Pr(X \leq x, Y \leq y)$$

In general, the marginal dfs may be recovered from the joint df via

$$F_X(x) = F_{XY}(x,\infty),\ F_Y(y) = F_{XY}(\infty, y)$$

The joint density f_{XY} is defined by

$$f_{XY}(x,y) = \frac{\partial^2}{\partial x\,\partial y} F_{XY}(x,y)$$

The random variables X and Y are independent if

$$F_{XY}(x,y) = F_X(x)F_Y(y) \tag{19.1}$$

for all values of x and y.

19.3.2 Copulas and Sklar's Theorem

A *bivariate copula* is a bivariate df C defined on $I^2 = [0,1] \times [0,1]$ with uniformly distributed margins; that is,

$$C(u,v) = \Pr(U \leq u, V \leq v)$$

where $U, V \sim U(0,1)$. As a result, it satisfies the following properties:

- $C(u,0) = C(0,v) = 1$, $C(1,v) = v$, $C(u,1) = u$ for every $u, v \in [0,1]$.
- $0 \leq C(u,v) \leq 1$.
- For every $u_1 \leq u_2$, $v_1 \leq v_2$, and $u_1, u_2, v_1, v_2 \in [0,1]$, the following inequality holds: $C(u_1,v_1) - C(u_2,v_1) - C(u_1,v_2) + C(u_2,v_2) \geq 0$.

The last property ensures that $\Pr(u_1 \leq U \leq u_2, v_1 \leq V \leq v_2) \geq 0$.

Although not immediately apparent from its definition, the idea of a copula is to separate a joint df F_{XY} into a part that describes the dependence between X and Y and parts that only describe the marginal behavior. To see this, X and Y may be transformed into uniform random variables U and V via $U = F_X(X)$ and $V = F_Y(Y)$. Let the joint df of (U,V) be the copula C. Then, it follows that

$$\begin{aligned} F_{XY}(x,y) &= \Pr(X \leq x, Y \leq y) \\ &= \Pr(F_X(X) \leq F_X(x), F_Y(Y) \leq F_Y(y)) \\ &= C(F_X(x), F_Y(x)) = C(u,v) \end{aligned}$$

and so the joint df F_{XY} can be described by the margins F_X and F_Y and the copula C. The copula C captures the dependence structure between X and Y.

Sklar's Theorem

Let F_{XY} be a joint df with margins F_X and F_Y. Then there exists a copula C such that for all $x, y \in [-\infty, \infty]$,

$$F_{XY}(x,y) = C(F_X(x), F_Y(y)) \tag{19.2}$$

If F_X and F_Y are continuous, then C is unique. Otherwise, C is uniquely defined on Range $F_X \times$ Range F_Y. Conversely, if C is a copula and F_X and

F_Y are univariate dfs, then F_{XY} defined in (19.2) is a joint df with margins F_X and F_Y.

Sklar's theorem (Sklar, 1959) shows that the copula associated with a continuous df F_{XY} couples the margins F_X and F_Y with a dependence structure to uniquely create F_{XY}. As such, it is often stated that the copula of X and Y is the df C of $F_X(x)$ and $F_Y(y)$.

The copula C of X and Y has the property that it is invariant to strictly increasing transformations of the margins F_X and F_Y; that is, if T_X and T_Y are strictly increasing functions, then $T_X(X)$ and $T_Y(Y))$ have the same copula as X and Y. This property of copulas is useful for defining measures of dependence.

Examples of Simple Copulas

If X and Y are independent, then their copula satisfies

$$C(u,v) = u \cdot v \tag{19.3}$$

The copula (19.3) is called the *independent copula* or *product copula*. Its form follows from the definition of independence given in (19.1).

Suppose that X and Y are perfectly positively dependent or *comonotonic*. This occurs if

$$Y = T(X)$$

and T is a strictly increasing transformation. Then the copula for X and Y satisfies

$$C(u,v) = \min(u,v)$$

Notice that this is df for the pair (U,U), where $U \sim U(0,1)$.

Finally, suppose that X and Y are perfectly negatively dependent or *countermonotonic*. This occurs if

$$Y = T(X)$$

and T is a strictly decreasing transformation. Then the copula for X and Y satisfies

$$C(u,v) = \max(u+v-1,0)$$

The above is the df for the pair $(U, 1-U)$.

The copulas for comonotonic and countermonotonic random variables form the so-called Fréchet bounds for any copula $C(u,v)$:

$$\max(u+v-1,0) \le C(u,v) \le \min(u,v)$$

Copula Density

The copula density is defined by

$$c(u,v) = \frac{\partial^2}{\partial u\, \partial v} C(u,v)$$

Let F_{XY} be a joint df with margins F_X and F_Y defined by (19.2). Then, using the chain rule, the joint density of X and Y may be recovered using

$$\begin{aligned} f_{XY}(x,y) &= \frac{\partial^2}{\partial x\,\partial y}F_{XY}(x,y) \qquad (19.4) \\ &= \frac{\partial^2}{\partial u\,\partial v}C(F_X(x),F_Y(y))\frac{\partial F_X}{\partial x}\frac{\partial F_Y}{\partial y} \\ &= c(F_X(x),F_Y(y))\cdot f_X(x)f_Y(y) \end{aligned}$$

The above result shows that it is always possible to specify a bivariate density by specifying the marginal densities and a copula density.

19.3.3 Dependence Measures and Copulas

Dependence measures for financial risk management are nicely surveyed in Embrechts, Lindskog, and McNeil (2003). For two random variables X and Y, they list four desirable properties of a general, single number measure of dependence $\delta(X,Y)$:

1. $\delta(X,Y)=\delta(Y,X)$.
2. $-1\leq\delta(X,Y)\leq 1$.
3. $\delta(X,Y)=1$ if X and Y are comonotonic; $\delta(X,Y)=-1$ if X and Y are countermonotonic.
4. If T is strictly monotonic, then

$$\delta(T(X),Y)=\begin{cases}\delta(X,Y), & T\text{ increasing}\\ -\delta(X,Y), & T\text{ decreasing}\end{cases}$$

They point out that the usual (Pearson) linear correlation only satisfies the first two properties. They show that the rank correlation measures Spearman's rho and Kendall's tau satisfy all four properties.

Pearson's Linear Correlation

The Pearson correlation coefficient

$$\rho=\frac{\text{cov}(X,Y)}{\sqrt{\text{var}(X)\text{var}(Y)}}$$

gives a scalar summary of the linear dependence between X and Y. If $Y=a+bX$, then $\rho=\pm1$. If X and Y are independent, then $\rho=0$. Embrechts, McNeil, and Straumann (2000) summarize the following shortcomings of linear correlation:

- ρ requires that both $\text{var}(X)$ and $\text{var}(Y)$ exist.

- $\rho = 0$ does not imply independence. Only if X and Y are bivariate normal does $\rho = 0$ imply independence.
- ρ is not invariant under nonlinear strictly increasing transformations.
- Marginal distributions and correlation do not determine the joint distribution. This is only true for the bivariate normal distribution.
- For given marginal distributions F_X and F_Y, $\rho \in [\rho_{\min}, \rho_{\max}]$ and it may be the case that $\rho_{\min} > -1$ and $\rho_{\max} < 1$.

Kendall's Tau and Spearman's Rho

Suppose the random variables X and Y represent financial returns or payoffs. It is often the case that both X and Y take either large or small values together, whereas it is seldom the case that X takes a large value and, at the same time, Y takes a small value (or vice versa). The concept of *concordance* is used to measure this type of association. Concordance measures have the useful property of being invariant to increasing transformations of X and Y. As a result, concordance measures may be expressed as a function of the copula between X and Y. Since the linear correlation ρ is not invariant to increasing transformations of X and Y, it is does not measure concordance. Two common measures of concordance are Kendall's tau statistic and Spearman's rho statistic.

Let F be a continuous bivariate df, and let (X_1, Y_1) and (X_2, Y_2) be two independent pairs of random variables from this distribution. The vectors (X_1, Y_1) and (X_2, Y_2) are said to be *concordant* if $X_1 > X_2$ whenever $Y_1 > Y_2$ and $X_1 < X_2$ whenever $Y_1 < Y_2$; they are said to be *discordant* in the opposite case. *Kendall's tau statistic* for the distribution F is defined as

$$\tau = \Pr\{(X_1 - X_2)(Y_1 - Y_2) > 0\} - \Pr\{(X_1 - X_2)(Y_1 - Y_2) < 0\}$$

If C is the copula associated with F, then it can be shown that

$$\tau = 4 \int\int_{I^2} C\, dC - 1 = 4 \int\int_{I^2} C(u,v)c(u,v)\, du\, dv - 1 \tag{19.5}$$

where $c(u,v)$ is the copula density. The empirical estimate[3] of τ for a sample of size n is the number of the sample's concordant pairs minus the number of discordant pairs divided by the total number of pairs $\binom{n}{2}$:

$$\hat{\tau} = \frac{1}{\binom{n}{2}} \sum_{1 \le i \le j \le n} \operatorname{sign}\left(\left(x_i - x_j\right)\left(y_i - y_j\right)\right) \tag{19.6}$$

[3]A pair (u_i, v_i) and (u_j, v_j) of the sample is called concordant if either $u_i < u_j$ and $v_i < v_j$ or $u_i > u_j$ and $v_i > v_j$. It is called discordant if either $u_i < u_j$ and $v_i > v_j$ or $u_i > u_j$ and $v_i < v_j$.

For a pair of random variables (X, Y) with joint df F and marginal distributions F_X and F_Y, *Spearman's rho statistic*, ρ_S, is defined as the (Pearson) correlation between $F_X(X)$ and $F_Y(Y)$. It is a measure of *rank correlation* in terms of the integral transforms of X and Y. For a copula associated with X and Y, it can be shown that

$$\rho_S = 12 \int \int_{I^2} C(u, v)\, du\, dv - 3 \tag{19.7}$$

For a sample of size n, ρ_S may be estimated using

$$\hat{\rho}_S = \frac{12}{n\,(n^2-1)} \sum_{i=1}^{n} \left(\text{rank}(x_i) - \frac{n+1}{2}\right)\left(\text{rank}(y_i) - \frac{n+1}{2}\right) \tag{19.8}$$

Although both τ and ρ_S are measures of concordance, their values can be quite different. Nelsen (1999) summarized the relationship between τ and ρ_S with the following inequalities:

$$\begin{aligned} \frac{3\tau - 1}{2} &\leq \rho_S \leq \frac{1 + 2\tau - \tau^2}{2}, \text{ for } \tau > 0 \\ \frac{\tau^2 + 2\tau - 1}{2} &\leq \rho_S \leq \frac{1 + 3\tau}{2}, \text{ for } \tau < 0 \end{aligned}$$

Example 123 *Empirical estimates of Kendall's tau and Spearman's rho*

In S-PLUS, estimates of Kendall's tau and Spearman's rho using (19.6) and (19.8) may be computed using the function `cor.test`. For example, to compute these statistics from the daily returns on BMW and Siemens, use

```
> german.tau = cor.test(bmw[nz.idx],siemens[nz.idx],
+                method="k")$estimate
> german.tau
      tau
 0.4729532
> german.rho = cor.test(bmw[nz.idx],siemens[nz.idx],
+                method="spearman")$estimate
> german.rho
      rho
 0.6499582
```

The estimates $\hat{\tau} = 0.4729$ and $\hat{\rho}_S = 0.6499$ are both smaller than the Pearson correlation estimate $\hat{\rho} = 0.6517$.

Tail Dependence Measures

Tail dependence measures are used to capture dependence in the joint tail of bivariate distributions. The coefficient of upper tail dependence may be defined as

$$\lambda_u(X, Y) = \lim_{q \to 1} \Pr(Y > \text{VaR}_q(Y) | X > \text{VaR}_q(X))$$

where the values-at-risk $\mathrm{VaR}_q(X)$ and $\mathrm{VaR}_q(Y)$ denote the $100\cdot q$th percent quantiles of X and Y, respectively. Loosely speaking, $\lambda_u(X,Y)$ measures the probability that Y is above a high quantile given that X is above a high quantile. Similarly, the coefficient of lower tail dependence is

$$\lambda_l(X,Y) = \lim_{q\to 0} \Pr(Y \le \mathrm{VaR}_q(Y) | X \le \mathrm{VaR}_q(X))$$

and measures the probability that Y is below a low quantile given that X is below a low quantile. It can be shown (Joe, 1997 p. 178) that the coefficients of tail dependence are functions of the copula C given by

$$\lambda_u = \lim_{q\to 1} \frac{1-2q+C(q,q)}{1-q} \tag{19.9}$$

$$\lambda_l = \lim_{q\to 0} \frac{C(q,q)}{q} \tag{19.10}$$

If $\lambda_u \in (0,1]$, then there is upper tail dependence; if $\lambda_u = 0$, then there is independence in the upper tail. Similarly, if $\lambda_l \in (0,1]$, then there is lower tail dependence; if $\lambda_l = 0$, then there is independence in the lower tail.

19.4 Parametric Copula Classes and Families

In this section, the bivariate parametric copula families implemented in `S+FinMetrics/EVANESCE` are defined. Once a copula family is defined, functions are available to construct joint cumulative and probability density functions, generate random variables, compute Kendall's tau and Spearman's rho, compute the tail index parameters, and to estimate the copula parameters by the method of maximum likelihood.

19.4.1 Normal Copula

One of the most frequently used copulas for financial modeling is the copula of a bivariate normal distribution with the correlation parameter δ defined by

$$\begin{aligned} C(u,v) &= \int_{-\infty}^{\Phi^{-1}(u)} dx \int_{-\infty}^{\Phi^{-1}(v)} dy \, \frac{1}{2\pi\sqrt{1-\delta^2}} \exp\left\{-\frac{x^2-2\delta xy+y^2}{2(1-\delta^2)}\right\} \\ &= \Phi_\delta(\Phi^{-1}(u), \Phi^{-1}(v)) \end{aligned} \tag{19.11}$$

where $\Phi^{-1}(\cdot)$ is the quantile function of the standard normal distribution and Φ_δ is the joint cumulative distribution function of a standard bivariate normal distribution with correlation coefficient δ $(0 \le \delta \le 1)$. From Sklar's theorem, the *normal copula* generates the bivariate standard normal distribution if and only if the margins are standard normal. For any other margins, the normal copula does not generate a bivariate standard normal distribution. The normal copula is heavily used in financial applications. See, for example, the technical documents describing J.P. Morgan's RiskMetrics$^{\text{TM}}$ system (RiskMetrics, 1995).

For the normal copula, Kendall's tau and Spearman's rho are given by

$$\begin{aligned}\tau &= \frac{2}{\pi}\arcsin\delta \\ \rho_S &= \frac{6}{\pi}\arcsin\frac{\delta}{2}\end{aligned}$$

In addition, except for the case $\delta = 1$, the normal copula does not display either lower or upper tail dependence:

$$\lambda_L = \lambda_U = \begin{cases} 0 & \text{for } \delta < 1 \\ 1 & \text{for } \delta = 1 \end{cases}$$

19.4.2 Normal Mixture Copula

Consider two pairs of random variables (U_1, V_1) and (U_2, V_2) that are independent of each other. The joint distribution of the two pairs are given by normal copulas with parameters δ_1 and δ_2, respectively; that is, $(U_1, V_1) \sim C_{\delta_1}$ and $(U_2, V_2) \sim C_{\delta_2}$, where C_δ denotes the normal copula (19.11) with parameter δ. Let (X, Y) be a random pair such that it is equal to (U_1, V_1) with probability p and it is equal to (U_2, V_2) with probability $(1 - p)$. Note that since the marginal distributions of U_1, V_1, U_2,and V_2 are uniform, so are the marginals of X, and Y. The joint distribution of (X, Y) is given by the following *normal mixture copula*:

$$C(u, v) = pC_{\delta_1}(u, v) + (1 - p)C_{\delta_2}(u, v)$$

where $0 \leq p,\ \delta_1,\ \delta_2 \leq 1$.

19.4.3 Extreme Value Copula Class

A copula is said to be an *extreme value* (EV) *copula* if for all $t > 0$, the scaling property

$$C(u^t, v^t) = (C(u, v))^t$$

holds for all $(u, v) \in I^2$. Let $(X_1, Y_1), (X_2, Y_2), \ldots, (X_n, Y_n)$ be iid random pairs from an EV copula C and define $M_n = \max(X_1, \ldots, X_n)$ and $N_n = \max(Y_1, \ldots, Y_n)$. Then C is also a copula associated with the random pair (M_n, N_n). This property is called *max-stability.* In can be shown (e.g. Joe, 1997 p. 175) that EV copulas can be represented in the form

$$C(u, v) = \exp\left\{\ln(uv)A\left(\frac{\ln(u)}{\ln(uv)}\right)\right\}$$

where $A(\cdot) : [0,1] \to [\frac{1}{2},1]$ is a convex function such that $\max(t, t-1) \leq A(t) \leq 1$ for all $t \in [0,1]$. The function $A(t)$ is called the *dependence function.* The following EV copulas are implemented in `S+FinMetrics/EVANESCE`.

Gumbel Copula

Probably the most common EV copula is the *Gumbel copula* (Gumbel, 1960) with parameter δ:

$$C(u,v) = \exp\left\{-[(-\ln(u)^{\delta} + (-\ln(v)^{\delta}]^{1/\delta}\right\}, \ \delta \geq 1$$

The dependence function of the Gumbel copula has the form

$$A(t) = (t^{\delta} + (1-t)^{\delta})^{1/\delta}$$

The parameter δ controls the strength of dependence. When $\delta = 1$, there is no dependence; when $\delta = +\infty$, there is perfect dependence. It can be shown that Kendall's tau is given by

$$\tau = 1 - \delta^{-1}$$

Further, the Gumbel copula exhibits upper tail dependency with

$$\lambda_U = 2 - 2^{1/\delta}$$

Galambos Copula

The *Galambos copula* (Galambos, 1975) with parameter δ has the form

$$C(u,v) = uv\exp\left\{[(-\ln(u)^{-\delta} + (-\ln(v)^{-\delta}]^{-1/\delta}\right\}, \ 0 \leq \delta < \infty$$

The dependence function for this copula is

$$A(t) = 1 - (t^{-\delta} + (1-t)^{-\delta})^{-1/\delta}$$

Hüsler and Reiss Copula

The *Hüsler and Reiss copula* (Hüsler and Reiss, 1987) with parameter δ is

$$C(u,v) = \exp\left\{-\tilde{u}\Phi\left[\frac{1}{\delta} + \frac{1}{2}\delta\ln\left(\frac{\tilde{u}}{\tilde{v}}\right)\right] - \tilde{v}\Phi\left[\frac{1}{\delta} + \frac{1}{2}\delta\ln\left(\frac{\tilde{v}}{\tilde{u}}\right)\right]\right\}$$

where $0 \leq \delta \leq \infty, \tilde{u} = -\ln u, \tilde{v} = -\ln v$, and Φ is the df of a standard normal random variable. The dependence function for this copula is

$$\begin{aligned} A(t) &= t\Phi\left[\frac{1}{\delta} + \frac{1}{2}\delta\ln\left(\frac{t}{1-t}\right)\right] \\ &\quad +(1-t)\Phi\left[\frac{1}{\delta} - \frac{1}{2}\delta\ln\left(\frac{t}{1-t}\right)\right] \end{aligned}$$

Twan Copula

The *Twan copula* (Twan, 1988) is an asymmetric extension of the Gumbel copula with parameters α, β, and r. It has dependence function

$$A(t) = 1 - \beta + (\beta - \alpha) + \{\alpha^r t^r + \beta^r (1-t)^r\}^{1/r}$$

where $0 \leq \alpha, \beta \geq 1$, and $1 \leq r \leq \infty$.

BB5 Copula

Joe (1997) defined the *BB5 copula* as a two-parameter extension of the Gumbel copula. It has the form

$$C(u,v) = \exp\left\{ - \left[\tilde{u}^\theta + \tilde{v}^\theta - (\tilde{u}^{-\theta\delta} + \tilde{v}^{-\theta\delta})^{-1/\delta} \right]^{1/\theta} \right\}$$

where $\delta > 0, \theta \geq 1, \tilde{u} = -\ln u$, and $\tilde{v} = -\ln v$. The dependence function for this copula is

$$A(t) = \left[t^\theta + (1-t)^\theta - (t^{-\delta\theta} + (1-t)^{-\delta\theta})^{-1/\delta} \right]^{1/\theta}$$

19.4.4 Archimedean Copulas

Archimedean copulas (see Nelsen, 1999, Chap. 4) are copulas that may be written in the form

$$C(u,v) = \phi^{-1}\left[\phi(u) + \phi(v)\right]$$

for a function $\phi : I \to \mathbb{R}^+$ that is continuous, strictly decreasing, and convex and satisfies $\phi(1) = 0$. The function ϕ is called the *Archimedean generator*, and ϕ^{-1} is its inverse function. For example, the Gumbel copula is an Archimedean copula with generator function $\phi(t) = (-\ln t)^\delta$. The density of an Archimedean copula may be determined using

$$c(u,v) = \frac{-\phi''\left(C(u,v)\right)\phi'(u)\phi'(v)}{\left(\phi'(C(u,v))\right)^3}$$

where ϕ' and ϕ'' denote the first and second derivatives of ϕ, respectively.

Genest and MacKay (1986) showed that Kendall's tau may be computed using

$$\tau = 4\int_I \frac{\phi(v)}{\phi'(v)}\, dv + 1$$

Prior to their use in financial applications, Archimedean copulas have been successfully used in actuarial applications (see Frees and Valdez, 1998). The following Archimedean copulas are implemented in `S+FinMetrics/EVANESCE`.

Frank Copula

The *Frank copula* (Frank, 1979) has the following distribution function:

$$C(u,v) = -\delta \log \left(\left[\eta - (1 - e^{-\delta u})(1 - e^{-\delta u}) \right] / \eta \right)$$

where $0 < \delta < \infty$ and $\eta = 1 - e^{-\delta}$. The generator function is given by

$$\phi(t) = -\ln \left(\frac{e^{-\delta t} - 1}{e^{-\delta} - 1} \right)$$

The Frank copula does not exhibit lower or upper tail dependency.

Kimeldorf-Sampson (Clayton) Copula

The *Kimeldorf and Sampson copula* (Kimeldorf and Sampson, 1975) has the following form:

$$C(u,v) = \left(u^{-\delta} + v^{-\delta} - 1 \right)^{-1/\delta}$$

where $0 < \delta < \infty$, and the generator function is

$$\phi(t) = t^{-\delta} - 1$$

This copula is also known as the *Clayton copula* (Clayton, 1978).

Kendall's tau is given by

$$\tau = \frac{\delta}{\delta + 2}$$

and it exhibits only lower tail dependency:

$$\lambda_L = 2^{-1/\delta}$$

Joe Copula

The *Joe copula* (Joe, 1993) has the form

$$C(u,v) = 1 - \left((1-u)^{\delta} + (1-v)^{\delta} - (1-u)^{\delta}(1-v)^{\delta} \right)^{1/\delta}$$

where $\delta \geq 1$ with generator function

$$\phi(t) = -\ln(1 - (1-t)^{\delta})$$

BB1 Copula

The *BB1 copula* (Joe, 1997) is given by

$$C(u,v) = \left(1 + \left[(u^{-\theta} - 1)^{\delta} + (v^{-\theta} - 1)^{\delta} \right]^{-1/\theta} \right)$$

with $\theta > 0$ and $\delta \geq 1$ and generator function

$$\phi(t) = (t^{-\theta} - 1)^{\delta}$$

BB2 Copula

The *BB2 copula* (Joe, 1997) has the form

$$C(u,v) = \left[1 + \delta^{-1} \ln\left(e^{\delta u^{-\theta}} + e^{\delta v^{-\theta}} - 1\right)\right]^{1/\theta}$$

with $\theta > 0$, $\delta > 0$ and generator function

$$\phi(t) = e^{(t^{-\theta}-1)} - 1$$

BB3 Copula

The *BB3 copula* (Joe, 1997) has the form of

$$C(u,v) = \exp\left\{-\left[\delta^{-1} \ln\left(e^{\delta \tilde{u}^{\theta}} + e^{\delta \tilde{v}^{\theta}} - 1\right)\right]^{1/\theta}\right\}$$

with $\theta > 1$, $\delta > 0$, $\tilde{u} = -\ln u$, and $\tilde{v} = -\ln v$. The generator function is

$$\phi(t) = \exp\left\{\delta(-\ln t)^{\theta}\right\} - 1$$

BB6 Copula

The *BB6 copula* (Joe, 1997) has the form of

$$\begin{aligned} C(u,v) \quad &= \quad 1 - \left(1 - \exp\left\{-\left[\left(-\ln\left(1-(1-u)^{\theta}\right)\right)^{\delta}\right.\right.\right. \\ &\qquad \left.\left.\left. + \left(-\ln\left(1-(1-v)^{\theta}\right)\right)^{\delta}\right]^{1/\delta}\right\}\right)^{1/\theta} \end{aligned}$$

where $\theta \geq 1$, $\delta \geq 0$ and has generator function

$$\phi(t) = \left[-\ln\left(1-(1-t)^{\theta}\right)\right]^{\delta}$$

BB7 Copula

The *BB7 copula* (Joe, 1997) has the form of

$$C(u,v) = 1 - \left(1 - \left[\left(1-(1-u)^{\theta}\right)^{-\delta} + \left(1-(1-v)^{\theta}\right)^{-\delta} - 1\right]^{-1/\delta}\right)^{1/\theta}$$

where $\theta \geq 1$ and $\delta > 0$ and generator function

$$\phi(t) = \left(1-(1-t)^{\theta}\right)^{-\delta} - 1$$

19.4.5 Archimax Copulas

Capéraà, Fourgères, and Genest (2000) combined the EV and Archimedean copula classes into a single class called *Archimax copulas*. The Archimax copulas have the form

$$C(u,v) = \phi^{-1}\left[(\phi(u)+\phi(v))\,A\left(\frac{\phi(u)}{\phi(u)+\phi(v)}\right)\right]$$

where $A(t)$ is a valid dependence function and ϕ is a valid Archimedean generator. Archimax copulas reduce to Archimedean copulas for $A(t) = 1$ and to EV copulas for $\phi(t) = -\ln(t)$. `S+FinMetrics/EVANESCE` implements the following Archimax copula due to Joe (1997).

BB4 Copula

The Archimax copula with

$$\phi(t) = t^{-\theta} - 1 \text{ and } A(t) = 1 - (t^{-\delta} + (1-t)^{-\delta})^{-1/\delta}$$

is called the *BB4 copula* and has the form

$$C(u,v) = \left(u^{-\theta} + v^{-\theta} - 1 - \left[(u^{-\theta}-1)^{-\delta} + (v^{-\theta}-1)^{-\delta}\right]^{-1/\delta}\right)^{-1/\theta}$$

with $\theta \geq 0$ and $\delta > 0$.

19.4.6 Representation of Copulas in `S+FinMetrics`

The `EVANESCE` implementation in `S+FinMetrics` defines an SV4 object class "`copula`", and its child classes "`ev.copula`", "`archm.copula`", "`normal.copula`", "`normal.mix.copula`", "`bb4.copula`", and "`empirical.copula`". These classes are referred to as *copula types.* Each parametric family of copulas discussed in the previous subsection that belongs to one of these copula types is also defined as a separate subclass and is named "`family.name.copula`". For example, the Gumbel copula family is a "`copula`" object, with child class "`ev.copula`" and subclass "`gumbell.copula`". If the copula family defines an EV copula, it is considered to be of "`ev.copula`" type. If it is an Archimedean copula but not an EV copula, then it inherits from "`archm.copula`". A copula's type will determine what internal functions are used for various operations and computations on the copula (e.g., generation of random observations from the copula, computation of Kendall's tau, etc.). For example, each parametric copula of class `archm.copula` has `S+FinMetrics/EVANESCE` method functions for computing ϕ, ϕ^{-1}, ϕ', ϕ'', and $\phi'^{(-1)}$. See the online help for `copula.object` for more details on these functions.

Objects of class "`copula`" for a particular family are created using the constructor functions listed in Table 19.1. In addition to the parametric

Function	Parameters	Inherits
normal.copula	delta	copula
normal.mix.copula	p, delta1, delta2	copula
bb4.copula	theta, delta	copula
gumbel.copula	delta	copula, ev.copula
galambos.copula	delta	copula, ev.copula
husler.reiss.copula	delta	copula, ev.copula
bb5.copula	delta, theta	copula, ev.copula
twan.copula	a, b, r	copula, ev.copula
frank.copula	delta	copula, archm.copula
kimeldorf.sampson. copula	delta	copula, archm.copula
joe.copula	theta	copula, archm.copula
bb1.copula	theta, delta	copula, archm.copula
bb2.copula	theta, delta	copula, archm.copula
bb3.copula	theta, delta	copula, archm.copula
bb6.copula	theta, delta	copula, archm.copula
bb7.copula	theta, delta	copula, archm.copula

TABLE 19.1. Copula constuctor functions

Slot name	Description
parameters	A vector with values of the parameters
param.names	Names of the greek letter parameters for the copula
param.lowbnd	A vector the same length of parameters containing the values of the lower bounds for the parameters
param.upbnd	A vector the same length of parameters containing the values of the upper bounds for the parameters
message	Name of the parametric copula family and copula subclass if applicable

TABLE 19.2. Slots for "copula" objects

family constructor functions, there are also copula child class constructor functions `ev.copula` and `archm.copula`. Each object of class "`copula`" has slots described in Table 19.2.

Example 124 *Constructing copula objects.*

To create a "`copula`" object representing a normal copula (19.11) with $\delta = 0.7$, use the constructor function `normal.copula`:

```
> ncop.7 = normal.copula(delta=0.7)
> class(ncop.7)
[1] "normal.copula"
> inherits(ncop.7, what="copula")
[1] T
```

```
> slotNames(ncop.7)
[1] "parameters"   "param.names"  "param.lowbnd" "param.upbnd"
[5] "message"
> ncop.7@parameters
[1] 0.7
> ncop.7@param.lowbnd
[1] 0
> ncop.7@param.upbnd
[1] 1
> ncop.7@message
[1] "Normal copula family"
```

The `print` method gives a basic description of the copula:

```
> ncop.7
    Normal copula family.
    Parameters :
       delta  =  0.7
```

An object representing the Gumbel copula with $\delta = 2$ may be created in a number of ways. It may be created using the `gumbel.copula` constructor function:

```
> gumbel.cop.2 = gumbel.copula(delta=2)
> inherits(gumbel.cop.2,what="copula")
[1] T
> inherits(gumbel.cop.2, what="ev.copula")
[1] T
> gumbel.cop.2
    Gumbel copula family; Extreme value copula.
    Parameters :
       delta  =  2
```

Since the Gumbel copula is an EV copula, it may also be created using the `ev.copula` constructor function:

```
> gcop.2 = ev.copula(family="gumbel", param=2)
> class(gcop.2)
[1] "gumbel.copula"
> gcop.2
    Gumbel copula family; Extreme value copula.
    Parameters :
       delta  =  2
```

Visualizing Copulas

Table 19.3 lists the `S+FinMetrics/EVANESCE` method functions for visualizing bivariate copulas. The surface plot of the bivariate CDF $C(u, v)$

Method Function	Description
`persp.pcopula`	Surface plot of copula CDF $C(u,v)$
`contour.plot`	Plot contours of copula CDF $C(u,v)$
`contour.pcopula`	
`persp.dcopula`	Surface plot of copula density $c(u,v)$
`contour.dcopula`	Plot contours of copula density $c(u,v)$

TABLE 19.3. Method functions for visualizing bivariate copulas

produced by the function `persp.pcopula` is often not very informative about the dependence properties of a copula. Instead, the contour plots of the level sets

$$\{(u,v) \in I^2 : C(u,v) = \kappa\},\ \kappa \text{ is constant}$$

produced by the functions `contour.plot` and `contour.pcopula` are much more informative.

Example 125 *Visualizing copulas*

The normal copula with $\delta = 0.7$ may be visualized using

```
> persp.dcopula(ncop.7)
> contour.dcopula(ncop.7)
> persp.pcopula(ncop.7)
> contour.pcopula(ncop.7)
```

The resulting figures are shown in Figure 19.9. Notice how the surface plot of the CDF is tent-shaped, with level 0 at the point $(0,0)$ and level 1 at the point $(1,1)$. The level sets of the CDF are convex and their angled shape indicates moderate dependence. The copula density has two large peaks at the points $(0,0)$ and $(1,1)$ and almost no mass in the upper left and lower right quadrants of I^2, which reflects positive dependence.

Figure 19.10 illustrates the normal copula with $\delta = 0.0001$ created by

```
> normal.cop.0 = normal.copula(delta=0.0001)
```

This copula is essentially the independent copula $C(u,v) = uv$. Notice how the level curves of the CDF are less convex than the level curves of the normal copula with $\delta = 0.7$. Also, the copula density has a relatively flat mass over the interior of I^2.

Figure 19.11 shows the Gumbel copula with $\delta = 2$. The plots look similar to those for the normal copula with $\delta = 0.7$. One noticeable difference is the asymmetric peaks in the density at the points $(0,0)$ and $(1,1)$. The Gumbel density has a much larger peak at the point $(1,1)$ than at the point $(0,0)$. This reflects upper tail dependence in the copula.

Figure 19.12 shows the Kimeldorf-Sampson (Clayton) copula with $\delta = 2$. The lower tail dependence of this copula is clearly shown in the CDF and

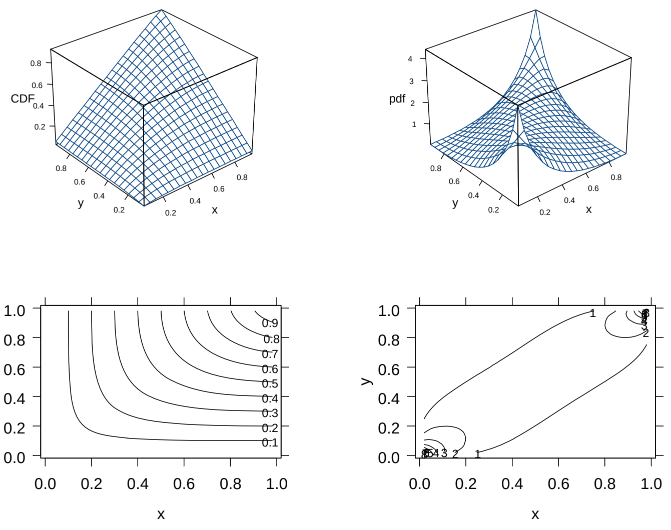


FIGURE 19.9. Surface and contour plots of $C(u, v)$ and $c(u, v)$ for the normal copula with $\delta = 0.7$.

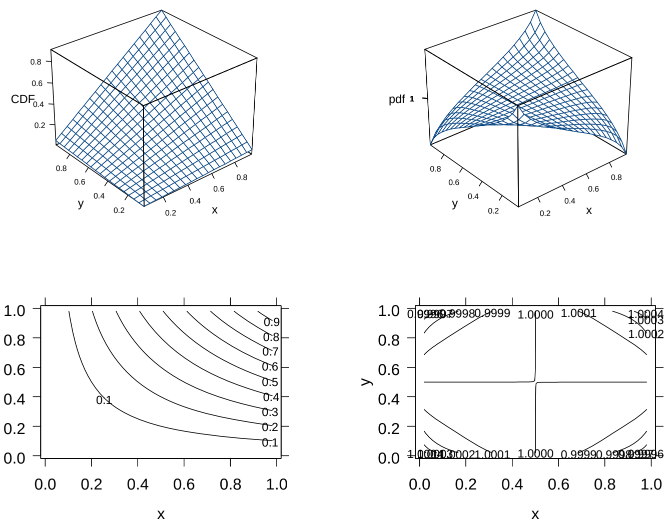


FIGURE 19.10. Surface and contour plots of $C(u, v)$ and $c(u, v)$ for the normal copula with $\delta = 0$.

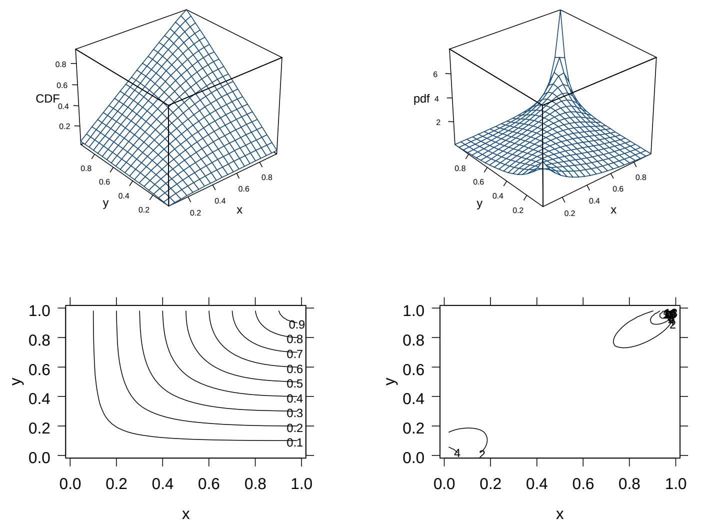


FIGURE 19.11. Surface and contour plots of $C(u, v)$ and $c(u, v)$ for the Gumbel copula with $\delta = 0$.

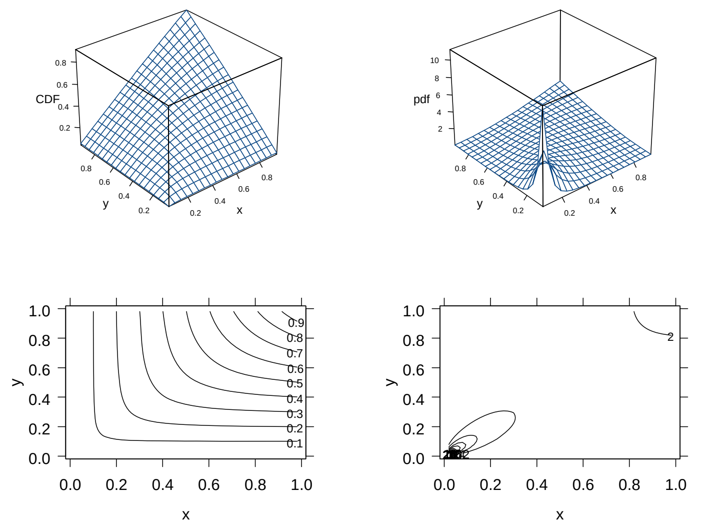


FIGURE 19.12. Surface and contour plots of $C(u, v)$ and $c(u, v)$ for the Clayton copula with $\delta = 0$.

density plots. The level sets of the CDF are pushed much closer to the origin where the density has a strong peak.

The reader is encouraged to experiment with plots of the other parametric copula families implemented in `S+FinMetrics`.

Simulating from Copulas

The `S+FinMetrics/EVANESCE` method functions `pcopula` and `dcopula` compute the copula CDF and pdf at specified points. The function `rcoupla` generates random variables with uniform marginal distributions and joint CDF determined by a specified copula. The algorithms used for simulating from parametric copulas are described in Genest and Revest (1993) and Cherubini, Luciano, and Vecchiato (2004).

Example 126 *Simulating uniform random variables from specified copulas*

To compute 2000 random pairs with uniform marginal distributions and joint distributions specified by the independent copula, normal copula with $\delta = 0.7$, Gumbel copula with $\delta = 2$ and Clayton copula with $\delta = 2$, respectively, use

```
> set.seed(123)
> normal.7.sim = rcopula(ncop.7,2000)
> normal.0.sim = rcopula(normal.cop.0,2000)
> gumbel.2.sim = rcopula(gumbel.cop.2,2000)
> ks.2.sim = rcopula(ks.cop.2,2000)
```

Figure 19.13 shows these pairs. Notice how the distribution of the uniformly distributed pairs in I^2 reflects the dependence structure of the copulas.

Computing Dependence Measures

The dependence measures introduced in Section 19.3 may be computed for selected copulas using the functions `Kendalls.tau`, `Spearmans.rho` and `tail.index`. These functions make use of the fact that for certain copulas, the dependence measures either have closed-form solutions based on the copula parameters or have representations in terms of the generator function ϕ and its inverse and derivatives.

Example 127 *Computing dependence measures for various copulas*

Kendall's tau, Spearman's rho, and the tail index parameters λ_L and λ_U for the normal copula with $\delta = 0.7$ may be computed using

```
> Kendalls.tau(ncop.7)
[1] 0.4936334
> Spearmans.rho(ncop.7)
[1] 0.6829105
> tail.index(ncop.7)
```

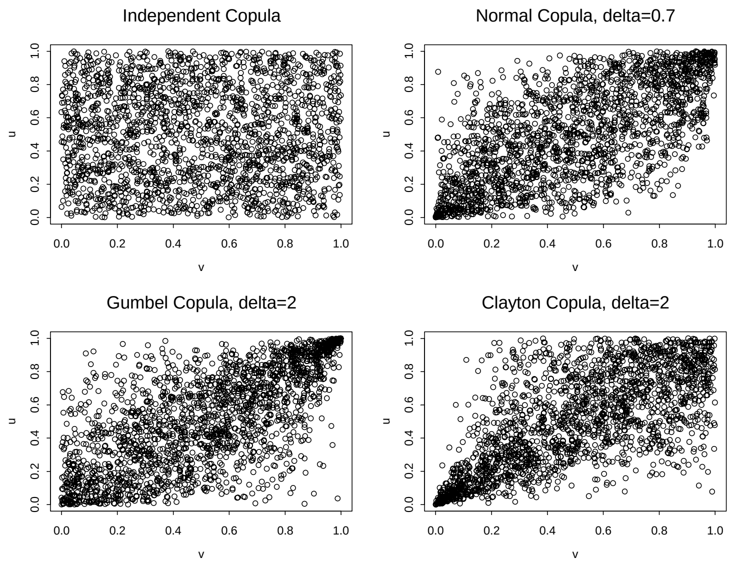


FIGURE 19.13. Random simulations from specified copulas.

```
 lower.tail upper.tail
          0          0
```

The corresponding estimates for the Gumbel copula with $\delta = 2$ are

```
 Kendalls.tau(gcop.2)
[1] 0.5
> Spearmans.rho(gcop.2)
[1] 0.6822338
> tail.index(gcop.2)
 lower.tail upper.tail
          0  0.5857864
```

For the Gumbel copula, the dependence measures are increasing functions of δ. For example, setting $\delta = 10$ produces

```
> gcop.10 = gumbel.copula(delta=10)
> Kendalls.tau(gcop.10)
[1] 0.9
> Spearmans.rho(gcop.10)
[1] 0.9854924
> tail.index(gcop.10)
 lower.tail upper.tail
          0  0.9282265
```

Method Function	**Description**
`pbivd`	Compute CDF for bivariate distribution
`dbivd`	Compute pdf for bivariate distribution
`rbivd`	Simulate from bivariate distribution
`persp.pbivd`	Surface plot of CDF
`contour.pbivd`	Contour plot of CDF
`persp.dbivd`	Surface plot of pdf
`contour.dbivd`	Contour plot of pdf

TABLE 19.4. Method functions for ”bivd” objects

19.4.7 Creating Arbitrary Bivariate Distributions

Consider the problem of creating an *arbitrary* bivariate distribution for two random variables X and Y. Sklar's theorem states that it is always possible to specify a bivariate distribution $F(x, y)$ by specifying

- the marginal distributions $F_X(x)$ and $F_Y(y)$
- a copula $C(u, v)$

Equations (19.2) and (19.4) show how this is done. In `S+FinMetrics/EVANESCE`, the constructor function `bivd` allows the user to build an arbitrary bivariate distribution from specified margins and a copula. The arguments expected by `bivd` are

```
> args(bivd)
function(cop, Xmarg = "unif", Ymarg = Xmarg, param.Xmarg =
          c(0, 1), param.Ymarg = param.Xmarg)
```

where `cop` represents a copula object, `Xmarg` and `Ymarg` specify the marginal distributions for X and Y, respectively, and `param.Xmarg` and `param.Ymarg` specify the parameters defining the marginal distributions. If the marginal distribution name is specified as "`dist`", it is assumed that the functions `ddist`, `pdist`, and `qdist` are implemented in `S-PLUS`. For example, if `Xmarg="norm"`, then the functions `dnorm`, `pnorm`, and `qnorm` should be available for correct performance of the bivariate distribution functions. The distribution names are not limited to those available in standard `S-PLUS`. The user may create the appropriate density and quantile functions. The arguments `param.Xmarg` and `param.Ymarg` specify the parameters required by the functions named by `Xmarg` and `Ymarg.` For example, if `Xmarg="norm"`, then the user must specify in a vector the values of the parameters `mean` and `sd` required by the functions `dnorm`, `pnorm`, and `qnorm`. The function `bivd` creates an object of class "`bivd`" with child class method functions listed in Table 19.4.

Example 128 *Create bivariate distribution with $N(3, 4^2)$ and t_3 margins, and Gumbel copula with $\delta = 2$*

To create a "`bivd`" object representing a bivariate distribution with F_X a normal distribution with $\mu = 3$ and $\sigma = 4$, F_Y a central Student's-t distribution with $\upsilon = 3$ degrees of freedom, and a Gumbel copula with $\delta = 2$, use

```
> gumbel.biv <- bivd(cop=gumbel.copula(2), Xmarg="norm",
+                    Ymarg="t", param.Xmarg=c(3,4),
+                    param.Ymarg=3)
> class(gumbel.biv)
[1] "gumbel.bivd"
```

Notice that the class of the object created by `bivd` is based on the class of the "`copula`" object specified by the `cop` argument. The information for the marginal normal distribution for X will be computed using the `S-PLUS` functions `dnorm`, `pnorm`, and `qnorm`, respectively. The corresponding information for the marginal Student's-t distribution for Y will be computed using the `S-PLUS` functions `dt`, `pt`, and `qt`, respectively. The bivariate distribution represented by the object `gumbel.bivd` may be visualized using

```
> persp.dbivd(gumbel.biv)
> contour.dbivd(gumbel.biv)
> persp.pbivd(gumbel.biv)
> contour.pbivd(gumbel.biv)
```

and these plots are given in Figure 19.14.

Joint orthant probabilities of the form $\Pr\{X \leq x_0, Y \leq y_0\}$ may be computed using the function `pbivd`. For example,

```
> pbivd(gumbel.biv, x = c(-2, 0), y = c(-2, 0))
[1] 0.03063639 0.19430934
```

gives $\Pr\{X \leq -2, Y \leq -2\} = 0.0306$ and $\Pr\{X \leq 0, Y \leq 0\} = 0.1943$, respectively.

A simulated sample of size 2000 from the bivariate distribution represented by the object `gumbel.bivd` may be created using the function `rbivd` as follows:

```
> set.seed(123)
> gumbel.biv.sim = rbivd(gumbel.biv,n=2000)
> class(gumbel.biv.sim)
[1] "data.frame"
> names(gumbel.biv.sim)
[1] "x" "y"
```

Some graphical diagnostics of the simulated data are given in Figure 19.15. The lower right-hand panel shows a scatter plot of the uniform random variables computed using $u = F_X(x)$ and $v = F_Y(y)$:

```
> U.x = pnorm(gumbel.biv.sim$x,mean=3,sd=4)
> V.y = pt(gumbel.biv.sim$y,df=3)
```

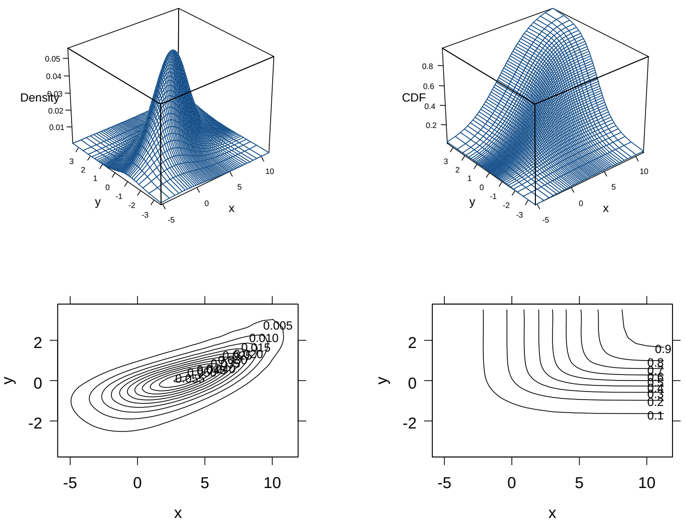


FIGURE 19.14. Bivariate distribution with $N(3,4^2)$ and t_3 marginals, and Gumbel copula with $\delta = 2$.

This plot reflects the upper tail dependence induced by the Gumbel copula.

19.4.8 Simulating from Arbitrary Bivariate Distributions

In the previous subsection, random draws from an arbitrary bivariate distribution were obtained using the `rbivd` method function for objects of class "`bivd`". The creation of a "`bivd`" object representing an arbitrary bivariate distribution requires a valid "`copula`" object and `S-PLUS` functions for computing the density, cumulative distribution, and quantiles. If one or more of the `S-PLUS` distribution functions for the margins are not available, then it is not possible to create the `bivd` object, and simulations using `rbivd` cannot be made. However, if `S-PLUS` functions for computing the quantiles of the marginal distributions are available, then it is still possible to compute simulated observations from the desired bivariate distribution. The process is as follows[4]:

1. Create a "`copula`" object using one of the copula constructor functions from Table 19.1.

[4]This process is carried out automatically in the function `rbivd` from the information in the "`bivd`" object.

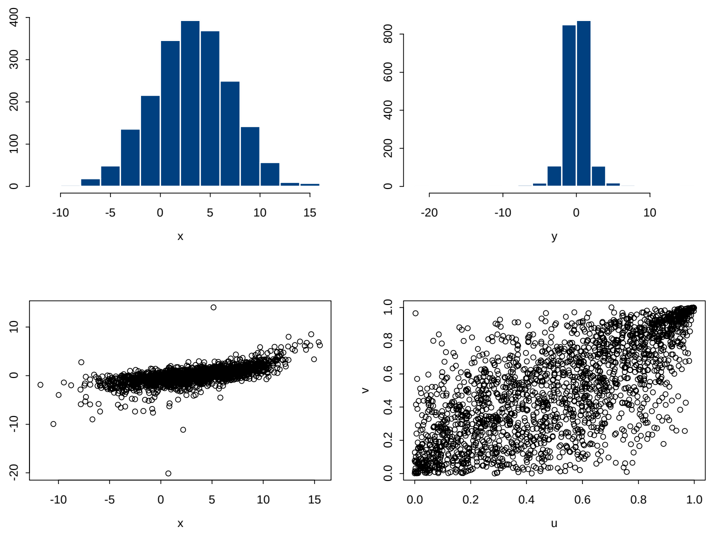


FIGURE 19.15. Graphical diagnostics of simulated data from bivariate distribution with $N(3, 4^2)$ and t_3 marginals, and Gumbel copula with $\delta = 2$.

2. Simulate a random sample of univariate pairs from the specified copula using the function `rcopula`.

3. Use the `S-PLUS` quantile functions for the marginal distributions evaluated on the simulated univariate pairs to generate observations from the desired bivariate distribution.

Example 129 *Simulate from bivariate distribution with GPD margins and normal copula*

Figure 19.8 shows a scatter plot of simulated returns from a bivariate distribution constructed from independent GPD margins fit to the BMW and Siemens returns. The GPDs capture the tail behavior of the marginal distributions, but the (implicit) independent copula does not capture the observed positive dependence in the bivariate distribution. Instead of simulating from a bivariate distribution with GPD margins and an independent copula, consider simulating from a distribution with the same margins and a normal copula with $\delta = 0.7$:

```
> n.obs <- seriesLength(bmw[nz.idx])
> set.seed(123)
> u.data <- rcopula(ncop.7,n.obs)
> bmw.cop.sim <- gpd.2q(u.data$x, gpd.bmw.fit2)
> siemens.cop.sim <- gpd.2q(u.data$y, gpd.siemens.fit2)
```

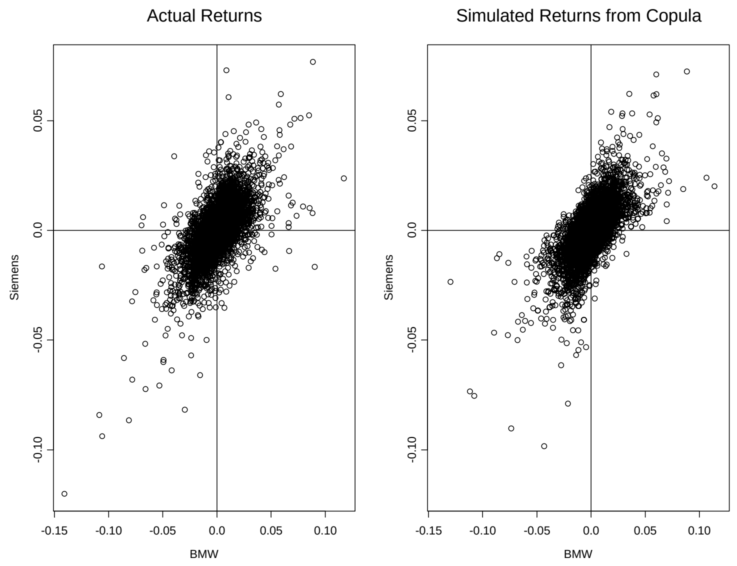


FIGURE 19.16. Scatter plots of actual returns and simulated returns from a bivariate distribution constructed from GPD margins and a normal copula with $\delta = 0.7$.

In the above code, the `S+FinMetrics/EVANESCE` function `gpd.2q` is used to compute the quantiles of the two-tailed GPD fit to the observed returns. Figure 19.16 shows scatter plots of the actual returns and the simulated returns. The simulated data from the bivariate distribution constructed from fitted GPD margins and a normal copula with $\delta = 0.7$ mimics the actual returns surprisingly well.

19.5 Fitting Copulas to Data

19.5.1 Empirical Copula

Deheuvels (1978) proposed the following nonparametric estimate of a copula C. Let $u_{(1)} \leq u_{(2)} \leq \cdots \leq u_{(n)}$ and $v_{(1)} \leq v_{(2)} \leq \cdots \leq v_{(n)}$ be the order statistics of the univariate samples from a copula C. The *empirical copula* $\hat{C}$ is defined at the points $\left(\frac{i}{n}, \frac{j}{n}\right)$ by

$$\hat{C}\left(\frac{i}{n}, \frac{j}{n}\right) = \frac{1}{n}\sum_{k=1}^{n} 1_{\{u_k \leq u_{(i)}, v_k \leq v_{(j)}\}}, \quad i, j = 1, 2, \ldots, n \tag{19.12}$$

Deheuvels proved that $\hat{C}$ converges uniformly to C as the sample size tends to infinity. The *empirical copula frequency* $\hat{c}$ is given by

$$\hat{c}\left(\frac{i}{n}, \frac{j}{n}\right) = \begin{cases} \frac{1}{n} & \text{if } (u_{(i)}, v_{(j)}) \text{ is an element of the sample} \\ 0 & \text{otherwise} \end{cases}$$

Nelsen (1999) showed that estimates of Spearman's rho and Kendall's tau for a sample of size n may be computed from the empirical copula using

$$\begin{aligned} \hat{\rho}_S &= \frac{12}{n^2-1}\sum_{j=1}^{n}\sum_{i=1}^{n}\left[\hat{C}\left(\frac{i}{n}, \frac{j}{n}\right) - \frac{i}{n}\cdot\frac{j}{n}\right] \\ \hat{\tau} &= \frac{2n}{n-1}\sum_{j=2}^{n}\sum_{i=2}^{n}\left[\hat{C}\left(\frac{i}{n}, \frac{j}{n}\right)\hat{C}\left(\frac{i-1}{n}, \frac{j-1}{n}\right)\right. \\ &\quad \left. -\hat{C}\left(\frac{i}{n}, \frac{j-1}{n}\right)\hat{C}\left(\frac{i-1}{n}, \frac{j}{n}\right)\right] \end{aligned}$$

The tail index parameters (19.9) and (19.10) may be inferred from the empirical copula by plotting

$$\begin{aligned} \hat{\lambda}_U(q) &= \frac{1-2q+\hat{C}(q,q)}{1-q} \\ \hat{\lambda}_L(q) &= \frac{\hat{C}(q,q)}{q} \end{aligned}$$

as functions of q and visually observing convergence as $q \to 1$ and $q \to 0$, respectively.

Example 130 *Empirical copula for BMW and Siemens daily returns*

Consider computing the empirical copula (19.12) for the BMW and Siemens returns based on the estimated semiparametric GPD models for the marginal distributions. The empirical copula (19.12) requires a sample of univariate random variables based on the estimated CDFs:

$$(\hat{u}_i, \hat{v}_i) = (\hat{F}_X(x_i), \hat{F}_Y(y_i)),\ i = 1, \ldots, n \tag{19.13}$$

From the estimated semiparametric GPD models for $\hat{F}_X$ and $\hat{F}_Y$, the uniform pairs (19.13) may be computed using the `S+FinMetrics/EVANESCE` function `gpd.2p` as follows:

```
> U.bmw.gpd <- gpd.2p(bmw[nz.idx], gpd.bmw.fit2)
> V.siemens.gpd <- gpd.2p(siemens[nz.idx], gpd.siemens.fit2)
```

The empirical copula (19.12) may then be computed using the `S+FinMetrics/EVANESCE` function `empirical.copula` as follows:

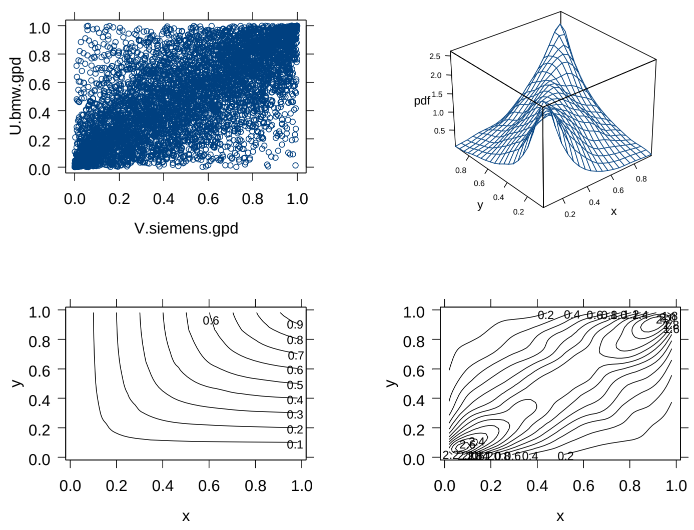


FIGURE 19.17. Empirical copula for BMW and Siemens daily returns based on estimated GPD marginals.

```
> empcop.bs <- empirical.copula(x = U.bmw.gpd,
+                               y = V.siemens.gpd)
> class(empcop.bs)
[1] "empirical.copula"
> slotNames(empcop.bs)
[1] "x" "y"
```

The values in the slots `x` and `y` are the same as the inputs. The function `empirical.copula` produces an object of class "`empirical.copula`" with method functions available for visualization, quantile evaluation, and the computation of dependence measures. The empirical copula may be visualized using

```
> plot(empcop.bs@x,empcop.bs@y)
> contour.pcopula(empcop.bs)
> persp.dcopula(empcop.bs)
> contour.dcopula(empcop.bs)
```

and the resulting plots are given in Figure 19.17. Notice that many of the features of the empirical copula are similar to those from a normal copula with $\delta = 0.7$ (see Figure 19.9).

Estimates of Kendall's tau, Spearman rho, and the tail index parameters λ_L and λ_U may also be computed from the empirical copula:

```
> Kendalls.tau(empcop.bs)
[1] 0.472954
> Spearmans.rho(empcop.bs)
[1] 0.6499589
> tail.index(empcop.bs)
```

The function `tail.index` produces a plot (not shown) showing estimates of λ_L and λ_U based on (19.10) and (19.9) as $q \to 0$ and $q \to 1$, respectively.

19.5.2 Maximum Likelihood Estimation

Let $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$ denote a random sample from a bivariate distribution F with marginal distributions F_X and F_Y and copula C with density c. Recall, the joint density of (x_i, y_i) may be represented as

$$f(x_i, y_i; \boldsymbol{\eta}) = c(F_X(x_i; \boldsymbol{\alpha}_x), F_Y(y_i; \boldsymbol{\alpha}_y); \boldsymbol{\theta}) f_X(x_i; \boldsymbol{\alpha}_x) f_Y(y_i; \boldsymbol{\alpha}_y)$$

where $\boldsymbol{\alpha}_x$ are the parameters for the marginal distribution F_X, $\boldsymbol{\alpha}_y$ are the parameters for the marginal distribution F_Y, $\boldsymbol{\theta}$ are the parameters for the the copula density c, and $\boldsymbol{\eta} = (\boldsymbol{\alpha}'_{\mathbf{x}}, \boldsymbol{\alpha}'_y, \boldsymbol{\theta}')'$ are the parameters of the joint density. The *exact log-likelihood* function is then

$$\begin{aligned} l(\boldsymbol{\eta}; \mathbf{x}, \mathbf{y}) &= \sum_{i=1}^{n} \ln c(F_X(x_i; \boldsymbol{\alpha}_x), F_Y(y_i; \boldsymbol{\alpha}_y); \boldsymbol{\theta}) \qquad (19.14) \\ &\quad + \sum_{i=1}^{n} (\ln f_X(x_i; \boldsymbol{\alpha}_x) + \ln f_Y(y_i; \boldsymbol{\alpha}_y)) \end{aligned}$$

and the exact maximum likelihood estimator (MLE) is defined as

$$\hat{\boldsymbol{\eta}}_{MLE} = \arg\max_{\boldsymbol{\eta}} \; l(\boldsymbol{\eta}; \mathbf{x}, \mathbf{y})$$

The numerical computation of the exact MLE may be difficult if there are many parameters in the marginal models and in the copula. Instead of maximizing the exact likelihood (19.14) as a function of $\boldsymbol{\eta}$, the copula parameters $\boldsymbol{\theta}$ may be estimated using a two-stage procedure. First, the marginal distributions F_X and F_Y are estimated. This could be done using parametric models (e.g., normal or Student's-t distributions), the empirical CDF, or a combination of an empirical CDF with an estimated generalized Pareto distribution for the tail. Next, given estimates $\hat{F}_X$ and $\hat{F}_Y$, a pseudo sample of observations from the copula

$$(\hat{u}_i, \hat{v}_i) = (\hat{F}_X(x_i), \hat{F}_Y(y_i)), \; i = 1, \ldots, n$$

is formed. Then, for a specified parametric copula $C(u, v; \boldsymbol{\theta})$ with density $c(u, v; \boldsymbol{\theta})$ and unknown parameters $\boldsymbol{\theta}$, the log-likelihood

$$l(\boldsymbol{\theta} : \hat{\mathbf{u}}, \hat{\mathbf{v}}) = \sum_{i=1}^{n} \ln c(\hat{u}_i, \hat{v}_i; \boldsymbol{\theta}) \qquad (19.15)$$

is maximized using standard numerical methods. This two-step method, due to Joe and Xu (1996), is called the *inference functions for margins* (IFM) method and the resulting estimator of $\boldsymbol{\theta}$ is called the *IFM estimator* (IFME). Properties of the IFME are discussed in Joe (1997) and Cherubini, Luciano, and Vecchiato (2004). Under standard regularity conditions, the IFME is consistent and asymptotically normally distributed. In particular, Joe (1997) showed that the IFME often nearly as efficient as the exact MLE.

19.5.3 *Fitting Copulas Using the* ***S+FinMetrics/EVANESCE*** *Function* ***fit.copula***

The IFM for fitting bivariate copulas is implemented in the `S+FinMetrics/ EVANESCE` function `fit.copula`. The arguments expected by `fit.copula` are

```
> args(fit.copula)
function(data, family = "normal", plot = F, init.est = NA,
trace = T, scale = 1, gradient = NULL, hessian = NULL,
control = NULL, ...)
```

where `data` is an object of class “`empirical.copula`” and “`family`” is the name of a parametric family of copulas to fit to the data. Table 19.1 lists the families available to be fit. Setting the optional argument `plot=T` produces a contour plot of the empirical copula overlaid with a contour plot of the fitted copula. The remaining parameters control aspects of the optimization performed by the `S-PLUS` function `nlminb`. The function returns an object of class “`copulaFit`” for which there are `print`, `IC`, and `compare` methods.

Example 131 *Estimate copulas for BMW and Siemens returns*

The visualization plots of the empirical copula for the BMW and Siemens returns, based on the fitted GPDs for the marginal distributions, suggest that a normal copula may be appropriate. To fit a normal copula with parameter δ by the IFM, use

```
>  cop.normal.fit = fit.copula(empcop.bs, family="normal",
+                              plot=T)
Iteration  0 : objective =  -1473.311
...
Iteration  7 : objective =  -1510.05
RELATIVE FUNCTION CONVERGENCE

> class(cop.normal.fit)
[1] "copulaFit"
> names(cop.normal.fit)
[1] "data"    "copula"  "vcov"    "loglike"
```

```
> cop.normal.fit
Fit Bivariate Copula by MLE --


Log-likelihood value:  1510

    Normal copula family.
    Parameters :
       delta  =  0.6567


Parameter Estimates, Standard Errors and t-ratios:
         Value Std.Error  t value
delta   0.6567   0.0065  101.0596
```

The IFME of δ is 0.6567, with an asymptotic standard error of 0.0065. The close match between the contour plot of the fitted copula and the empirical copula, given in Figure 19.18, indicates a good fit to the data. Kendall's tau and Spearman's rho for the fitted copula are

```
>  Kendalls.tau(cop.normal.fit$copula)
[1] 0.4561202
> Spearmans.rho(cop.normal.fit$copula)
[1] 0.6389838
```

These values are very close to the empirical estimates from the returns computed earlier.

For comparison purposes, consider fitting a Gumbel copula with parameter δ to the BMW and Siemens returns:

```
> cop.gumbel.fit <- fit.copula(empcop.bs,family="gumbel",
+                               plot=T)
Iteration  0 : objective =  -1348.776

Iteration  5 : objective =  -1410.845
RELATIVE FUNCTION CONVERGENCE

> cop.gumbel.fit
Fit Bivariate Copula by MLE --


Log-likelihood value:  1411

    Gumbel copula family; Extreme value copula.
    Parameters :
       delta  =  1.7717
```

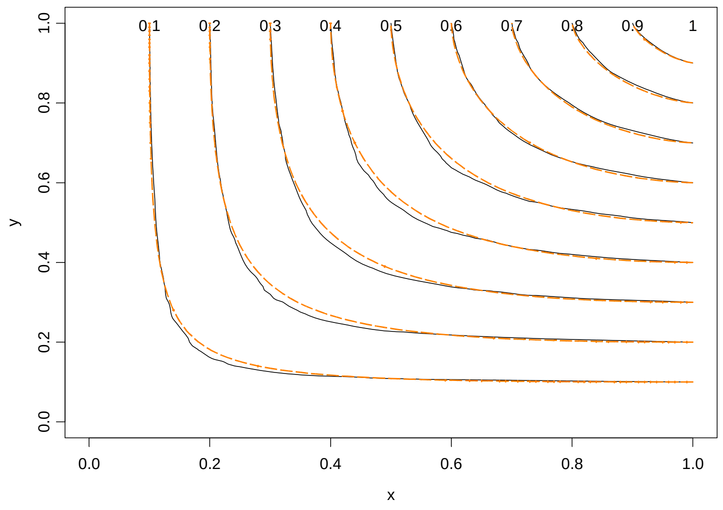


FIGURE 19.18. Contour plots of empirical copula and fitted normal copula for the daily returns on BMW and Siemens.

```
Parameter Estimates, Standard Errors and t-ratios:
        Value Std.Error t value
delta  1.7717  0.0196   90.5103
```

The IFME of δ is 1.7717, with an asymptotic standard error of 0.0196. The log-likelihood value of the fitted Gumbel copula is much lower than the log-likelihood value of the normal copula. Figure 19.19 shows that the contours of the fitted Gumbel copula do not match the contours of the empirical copula nearly as well as the normal copula.

The fits of the normal and Gumbel copula may be compared using the `compare.copulaFit` method function:

```
> compare.copulaFit(cop.normal.fit, cop.gumbel.fit)
                loglike       AIC       BIC        HQ
cop.normal.fit 1510.050 -3018.100 -3011.516 -3015.800
cop.gumbel.fit 1410.845 -2819.689 -2813.104 -2817.389
```

The normal copula is clearly a better fit to the bivariate returns than the Gumbel copula. It is emphasized here that the normal copula is a good fit to the bivariate returns *conditional* on using semiparametric GPD models for

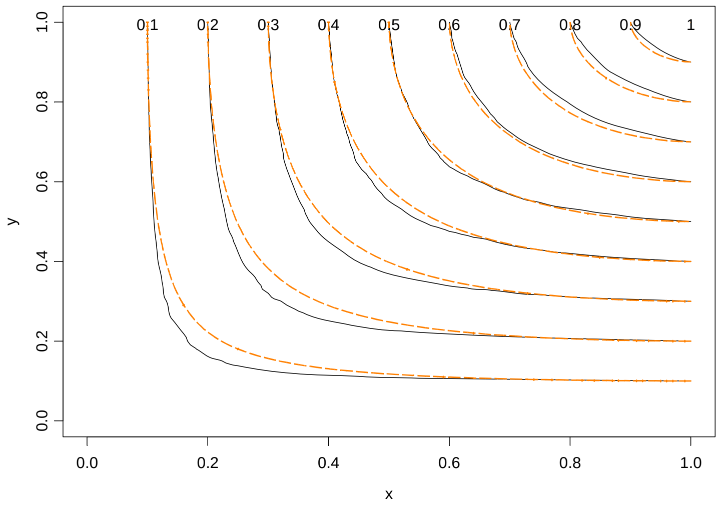


FIGURE 19.19. Contour plots of empirical copula and fitted Gumbel copula for the daily returns on BMW and Siemens.

the marginal distributions. If different models are used to fit the marginals, then the normal copula may not provide the best fit.

19.6 Risk Management Using Copulas

Cherubini, Luciano, and Vecchiato (2004) gave detailed examples of using copulas to compute risk measures for general portfolios of derivative securities including credit derivatives. In this section, the simple problem of computing common risk measures for a portfolio of two assets using copulas is described.

19.6.1 Computing Portfolio Risk Measures Using Copulas

Consider a one-period investment in two assets named asset 1 and asset 2. Let λ_1 and λ_2 denote the shares of wealth invested in asset 1 and asset 2, respectively, such that $\lambda_1 + \lambda_2 = 1$. Let X and Y denote the one-period continuously compounded (log) returns defined by

$$X = \ln\left(\frac{P_{1t+1}}{P_{1t}}\right),\ Y = \ln\left(\frac{P_{2t+1}}{P_{2t}}\right)$$

where P_{1t} and P_{2t} denote the prices of assets 1 and 2 at time period t, respectively. Then, the one-period log-return on the portfolio is given by

$$R = \ln(\lambda_1 e^X + \lambda_2 e^Y) \tag{19.16}$$

Since R is a nonlinear transformation of the random variables X and Y, the df of R is generally unknown.

Two common portfolio risk measures, introduced in Chapter 5, are *value-at-risk* (VaR) and *expected shortfall* (ES). For a portfolio with random return R and df F_R, the VaR and ES for a given loss probability $1-q$ are

$$\begin{aligned} \text{VaR}_q &= F_{-R}^{-1}(q) && (19.17) \\ \text{ES}_q &= E[-R| - R > \text{VaR}_q] && (19.18) \end{aligned}$$

Since the df F is generally unknown, analytic expressions for VaR_q and ES_q are generally not available.

19.6.2 Computing VaR and ES by Simulation

If it is possible to generate random simulations from the joint df of X and Y, then it is easy to compute numerical approximations to (19.17) and (19.18) using (19.16). Let $\{r_1, \ldots, r_n\}$ denote a n simulated values of R based on n simulated random pairs of X and Y. Then the simulation-based estimates of (19.17) and (19.18) are

$$\begin{aligned} \widehat{\text{VaR}_q} &= 100 \cdot q\text{th empirical quantile of } -r_t \text{ values} \\ \widehat{\text{ES}_q} &= \text{mean of } -r_t \text{ values greater than } \widehat{\text{VaR}_q} \end{aligned}$$

If the joint df of X and Y is specified by the marginal dfs F_X and F_Y and the copula C, then simulated values for X and Y may be obtained using the procedures described in Section 19.4.

Example 132 *Simulation-based VaR and ES estimates*

Let X and Y denote the daily log-returns on two assets. Consider a portfolio with $\lambda_1 = 0.7$ and $\lambda_2 = 0.3$. Suppose $X \sim N(0,00034, 0.0147^2)$ and $Y \sim N(0,00021, 0.01140^2)$. Furthermore, suppose the copula for X and Y is the Clayton copula with $\delta = 2$. Since the Clayton copula exhibits a lower tail dependence, there is a greater probability of large negative values of X and Y together than is predicted by the bivariate normal distribution with a similar correlation structure. The following commands may be used to compute the simulation-based estimates of (19.17) and (19.18) for $q = 0.95$ based on $n = 10000$ simulated values of the portfolio log-return R:

```
> # create bivd object
> clayton.biv <- bivd(cop=kimeldorf.sampson.copula(delta=2),
+                     Xmarg = "norm", Ymarg = "norm",
+                     param.Xmarg=c(0.00034, 0.0147),
+                     param.Ymarg=c(0.00021, 0.01140))
> # simulate from bivariate distn and compute log-return
> set.seed(123)
> xy.sim = rbivd(clayton.biv,10000)
> R.sim = log(0.7*exp(xy.sim$x) + 0.3*exp(xy.sim$y))
> VaR.95.est = quantile(-R.sim,probs=0.95)
> ES.95.est = -mean(R.sim[-R.sim > VaR.95.est])
> VaR.95.est
        95%
 0.02215953
> ES.95.est
[1] 0.02776794
```

With 5% probability, the portfolio could lose 2.216% or more in one day. If the return is less than -2.216%, then, on average, the loss is 2.777%

Example 133 *VaR and ES estimates for portfolio of BMW and Siemens stocks*

The S+FinMetrics/EVANESCE function VaR.exp.sim automates the calculation of VaR_q and ES_q for a portfolio of two assets when the bivariate dfs of X and Y are described by GPD margins and a given parametric copula C. This type of bivariate model with a normal copula was used to describe the joint df of the daily returns on BMW and Siemens stocks. Consider a portfolio with 70% of wealth invested in BMW ($\lambda_1 = 0.7$) and the remainder invested in Siemens ($\lambda_2 = 0.3$). Then, simulation-based estimates of (19.17) and (19.18) for $q = 0.99$ and $q = 0.95$ based on $n = 10,000$ simulated values of the portfolio log-return R may be computed using

```
> VaR.out = VaR.exp.sim(n=10000,Q=c(0.01,0.05),
+                       copula=cop.normal.fit$copula,
+                       x.est=gpd.bmw.fit2,
+                       y.est=gpd.siemens.fit2,
+                       lambda1=0.7,
+                       lambda2=0.3)
> VaR.out
 Simulation size VaR Q=0.01 VaR Q=0.05  ES Q=0.01 ES Q=0.05
           10000 0.03527972 0.01949605 0.04730781  0.029359
```

19.7 References

Capéraà, P., A.-L. Fourgères, and C. Genest (2000). "Bivariate Distributions with Given Extreme Value Attractor," *Journal of Multivariate Analysis*, 72, 30–49.

Carmona, R. (2004). *Statistical Analysis of Financial Data in Splus.* Springer-Verlag, New York.

Carmona, R. and J. Morrisson (2001). "Heavy Tails and Copulas with Evanesce," ORFE Tech. Report, Princeton University, Princeton, NJ.

Cherubini, U., E. Luciano and W. Vecchiato (2004). *Copula Methods in Finance.* John Wiley & Sons, New York.

Clayton, D.G. (1978). "A Model for Association in Bivariate Life Tables and Its Application in Epidemiological Studies of Familial Tendency in Chronic Disease Incidence," *Biometrika*, 65, 141-151.

Deheuvels, P. (1978). "Caractérisation Compléte des lois Extrêmes Multivariées et de la Convergence des Types Extêmes," *Publications de l'Institut de. Statistique de l'Université de Paris*, 23, 1–36.

Embrechts, P. C., F. Lindskog and A. McNeil (2003). "Modelling Dependence with Copulas and Applications to Risk Management," in S.T. Rachev (ed.) *Handbook of Heavy Tailed Distributions in Finance*, Pronide Publisher.

Embrechts, P., A. McNeil and D. Straumann (2000). "Correlation and Dependence in Risk Management: Properties and Pitfalls," in M. Dempster and H. K. Moffatt (eds.) *Risk Management: Value at Risk and Beyond*, Cambridge University Press, Cambridge.

Frank, M.J. (1979). "On the Simultaneous Associativity of $f(x, y)$ and $x + y - f(x, y)$," *Aequationes Mathematiques*, 19, 194–226.

Frees, E.W. and E. Valdez (1998). "Understanding Relationships Using Copulas," *North American Actuarial Journal*, 2, 1-25.

Galambos, J. (1975). "Order Statistics of Samples from Multivariate Distributions," *Journal of American Statistical Association*, 70, 674–680.

Genest, C. and R. J. MacKay (1986). "Copules Archimédiennes et Familles de lois Bidimensionnelles dont les Marges sont Données," *Canadian Journal of Statistics*, 14, 145–159, Genest, C. and L.-P. Rivest (1993). “Statistical Inference Procedures for Bivariate Archimedean Copulas,” *Journal of American Statistical Association*, 88(423), 1034–1043.

Gumbel, E. J. (1960). “Distributions des Valeurs Extrêmes en Plusiers Dimensions,” *Publications de l’Institut de. Statistique de l’Université de Paris*, 9, 171–173.

Hüsler, J. and R.-D. Reiss (1989). “Maxima of Normal Random Vectors: Beween Independence and Complete Dependence, *Statistics and Probability Letters*, 7, 283–286.

Joe, H. (1993). “Parametric Families of Multivariate Distributions with Given Margins,” *Journal of Multivariate Analysis*, 46, 262–282.

Joe, H. (1997). *Multivariate Models and Dependence Concepts.* Chapman & Hall, London.

Joe, H. and J. Xu (1996). “The Estimation Method of Inference Functions for Margins for Multivariate Models,” Technical Report No. 166, Department of Statistics, University of British Columbia, Vancouver.

Kimeldorf, G. and Sampson, A.R. (1975). “Uniform Representations of Bivariate Distributions,” *Communications in Statistics*, 4, 617-627.

Morrison, J.E. (2001). *Extreme Value Statistics with Applications in Hydrology and Financial Engineering*, Ph.D. thesis, Princeton University, Princeton, NJ.

Nelsen, R. B. (1999). *An Introduction to Copulas,* Lecture Notes in Statistics. Springer-Verlag, New York.

RiskMetrics, 1995. “RiskMetrics Technical Document,” 3rd ed., J.P. Morgan, New York.

Sklar, A. (1959). “Fonctions de Répartition à n Dimensions et Leurs Marges,” *Publications de l’Institut de. Statistique de l’Université de Paris*, 8, 229–231.

Tawn, J. A. (1988). “Bivariate Extreme Value Theory: Models and Estimation,” *Biometrika*, 75, 397–415.

20
Continuous-Time Models for Financial Time Series

20.1 Introduction

Many stochastic models in modern finance are represented in continuous-time. In these models, the dynamic behavior of the underlying random factors is often described by a system of *stochastic differential equations* (SDEs). The leading example is the option pricing model of Black and Scholes (1973), in which the underlying stock price evolves according to a geometric SDE. For asset pricing purposes, continuous-time financial models are often more convenient to work with than discrete-time models. For practical applications of continuous-time models, it is necessary to solve, either analytically or numerically, systems of SDEs. In addition, the simulation of continuous-time financial models is necessary for estimation using the Efficient Method of Moments (EMM) described in Chapter 23. This chapter discusses the most common types of SDEs used in continuous-time financial models and gives an overview of numerical techniques for simulating solution paths to these SDEs.

This chapter begins with two introductory sections: the first giving some basic background material on SDEs and their solutions, and the second presenting some common numerical schemes for simulating solutions to SDEs using discretization methods. Then, the `S+FinMetrics` functions that are available for simulating general systems of SDEs, as well as those tailored to some of the more popular models for financial data, are described and illustrated through examples.

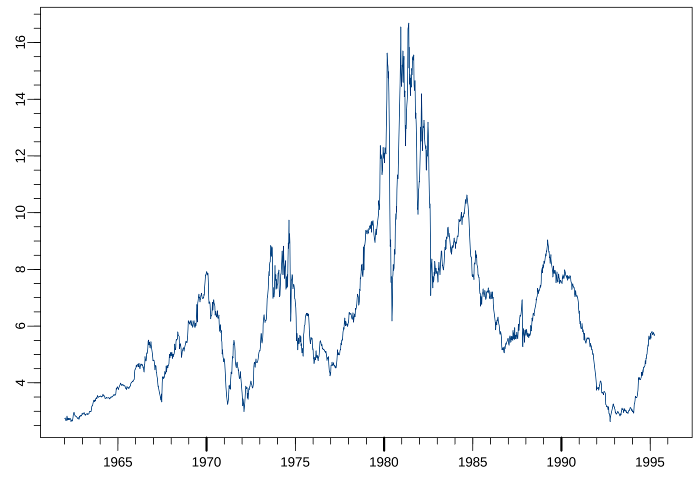


FIGURE 20.1. U.S. 3-month T-Bill rate, observed weekly, 1962 - 1995.

Merton (1990) and Duffie (1996) gave comprehensive treatments of continuous -time models in finance. The discussion and treatment of SDEs and methods for simulating their solutions presented in this chapter is meant to be intuitive and nontechnical and similar to the treatment in Neftci (1996), Baxter and Rennie (1996), and Seydel (2002). A thorough technical treatment of SDEs is given in Kloeden and Platten (1999) and Øksendal (1998).

20.2 SDEs: Background

Stochastic differential equations provide a framework for modeling continuous, dynamically changing quantities that exhibit randomness, or *volatility*, such as the U.S. Treasury bill rates shown in Figure 20.1. Applications of SDEs span the social and physical sciences. See Kloeden and Platten (1999) and Øksendal (1998) for numerous examples.

An SDE is represented by an equation of the form

$$dX_t = a(t, X_t)\,dt + b(t, X_t)\,dW_t \tag{20.1}$$

with a deterministic component defined by the function $a(t, X_T)$ (the instantaneous drift) and a random component defined by function $b(t, X_t)$ (the instantaneous diffusion). When $b \equiv 0$, the SDE reduces to an *ordinary differential equation* (ODE). The stochastic differential dW_t represents an infinitessimal increment of *Brownian motion*, W_t, also called the *Wiener process.* The Wiener process, first introduced in Chapter 4, is a continuous random variable such that for $s, t \geq 0$, $W_{s+t} - W_s$ is distributed as normal, $N(0, t)$, and is independent of the history of what the process did up to s. The formal definition of (20.1) comes from writing down its integral representation,

$$X_t = X_{t0} + \int_{t0}^{t} a(s, X_s)\, ds + \int_{t0}^{t} b(s, X_s)\, dW_s \tag{20.2}$$

and then defining the right-hand stochastic integral in terms of a limit of sums including finite Brownian increments. This leads to the "Ito" or "Stratonovich" calculus, depending on the definition of the integral. For technical details, see Kloeden and Platen (1999). For a financial approach, see Neftci (1996), and for a readable introduction, see Baxter and Rennie (1996).

A solution X_t to (20.1) or (20.2) is called a *stochastic process*, which can be thought of as being indexed by t and the different realizations W of Brownian motion. For fixed t, X_t is a random variable. For a fixed realization $W(t)$ of Brownian motion, one obtains a *sample path* $X(t)$.

For example,

$$dX_t = k\, dt + \sigma\, dW_t \tag{20.3}$$

has the explicit solution

$$X_t = kt + \sigma W_t \tag{20.4}$$

This is just the straight-line solution to $dX_t = k\, dt$ plus some Gaussian noise. Figure 20.2 shows different sample paths for an example of this simple SDE, and the superimposed straight line.

Equation (20.3) is an example of a *scalar* SDE. In general, (20.1) may be vector-valued, with X_t of dimension N and the Brownian motion dW_t of dimension M. In that case, the drift is an N-vector and the diffusion is a matrix of dimension $N \times M$. The individual components of X_t are sometimes called the *factors* of the model.

20.3 Approximating Solutions to SDEs

Stochastic differential equations like (20.3) with explicit solutions are rare. As with ODEs, there are a variety of methods for approximating solutions using discretization. `S+FinMetrics` supports three: Euler's method and the so-called "strong order 1" and "weak order 2" explicit methods of Platen (a reference for this entire section is Kloeden and Platen (1996)).

Euler's method is the simplest of the three and is a straightforward extension of the ODE technique of the same name. For a discretization step

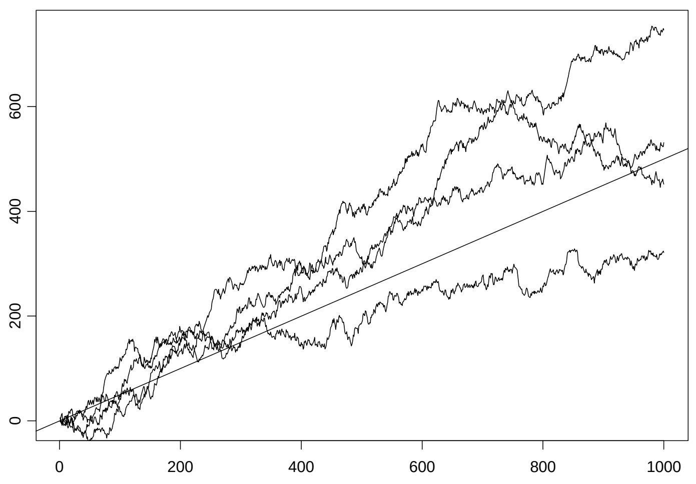


FIGURE 20.2. Sample paths for $X_t = \frac{1}{2}dt + 5W_t$ with $X_0 = 0$, and the straight line $X = t/2$.

size of Δt and $t_n = t_0 + n\Delta t$, the Euler's method approximation to (20.1), $Y_n \cong X_{t_n}$, is given by

$$Y_{n+1} = Y_n + a(t_n, Y_n)\Delta t + b(t_n, Y_n)\Delta W_n \tag{20.5}$$

where $Y_0 = X_0$ is the initial condition. Increments of the Brownian motion, $\Delta W_n = W_{t_{n+1}} - W_{t_n}$, can be simulated using pseudo randomly generated normal variates, since by the defining properties of Brownian motion, $\Delta W_n^j = N(0, \Delta t)$ for $j = 1, \ldots, M$. Each discretization step for Euler's method thus requires M random normals.

For the other two schemes, the notions of strong and weak convergence of approximations to solutions of SDEs need to be defined. The former concerns pathwise approximations, whereas the latter focuses on matching expectations of moments (or some other functional) of solutions.

A discrete approximation scheme Y^δ with discretization step size $\delta = \Delta t$ is said to *converge strongly* to X at time T with order $\gamma > 0$ if there exists a positive constant C that does not depend on δ such that

$$E(|X_T - Y^\delta(T)|) \leq C\delta^\gamma$$

The quantity $E(|X_T - Y^\delta(T)|)$ measures the closeness of sample paths at the endpoint T.

The approximation Y^δ is said to *converge weakly* with order $\beta > 0$ to X at time T if for every function g of sufficient regularity, there exists a positive constant C independent of δ such that

$$|E(g(X_T)) - E(g(Y^\delta(T)))| \leq C\delta^\beta$$

Thus, weak convergence of order β implies convergence of moments of all order (taking $g(X) = |X|^q$).

Under suitable smoothness and growth conditions on the drift and diffusion terms of the SDE, Euler's method is convergent of strong order 0.5 and weak order 1. Other schemes are usually devised to converge either strongly or weakly, and it is a matter of the type of problem that one wants to solve as to which is more appropriate. Higher-order schemes typically involve higher order derivatives of the drift and diffusion functions. "Implicit" approximation schemes require analytic expressions for these higher-order derivatives, whereas "explicit" schemes use approximations of these derivatives and thus require only the values of the drift and diffusion functions themselves. Only explicit schemes are implemented in `S+FinMetrics`.

The kth component of the *strong order 1 scheme* of Platen is given by

$$Y_{n+1}^k = Y_n^k + a^k\Delta t + \sum_{j=1}^{M} b^{k,j}\Delta W_n^j + \frac{1}{\sqrt{\Delta t}} \sum_{j_1,j_2=1}^{M} \{b^{k,j_2}(t_n, \Upsilon_n^{j_1}) - b^{k,j_2}\} I_{(j_1,j_2)} \tag{20.6}$$

with the supporting values

$$\Upsilon_n^j = Y_n + a\Delta t + b^j\sqrt{\Delta t}$$

Here, the a's and b's without arguments are taken to be evaluated at (t_n, Y_n), and b^j denotes the jth column of b. The multiple Ito stochastic integral

$$I_{(j_1,j_2)} = \int_{t_n}^{t_{n+1}} \int_{t_n}^{t_{n+1}} dW_{s_2}^{j_1}\ dW_{s_1}^{j_2}$$

can be evaluated exactly when $j_1 = j_2$ as

$$I_{(j_1,j_1)} = \frac{1}{2}\left\{(\Delta W_n^{j_1})^2 - \Delta t\right\}$$

For $j_1 \neq j_2$, however, $I_{(j_1,j_2)}$ must be approximated, using

$$\begin{aligned} I_{(j_1,j_2)}^p &= \Delta t\left(\frac{1}{2}\xi_{j_1}\xi_{j_2} + \sqrt{\rho_p}(\mu_{j_1,p}\xi_{j_2} - \mu_{j_2,p}\xi_{j_1})\right) \qquad (20.7)\\ &+ \frac{\Delta t}{2\pi}\sum_{r=1}^{p}\frac{1}{r}\left(\zeta_{j_1,r}(\sqrt{2}\xi_{j_2} + \eta_{j_2,r}) - \zeta_{j_2,r}(\sqrt{2}\xi_{j_1} + \eta_{j_1,r})\right) \end{aligned}$$

where

$$\rho_p = \frac{1}{12} - \frac{1}{2\pi^2}\sum_{r=1}^{p}\frac{1}{r^2}$$

and ξ_j, $\mu_{j,p}$, $\eta_{j,r}$, and $\zeta_{j,r}$ are independent $N(0,1)$ and $\xi_j = \frac{1}{\sqrt{\Delta t}}\Delta W_n^j$, for $j = 1,\ldots,M$ and $r = 1,\ldots,p$. The choice of p affects the accuracy of the approximation. Gallant (2003) recommended that $p > \frac{0.05}{\Delta t}$. Altogether, each discretization step of (20.6) requires $2M(p+1)$ random normals.

That is a lot of overhead. In fact, $p > 50$ in typical applications, which makes very long simulations such as those required by EMM prohibitively expensive using this scheme. However, the computation of $I^p_{(j_1,j_2)}$ can be avoided in the special case of *diagonal noise*; that is, when $M = N$ and b is a diagonal matrix with $\frac{d\,b^{j,j}}{dX^k} = 0$ for $k \neq j$. In that case, the strong order 1 scheme reduces to

$$Y^k_{n+1} = Y^k_n + a^k\Delta t + b^{k,k}\Delta W^k_n + \frac{1}{2\sqrt{\Delta t}}\{b^{k,k}(t_n,\Upsilon^k_n) - b^{k,k}\}\{(\Delta W^k_n)^2 - \Delta t\} \tag{20.8}$$

which requires M random normals for each discretization step.

A sub-case of diagonal noise is *additive noise*, when the diffusion b is constant. In that case, the term on the right of (20.8) drops out and the strong order 1 scheme reduces to Euler's method.

The *weak order 2 scheme* applies to *autonomous* SDEs; that is, when $a = a(X_t)$ and $b = b(X_t)$ are not explicitly dependent on the variable t. Under that assumption, the weak order 2 scheme is given by

$$\begin{aligned} Y_{n+1} = Y_n &+ \frac{1}{2}(a(\Upsilon) + a)\Delta t \\ &+\frac{1}{4}\sum_{j=1}^{M}[\left(b^j\left(R^j_+\right) + b^j\left(R^j_-\right) + 2b^j\right)\Delta W^j_n \\ &+ \sum_{r=1,r\neq j}^{M}\left(b^j\left(U^j_+\right) + b^j\left(U^j_-\right) - 2b^j\right)\Delta W^j_n] \\ &+\frac{1}{4}\sum_{j=1}^{M}[\left(b^j\left(R^j_+\right) - b^j\left(R^j_-\right)\right)\left\{\left(\Delta W^j_n\right)^2 - \Delta t\right\} \\ &+ \sum_{r=1,r\neq j}^{M}\left(b^j\left(U^j_+\right) - b^j\left(U^j_-\right)\right)\left\{\Delta W^j_n\Delta W^r_n + V_{r,j}\right\}]\,(\Delta t)^{-1/2} \end{aligned} \tag{20.9}$$

with supporting values

$$\begin{aligned} \Upsilon &= Y_n + a\Delta t + b\Delta W_n \\ R^j_\pm &= Y_n + a\Delta t \pm b^j\sqrt{\Delta t} \\ U^j_\pm &= Y_n + b^j\sqrt{\Delta t} \end{aligned}$$

and with $\Delta W^j_n = N(0,\Delta t)$ and V_{j_1,j_2} distributed as two-point random variables with $P(V_{j_1,j_2} = \pm\Delta t) = \frac{1}{2}$ for $j_2 = 1,\ldots,j_1-1$, $V_{j_1,j_1} = -\Delta t$, and $V_{j_1,j_2} = -V_{j_2,j_1}$. Each discretization step thus requires M random normals for the $\{\Delta W_n^j\}$ and $M(M-1)/2$ uniform random numbers for the $\{V_{j_1,j_2}\}$.

In the case of additive noise, the weak order 2 scheme reduces to

$$Y_{n+1} = Y_n + \frac{1}{2}\left\{a\left(Y_n + a\Delta t + b\Delta W_n\right) + a\right\}\Delta t + b\Delta W_n. \tag{20.10}$$

This requires M random normals and no random uniforms, per discretization the step.

20.4 `S+FinMetrics` Functions for Solving SDEs

`S+FinMetrics` includes functions that implement one or more of the three schemes just described for simulating sample paths to both general and specific systems of SDEs. Most of the simulators follow a specific template, as they are designed to be used with the `S+FinMetrics` function `EMM` for model estimation. The calling sequence for these "`gensim`" functions is as follows:

```
xxx.gensim(rho, n.sim, n.var, n.burn, aux)
```

This is a very general framework for supporting simulations of models that are not necessarily defined by systems of SDEs. Here, `rho` is a vector of parameters that define the model to be simulated, `n.sim` is the length of the simulation, `n.var` is the number of variables in the simulation, `n.burn` is the number of "burn-in" simulation steps that are to be performed before the `n.sim` steps to be returned, and `aux` is a list designed to be a catch all container for any other parameters that a particular simulator may require. Usually, `aux` is supplied by a separate `S-PLUS` function that is tailored to the particular `gensim` simulator. Specific examples in the following sections will illustrate and further explain these arguments.

Functions for performing simulations for specific systems of SDEs are described in Section 20.4.1. Section 20.4.2 describes functions for simulating general systems of SDEs.

20.4.1 Problem-Specific Simulators

`S+FinMetrics` includes several `gensim` functions that are tailored to specific systems of SDEs, namely the Ornstein-Uhlenbeck (also called Vasicek), CIR (Cox-Ingersoll-Ross), and IRD (two-factor interest rate diffusion). Examine the first example carefully, since the other `gensim` functions use many of the same arguments and follow the same behavior.

Ornstein-Uhlenbeck Process

The Ornstein-Uhlenbeck (OU), or Vasicek, process,

$$dr_t = \kappa(\theta - r_t)\,dt + \sigma\,dW_t \tag{20.11}$$

is a simple single-factor model for interest rate behavior originally described in Vasicek (1977). There are three parameters, κ, θ, and σ, which are usually all strictly positive. The long-run behavior of any solution is to hover around θ (the *level*). This property is called *mean reversion*; mean-reverting SDEs are also called *stationary.* The parameter κ controls the rate of mean reversion, whereas σ controls the volatility.

The OU SDE has the following exact solution:

$$r_t = \theta + e^{-\kappa t}(r_0 - \theta) + \sigma \int_0^t e^{\kappa(s-t)}\, dW_s \tag{20.12}$$

The latter stochastic integral must still be simulated. Functions `OU.gensim` and `OU.aux` are available for simulating exact solutions to the OU process in the `gensim` framework.

Example 134 *Simulate exact solution to the OU process*

To simulate 1000 values from (20.12) calibrated to match annualized data sampled weekly, use

```
# In OU.gensim, rho is packed as c(kappa, theta, sigma)
> kappa = .4; theta = .08; sigma = .1
> sim.ou = OU.gensim(rho = c(kappa, theta, sigma),
+                    n.sim = 1000, n.burn = 500,
+                    aux = OU.aux(X0 = .1, ndt = 25,
+                    seed = 1, t.per.sim = 1/52))
# Plot the simulation
> tsplot(sim.ou)
# Plot the long-run mean
> abline(h=theta)
```

See Figure 20.3 for the simulated solution. The auxiliary information for the simulator is supplied by the `S+FinMetrics` function `OU.aux`. The argument `ndt` defines the number of discretization steps per simulation step, and argument `t.per.sim` defines the time length of each simulation step. The internal discretization step size is thus $\Delta t =$ `t.per.sim/ndt`. In the above simulation, the 1000 simulated values of r_t are separated by an implied time of 1/52 (1 week, if the parameters are assumed to generate annualized data), and successive values are separated by 25 internal discretization steps. With a burn-in period of 500 simulation steps, that is a total of $(1000{+}100)\times 25 = 27,500$ discretization steps of size $\Delta t = 1/(52 \times 25) \simeq 0.0008$ for the entire simulation. Notice that the simulated solution generates negative values for r_t.

Other arguments to `OU.aux`, including additional optional arguments not used in the above example, are listed in Table 20.1. Note that the exact solution approximation still requires a vector `z` of random normals (one per discretization step, the same as Euler's method) in order to approximate the stochastic integral in (20.12).

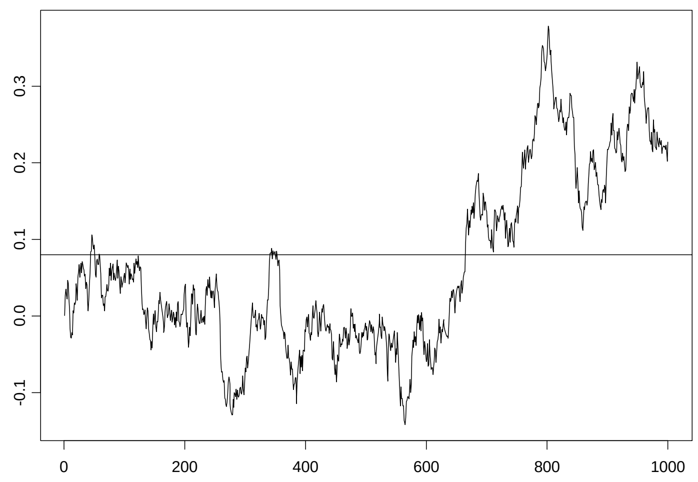


FIGURE 20.3. OU sample path; horizontal line is the long-run mean.

CIR Process

The Cox-Ingersoll-Ross (CIR) (1985) SDE,

$$dr_t = \kappa(\theta - r_t)\,dt + \sigma\sqrt{r_t}\,dW_t \tag{20.13}$$

is almost identical to the OU model, except that volatility $\sigma\sqrt{r_t}$ increases and decreases with r_t, whereas for OU processes it is constant. The CIR model guarantees that r_t is non-negative (as long as κ and θ are strictly positive). The `gensim` functions for CIR are `CIR.gensim` and `CIR.aux`,

Argument	**Description**
`X0`	Initial condition for the SDE.
`z`	Vector of random normals, of length `ndt*(n.sim + n.burn)`, required by the exact Approximation. If not specified, then it is generated internally.
`seed`	Random number seed, for reproducibility in case `z` is generated internally. Default value is 0.
`lbound, ubound`	The simulation is truncated to lie in the interval `[lbound, ubound]`. Default is `[-100, 100]`.

TABLE 20.1. Arguments for `OU.aux`

and they always use Euler's method. Their usage is identical to that of `OU.gensim` and `OU.aux`.

Example 135 *Simulate solution to CIR model using Euler's method*

To simulate 1000 values from (20.13) based on Euler's method calibrated to match annualized data sampled weekly, use

```
> kappa = .4; theta = .08; sigma = .1
> rho = c(kappa, theta, sigma)

# pass in random normals for Brownian motion, although
# CIR.gensim can generate these internally on the fly.
> ndt = 25; n.sim = 1000; n.burn = 500
> set.seed(1)
> z = rnorm(ndt*(n.burn + n.sim))

> sim.cir = CIR.gensim(rho = rho, n.sim = n.sim,
+                      n.burn = n.burn,
+                      aux = CIR.aux(X0 = theta,
+                          ndt = ndt, z = z,
+                          t.per.sim = 1/52))
# Plot the simulation
> tsplot(sim.cir)
# Plot the long-run mean
> abline(h=theta)
```

See Figure 20.4 for the simulated solution. Compare with the OU example with the same parameters of Figure 20.3, where the same random seed, and hence the same random normal vector, is used to define the discretized sample path. Note the difference in scale between the two plots and that simulated values for r_t from the CIR model are always positive.

The presence of the square root in (20.13) can be problematic for simulation. For although exact solutions to CIR processes are guaranteed to stay positive, discretization error can lead to negative values for r_t under the square root sign. The function `CIR.gensim` handles this by truncating the instantaneous standard deviation $\sigma\sqrt{r_t}$ at 0 whenever discretized values of r_t are pushed negative.

IRD Process

Gallant and Tauchen (2002) presented the following two-factor interest rate diffusion (IRD) model for modeling the short-term interest rate, s_t.

$$\begin{pmatrix} dv_t \\ ds_t \end{pmatrix} = \begin{pmatrix} a_v + a_{vv}v_t + a_{vs}s_t \\ a_s + a_{ss}s_t \end{pmatrix} dt \tag{20.14}$$

$$+ \begin{pmatrix} b_{1v} + b_{1vv}v_t + b_{1vs}s_t & b_{2v} \\ 0 & (b_{2s} + b_{2ss}s_t)e^{v_t} \end{pmatrix} \begin{pmatrix} dW_{1,t} \\ dW_{2,t} \end{pmatrix} \tag{20.15}$$

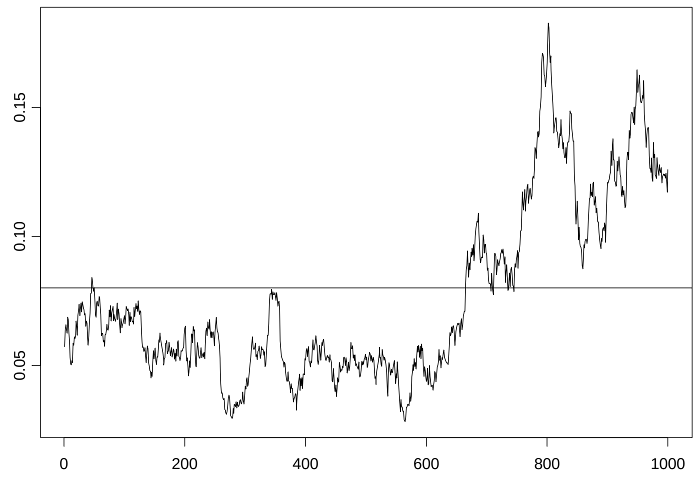


FIGURE 20.4. CIR sample path; horizontal line is the long-run mean.

The component v_t is an unobserved factor contributing to the volatility of s_t. There are 11 parameters describing the process. Normalization is usually accomplished by setting $a_v = -a_{vv}$. Functions `IRD.gensim` and `IRD.aux` simulate solutions to (20.14) and (20.15) using Euler's method.

Example 136 *Simulate solution to two-factor IRD using Euler's method*

To simulate 1000 values of v_t and s_t from (20.14) and (20.15) based on Euler's method, calibrated to match annualized data in percent sampled weekly, use

```
# Set values for the parameters
> av = .18; avv = -av; avs = -.0088; as = .019; ass = -.0035
> b1v = .69; b1vv = 0; b1vs = -.063; b2v = 0; b2s = .038
> b2ss = -.017

# Generate the random normals for this simulation
> n.sim = 10000; n.burn = 10000; ndt = 25
> set.seed(0)
> ird.z = rnorm(2*ndt*(n.sim + n.burn))

# IRD.gensim assumes rho is packed as follows.  Note that av
# is missing from the list, as the normalization av = -avv
# is assumed.
```

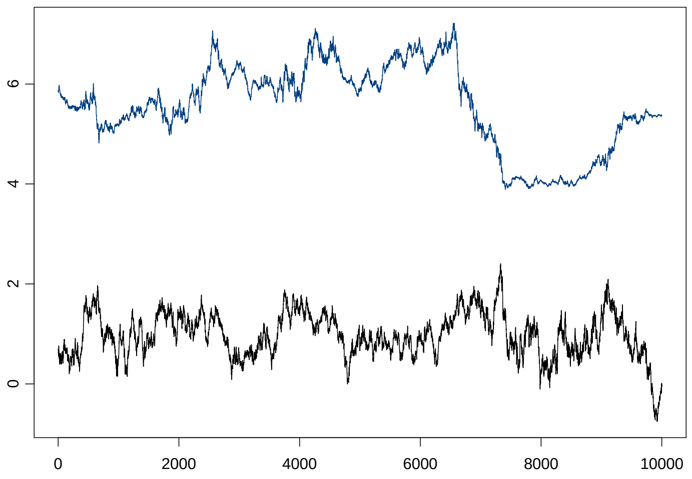


FIGURE 20.5. Two factor interest rate diffusion sample path. The short rate, s_t, appears on top.

```
> rho = c(avv, avs, as, ass, b1v, b1vv, b1vs, b2v, b2s, b2ss)
> ird.sim = IRD.gensim(rho = rho, n.sim = n.sim,
+                n.burn = n.burn, n.var = 2,
+                aux = IRD.aux(ndt = ndt, t.per.sim = 1/52,
+                z = ird.z, X0 = c(1, 5), returnc = c(T, T)))
```

The simulation is stored in the object `ird.sim` as a single vector of length 20000 (= `n.var*n.sim`), with the first two entries corresponding to $\{v_{t_0}, s_{t_0}\}$, followed by $\{v_{t_1}, s_{t_1}\}$, and so on. The following plots both factors:

```
# Plot v(t) and s(t)
> tsplot(t(matrix(ird.sim, nrow = 2)))
```

and the resulting plot is shown in Figure 20.5. Note that the initial condition argument `X0` is a vector for both factors. Argument `returnc` appears in this and other auxiliary functions for multi-factor models; it is a logical vector of length equal to the number of factors describing which factors should be returned in the simulation. In this case, it is specified that all factors be returned; the default is to return only the short rate component (`returnc = c(F,T)`), since the observed factor is the only one that would be used for estimation purposes. The number of `T`'s in `returnc` needs to match argument `n.var`.

20.4.2 General Simulators

The S+FinMetrics functions euler.pcode.gensim, strong1.pcode.gensim, and weak2.pcode.gensim simulate sample paths for solutions to univariate and multivariate SDEs with nearly arbitrary drift and diffusion terms using Euler's method, the strong order 1 scheme, and the weak order 2 scheme, respectively. The function euler1d.pcode.gensim simulates solutions to univariate SDE's using Euler's method. A pseudo code grammar is used to define mathematical expressions for the drift and diffusion. The price to pay for this flexibility is speed; these functions are two to five times slower than compiled C. Accompanying the pcode.gensim functions are the pcode.aux functions euler.pcode.aux, strong1.pcode.aux, and weak2.pcode.aux used to specify auxiliary information required for the simulation. The arguments to euler.pcode.aux and weak2.pcode.aux are listed in Table 20.2. The arguments to strong1.pcode.aux include the argument p, which corresponds to p in (20.7). The arguments to weak2.pcode.aux include the argument u for passing in uniform random numbers for defining the $\{V_{j_1,j_2}\}$ in (20.9) (these are generated internally by default). These arguments are explained in the following examples.

Example 137 *Simulate solution to OU process using pcode.gensim functions*

The auxiliary information required to simulate 1000 values for the solution to the OU process (20.11) calibrated to annualized weekly data is

```
> n.sim = 1000; n.burn = 500; ndt = 25
> set.seed(1)
> z = rnorm(ndt*(n.sim + n.burn))
> ou.names = c("kappa", "theta", "sigma")
> ou.eu.aux1 = euler.pcode.aux(ndt=25, t.per.sim=1/52,
+              X0 = 0.1, z = z,
+              drift.expr   = expression(kappa*(theta - X)),
+              diffuse.expr = expression(sigma),
+              rho.names = ou.names)
```

The n.sim=1000 simulated values of r_t are separated by an implied time of t.per.sim=1/52 (1 week, since the parameters are assumed to generate annualized data), and successive values are separated by ndt=25 internal discretization steps. With a burn-in period of n.burn=500 simulation steps, that is a total of $(1000 + 100) \times 25 = 27,500$ discretization steps of size $\Delta t = 1/(52 \times 25) \simeq 0.0008$ for the entire simulation. The random normal variables required for the simulation are precomputed and supplied in the vector z. The drift and diffusion are defined by using the S-PLUS expression function in the arguments drift.expr and diffuse.expr. The state vector is referred to by X. The model parameter names are specified in the vector assigned to the rho.names argument. Notice that

Argument	Description
`drift.expr`	Expression for drift function.
`diffuse.expr`	Expression for diffusion function.
`rho.names`	Vector of strings that can be used in drift and diffuse. expressions for accessing elements of the rho vector.
`M`	Dimension of the Brownian motion process.
`N`	Number of factors in the SDE.
`t.per.sim`	Time length of each simulation step.
`ndt`	Number of discretization steps per simulation step.
`z`	Vector of random normals. Length of `z` is: `(n.sim + n.burn)ndt*M` for Euler's method; `(n.sim + n.burn)2*ndt*M*(p + 1)` for Strong 1; `(n.sim + n.burn)*ndt*M` for Weak 2.
`u`	Vector of random uniforms, for Weak 2 scheme, of length`(n.burn+n.sim)*ndt*M*(M-1)/2`.
`p`	Value to control approximation to multiple Ito integral used by Strong order 1 scheme. The default value is `ceiling(0.05/(t.per.sim/ndt))`
`seed`	Random number seed, for reproducibility in case `z` and/or `u` is generated internally. Default value is 0.
`X0`	Initial condition for the SDE. Vector of length `N`.
`returnnc`	Logical vector of length `N` indicating which components of the simulation to return.
`lbound`, `ubound`	The simulation is truncated to lie in the interval `[lbound, ubound]`. Default is `[-100, 100]`.

TABLE 20.2. Arguments for `pcode.aux` functions

the drift and diffusion terms are specified in a way that is analogous to their mathematical representation. For the OU process, these terms are $a(t, X_t) = \kappa(\theta - X_t)$ and $b(t, X_t) = \sigma$, respectively.[1] The expressions for the drift and diffusion support the arithmetic operations +, -, *, /, and ^, basic functions such as trigonometric and exponential, max/min, and logical functions and conditionals. See the online help file for `euler.pcode.aux` for a complete list.

The expressions for the drift and diffusion specified in the call to `euler.pcode.aux` may be tested using the function `euler.pcode.test`. For example, to evaluate the drift and diffusion for the OU process with parameters $\kappa = 0.4$, $\theta = 0.08$, and $\sigma = 0.1$ at $t = 0$ and $r_0 = 0.08$, use

```
> rho.test = c(0.4, 0.08, 0.1)
> euler.pcode.test(rho.test,X=0.08,aux=ou.eu.aux1)
```

[1]If a vector of parameter names is not specified, the parameters are assumed to be in a vector called `rho` and the components of `rho` can be used directly in the expressions; for example, `rho[1]` for `kappa`, `rho[2]` for `theta` and `rho[3]` for `sigma`.

```
$drift:
[1] 0

$diffuse:
     [,1]
[1,]  0.1
```

With the auxiliary information specified, the simulated solution to the OU process may be computed using either `euler1d.pcode.gensim` or `euler.pcode.gensim`:

```
> sim.ou.eu1 = euler1d.pcode.gensim(rho=c(0.4, 0.08, 0.1),
+              n.sim = n.sim, n.burn = n.burn,
+              aux = ou.eu.aux1)

> sim.ou.eu = euler.pcode.gensim(rho = c(0.4, 0.08, 0.1),
+             n.sim = n.sim, n.burn = n.burn,
+             aux = ou.eu.aux1)
```

The simulated solutions look identical to Figure 20.3. For univariate SDEs, the function `euler1d.pcode.gensim` is slightly more efficient than `euler.pcode.gensim`. Simulations produced using `strong1.pcode.gensim` and `weak2.pcode.gensim` work essentially the same and have nearly the same arguments. For the OU process, which has additive noise, the strong order 1 scheme reduces to Euler's method.

Some of the simulated values of r_t in (20.11) are negative. To create simulated values for r_t that are bounded from below by zero, set `lbound=0` in the auxiliary list variable `ou.eu.aux1`:

```
> ou.eu.aux1$lbound = 0
> sim.ou.eu1 = euler1d.pcode.gensim(rho=c(kappa,theta,sigma),
+              n.sim = n.sim, n.burn = n.burn,
+              aux = ou.eu.aux1)
> tsplot(sim.ou.eu1)
```

The simulated solution path truncated at zero is illustrated in Figure 20.6.

Example 138 *Compare Euler's method and weak order 2 scheme for OU process*

To compare Euler's method and the weak order 2 scheme on the OU process of the previous example, use

```
> sim.ou.wk = weak2.pcode.gensim(rho = c(kappa, theta, sigma),
+           n.sim = n.sim, n.burn = n.burn, n.var = 1,
+           aux = weak2.pcode.aux(ndt = ndt, t.per.sim = 1/52,
+           N = 1, M = 1, X0 = .1, z = z,
+           drift.expr   = expression(kappa*(theta - X)),
+           diffuse.expr = expression(sigma),
```

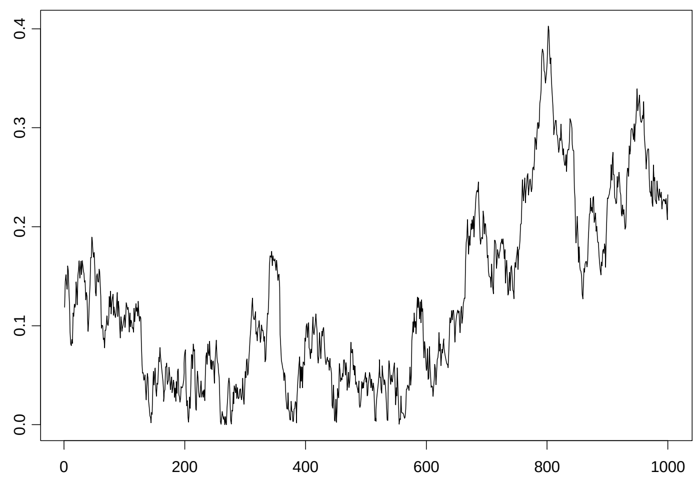


FIGURE 20.6. Simulated OU sample path with negative values of r_t replaced with zeros.

```
+           rho.names = c("kappa", "theta", "sigma")))
# Plot the simulation
> tsplot(sim.ou.wk)
# Plot the absolute differences with the exact solution.
> tsplot(abs(sim.ou.wk - sim.ou))
> lines(abs(sim.ou.eu - sim.ou), col = 2)
```

Since the OU process has additive noise the weak 2 scheme reduces to (20.10) and no uniform random numbers are required to produce the simulation. The weak order 2 simulation (not shown) looks identical when plotted next to the exact simulation of Figure 20.3. Figure 20.7 shows absolute differences between the weak and exact simulations and between Euler's method and exact simulations. Note that this is not really a fair comparison, since weak convergence is aiming for better average behavior over all paths, but it is, nevertheless, instructive to see how the two methods compare on a single path.

Example 139 *Simulate solutions to generalized CIR model*

Chan, Karolyi, Longstaff, and Sanders (1992), hereafter CKLS, considered estimating the parameters of the continuous-time interest rate diffusion model

$$dr_t = (\alpha_0 + \beta_0 r_t)dt + \sigma_0 r_t^{\gamma_0} \, dW_t \tag{20.16}$$

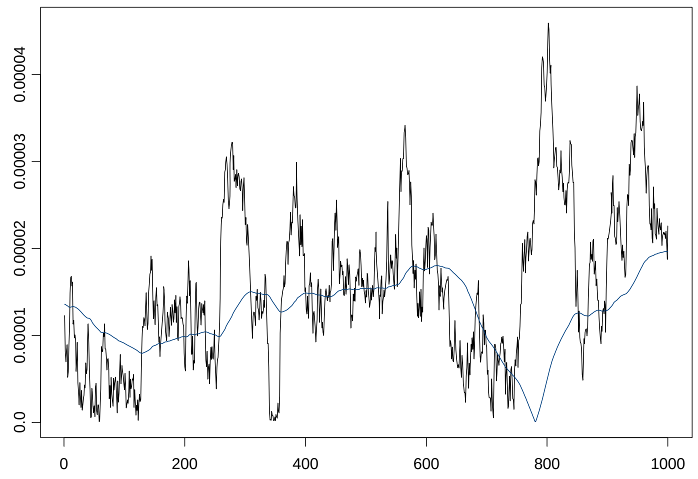


FIGURE 20.7. Comparison of Euler's method and the weak order 2 scheme for simulating an OU sample path. The smooth curve is the absolute difference between the Euler and exact simulations. The more volatile curve is the absolute difference between the weak order 2 and exact simulations.

The process (20.16) is sometimes referred to as the *generalized CIR process*. In (20.16), the drift function $(\alpha_0 + \beta_0 r_t)\ dt$ may be reparameterized as $\kappa_0(\mu_0 - r_t)\ dt$, where $\alpha_0 = \kappa_0\mu_0$ and $\beta_0 = -\kappa_0$. The parameter μ_0 is the long-run mean, and the parameter κ_0 determines the speed of mean reversion. The model (20.16) encompasses a variety of models that have been proposed for the short-term interest rate. These models and the corresponding parameter restrictions on (20.16) are summarized in Table 20.3: The Merton model is a simple Brownian motion, the Vasicek model is an OU process, and the Dothan model may be viewed as a restricted version of the Merton or Vasicek models. These models all imply constant volatility for r_t. The remaining models allow for stochastic volatility. The CIR SR model is the so-called "square root" model, the GBM is the geometric Brownian motion model, and the CEV model is the constant elasticity of variance model.

Simulated solutions from (20.16) for the nine models in Table 20.3, based on the estimated parameters from CKLS Table III, using Euler's method may be produced using

```
> n.sim.ckls = 306; n.burn.ckls = 0; ndt.ckls = 25
> set.seed(123)
> z.ckls = rnorm(ndt*(n.sim.ckls + n.burn.ckls))
```

$dr_t = (\alpha + \beta r)dt + \sigma r_t^{\gamma} dW_t$

Model	**Specification**	$\boldsymbol{\alpha}$	$\boldsymbol{\beta}$	$\boldsymbol{\gamma}$
Merton	$dr_t = \alpha dt + \sigma dW_t$		0	0
Vasicek	$dr_t = (\alpha + \beta r_t)dt + \sigma dW_t$			0
CIR SR	$dr_t = (\alpha + \beta r_t)dt + \sigma r_t^{1/2} dW_t$			1/2
Dothan	$dr_t = \sigma dW_t$	0	0	0
GBM	$dr_t = \beta r_t dt + \sigma r_t dW_t$	0		1
Brennan-Schwartz	$dr_t = (\alpha + \beta r)dt + \sigma r_t dW_t$			1
CIR VR	$dr_t = \sigma r_t^{3/2} dW_t$	0	0	3/2
CEV	$dr_t = \beta r_t dt + \sigma r_t^{\gamma} dW_t$	0		

TABLE 20.3. Parameter restrictions imposed by alternative short-term interest rate models

```
> gcir.names = c("alpha","beta","sigma","gamma")
> gcir.aux=euler.pcode.aux(
+               drift.expr = expression(alpha + beta*X),
+               diffuse.expr = expression(sigma*X^gamma),
+               rho.names = gcir.names,
+               t.per.sim = 1/12, ndt = 25,
+               z = z.ckls,
+               X0 = 0.067)

# generalized cir model
> rho.gcir = c(0.0408,-0.5921,sqrt(1.6704),1.4999)
> gcir.sim = euler1d.pcode.gensim(rho.gcir,n.sim = n.sim.ckls,
+           n.burn = n.burn.ckls, aux = gcir.aux)
# merton model
> rho.merton = c(0.0055, 0.0, sqrt(0.0004), 0.0)
> merton.sim = euler1d.pcode.gensim(rho.merton,
+     n.sim = n.sim.ckls, n.burn = n.burn.ckls,aux = gcir.aux)
# vasicek model
> rho.vasicek = c(0.0154, -0.1779, sqrt(0.0004), 0.0)
> vasicek.sim = euler1d.pcode.gensim(rho.vasicek,
+     n.sim = n.sim.ckls, n.burn = n.burn.ckls,aux = gcir.aux)
# cir.sr model
> rho.cir.sr = c(0.0189, -0.2339, sqrt(0.0073), 0.5)
> cir.sr.sim = euler1d.pcode.gensim(rho.cir.sr,
+     n.sim = n.sim.ckls, n.burn = n.burn.ckls,aux = gcir.aux)
# dothan model
> rho.dothan = c(0.0, 0.0, sqrt(0.1172), 1)
> dothan.sim = euler1d.pcode.gensim(rho.dothan,
+     n.sim = n.sim.ckls, n.burn = n.burn.ckls,aux = gcir.aux)
# gbm model
> rho.gbm = c(0.0, 0.1101, sqrt(0.1185), 1)
```

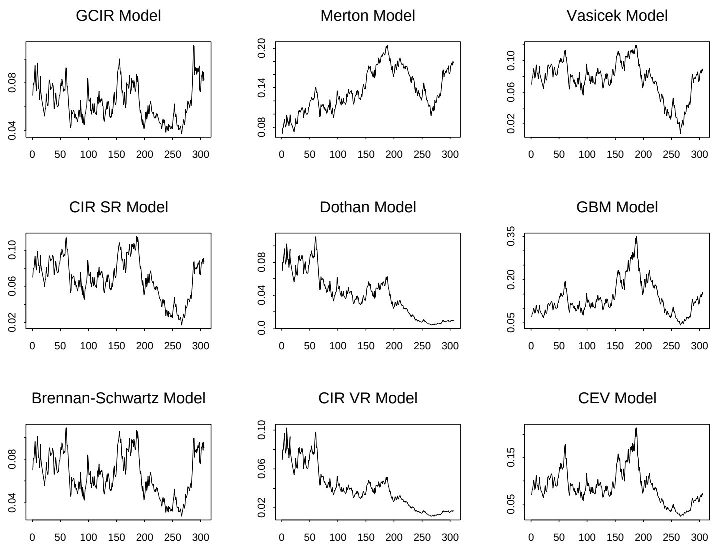


FIGURE 20.8. Simulated solutions from GCIR and restricted models based on Euler's method

```
> gbm.sim = euler1d.pcode.gensim(rho.gbm,n.sim = n.sim.ckls,
+           n.burn = n.burn.ckls,aux = gcir.aux)
# bs model
> rho.bs = c(0.0242, -0.3142, sqrt(0.1185), 1)
> bs.sim = euler1d.pcode.gensim(rho.bs,n.sim = n.sim.ckls,
+          n.burn = n.burn.ckls,aux = gcir.aux)
# cir.vr model
> rho.cir.vr = c(0.0, 0.0, sqrt(1.5778), 1.5)
> cir.vr.sim = euler1d.pcode.gensim(rho.cir.vr,
+     n.sim = n.sim.ckls, n.burn = n.burn.ckls,aux = gcir.aux)
# cev model
> rho.cev = c(0.0, 0.1026, sqrt(0.5207), 1.2795)
> cev.sim = euler1d.pcode.gensim(rho.cev,n.sim = n.sim.ckls,
+           n.burn = n.burn.ckls,aux = gcir.aux)
```

The simulated solutions are illustrated in Figure 20.8.

Example 140 *Simulate solutions to two-factor IRD*

Andersen and Lund (1997) considered the following two-factor extension of the CKLS model:

$$\begin{aligned} d\log\sigma_t^2 &= \kappa_2(\alpha - \log\sigma_t^2)dt + \xi dW_{1t} \\ dr_t &= \kappa_1(\mu - r_t)dt + \sigma_t r_t^\gamma dW_{2t},\ \gamma > 0 \end{aligned} \tag{20.17}$$

where W_{1t} and W_{2t} are independent Brownian motion processes, to model weekly observations on the U.S. 3-month T-bill rate. The specification implies mean reversion of the interest rate level as well as the (log-) volatility. Define $v_t = \log \sigma_t^2 = 2 \log \sigma_t$. Then, $\exp(v_t/2) = \sigma_t$ and (20.17) may be re-expressed as

$$\begin{aligned} dv_t &= \kappa_2(\alpha - v_t)dt + \xi dW_{1t} \\ dr_t &= \kappa_1(\mu - r_t)dt + \exp(v_t/2) r_t^\gamma dW_{2t}, \ \gamma > 0 \end{aligned} \tag{20.18}$$

Let $\mathbf{X}_t = (v_t, r_t)'$ and $\mathbf{W}_t = (W_{1t}, W_{2t})'$ so that (23.32) may be expressed in matrix form as

$$d\mathbf{X}_t = \mathbf{a}(\mathbf{X}_t, t)dt + \mathbf{b}(\mathbf{X}_t, t)d\mathbf{W}_t$$

where

$$\mathbf{a}(\mathbf{X}_t, t) = \begin{pmatrix} \kappa_2(\alpha - v_t) \\ \kappa_1(\mu - r_t) \end{pmatrix}, \ \mathbf{b}(\mathbf{X}, t) = \begin{pmatrix} \xi & 0 \\ 0 & \exp(v_t/2) r_t^\gamma \end{pmatrix} \tag{20.19}$$

are the drift and diffusion functions.

Simulated solutions for $\mathbf{X}_t$ from (20.17) may be easily computed using the general `pcode.gensim` functions. For example, the auxiliary information to set the drift and diffusion function (20.19) for Euler's method and to calibrate the simulation to the weekly data on 3-month T-bills is

```
> ndt = 25; t.per.sim = 1/52; n.var = 2; n.sim = 2000
> n.burn = 100
> set.seed(0)
> z.euler = rnorm(ndt*n.var*(n.sim + n.burn))
> rho.AL2.names = c("k1","alpha","k2","xi","mu","gamma")
> ird.AL2.aux = euler.pcode.aux(
+           N = n.var, M = n.var,
+           t.per.sim = t.per.sim, ndt = ndt,
+           X0 = c(6,-0.3), z = z.euler,
+           returnc = c(T,T),
+           rho.names = rho.AL2.names,
+           drift.expr = expression(k1*(mu - X[1]),
+                                   k2*(alpha-X[2])),
+           diffuse.expr = expression((exp(X[2]/2)*X[1]^gamma,
+                                       0.0, 0.0, xi))
```

The model (20.17) has two random factors and the dimension of $\mathbf{W}_t$ is the same as $\mathbf{X}_t$, so `N=2` and `M=2`. Since the annualized rates are assumed to be observed weekly, `t.per.sim=1/52`. The number of discretization points, `ndt`, for each $\mathbf{X}_t$ is 25. The initial value for $\mathbf{X}_t$ is set using `X0=c(6,-0.3)`. Setting `returnc=c(T,T)` returns simulated solutions for both r_t and v_t. The drift and diffusion functions (20.19) are specified in the components `drift.expr` and `diffuse.expr`. The parameters used in the expressions are specified in the component `rho.names`, and the two state variables in $\mathbf{X}_t$ are accessed as `X[1]` and `X[2]`, respectively. The drift and diffusion functions evaluated at $t = 0$, $\mathbf{X}_0 = (6, -0.3)$ and the parameters estimated by Andersen and Lund

```
> rho.AL2 = c(0.163,-0.282,1.04,1.27,5.95,0.544)
```

are

```
> euler.pcode.test(rho.AL2, N = 2, M = 2, t = 0,
+                  X = c(6, -0.3), aux = ird.AL2.aux)
$drift:
[1] -0.00815  0.01872

$diffuse:
         [,1] [,2]
[1,] 2.281235 0.00
[2,] 0.000000 1.27
```

To produce 2000 simulated values for r_t and v_t from (20.17) using Euler's method, with 100 burn-in values, based on the parameters estimated by Andersen and Lund, use the following commands:

```
> ird.AL2.sim = euler.pcode.gensim(rho = rho.AL2,
+               n.sim = n.sim, n.var = n.var, n.burn = n.burn,
+               aux = ird.AL2.aux)
> class(ird.AL2.sim)
[1] "numeric"
> length(ird.AL2.sim)
[1] 4000
```

The simulation is stored in the object `ird.AL2.sim` as a single vector of length 4000 (= `n.var*n.sim`), with the first two entries corresponding to $\{r_{t_0}, v_{t_0}\}$, followed by $\{r_{t_1}, v_{t_1}\}$, and so on. The simulated solutions for r_t and v_t may be placed in a 2000×2 matrix using

```
> ird.AL2.sim = matrix(ird.AL2.sim, n.sim, n.var, byrow=T)
```

Figure 20.9 shows the solutions.

Simulations from the strong order 1 scheme require much more overhead than simulations from Euler's method. In this example, if p in (20.7) is set according to the rule

```
> p.strong = ceiling(0.05/(t.per.sim/ndt))
> p.strong
[1] 65
```

then the number of random normals to compute for one simulated path is

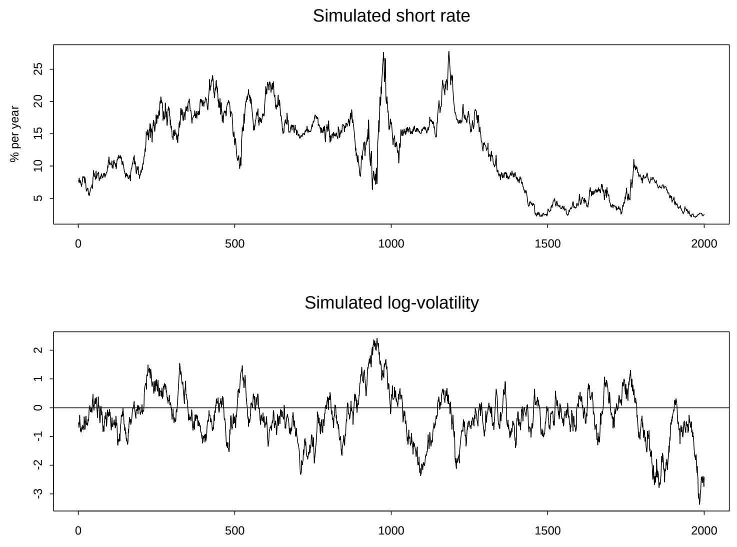


FIGURE 20.9. Simulated solutions from (20.17) based on Euler's method.

```
> 2*ndt*n.var*(p.strong + 1)*(n.sim + n.burn)
[1] 13860000
```

Euler's method only requires

```
> ndt*n.var*(n.sim + n.burn)
[1] 105000
```

values. To produce 2000 simulated values for r_t and v_t from (20.17) using the strong order 1 scheme, with 100 burn-in values, use the following commands:

```
> set.seed(0)
> z.strong = rnorm(2*ndt*n.var*(p.strong + 1)*(n.sim + n.burn))
> ird.AL2.aux = strong1.pcode.aux(
+          N = n.var, M = n.var,
+          t.per.sim = t.per.sim, ndt = ndt,
+          p = p.strong, z = z.strong,
+          X0 = c(6,-0.3),
+          returnc = c(T,T),
+          rho.names = rho.AL2.names,
+          drift.expr = expression(
+                       k1*(mu - X[1]),
+                       k2*(alpha - X[2])),
+          diffuse.expr = expression(
+                           (exp(X[2]/2))*X[1]^gamma,
+                           0.0,
+                           0.0,
+                           xi))
> ird.AL2.simS = strong1.pcode.gensim(rho = rho.AL2,
+              n.sim = n.sim, n.var = n.var, n.burn = n.burn,
+              aux = ird.AL2.aux)
> ird.AL2.simS = matrix(ird.AL2.simS, n.sim, n.var, byrow = T)
```

The overhead for the weak order 2 scheme is in between the overhead for Euler's method and the strong order 1 scheme. The weak order 2 scheme requires

```
> ndt*n.var*(n.sim + n.burn)
[1] 105000
```

random normal variables and

```
> (n.burn+n.sim)*ndt*n.var*(n.var-1)/2
[1] 52500
```

random uniform variables. To produce 2000 simulated values for r_t and v_t from (20.17) using the weak order 2 scheme, with 100 burn-in values, use the following commands:

```
> set.seed(0)
> z.weak = rnorm(ndt*n.var*(n.sim + n.burn))
> u.weak = runif((n.burn+n.sim)*ndt*n.var*(n.var-1)/2)
> ird.AL2.aux = weak2.pcode.aux(
+           N = n.var, M = n.var,
+           t.per.sim = t.per.sim, ndt = ndt,
+           z = z.weak, u = u.weak,
+           X0 = c(6,-0.3),
+           returnc=c(T,T),
+           rho.names = rho.AL2.names,
+           drift.expr = expression(
+                          k1*(mu - X[1]),
+                          k2*(alpha-X[2])),
+           diffuse.expr = expression(
+                           (exp(X[2]/2))*X[1]^gamma,
+                           0.0,
+                           0.0,
+                            xi))
> ird.AL2.simW = weak2.pcode.gensim(rho = rho.AL2,
+              n.sim = n.sim, n.var = n.var, n.burn = n.burn,
+               aux = ird.AL2.aux)
> ird.AL2.simW = matrix(ird.AL2.simW, n.sim, n.var, byrow=T)
```

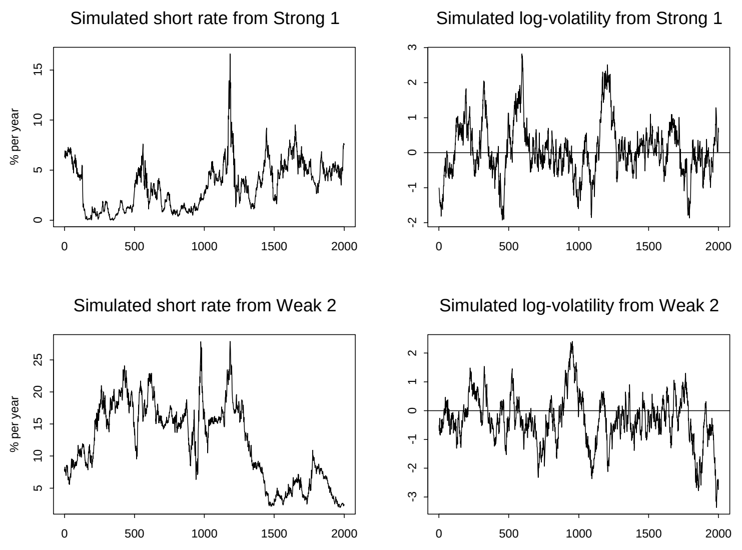


FIGURE 20.10. Simulated solutions from (23.32) based on the strong order 1 scheme and the weak order 2 scheme.

Figure 20.10 shows the simulated solution from the strong order 1 and weak order 2 schemes. Notice that the paths from Euler's method and the weak order 2 scheme are almost identical, whereas the strong order 1 path is quite different.

20.5 References

Andersen, T.G. and J. Lund (1997). "Estimating Continuous-Time Stochastic Volatility Models of the Short-Term Interest Rate," *Journal of Econometrics*, 77, 343-377.

Baxter, M.W. and A.J.O. Rennie (1996) *Financial Calculus: An Introduction to Derivative Pricing.* Cambridge University Press, Cambridge.

Black, F. and M. Scholes (1973). "The Pricing of Options and Corporate Liabilities," *Journal of Political Economy*, 81, 637-654.

Chan, K.C., G.A. Karolyi, F.A. Longstaff and A.B. Sanders (1992). "An Empirical Comparison of Alternative Models of the Term Structure of Interest Rates," *Journal of Finance*, 47, 1209-1227.

Cox, J.C., J.E. Ingersoll and S.A. Ross (1985). "A Theory of the Term Structure of Interest Rates," *Econometrica*, 53(2), 385–407.

Duffie, D. (1996). *Dynamic Asset Pricing*, 2nd ed., Princeton University Press, Princeton, NJ.

Gallant, A.R. (2003). Original FORTRAN routines `stng1.f` and `weak2.f`. Downloaded from `ftp.econ.duke.edu`, directory `pub/arg/libf`, . Copyright (C) 1995. A. Ronald Gallant, P.O. Box 659, Chapel Hill NC 27514-0659, USA. Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation.

Gallant, A.R. and G. Tauchen (2002). "EMM: A Program for Efficient Method of Moments Estimation, Version 1.6, User's Guide," Working paper, University of North Carolina at Chapel Hill. Current revision available at `www.unc.edu/~arg`.

Kloeden, P.E. and E. Platen (1999). *Numerical Solution of Stochastic Differential Equations,* 3rd ed. Volume 23 of Applications of Mathematics, Stochastic Modelling and Applied Probability, Springer-Verlag, New York.

Merton, R. (1990). *Continuous Time Finance.* Blackwell, Cambridge.

Neftci, S.N. (1996). *An Introduction to the Mathematics of Financial Derivatives.* Academic Press, San Diego.

Øksendal, B.K. (1998). *Stochastic Differential Equations. An Introduction with Applications*, 5th ed. Universitext. Springer-Verlag, New York.

Seydel, R. (2002). *Tools for Computational Finance.* Springer-Verlag, Berlin.

Vasicek, O.A. (1977). "An Equilibrium Characterization of the Term Strucure," *Journal of Financial Economics*, 5, 177–188.

21
Generalized Method of Moments

21.1 Introduction

This chapter describes generalized method of moments (GMM) estimation for linear and nonlinear models with applications in economics and finance. The GMM estimation was formalized by Hansen (1982), and since has become one of the most widely used methods of estimation for models in economics and finance. Unlike the maximum likelihood estimation (MLE), GMM does not require complete knowledge of the distribution of the data. Only specified moments derived from an underlying model are needed for the GMM estimation. In some cases in which the distribution of the data is known, MLE can be computationally very burdensome, whereas GMM can be computationally very easy. The log-normal stochastic volatility model is one example. In models for which there are more moment conditions than model parameters, the GMM estimation provides a straightforward way to test the specification of the proposed model. This is an important feature that is unique to the GMM estimation.

This chapter is organized as follows. The GMM estimation for linear models is described in Section 21.2. Section 21.3 describes methods for estimating the efficient weight matrix. Sections 21.4 and 21.5 give examples of estimation and inference using the `S+Finmetrics` function `GMM`. Section 21.6 describes the GMM estimation and inference for nonlinear models. Section 21.7 provides numerous examples of the GMM estimation of nonlinear models in finance, including Euler equation asset pricing models, discrete-time stochastic volatility models, and continous-time interest rate diffusion models.

The theory and notation for GMM presented herein follows the excellent treatment given in Hayashi (2000). Other good textbook treatments of GMM at an intermediate level are given in Hamilton (1994), Ruud (2000), Davidson and MacKinnon (2004), and Greene (2004). The most comprehensive textbook treatment of GMM is Hall (2005). Excellent surveys of recent developments in GMM are given in the special issues of the *Journal of Business and Economic Statistics* (1996, 2002). Discussions of GMM applied to problems in finance are given in Ogaki (1992), Ferson (1995), Andersen and Sorensen (1996), Campbell, Lo, and MacKinlay (1997), James and Webber (2000), Cochrane (2001), Jagannathan and Skoulakis (2002), and Hall (2005).

21.2 Single Equation Linear GMM

Consider the linear regression model

$$y_t = \mathbf{z}_t'\boldsymbol{\delta}_0 + \varepsilon_t,\ t = 1, \ldots, n \tag{21.1}$$

where $\mathbf{z}_t$ is an $L \times 1$ vector of explanatory variables, $\boldsymbol{\delta}_0$ is a vector of unknown coefficients, and ε_t is a random error term. The model (21.1) allows for the possibility that some or all of the elements of z_t may be correlated with the error term ε_t (i.e., $E[z_{tk}\varepsilon_t] \neq 0$ for some k). If $E[z_{tk}\varepsilon_i] \neq 0$, then z_{tk} is called an *endogenous variable*. It is well known that if $\mathbf{z}_t$ contains endogenous variables, then the least squares estimator of $\boldsymbol{\delta}_0$ in (21.1) is biased and inconsistent.

Associated with the model (21.1), it is assumed that there exists a $K \times 1$ vector of *instrumental variables* $\mathbf{x}_t$ that may contain some or all of the elements of $\mathbf{z}_t$. Let $\mathbf{w}_t$ represent the vector of unique and nonconstant elements of $\{y_t, \mathbf{z}_t, \mathbf{x}_t\}$. It is assumed that $\{\mathbf{w}_t\}$ is a stationary and ergodic stochastic process.

The instrumental variables $\mathbf{x}_t$ satisfy the set of K orthogonality conditions

$$E[\mathbf{g}_t(\mathbf{w}_t, \boldsymbol{\delta}_0)] = E[\mathbf{x}_t\varepsilon_t] = E[\mathbf{x}_t(y_t - \mathbf{z}_t'\boldsymbol{\delta}_0)] = \mathbf{0} \tag{21.2}$$

where $\mathbf{g}_t(\mathbf{w}_t, \boldsymbol{\delta}_0) = \mathbf{x}_t\varepsilon_t = \mathbf{x}_t(y_t - \mathbf{z}_t'\boldsymbol{\delta}_0)$. Expanding (21.2) gives the relation

$$\boldsymbol{\Sigma}_{xy} = \boldsymbol{\Sigma}_{xz}\boldsymbol{\delta}_0$$

where $\boldsymbol{\Sigma}_{xy} = E[\mathbf{x}_t y_t]$ and $\boldsymbol{\Sigma}_{xz} = E[\mathbf{x}_t\mathbf{z}_t']$. For identification of $\boldsymbol{\delta}_0$, it is required that the $K \times L$ matrix $E[\mathbf{x}_t\mathbf{z}_t'] = \boldsymbol{\Sigma}_{xz}$ be of full rank L. This *rank condition* ensures that $\boldsymbol{\delta}_0$ is the unique solution to (21.2). Note that if $K = L$, then $\boldsymbol{\Sigma}_{xz}$ is invertible and $\boldsymbol{\delta}_0$ may be determined using

$$\boldsymbol{\delta}_0 = \boldsymbol{\Sigma}_{xz}^{-1}\boldsymbol{\Sigma}_{xy}$$

A necessary condition for the identification of $\boldsymbol{\delta}_0$ is the *order condition*

$$K \geq L \tag{21.3}$$

which simply states that the number of instrumental variables must be greater than or equal to the number of explanatory variables in (21.1). If $K = L$, then $\boldsymbol{\delta}_0$ is said to be (apparently) just identified; if $K > L$ then $\boldsymbol{\delta}_0$ is said to be (apparently) overidentified; if $K < L$ then $\boldsymbol{\delta}_0$ is not identified. The word "apparently" in parentheses is used to remind the reader that the rank condition

$$\text{rank}(\boldsymbol{\Sigma}_{xz}) = L \tag{21.4}$$

must also be satisfied for identification.

In the regression model (21.1), the error terms are allowed to be conditionally heteroskedastic as well as serially correlated. For the case in which ε_t is conditionally heteroskedastic, it is assumed that $\{\mathbf{g}_t\} = \{\mathbf{x}_t\varepsilon_t\}$ is a stationary and ergodic martingale difference sequence (MDS) satisfying

$$E[\mathbf{g}_t\mathbf{g}_t'] = E[\mathbf{x}_t\mathbf{x}_t'\varepsilon_t^2] = \mathbf{S}$$

where $\mathbf{S}$ is a nonsingular $K \times K$ matrix. The matrix $\mathbf{S}$ is the asymptotic variance-covariance matrix of the sample moments $\bar{\mathbf{g}} = n^{-1}\sum_{t=1}^n \mathbf{g}_t(\mathbf{w}_t, \boldsymbol{\delta}_0)$. This follows from the central limit theorem for ergodic stationary martingale difference sequences (see Hayashi, 2000, p. 106)

$$\sqrt{n}\bar{\mathbf{g}} = \frac{1}{\sqrt{n}}\sum_{t=1}^n \mathbf{x}_t\varepsilon_t \xrightarrow{d} N(\mathbf{0}, \mathbf{S})$$

where $\mathbf{S} = \text{avar}(\bar{\mathbf{g}})$ denotes the variance-covariance matrix of the limiting distribution of $\sqrt{n}\bar{\mathbf{g}}$.

For the case in which ε_t is serially correlated and possibly conditionally heteroskedastic as well, it is assumed that $\{\mathbf{g}_t\} = \{\mathbf{x}_t\varepsilon_t\}$ is a stationary and ergodic stochastic process that satisfies

$$\begin{aligned}
\sqrt{n}\bar{\mathbf{g}} &= \frac{1}{\sqrt{n}}\sum_{t=1}^n \mathbf{x}_t\varepsilon_t \xrightarrow{d} N(\mathbf{0}, \mathbf{S}) \\
\mathbf{S} &= \sum_{j=-\infty}^{\infty} \boldsymbol{\Gamma}_j = \boldsymbol{\Gamma}_0 + \sum_{j=1}^{\infty}(\boldsymbol{\Gamma}_j + \boldsymbol{\Gamma}_j')
\end{aligned}$$

where $\boldsymbol{\Gamma}_j = E[\mathbf{g}_t\mathbf{g}_{t-j}'] = E[\mathbf{x}_t\mathbf{x}_{t-j}'\varepsilon_t\varepsilon_{t-j}]$. In the above, $\text{avar}(\bar{\mathbf{g}}) = \mathbf{S}$ is also referred to as the *long-run variance* of $\bar{\mathbf{g}}$.

21.2.1 Definition of the GMM Estimator

The GMM estimator of $\boldsymbol{\delta}_0$ in (21.1) is constructed by exploiting the orthogonality conditions (21.2). The idea is to create a set of estimating equations for $\boldsymbol{\delta}_0$ by making sample moments match the population moments defined by (21.2). The sample moments based on (21.2) for an arbitrary value $\boldsymbol{\delta}$ are

$$\begin{aligned}\mathbf{g}_n(\boldsymbol{\delta}) &= \frac{1}{n}\sum_{t=1}^{n} g(\mathbf{w}_t, \boldsymbol{\delta}) = \frac{1}{n}\sum_{t=1}^{n} \mathbf{x}_t(y_t - \mathbf{z}_t'\boldsymbol{\delta}) \\ &= \begin{pmatrix} \frac{1}{n}\sum_{t=1}^{n} x_{1t}(y_t - \mathbf{z}_t'\boldsymbol{\delta}) \\ \vdots \\ \frac{1}{n}\sum_{t=1}^{n} x_{Kt}(y_t - \mathbf{z}_t'\boldsymbol{\delta}) \end{pmatrix}\end{aligned}$$

These moment conditions are a set of K linear equations in L unknowns. Equating these sample moments to the population moment $E[\mathbf{x}_t\varepsilon_t] = \mathbf{0}$ gives the estimating equations

$$\mathbf{S}_{xy} - \mathbf{S}_{xz}\boldsymbol{\delta} = \mathbf{0} \tag{21.5}$$

where $\mathbf{S}_{xy} = n^{-1}\sum_{t=1}^{n} \mathbf{x}_t y_t$ and $\mathbf{S}_{xz} = n^{-1}\sum_{t=1}^{n} \mathbf{x}_t\mathbf{z}_t'$ are the sample moments.

If $K = L$ ($\boldsymbol{\delta}_0$ is just identified) and $\mathbf{S}_{xz}$ is invertible, then the GMM estimator of $\boldsymbol{\delta}_0$ is

$$\hat{\boldsymbol{\delta}} = \mathbf{S}_{xz}^{-1}\mathbf{S}_{xy}$$

which is also known as the *indirect least squares* estimator. If $K > L$, then there may not be a solution to the estimating equations (21.5). In this case, the idea is to try to find a $\boldsymbol{\delta}$ that makes $\mathbf{S}_{xy} - \mathbf{S}_{xz}\boldsymbol{\delta}$ as close to zero as possible. To do this, let $\hat{\mathbf{W}}$ denote a $K \times K$ symmetric and positive definite (p.d.) weight matrix, possibly dependent on the data, such that $\hat{\mathbf{W}} \xrightarrow{p} \mathbf{W}$ as $n \to \infty$ with $\mathbf{W}$ symmetric and p.d. Then, the GMM estimator of $\boldsymbol{\delta}_0$, denoted $\hat{\boldsymbol{\delta}}(\hat{\mathbf{W}})$, is defined as

$$\hat{\boldsymbol{\delta}}(\hat{\mathbf{W}}) = \arg\min_{\boldsymbol{\delta}} \; J(\boldsymbol{\delta}, \hat{\mathbf{W}})$$

where

$$\begin{aligned} J(\boldsymbol{\delta}, \hat{\mathbf{W}}) &= n\mathbf{g}_n(\boldsymbol{\delta})'\hat{\mathbf{W}}\mathbf{g}_n(\boldsymbol{\delta}) \qquad (21.6) \\ &= n(\mathbf{S}_{xy} - \mathbf{S}_{xz}\boldsymbol{\delta})'\hat{\mathbf{W}}(\mathbf{S}_{xy} - \mathbf{S}_{xz}\boldsymbol{\delta}) \end{aligned}$$

Since $J(\boldsymbol{\delta}, \hat{\mathbf{W}})$ is a simple quadratic form in $\boldsymbol{\delta}$, straightforward calculus may be used to determine the analytic solution for $\hat{\boldsymbol{\delta}}(\hat{\mathbf{W}})$:

$$\hat{\boldsymbol{\delta}}(\hat{\mathbf{W}}) = (\mathbf{S}_{xz}'\hat{\mathbf{W}}\mathbf{S}_{xz})^{-1}\mathbf{S}_{xz}'\hat{\mathbf{W}}\mathbf{S}_{xy} \tag{21.7}$$

Asymptotic Properties

Under standard regularity conditions (see Hayashi, 2000, Chap. 3), it can be shown that

$$\begin{gathered}\hat{\boldsymbol{\delta}}(\hat{\mathbf{W}}) \xrightarrow{p} \boldsymbol{\delta}_0 \\ \sqrt{n}\left(\hat{\boldsymbol{\delta}}(\hat{\mathbf{W}}) - \boldsymbol{\delta}_0\right) \xrightarrow{d} N(\mathbf{0}, \mathrm{avar}(\hat{\boldsymbol{\delta}}(\hat{\mathbf{W}})))\end{gathered}$$

where

$$\text{avar}(\hat{\boldsymbol{\delta}}(\hat{\mathbf{W}})) = (\boldsymbol{\Sigma}_{xz}'\mathbf{W}\boldsymbol{\Sigma}_{xz})^{-1}\boldsymbol{\Sigma}_{xz}'\mathbf{W}\mathbf{S}\mathbf{W}\boldsymbol{\Sigma}_{xz}(\boldsymbol{\Sigma}_{xz}'\mathbf{W}\boldsymbol{\Sigma}_{xz})^{-1} \quad (21.8)$$

A consistent estimate of $\text{avar}(\hat{\boldsymbol{\delta}}(\hat{\mathbf{W}}))$, denoted $\widehat{\text{avar}}(\hat{\boldsymbol{\delta}}(\hat{\mathbf{W}}))$, may be computed using

$$\widehat{\text{avar}}(\hat{\boldsymbol{\delta}}(\hat{\mathbf{W}})) = (\mathbf{S}_{xz}'\hat{\mathbf{W}}\mathbf{S}_{xz})^{-1}\mathbf{S}_{xz}'\hat{\mathbf{W}}\hat{\mathbf{S}}\hat{\mathbf{W}}\mathbf{S}_{xz}(\mathbf{S}_{xz}'\hat{\mathbf{W}}\mathbf{S}_{xz})^{-1} \quad (21.9)$$

where $\hat{\mathbf{S}}$ is a consistent estimate for $\mathbf{S} = \text{avar}(\bar{\mathbf{g}})$.

The Efficient GMM Estimator

For a given set of instruments $\mathbf{x}_t$, the GMM estimator $\hat{\boldsymbol{\delta}}(\hat{\mathbf{W}})$ is defined for an arbitrary positive definite and symmetric weight matrix $\hat{\mathbf{W}}$. The asymptotic variance of $\hat{\boldsymbol{\delta}}(\hat{\mathbf{W}})$ in (21.8) depends on the chosen weight matrix $\hat{\mathbf{W}}$. A natural question to ask is: What weight matrix $\mathbf{W}$ produces the smallest value of $\text{avar}(\hat{\boldsymbol{\delta}}(\hat{\mathbf{W}}))$? The GMM estimator constructed with this weight matrix is called the *efficient GMM estimator*. Hansen (1982) showed that efficient the GMM estimator results from setting $\hat{\mathbf{W}} = \hat{\mathbf{S}}^{-1}$ such that $\hat{\mathbf{S}} \xrightarrow{p} \mathbf{S}$. For this choice of $\hat{\mathbf{W}}$, the asymptotic variance formula (21.8) reduces to

$$\text{avar}(\hat{\boldsymbol{\delta}}(\hat{\mathbf{S}}^{-1})) = (\boldsymbol{\Sigma}_{xz}'\mathbf{S}^{-1}\boldsymbol{\Sigma}_{xz})^{-1} \quad (21.10)$$

of which a consistent estimate is

$$\widehat{\text{avar}}(\hat{\boldsymbol{\delta}}(\hat{\mathbf{S}}^{-1})) = (\mathbf{S}_{xz}'\hat{\mathbf{S}}^{-1}\mathbf{S}_{xz})^{-1} \quad (21.11)$$

The efficient GMM estimator is defined as

$$\hat{\boldsymbol{\delta}}(\hat{\mathbf{S}}^{-1}) = \arg\min_{\boldsymbol{\delta}} \; n\mathbf{g}_n(\boldsymbol{\delta})'\hat{\mathbf{S}}^{-1}\mathbf{g}_n(\boldsymbol{\delta})$$

which requires a consistent estimate of $\mathbf{S}$. However, a consistent estimation of $\mathbf{S}$, in turn, requires a consistent estimate of $\boldsymbol{\delta}_0$. To see this, consider the case in which ε_t in (21.1) is conditionally heteroskedastic so that $\mathbf{S} = E[\mathbf{g}_t\mathbf{g}_t'] = E[\mathbf{x}_t\mathbf{x}_t'\varepsilon_t^2]$. A consistent estimate of $\mathbf{S}$ has the form

$$\hat{\mathbf{S}} = \frac{1}{n}\sum_{t=1}^{n}\mathbf{x}_t\mathbf{x}_t'\hat{\varepsilon}_t^2 = \frac{1}{n}\sum_{t=1}^{n}\mathbf{x}_t\mathbf{x}_t'(y_t - \mathbf{z}_t'\hat{\boldsymbol{\delta}})^2$$

such that $\hat{\boldsymbol{\delta}} \xrightarrow{p} \boldsymbol{\delta}_0$. Similar arguments hold for the case in which $\mathbf{g}_t = \mathbf{x}_t\varepsilon_t$ is a serially correlated and heteroskedastic process.

Two-Step Efficient GMM

The two-step efficient GMM estimator utilizes the result that a consistent estimate of $\boldsymbol{\delta}_0$ may be computed by GMM with an arbitrary positive definite and symmetric weight matrix $\hat{\mathbf{W}}$ such that $\hat{\mathbf{W}} \xrightarrow{p} \mathbf{W}$. Let

$\hat{\boldsymbol{\delta}}(\hat{\mathbf{W}})$ denote such an estimate. Common choices for $\hat{\mathbf{W}}$ are $\hat{\mathbf{W}} = \mathbf{I}_k$ and $\hat{\mathbf{W}} = \mathbf{S}_{xx}^{-1} = (n^{-1}\mathbf{X}'\mathbf{X})^{-1}$, where $\mathbf{X}$ is an $n \times k$ matrix with the tth row equal to $\mathbf{x}_t'$[1]. Then, a first step consistent estimate of $\mathbf{S}$ is given by

$$\hat{\mathbf{S}}(\hat{\mathbf{W}}) = \frac{1}{n}\sum_{t=1}^{n}\mathbf{x}_t\mathbf{x}_t'(y_t - \mathbf{z}_t'\hat{\boldsymbol{\delta}}(\hat{\mathbf{W}}))^2 \tag{21.12}$$

The *two-step efficient GMM estimator* is then defined as

$$\hat{\boldsymbol{\delta}}(\hat{\mathbf{S}}^{-1}(\hat{\mathbf{W}})) = \arg\min_{\boldsymbol{\delta}}\ n\mathbf{g}_n(\boldsymbol{\delta})'\hat{\mathbf{S}}^{-1}(\hat{\mathbf{W}})\mathbf{g}_n(\boldsymbol{\delta}) \tag{21.13}$$

Iterated Efficient GMM

The *iterated efficient GMM estimator* uses the two-step efficient GMM estimator $\hat{\boldsymbol{\delta}}(\hat{\mathbf{S}}^{-1}(\hat{\mathbf{W}}))$ to update the estimation of $\mathbf{S}$ in (21.12) and then recomputes the estimator in (21.13). The process is repeated (iterated) until the estimates of $\boldsymbol{\delta}_0$ do not change significantly from one iteration to the next. Typically, only a few iterations are required. The resulting estimator is denoted $\hat{\boldsymbol{\delta}}(\hat{\mathbf{S}}_{\text{iter}}^{-1})$. The iterated efficient GMM estimator has the same asymptotic distribution as the two-step efficient estimator. However, in finite samples, the two estimators may differ. As Hamilton (1994, p. 413) pointed out, the iterated GMM estimator has a practical advantage over the two-step estimator in that the resulting estimates are invariant with respect to the scale of the data and to the initial weighting matrix $\hat{\mathbf{W}}$.

Continuous Updating Efficient GMM

This estimator simultaneously estimates $\mathbf{S}$, as a function of $\boldsymbol{\delta}$, and $\boldsymbol{\delta}$. It is defined as

$$\hat{\boldsymbol{\delta}}(\hat{\mathbf{S}}_{\text{CU}}^{-1}) = \arg\min_{\boldsymbol{\delta}}\ n\mathbf{g}_n(\boldsymbol{\delta})'\hat{\mathbf{S}}^{-1}(\boldsymbol{\delta})\mathbf{g}_n(\boldsymbol{\delta}) \tag{21.14}$$

where the expression for $\hat{\mathbf{S}}(\boldsymbol{\delta})$ depends on the estimator used for $\mathbf{S}$. For example, with conditionally heteroskedastic errors, $\hat{\mathbf{S}}(\delta)$ takes the form

$$\hat{\mathbf{S}}(\boldsymbol{\delta}) = \frac{1}{n}\sum_{t=1}^{n}\mathbf{x}_t\mathbf{x}_t'(y_t - \mathbf{z}_t'\boldsymbol{\delta})^2$$

Hansen, Heaton, and Yaron (1996) call $\hat{\boldsymbol{\delta}}(\hat{\mathbf{S}}_{\text{CU}}^{-1})$ the *continuous updating* (CU) *efficient GMM estimator*. This estimator is asymptotically equivalent to the two-step and iterated estimators, but may differ in finite samples.

[1]In the function `GMM`, the default initial weight matrix is the identity matrix. This can be changed by supplying a weight matrix using the optional argument `w=my.weight.matrix`. Using $\hat{\mathbf{W}} = \mathbf{S}_{xx}^{-1}$ is often more numerically stable than using $\hat{\mathbf{W}} = \mathbf{I}_k$.

The CU efficient GMM estimator does not depend on an initial weight matrix $\mathbf{W}$, and like the iterated efficient GMM estimator, the numerical value of CU estimator is invariant to the scale of the data. It is computationally more burdensome than the iterated estimator, especially for large nonlinear models, and is more prone to numerical instability. However, Hansen, Heaton, and Yaron found that the finite sample performance of the CU estimator, and test statistics based on it, is often superior to the other estimators. The good finite sample performance of the CU estimator relative to the iterated GMM estimator may be explained by the connection between the CU estimator and empirical likelihood estimators. See Imbens (2002) and Newey and Smith (2004) for a further discussion on the relationship between GMM estimators and empirical likelihood estimators.

21.2.2 Specification Tests in Overidentified Models

An advantage of the GMM estimation in overidentified models is the ability to test the specification of the model. The following subsections summarize the common statistics used for evaluating the basic model specification.

The J-Statistic

The *J-statistic*, introduced in Hansen (1982), refers to the value of the GMM objective function evaluated using an efficient GMM estimator:

$$J = J(\hat{\boldsymbol{\delta}}(\hat{\mathbf{S}}^{-1}), \hat{\mathbf{S}}^{-1}) = n\mathbf{g}_n(\hat{\boldsymbol{\delta}}(\hat{\mathbf{S}}^{-1}))'\hat{\mathbf{S}}^{-1}\mathbf{g}_n(\hat{\boldsymbol{\delta}}\hat{\mathbf{S}}^{-1}) \tag{21.15}$$

where $\hat{\boldsymbol{\delta}}(\hat{\mathbf{S}}^{-1})$ denotes any efficient GMM estimator of $\boldsymbol{\delta}_0$ and $\hat{\mathbf{S}}$ is a consistent estimate of $\mathbf{S}$. If $K = L$, then $J = 0$, and if $K > L$, then $J > 0$. Under regularity conditions (see Hayashi, 2000, Chap. 3) and if the moment conditions (21.2) are valid, then as $n \to \infty$

$$J \xrightarrow{d} \chi^2(K - L)$$

Hence, in a well-specified overidentified model with valid moment conditions the J-statistic behaves like a chi-square random variable with degrees of freedom equal to the number of overidentifying restrictions. If the model is misspecified and/or some of the moment conditions (21.2) do not hold (e.g., $E[x_{it}\varepsilon_t] = E[x_{it}(y_t - \mathbf{z}_t'\boldsymbol{\delta}_0)] \neq 0$ for some i), then the J-statistic will be large relative to a chi-square random variable with $K - L$ degrees of freedom.

The J-statistic acts as an omnibus test statistic for model misspecification. A large J-statistic indicates a misspecified model. Unfortunately, the J-statistic does not, by itself, give any information about how the model is misspecified.

Normalized Moments

If the model is rejected by the J-statistic, it is of interest to know why the model is rejected. To aid in the diagnosis of model failure, the magnitudes of the individual elements of the normalized moments $\sqrt{n}\mathbf{g}_n(\hat{\boldsymbol{\delta}}(\hat{\mathbf{S}}^{-1}))$ may point the reason why the model is rejected by the J-statistic. Under the null hypothesis that the model is correct and the orthogonality conditions are valid, the *normalized moments* satisfy

$$\sqrt{n}\mathbf{g}_n(\hat{\boldsymbol{\delta}}(\hat{\mathbf{S}}^{-1})) \xrightarrow{d} N(\mathbf{0}, \mathbf{S} - \boldsymbol{\Sigma}_{xz}[\boldsymbol{\Sigma}_{xz}'\mathbf{S}^{-1}\boldsymbol{\Sigma}_{xz}]^{-1}\boldsymbol{\Sigma}_{xz}')$$

As a result, for a well specified model, the individual moment t-ratios

$$t_i = \mathbf{g}_n(\hat{\boldsymbol{\delta}}(\hat{\mathbf{S}}^{-1}))_i / \mathrm{SE}(\mathbf{g}_n(\hat{\boldsymbol{\delta}}(\hat{\mathbf{S}}^{-1}))_i),\ i = 1, \ldots, K \tag{21.16}$$

where

$$\mathrm{SE}(\mathbf{g}_n(\hat{\boldsymbol{\delta}}(\hat{\mathbf{S}}^{-1}))_i = \left(\left[\hat{\mathbf{S}} - \hat{\boldsymbol{\Sigma}}_{xz}[\hat{\boldsymbol{\Sigma}}_{xz}'\hat{\mathbf{S}}^{-1}\hat{\boldsymbol{\Sigma}}_{xz}]^{-1}\hat{\boldsymbol{\Sigma}}_{xz}'\right]/T\right)_{ii}^{1/2}$$

are asymptotically standard normal. When the model is rejected using the J-statistic, a large value of t_i indicates misspecification with respect to the ith moment condition. Since the rank of $\mathbf{S} - \boldsymbol{\Sigma}_{xz}[\boldsymbol{\Sigma}_{xz}'\mathbf{S}^{-1}\boldsymbol{\Sigma}_{xz}]^{-1}\boldsymbol{\Sigma}_{xz}'$ is $K - L$, the interpretation of the moment t-ratios (21.16) may be difficult in models for which the degree of overidentification is small. In particular, if $K - L = 1$, then $t_1 = \cdots = t_K$.

21.2.3 Two-Stage Least Squares as an Efficient GMM Estimator

If, in the linear GMM regression model (21.1), the errors are conditionally homoskedastic, then

$$E[\mathbf{x}_t\mathbf{x}_t'\varepsilon_t^2] = \sigma^2\boldsymbol{\Sigma}_{xx} = \mathbf{S}$$

A consistent estimate of $\mathbf{S}$ has the form $\hat{\mathbf{S}} = \hat{\sigma}^2\mathbf{S}_{xx}$, where $\hat{\sigma}^2 \xrightarrow{p} \sigma^2$. Typically,

$$\hat{\sigma}^2 = n^{-1}\sum_{t=1}^{n}(y_t - \mathbf{z}_t'\hat{\boldsymbol{\delta}})^2$$

where $\hat{\boldsymbol{\delta}} \rightarrow \boldsymbol{\delta}_0$. The efficient GMM estimator becomes

$$\begin{aligned}\hat{\boldsymbol{\delta}}(\hat{\sigma}^{-2}\mathbf{S}_{xx}^{-1}) &= (\mathbf{S}_{xz}'\hat{\sigma}^{-2}\mathbf{S}_{xx}^{-1}\mathbf{S}_{xz})^{-1}\mathbf{S}_{xz}'\hat{\sigma}^{-2}\mathbf{S}_{xx}^{-1}\mathbf{S}_{xy} \\ &= (\mathbf{S}_{xz}'\mathbf{S}_{xx}^{-1}\mathbf{S}_{xz})^{-1}\mathbf{S}_{xz}'\mathbf{S}_{xx}^{-1}\mathbf{S}_{xy} \\ &= \hat{\boldsymbol{\delta}}(\mathbf{S}_{xx}^{-1})\end{aligned}$$

which does not depend on $\hat{\sigma}^2$. The estimator $\hat{\boldsymbol{\delta}}(\mathbf{S}_{xx}^{-1})$ is, in fact, identical to the *two-stage least squares* (TSLS) estimator of $\boldsymbol{\delta}_0$:

$$\begin{aligned}\hat{\boldsymbol{\delta}}(\mathbf{S}_{xx}^{-1}) &= (\mathbf{S}_{xz}'\mathbf{S}_{xx}^{-1}\mathbf{S}_{xz})^{-1}\mathbf{S}_{xz}'\mathbf{S}_{xx}^{-1}\mathbf{S}_{xy} \\ &= (\mathbf{Z}'\mathbf{P}_X\mathbf{Z})^{-1}\mathbf{Z}'\mathbf{P}_X\mathbf{y} \\ &= \hat{\boldsymbol{\delta}}_{\mathrm{TSLS}}\end{aligned}$$

where $\mathbf{Z}$ denotes the $n \times L$ matrix of observations with tth row $\mathbf{z}_t'$, $\mathbf{X}$ denotes the $n \times k$ matrix of observations with the tth row $\mathbf{x}_t'$, and $\mathbf{P}_X = \mathbf{X}(\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'$ is the idempotent matrix that projects onto the columns of $\mathbf{X}$.

Using (21.10), the asymptotic variance of $\hat{\boldsymbol{\delta}}(\mathbf{S}_{xx}^{-1}) = \hat{\boldsymbol{\delta}}_{\mathrm{TSLS}}$ is

$$\mathrm{avar}(\hat{\boldsymbol{\delta}}_{\mathrm{TSLS}}) = (\boldsymbol{\Sigma}_{xz}'\mathbf{S}^{-1}\boldsymbol{\Sigma}_{xz})^{-1} = \sigma^2(\boldsymbol{\Sigma}_{xz}'\boldsymbol{\Sigma}_{xx}^{-1}\boldsymbol{\Sigma}_{xz})^{-1}$$

Although $\hat{\boldsymbol{\delta}}(\mathbf{S}_{xx}^{-1})$ does not depend on $\hat{\sigma}^2$, a consistent estimate of the asymptotic variance does:

$$\widehat{\mathrm{avar}}(\hat{\boldsymbol{\delta}}_{\mathrm{TSLS}}) = \hat{\sigma}^2(\mathbf{S}_{xz}'\mathbf{S}_{xx}^{-1}\mathbf{S}_{xz})^{-1}$$

Similarly, the J-statistic also depends on $\hat{\sigma}^2$ and takes the form

$$J(\hat{\boldsymbol{\delta}}_{\mathrm{TSLS}}, \hat{\sigma}^{-2}\mathbf{S}_{xx}^{-1}) = n\frac{(\mathbf{s}_{xy} - \mathbf{S}_{xz}\hat{\boldsymbol{\delta}}_{\mathrm{TSLS}})'\mathbf{S}_{xx}^{-1}(\mathbf{s}_{xy} - \mathbf{S}_{xz}\hat{\boldsymbol{\delta}}_{\mathrm{TSLS}})}{\hat{\sigma}^2}$$

The TSLS J-statistics is also known as Sargan's statistic (see Sargan 1958).

21.3 Estimation of S

To compute any of the efficient GMM estimators, a consistent estimate of $\mathbf{S} = \mathrm{avar}(\bar{\mathbf{g}})$ is required. The method used to estimate $\mathbf{S}$ depends on the time series properties of the population moment conditions $\mathbf{g}_t$. Two cases are generally considered. In the first case, $\mathbf{g}_t$ is assumed to be serially uncorrelated but may be conditionally heteroskedastic. In the second case, $\mathbf{g}_t$ is assumed to be serially correlated as well as potentially conditionally heteroskedastic. The following subsections discuss the estimation of $\mathbf{S}$ in these two cases. Similar estimators were discussed in the context of linear regression in Chapter 6, Section 5. In what follows, the assumption of a linear model (21.1) is dropped and the K moment conditions embodied in the vector $\mathbf{g}_t$ are assumed to be nonlinear functions of $q \leq K$ model parameters $\boldsymbol{\theta}$ and are denoted $\mathbf{g}_t(\boldsymbol{\theta})$. The moment conditions satisfy $E[\mathbf{g}_t(\boldsymbol{\theta}_0)] = \mathbf{0}$ and $\mathbf{S} = \mathrm{avar}(\bar{\mathbf{g}}) = \mathrm{avar}(n^{-1}\sum_{i=1}^{n}\mathbf{g}_t(\boldsymbol{\theta}_0))$.

Kernel	**Default** b_n	$w_{j,n}$
Truncated	$\text{int}\left[4\left(\frac{n}{100}\right)^{1/5}\right]$	$\begin{cases} 1 & \text{for } a_j < 1 \\ 0 & \text{for } a_j \geq 1 \end{cases}$
Bartlett	$\text{int}\left[4\left(\frac{n}{100}\right)^{1/4}\right]$	$\begin{cases} 1 - a_j & \text{for } a_j \leq 1 \\ 0 & \text{for } a_j > 1 \end{cases}$
Parzen	$\text{int}\left[4\left(\frac{n}{100}\right)^{4/25}\right]$	$\begin{cases} 1 - 6a_j^2 + 6a_j^3 & \text{for } 0 \leq a_j \leq 0.5 \\ 2(1-a_j)^3 & \text{for } 0.5 \leq a_j \leq 1 \\ 0 & \text{for } a_j > 1 \end{cases}$
Quadratic spectral	$\text{int}\left[4\left(\frac{n}{100}\right)^{4/25}\right]$	$\frac{25}{12\pi^2 d_i^2}\left[\frac{\sin(m_j)}{m_j} - \cos(m_j)\right]$
note: $a_j = i/(b_n+1)$, $d_j = j/b_n$, $m_j = 6\pi d_i/5$		

TABLE 21.1. Common kernel weights and default bandwidths

21.3.1 Serially Uncorrelated Moments

In many situations, the population moment conditions $\mathbf{g}_t(\boldsymbol{\theta}_0)$ form an ergodic-stationary MDS with an appropriate information set I_t. In this case,

$$\mathbf{S} = \text{avar}(\bar{\mathbf{g}}) = E[\mathbf{g}_t(\boldsymbol{\theta}_0)\mathbf{g}_t(\boldsymbol{\theta}_0)']$$

Following White (1980), a *heteroskedasticity consistent* (HC) estimate of $\mathbf{S}$ has the form

$$\hat{\mathbf{S}}_{\text{HC}} = \frac{1}{n}\sum_{t=1}^{n} \mathbf{g}_t(\hat{\boldsymbol{\theta}})\mathbf{g}_t(\hat{\boldsymbol{\theta}})' \tag{21.17}$$

where $\hat{\boldsymbol{\theta}}$ is a consistent estimate of $\boldsymbol{\theta}_0$.[2] Davidson and MacKinnon (1993, Sec. 16.3) suggested using a simple degrees-of-freedom corrected estimate of $\mathbf{S}$ that replaces n^{-1} in (21.17) with $(n-k)^{-1}$ to improve the finite sample performance of tests based on (21.11).

21.3.2 Serially Correlated Moments

If the population moment conditions $\mathbf{g}_t(\boldsymbol{\theta}_0)$ are an ergodic-stationary but serially correlated process then

$$\mathbf{S} = \text{avar}(\bar{\mathbf{g}}) = \boldsymbol{\Gamma}_0 + \sum_{j=1}^{\infty}(\boldsymbol{\Gamma}_j + \boldsymbol{\Gamma}_j')$$

where $\boldsymbol{\Gamma}_j = E[\mathbf{g}_t(\boldsymbol{\theta}_0)\mathbf{g}_{t-j}(\boldsymbol{\theta}_0)']$. In this case, a *heteroskedasticity and autocorrelation consistent* (HAC) estimate of $\mathbf{S}$ has the form

$$\hat{\mathbf{S}}_{\text{HAC}} = \hat{\boldsymbol{\Gamma}}_0(\hat{\boldsymbol{\theta}}) + \sum_{j=1}^{n-1} w_{j,n}(\hat{\boldsymbol{\Gamma}}_j(\hat{\boldsymbol{\theta}}) + \hat{\boldsymbol{\Gamma}}_j'(\hat{\boldsymbol{\theta}}))$$

[2]For example, $\hat{\boldsymbol{\theta}}$ may be an inefficient GMM estimate based on an arbitrary p.d. weight matrix.

where $w_{j,n}$ $(j = 1, \ldots, b_n)$ are kernel function weights, b_n is a non-negative bandwidth parameter that may depend on the sample size, $\hat{\mathbf{\Gamma}}_j(\hat{\boldsymbol{\theta}}) = \frac{1}{n}\sum_{t=j+1}^{n} \mathbf{g}_t(\hat{\boldsymbol{\theta}})\mathbf{g}_{t-j}(\hat{\boldsymbol{\theta}})'$, and $\hat{\boldsymbol{\theta}}$ is a consistent estimate of $\boldsymbol{\theta}_0$. Different HAC estimates of $\mathbf{S}$ are distinguished by their kernel weights and bandwidth parameter. The most common kernel functions are listed in Table 21.1. For all kernels except the quadratic spectral, the integer bandwidth parameter, b_n, acts as a lag truncation parameter and determines how many autocovariance matrices to include when forming $\hat{\mathbf{S}}_{\text{HAC}}$. Figure 21.1 illustrates the first 10 kernel weights for the kernels listed in Table 21.1 evaluated using the default values of b_n for $n = 100$. The choices of kernel and bandwidth determine the statistical properties of $\hat{\mathbf{S}}_{\text{HAC}}$. The truncated kernel is often used if the moment conditions follow a finite-order moving average process. However, the resulting estimate of $\mathbf{S}$ is not guaranteed to be positive definite. Use of the Bartlett, Parzen, or quadratic spectral kernel ensures that $\hat{\mathbf{S}}_{\text{HAC}}$ will be positive semidefinite. For these kernels, Andrews (1991) studied the asymptotic properties $\hat{\mathbf{S}}_{\text{HAC}}$. He showed that $\hat{\mathbf{S}}_{\text{HAC}}$ is consistent for $\mathbf{S}$ provided that $b_n \to \infty$ as $n \to \infty$. Furthermore, for each kernel, Andrews determined the rate at which $b_n \to \infty$ to asymptotically minimize the mean sequared error, $MSE(\hat{\mathbf{S}}_{\text{HAC}}, \mathbf{S})$. For the Bartlett, Parzen, and quadratic spectral kernels, the rates are $n^{1/3}$, $n^{1/5}$, and $n^{1/5}$, respectively. Using the optimal bandwidths, Andrews found that the $\hat{\mathbf{S}}_{\text{HAC}}$ based on the quadratic spectral kernel has the smallest asymptotic MSE, followed closely by $\hat{\mathbf{S}}_{\text{HAC}}$ based on the Parzen kernel.

Automatic Bandwidth Selection

Based on extensive Monte Carlo experiments, Newey and West (1994) concluded that the choice of bandwidth parameter was more important than the choice of kernel for the finite sample performance of $\hat{\mathbf{S}}_{\text{HAC}}$. Increasing b_n reduces the bias of $\hat{\mathbf{S}}_{\text{HAC}}$ but increases the variance. The default values for b_n in Table 21.1 are *ad hoc*, being motivated by the convergence rates of $\hat{\mathbf{S}}_{\text{HAC}}$ for the respective kernels. To overcome the *ad hoc* choice of b_n, Andrews (1991) and Newey and West (1994) propose data dependent automatic bandwidth selection procedures that asymptotically minimize $MSE(\hat{\mathbf{S}}_{\text{HAC}}, \mathbf{S})$. The details of these procedures are tedious and so are not repeated here. The interested reader is referred to den Haan and Levin (1997), who nicely summarize the procedures and give guidance to the practitioner. However, as den Haan and Levin point out, the so-called "automatic" bandwidth selection procedures still depend on a $K \times 1$ user-specified vector of weights $\mathbf{w}$ for the elements of $\mathbf{g}_t(\hat{\boldsymbol{\theta}})$ and a method of providing initial estimates of $\mathbf{S}$.

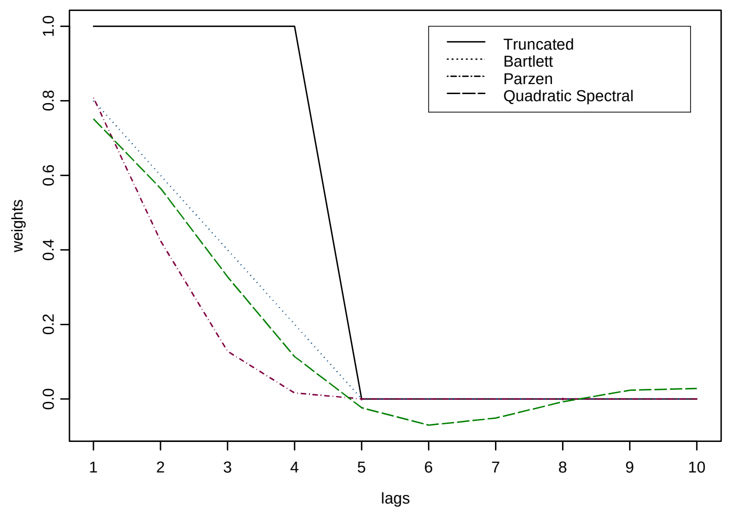


FIGURE 21.1. Kernel weights evaluated at default bandwidths for $n = 100$.

Prewhitening and Recoloring

If $\mathbf{g}_t(\hat{\boldsymbol{\theta}})$ is highly autocorrelated, Andrews and Monahan (1992) found that the finite sample behavior of $\hat{\mathbf{S}}_{\mathrm{HAC}}$ is often improved if a prewhitening and recoloring procedure is used. This procedure works as follows:

[1] Prewhiten $\mathbf{g}_t(\hat{\boldsymbol{\theta}})$ by estimating a single lag vector autoregression, VAR(1), for $\mathbf{g}_t(\hat{\boldsymbol{\theta}})$

$$\mathbf{g}_t(\hat{\boldsymbol{\theta}}) = \Phi \mathbf{g}_{t-1}(\hat{\boldsymbol{\theta}}) + e_t(\hat{\boldsymbol{\theta}})$$

and forming the residuals $e_t(\hat{\boldsymbol{\theta}}) = \mathbf{g}_t(\hat{\boldsymbol{\theta}}) - \hat{\Phi}\mathbf{g}_{t-1}(\hat{\boldsymbol{\theta}})$.

[2] Construct an HAC estimator for $e_t(\hat{\boldsymbol{\theta}})$:

$$\hat{\mathbf{\Sigma}}_{\mathrm{HAC}} = \frac{1}{n}\sum_{j=1}^{b_n} w_{j,n}(\hat{\mathbf{\Gamma}}_j + \hat{\mathbf{\Gamma}}_j')$$

where $\hat{\mathbf{\Gamma}}_j = \frac{1}{n}\sum_{t=j+1}^{n} e_t(\hat{\boldsymbol{\theta}})e_{t-j}(\hat{\boldsymbol{\theta}})$.

[3] Form the recolored HAC estimate for $\mathbf{S}$ using

$$\hat{\mathbf{S}}_{\mathrm{PWHAC}} = (\mathbf{I}_K - \hat{\Phi})^{-1}\hat{\mathbf{\Sigma}}_{\mathrm{HAC}}(\mathbf{I}_K - \hat{\Phi})^{-1\prime}$$

21.3.3 *Estimating* **S** *Using the* ***S+FinMetrics*** *Function* ***var.hac***

The HAC estimates of **S** for an $n \times k$ time series $\mathbf{x}_t$, based on the procedures described in the previous section, may be computed using the S+FinMetrics function var.hac.[3] The arguments expected by var.hac are

```
> args(var.hac)
function(x, bandwidth = NULL, window = "parzen", na.rm = F,
automatic = "none", df.correction = 0, prewhiten = F,
w = NULL, demean = T)
```

The optional arguments bandwidth and window are used to specify the bandwidth parameter b_n and kernel weight function $w_{j,n}$, respectively. Valid kernels are those listed in Table 21.1: "truncated", "bartlett", "parzen", and "qs". If the bandwidth is not specified, then the default value for b_n from Table 21.1 is used for the specified kernel. The argument df.correction specifies a non-negative integer to be subtracted from the sample size to perform a degrees-of-freedom correction. The argument automatic determines if the Andrews (1991) or Newey-West (1994) automatic bandwidth selection procedure is to be used to set b_n. If automatic is set to "andrews" or "nw", then the argument w must be supplied as a vector of weights for each variable in x. The Andrews-Monahan (1992) VAR(1) prewhitening and recoloring procedure is performed if prewhiten=T.

21.4 GMM Estimation Using the S+FinMetrics Function GMM

The GMM estimation of general linear and nonlinear models may be performed using the S+FinMetrics function GMM. The arguments expected by GMM are

```
> args(GMM)
function(start, moments, control = NULL, scale = NULL, lower
=  -Inf, upper = Inf, trace = T, method = "iterative",
w = NULL, max.steps = 100, w.tol = 0.0001, ts = F,
df.correction = T, var.hac.control = var.hac.control(),
w0.efficient = F, ...)
```

The required arguments are start, which is a vector of starting values for the parameters of the model, and moments, which is an S-PLUS function

[3]The function var.hac is an enhanced version of the S+FinMertics function asymp.var.

to compute the sample moment conditions used for estimating the model. The `moments` function must be of the form `f(parm,...)`, where `parm` is a vector of L parameters, and return a matrix of dimension $n \times K$ giving the GMM moment conditions $\mathbf{x}_t'\varepsilon_t$ for $t = 1, \ldots, n$. The optional arguments `control`, `scale`, `lower`, and `upper` are used by the `S-PLUS` optimization function `nlregb`. See the online help for `nlregb` and `nlregb.control` for details. Setting `trace=T` displays iterative information from the optimization. The argument `method` determines the type of GMM estimation to be performed. Valid choices are `"iterative"`, for iterated GMM estimation, and `"simultaneous"`, for continuous updating GMM estimation. If `method="iterative"` then the argument `max.steps` determines the number of iterations to be performed. The argument `w` specifies the weight matrix used for constructing the GMM objective function (21.6).[4] If `w=NULL`, then an estimate of the efficient weight matrix based on the asymptotic variance of the sample moment conditions will be used. In this case, the argument `ts` determines if an HC or HAC covariance estimator is computed and the arguments `df.correction` and `var.hac.control` control various options associated with these estimators. The user may supply a positive definite and symmetric weight matrix to be used as the initial weight matrix if `method="iterative"`. This weight matrix may be fixed throughout the estimation by setting `max.step=0`. If `method="interative"`, and `max.step=0` then the argument `w0.efficient` indicates whether the user-supplied weight matrix is an efficient weight matrix. This is useful for computing certain types of test statistics based on the GMM objective function. The argument `...` specifies any optional arguments that will be passed to the `moments` function used for computing the GMM moment conditions. Typically, these arguments specify the data used to compute the moment conditions.

The `GMM` function produces an object of class `"GMM"` for which there are `print` and `summary` methods, and extractor function `coef`.

Example 141 *Estimating the classical linear regression model by GMM*

Consider the classical linear regression model

$$y_t = \mathbf{x}_t'\boldsymbol{\beta}_0 + \varepsilon_t \tag{21.18}$$

where the explanatory variables $\mathbf{x}_t$ are assumed to be orthogonal to the error term. However, ε_t is allowed to be conditionally heteroskedastic and/or serially correlated. In this model, the explanatory variables are also the instrumental variables so that $\mathbf{z}_t = \mathbf{x}_t$ and $K = L$. The population orthogonality condition is

$$E[g_t(\mathbf{w}_t, \boldsymbol{\beta}_0)] = E[\mathbf{x}_t\varepsilon_t] = E[\mathbf{x}_t(y_t - \mathbf{x}_t'\boldsymbol{\beta}_0)] = \mathbf{0}$$

[4]The current version of `GMM` uses the inverse of `w` as the initial weight matrix.

where $g_t(\mathbf{w}_t, \boldsymbol{\beta}) = \mathbf{x}_t \varepsilon_t$ and $\mathbf{w}_t = (y_t, \mathbf{x}_t)'$. The sample moment condition used for estimation is

$$g_n(\mathbf{w}_t, \boldsymbol{\beta}) = \frac{1}{n} \sum_{t=1}^{n} \mathbf{x}_t (y_t - \mathbf{x}_t' \boldsymbol{\beta})$$

which gives rise to the GMM estimating equation

$$\mathbf{S}_{xy} - \mathbf{S}_{xx} \boldsymbol{\beta} = \mathbf{0}$$

Since $K = L$, the model is just identified, and, provided that $\mathbf{S}_{xx}$ is invertible, the GMM estimator is equivalent to the least squares estimator

$$\hat{\boldsymbol{\beta}} = \mathbf{S}_{xx}^{-1} \mathbf{S}_{xy}$$

The estimate $\hat{\boldsymbol{\beta}}$ is asymptotically normally distributed with asymptotic variance

$$\text{avar}(\hat{\boldsymbol{\beta}}) = (\boldsymbol{\Sigma}_{xx} \mathbf{S}^{-1} \boldsymbol{\Sigma}_{xx})^{-1}$$

where $\mathbf{S} = \text{avar}(\bar{\mathbf{g}})$. If ε_t is iid $(0, \sigma^2)$, say, then $\mathbf{S} = E[\mathbf{x}_t \mathbf{x}_t' \varepsilon_t^2] = \sigma^2 \boldsymbol{\Sigma}_{xx}$ and $\text{avar}(\hat{\boldsymbol{\beta}}) = \boldsymbol{\sigma}^2 \boldsymbol{\Sigma}_{xx}^{-1}$, which is the usual formula for $\text{avar}(\hat{\boldsymbol{\beta}})$ in the classical linear regression formula.

As an example of a simple linear regression model, consider the Capital Asset Pricing Model (CAPM)

$$R_t - r_{ft} = \alpha + \beta(R_{Mt} - r_{ft}) + \varepsilon_t, \; t = 1, \ldots, n \tag{21.19}$$

where R_t denotes the return on an asset, r_{ft} denotes the risk-free rate, and R_{Mt} denotes the return on a market portfolio proxy. Using the notation for the linear model (21.18), $y_t = R_t - r_{ft}$ and $x_t = (1, R_{Mt} - r_{ft})'$. The data for this example are the monthly excess returns on Microsoft stock and the S&P 500 index over the period February 1990 through December 2000 in the S+FinMetrics "timeSeries" object excessReturns.ts. Assuming that the error term is orthogonal to $R_{Mt} - r_{ft}$, the CAPM may be consistently estimated using ordinary least squares (OLS):

```
> ols.fit = OLS(MSFT~SP500, data = excessReturns.ts)
> ols.fit

Call:
OLS(formula = MSFT ~SP500, data = excessReturns.ts)

Coefficients:
 (Intercept)  SP500
 0.0175      1.5677

Degrees of freedom: 131 total; 129 residual
Time period: from Feb 1990 to Dec 2000
Residual standard error: 0.09094843
```

An S-PLUS function to compute the moment conditions for the linear regression model is

```
ols.moments = function(parm,y=NULL,x=NULL) {
  x = as.matrix(x)
  x * as.vector(y - x %*% parm)
}
```

where parm is an $L \times 1$ vector of parameter $\boldsymbol{\beta}$, y is an $n \times 1$ vector of observations on the dependent variable, and x is an $n \times L$ matrix of observations on the explanatory variables. The function returns an $n \times L$ matrix of moment conditions $\mathbf{x}_t' \varepsilon_t = \mathbf{x}_t'(y_t - \mathbf{x}_t'\boldsymbol{\beta})$ for $t = 1, \ldots, n$:

```
> excessReturns.df = excessReturns.ts@data
> ols.moments(c(1,1),y = excessReturns.df[,"MSFT"],
+             x = cbind(1, excessReturns.df[,"SP500"]))
numeric matrix: 131 rows, 2 columns.
            [,1]          [,2]
 [1,] -0.9409745 -0.0026713864
 [2,] -0.9027093 -0.0161178959
...
[131,] -1.2480621  0.0009318206
```

To estimate the CAPM regression (21.19) with GMM assuming heteroskedastic errors, use

```
> start.vals = c(0,1)
> names(start.vals) = c("alpha", "beta")
> gmm.fit = GMM(start.vals, ols.moments, max.steps = 1,
+           y = excessReturns.df[,"MSFT"],
+           x = cbind(1, excessReturns.df[,"SP500"]))
> class(gmm.fit)
[1] "GMM"
```

Notice how the data are passed to the function ols.moments through the ... argument. The object gmm.fit returned by GMM is of class "GMM" and has components

```
> names(gmm.fit)
 [1] "parameters"         "objective"
 [3] "message"            "grad.norm"
 [5] "iterations"         "r.evals"
 [7] "j.evals"            "scale"
 [9] "normalized.moments" "vcov"
[11] "method"             "df.J"
[13] "df.residual"        "call"
```

The online help for GMM gives a complete description of these components.[5] Typing the name of the "GMM" object invokes the print method on the fitted object:

```
> gmm.fit

Call:
GMM(start = start.vals, moments = ols.moments, max.steps = 1,
y = excessReturns.df[, "MSFT"], x = cbind(1,
excessReturns.df[, "SP500"]))

Coefficients:
  alpha   beta
 0.0175 1.5677

Test of Overidentification:
model is just-identified

Optimization Info:
 Number of Iterations: 1
 Convergence: x convergence
```

As expected, the GMM estimates of α and β are equivalent to the least squares estimates. Also, since the linear model is just identified, the GMM objective function (J-statistic) is identically zero at the optimum and, therefore, there is no test for overidentifying restrictions.

The summary method provides information about the statistical significance of the estimated parameters:

```
> summary(gmm.fit)

Call:
GMM(start = start.vals, moments = ols.moments, max.steps = 1,
y = excessReturns.df[, "MSFT"], x = cbind(1,
excessReturns.df[, "SP500"]))

Coefficients:
       Value Std.Error t value Pr(>|t|)
alpha 0.0175 0.0079    2.2175  0.0283
 beta 1.5677 0.1905    8.2274  0.0000
```

[5]Since the linear model is just identified, the weight matrix in the GMM objective function is irrelevant and so the weight.matrix component of gmm.fit is not returned. Also, the moments.vcov component is not returned since all of the normalized moments are equal to zero.

```
Test of Overidentification:
model is just-identified

Optimization Info:
 Number of Iterations: 1
 Convergence: x convergence
```

By default, the GMM function computes an HC estimate of the asymptotic variance of the sample moment conditions, and so values in the column labeled Std.Error are heteroskedasticity consistent (HC) standard errors. To be sure, these standard errors may be compared to those computed from the OLS fit with the White HC correction:

```
> summary(ols.fit,correction="white")

Call:
OLS(formula = MSFT ~SP500, data = excessReturns.ts)

Residuals:
     Min      1Q  Median      3Q     Max
 -0.3101 -0.0620 -0.0024  0.0581  0.2260

Coefficients:
             Value Std. Error t value Pr(>|t|)
(Intercept) 0.0175 0.0079     2.2175  0.0283
      SP500 1.5677 0.1905     8.2274  0.0000
```

To compute the GMM estimator using an HAC estimate of the asymptotic variance of the sample moment conditions, call the GMM function with the optional argument ts=T. The type of HAC estimate used is determined by the options set in var.hac.control. For example, to compute the OLS estimates with the usual Newey-West HAC standard errors use

```
> gmm.fit2 = GMM(c(0,1), ols.moments, max.steps = 1,
+  y = excessReturns.df[,"MSFT"],
+  x = cbind(1, excessReturns.df[,"SP500"]), ts = T,
+  var.hac.control = var.hac.control(window = "bartlett",
+  bandwidth = floor(4 * (nrow(excessReturns.df)/100)^(2/9))))
```

The standard errors for the above GMM estimates are identical to those returned by

```
> summary(ols.fit,correction="nw")
```

Example 142 *Estimating the instrumental variables regression model using GMM*

As in Campbell and Mankiw (1990), consider the stylized consumption function

$$\begin{aligned}\Delta c_t &= \delta_0 + \delta_1 \Delta y_t + \delta_2 r_t + \varepsilon_t,\ t = 1, \ldots, T \\ &= \boldsymbol{\delta}' \mathbf{z}_t + \varepsilon_t\end{aligned} \tag{21.20}$$

where c_t denotes the log of real per capita consumption (excluding durables), y_t denotes the log of real disposable income, and r_t denotes the ex post real interest rate (T-bill rate – inflation rate). Assume that $\{\Delta c_t, \Delta y_t, r_t\}$ are stationary and ergodic and that $\{\varepsilon_t, I_t\}$ is a stationary and ergodic martingale difference sequence (MDS), where $I_t = \{\Delta c_s, \Delta y_s, r_s\}_{s=1}^{t}$ denotes the observed information set at time t. In (21.20), the variables Δy_t and r_t are likely to be contemporaneously correlated with ε_t and so the least squares estimates of $\boldsymbol{\delta}$ are likely to be biased and inconsistent. Because $\{\varepsilon_t, I_t\}$ is a stationary and ergodic MDS, $E[\varepsilon_t | I_{t-1}] = 0$, which implies that any variable in I_{t-1} is a potential instrument. Furthermore, for any variable $x_{t-1} \subset I_{t-1}$, $\{x_{t-1}\varepsilon_t\}$ is an uncorrelated sequence.

The data for this example are annual data over the period 1960 to 1995 taken from Wooldridge (2002), and are in the "timeSeries" object consump.ts:

```
> colIds(consump.ts)
[1] "GC" "GY" "R3"
> consump.ts@documentation
[1] "GY = log growth rate of real income"
[2] "GC = log growth rate of real consumption"
[3] "R3 = real 3-month T-bill rate"
[4] "source: Wooldridge (2002), Introduction to"
[5] "Econometrics, 2nd Edition"
[6] "South-Western Thompson"
```

The following data frame consump is created for use with the function GMM:

```
> nobs = numRows(consump.ts)
> consump = seriesData(consump.ts)
> consump$const = rep(1,nobs)
```

An S-PLUS function to compute the linear instrumental variables regression model moment conditions $g(\mathbf{w}_t, \boldsymbol{\delta}) = \mathbf{x}_t(y_t - \mathbf{z}_t'\boldsymbol{\delta})$ for $t = 1, \ldots, n$, is

```
iv.moments = function(parm, y, X, Z) {
     # parm = L x 1 vector of parameters
     # y = n x 1 response vector
     # X = n x K matrix of instruments
     # Z = n x L matrix of explanatory variables
     X = as.matrix(X)
     Z = as.matrix(Z)
     X * as.vector(y - Z %*% parm)
}
```

Applying this function to the consumption data with $\boldsymbol{\delta} = (1,1,1)'$, $\mathbf{z}_t = (\Delta y_t, r_t, 1)'$, and $\mathbf{x}_t = (\Delta c_{t-1}, \Delta y_{t-1}, r_{t-1}, 1)'$ gives

```
> iv.moments(c(1,1,1),y = consump[2:nobs,1],
+            X = consump[1:(nobs-1),],
+            Z = consump[2:nobs,2:4])
             GC            GY            R3      const
 1 -0.014143206 -0.0070677434 -0.0124922035 -1.0156263
 2 -0.013609181 -0.0154800892 -0.0141370678 -1.0244252
...
35 -0.015828003 -0.0143904735 -0.0175688817 -1.0395788
```

To estimate the consumption function (21.20) by two-step efficient GMM using $\hat{\mathbf{W}} = \mathbf{I}_K$ as the initial weight matrix and assuming conditionally heteroskedastic errors, call `GMM` with the optional arguments `method = "iterative"` and `max.steps = 1`:

```
> start.vals = rep(0.1,3)
> names(start.vals) = c("GY","R3","const")
> gmm.fit.2step = GMM(start.vals, iv.moments,
+                     method = "iterative", max.steps=1,
+                     y = consump[2:nobs,1],
+                     X = consump[1:(nobs-1),],
+                     Z = consump[2:nobs,2:4])
1-step objective = 1.02951e-8
2-step objective = 1.57854
```

A summary of the model fit is

```
> summary(gmm.fit.2step)

Call:
GMM(start = start.vals, moments = iv.moments, method =
"iterative", max.steps = 1, y = consump[2:nobs, 1],
X = consump[1:(nobs - 1),  ], Z = consump[2:nobs,2:4])

Coefficients:
        Value Std.Error t value Pr(>|t|)
   GY  0.6277  0.1500    4.1852  0.0002
   R3 -0.0099  0.0981   -0.1009  0.9202
const  0.0071  0.0037    1.9134  0.0647

Test of Overidentification:
 J-stat Df P.value
 1.5785 1  0.209
```

```
Optimization Info:
Number of Iterative Steps: 2
```

The coefficient on Δy_t is 0.6277, with an estimated standard error of 0.1761, and the coefficient on r_t is slightly negative, with an estimated standard error of 0.1156. The J-statistic is 1.5785 and has a p-value of 0.209 based on the chi-square distribution with one degree of freedom. The data appear to support the single overidentifying restriction.

To estimate the consumption function (21.20) by iterated efficient GMM assuming conditionally heteroskedastic errors, call `GMM` with `method = "iterative"` and `max.steps` set to a large number[6]:

```
> gmm.fit.iter = GMM(start.vals, iv.moments,
+                    method = "iterative", max.steps = 100,
+                    y = consump[2:nobs,1],
+                    X = consump[1:(nobs-1),],
+                    Z = consump[2:nobs,2:4])
1-step objective = 1.02951e-8
2-step objective = 1.57854
...
13-step objective = 1.85567
```

To compute the continuously updated efficient GMM estimator (21.14), call `GMM` with `method = "simultaneous"`:

```
> start.vals = gmm.fit.iter$parameters
> gmm.fit.cu = GMM(start.vals, iv.moments,
+                  method = "simultaneous",
+                  y = consump[2:nobs,1],
+                  X = consump[1:(nobs-1),],
+                  Z = consump[2:nobs,2:4])
```

Good starting values are important for the CU estimator, and the above estimation uses the iterated GMM estimates as starting values.

Finally, to compute an inefficient one-step GMM estimator with $\mathbf{W} = \mathbf{I}_4$, use

```
> start.vals = rep(0.1,3)
> names(start.vals) = c("GY","R3","const")
> gmm.fit.1step = GMM(start.vals, iv.moments,
+                     method = "iterative", max.steps = 0,
+                     w = diag(4), w0.efficient = F,
```

[6]Notice that the one-step objective almost equal to zero. This is caused by using $\hat{\mathbf{W}} = \mathbf{I}_k$ as the initial weight matrix since the scaling of the individual moment conditions is very different. Using $\hat{\mathbf{W}} = \mathbf{S}_{xx}^{-1}$ generally provides a better scaling of the moment conditions.

$\Delta c_t = \delta_1 + \delta_2 \Delta y_t + \delta_3 r_t + \varepsilon_t$ $\mathbf{x}_t = (1, \Delta c_{t-1}, \Delta y_{t-1}, r_{t-1})'$, $E[\mathbf{x}_t \varepsilon_t] = 0$, $E[\mathbf{x}_t \mathbf{x}_t' \varepsilon_t^2] = \mathbf{S}$				
Estimator	$\boldsymbol{\delta}_1$	$\boldsymbol{\delta}_2$	$\boldsymbol{\delta}_3$	**J-stat**
2-step efficient	.007 (.004)	.627 (.150)	−.010 (.098)	1.578 (.209)
Iterated efficient	.008 (.004)	.591 (.144)	−.032 (.095)	1.855 (.173)
CU efficient	.008 (.003)	.574 (.139)	−.054 .095	1.747 (.186)
1-step inefficient	.003 (.005)	.801 (.223)	−.024 (.116)	– –

TABLE 21.2. Efficient GMM estimates of the consumption function parameters

```
+                    y = consump[2:nobs,1],
+                    X = consump[1:(nobs-1),],
+                    Z = consump[2:nobs,2:4])
1-step objective = 1.02951e-8
1-step objective = 1.02951e-8
Warning messages:
1: Maximum iterative steps exceeded. in: GMM(start.vals,
iv.moments, method = "iterative", max.steps =  ....
2: The J-Statistic is not valid since the weight matrix is
not efficient. in: GMM(start.vals, iv.moments, method
= "iterative", max.steps =  ....
```

Table 21.2 summarizes the different efficient GMM estimators for the parameters in (21.20). The results are very similar across the efficient estimations.[7]

Example 143 *Estimating the instrumental variables regression model using TSLS*

The TSLS estimator of $\boldsymbol{\delta}$ may be computed using the function `GMM` by supplying the fixed weight matrix $\mathbf{W} = \mathbf{S}_{xx}^{-1}$ as follows[8]:

```
> w.tsls = crossprod(consump[1:(nobs-1),])/nobs
> start.vals = rep(0.1,3)
> names(start.vals) = c("GY","R3","const")
> gmm.fit.tsls = GMM(start.vals,iv.moments,
+                    method = "iterative",max.steps = 0,
+                    w = w.tsls,w0.efficient = T,
```

[7]To match the default Eviews output for GMM, set the optional argument `df.correction=F`.

[8]Recall the `GMM` function uses the inverse of `w` as the weight matrix.

```
+                    y = consump[2:nobs,1],
+                    X = consump[1:(nobs-1),],
+                    Z = consump[2:nobs,2:4])
1-step objective = 1.12666e-4

> gmm.fit.tsls

Call:
GMM(start = start.vals, moments = iv.moments, method =
"iterative", w = w.tsls, max.steps = 0, w0.efficient
= T, y = consump[2:nobs, 1], X = consump[1:(nobs -1), ],
Z = consump[2:nobs, 2:4])

Coefficients:
      GY      R3   const
  0.5862 -0.0269  0.0081

Test of Overidentification:
 J-stat Df P.value
 0.0001 1  0.9915

Optimization Info:
Number of Iterative Steps: 1
```

The TSLS estimate of $\boldsymbol{\delta}$ is similar to the efficient iterated estimate. The J-statistic and the estimate of avar($\hat{\boldsymbol{\delta}}_{\text{TSLS}}$) computed using $\hat{\mathbf{W}} = \hat{\mathbf{S}}_{xx}^{-1}$, however, are not correct since $\mathbf{S}_{xx}^{-1}$ is proportional to the efficient weight matrix. To get the correct values for these quantities, a consistent estimate $\hat{\sigma}^2$ of σ^2 is required to form the efficient weight matrix $\hat{\sigma}^2\mathbf{S}_{xx}^{-1}$. This is easily accomplished using

```
# compute TSLS estimate of error variance
> y = as.vector(consump[2:nobs,1])
> X = as.matrix(consump[1:(nobs-1),])
> Z = as.matrix(consump[2:nobs,2:4])
> d.hat = coef(gmm.fit.tsls)
> e.hat = y - Z%*%d.hat
> df = nrow(Z) - ncol(Z)
> s2 = as.numeric(crossprod(e.hat)/df)
# compute correct efficient weight matrix for tsls
# that contains error variance term
> w.tsls2 = crossprod(X)*s2/nobs
> start.vals = rep(0.1,3)
> names(start.vals) = c("GY","R3","const")
> gmm.fit.tsls2 = GMM(start.vals,iv.moments,
+                      method = "iterative",max.steps = 0,
+                    w = w.tsls2,w0.efficient = T,
+                    y = consump[2:nobs,1],
+                    X = consump[1:(nobs-1),],
+                    Z = consump[2:nobs,2:4])
1-step objective = 2.01841

> summary(gmm.fit.tsls2)

Call:
GMM(start = start.vals, moments = iv.moments, method =
"iterative", w = w.tsls2, max.steps = 0, w0.efficient
= T, y = consump[2:nobs, 1], X = consump[1:(nobs -1), ],
Z = consump[2:nobs, 2:4])

Coefficients:
        Value Std.Error t value Pr(>|t|)
   GY  0.5862  0.1327    4.4177  0.0001
   R3 -0.0269  0.0753   -0.3576  0.7230
const  0.0081  0.0032    2.5285  0.0166

Test of Overidentification:
 J-stat Df P.value
 2.0184 1  0.1554
```

21.5 Hypothesis Testing for Linear Models

The following subsections discuss hypothesis testing in linear models estimated by GMM. The main types of hypothesis test are for coefficient restrictions, overidentification restrictions, subsets of orthogonality restrictions, and instrument relevance. Except for the tests for instrument relevance, the tests extend in a straightforward way to nonlinear models estimated by GMM.

21.5.1 Testing Restrictions on Coefficients

Hypothesis testing on coefficients in linear GMM models is surveyed in Newey and West (1987) and nicely summarized in Chapter 3 of Hayashi (2000).

Wald Statistics

Wald-type statistics are based on the asymptotic normality of the GMM estimator $\hat{\boldsymbol{\delta}}(\hat{\mathbf{W}})$ for an arbitrary weight matrix $\hat{\mathbf{W}}$. Simple tests on individual coefficients of the form

$$H_0 : \delta_k = \delta_k^0 \tag{21.21}$$

may be conducted using the asymptotic t-ratio

$$t_k = \frac{\hat{\delta}_k(\hat{\mathbf{W}}) - \delta_k^0}{\widehat{\mathrm{SE}}(\hat{\delta}_k(\hat{\mathbf{W}}))} \tag{21.22}$$

where $\widehat{\mathrm{SE}}(\hat{\delta}_k(\hat{\mathbf{W}}))$ is the square root of the kth diagonal element of (21.9). Under the null hypothesis (21.21), the t-ratio (21.22) has an asymptotic standard normal distribution.

Linear hypotheses of the form

$$H_0 : \mathbf{R}\boldsymbol{\delta} = \mathbf{r} \tag{21.23}$$

where $\mathbf{R}$ is a fixed $q \times L$ matrix of rank q and $\mathbf{r}$ is a fixed $q \times 1$ vector, may be tested using the Wald statistic

$$\mathrm{Wald} = n(\mathbf{R}\hat{\boldsymbol{\delta}}(\hat{\mathbf{W}}) - \mathbf{r})' \left[\mathbf{R}\widehat{\mathrm{avar}}(\hat{\boldsymbol{\delta}}(\hat{\mathbf{W}}))\mathbf{R}'\right]^{-1} (\mathbf{R}\hat{\boldsymbol{\delta}}(\hat{\mathbf{W}}) - \mathbf{r}) \tag{21.24}$$

where $\widehat{\mathrm{avar}}(\hat{\boldsymbol{\delta}}(\hat{\mathbf{W}}))$ is given by (21.9). Under the null (21.23), the Wald statistic (21.24) has a limiting chi-square distribution with q degrees of freedom. The Wald statistic (21.24) is valid for any consistent and asymptotically normal GMM estimator $\hat{\boldsymbol{\delta}}(\hat{\mathbf{W}})$ based on an arbitrary symmetric and positive definite weight matrix $\hat{\mathbf{W}} \xrightarrow{p} \mathbf{W}$. Usually, Wald statistics are computed using $\hat{\mathbf{W}} = \hat{\mathbf{S}}^{-1}$ so that $\widehat{\mathrm{avar}}(\hat{\boldsymbol{\delta}}(\hat{\mathbf{S}}^{-1}))$ is given by (21.11).

Nonlinear hypotheses of the form

$$H_0 : \mathbf{a}(\boldsymbol{\delta}_0) = \mathbf{0} \tag{21.25}$$

where $\mathbf{a}(\boldsymbol{\delta}_0) = \mathbf{0}$ imposes q nonlinear restrictions and $\partial\mathbf{a}(\boldsymbol{\delta}_0)/\partial\boldsymbol{\delta}'$ has full rank q, may be tested using the Wald statistic

$$\mathrm{Wald} = n\mathbf{a}(\hat{\boldsymbol{\delta}}(\hat{\mathbf{W}}))' \left[\frac{\partial\mathbf{a}(\hat{\boldsymbol{\delta}}(\hat{\mathbf{W}}))}{\partial\boldsymbol{\delta}'}\widehat{\mathrm{avar}}(\hat{\boldsymbol{\delta}}(\hat{\mathbf{W}}))\frac{\partial\mathbf{a}(\hat{\boldsymbol{\delta}}(\hat{\mathbf{W}}))'}{\partial\boldsymbol{\delta}}\right]^{-1} \mathbf{a}(\hat{\boldsymbol{\delta}}(\hat{\mathbf{W}})) \tag{21.26}$$

Under the null (21.25), the Wald statistic (21.26) has a limiting chi-square distribution with q degrees of freedom.

GMM LR-Type Statistics

Linear and nonlinear restrictions on the model coefficients $\boldsymbol{\delta}$ may also be tested using a likelihood ratio (LR)-type statistic. In efficient GMM estimation, the unrestricted objective function is $J(\hat{\boldsymbol{\delta}}(\hat{\mathbf{S}}^{-1}), \hat{\mathbf{S}}^{-1})$, for a consistent estimate $\hat{\mathbf{S}}$ of $\mathbf{S}$. The restricted efficient GMM estimator solves

$$\tilde{\delta}_{\mathrm{R}}(\hat{\mathbf{S}}^{-1}) = \arg\min_{\delta} \; J(\boldsymbol{\delta}, \hat{\mathbf{S}}^{-1}) \text{ subject to } H_0 \tag{21.27}$$

The GMM LR-type statistic is the difference between the restricted and unrestricted J-statistics:

$$\mathrm{LR}_{\mathrm{GMM}} = J(\tilde{\boldsymbol{\delta}}_{\mathrm{R}}(\hat{\mathbf{S}}^{-1}), \hat{\mathbf{S}}^{-1}) - J(\hat{\boldsymbol{\delta}}(\hat{\mathbf{S}}^{-1}), \hat{\mathbf{S}}^{-1}) \tag{21.28}$$

Under the null hypotheses (21.23) or (21.25), $\mathrm{LR}_{\mathrm{GMM}}$ has a limiting chi-square distribution with q degrees of freedom. As $n \to \infty$, it can be shown that Wald $- \mathrm{LR}_{\mathrm{GMM}} \xrightarrow{p} 0$, although the two statistics may differ in finite samples. For linear restrictions, Wald and $\mathrm{LR}_{\mathrm{GMM}}$ are numerically equivalent provided that the same value of $\hat{\mathbf{S}}$ is used to compute the restricted and unrestricted efficient GMM estimators. Typically $\hat{\mathbf{S}}$ computed under the unrestricted model is used in constructing $\mathrm{LR}_{\mathrm{GMM}}$. In this case, when the restricted efficient GMM estimator is computed by solving (21.27) the weight matrix $\hat{\mathbf{S}}_{\mathrm{UR}}^{-1}$ is held fixed during the estimation (no iteration is performed on the weight matrix). If $\mathrm{LR}_{\mathrm{GMM}}$ is computed using two different consistent estimates of $\mathbf{S}$, say $\hat{\mathbf{S}}$ and $\tilde{\mathbf{S}}$, then it is not guaranteed to be positive in finite samples but is asymptotically valid. The $\mathrm{LR}_{\mathrm{GMM}}$ statistic has the advantage over the Wald statistic for nonlinear hypotheses in that it is invariant to how the nonlinear restrictions are represented. Additionally, Monte Carlo studies have shown that $\mathrm{LR}_{\mathrm{GMM}}$ often performs better than Wald in finite samples. In particular, Wald tends to over reject the null hypothesis when it is true.

Example 144 *Testing the PIH*

The pure permanent income hypothesis (PIH) due to Hall (1978) states that c_t in (21.20) is a martingale so that $\Delta c_t = \varepsilon_t$ is a MDS. Hence, the PIH implies the linear restrictions

$$H_0 : \delta_1 = \delta_2 = 0$$

which are of the form (21.23) with

$$\mathbf{R} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}, \ \mathbf{r} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

If there are temporary income consumers, then $\delta_1 > 0$.

The Wald statistic (21.24) based on the iterated efficient GMM estimator may be computed using

```
> Rmat = matrix(c(1,0,0,1,0,0),2,3)
> rvec = c(0,0)
> dhat = coef(gmm.fit.iter)
> avarRbhat = Rmat %*% gmm.fit.iter$vcov %*% t(Rmat)
> Rmr = Rmat %*% dhat - rvec
> wald.stat = as.numeric(t(Rmr) %*% solve(avarRbhat) %*% Rmr)
> wald.stat
[1] 16.99482
```

Since there are $q = 2$ linear restrictions, the Wald statistic has an asymptotic chi-square distribution with two degrees of freedom. The p-value is

```
> 1 - pchisq(wald.stat,2)
[1] 0.0002039964
```

which suggests rejecting the PIH at any reasonable level of significance.

To compute the GMM LR-type statistic (21.28), one must compute restricted and unrestricted GMM estimates using the same estimate $\hat{\mathbf{S}}^{-1}$of the efficient weight matrix and evaluate the corresponding J-statistics. Consider computing (21.28) using the iterated efficient GMM estimate as the unrestricted estimate. Its J-statistic is $J(\hat{\boldsymbol{\delta}}(\hat{\mathbf{S}}^{-1}), \hat{\mathbf{S}}^{-1}) = 1.855$. To compute the restricted efficient GMM estimate with $\hat{\mathbf{S}}^{-1}$ from the unrestricted iterated efficient estimation as the weight matrix, use the following commands:

```
> s.ur = solve(gmm.fit.iter$weight.matrix)
> s.ur = (s.ur + t(s.ur))/2
> start.vals = 0.1
> names(start.vals) = c("const")
> gmm.fit.r = GMM(start.vals, iv.moments,
+                 method = "iterative", max.steps = 0,
+                 w = s.ur, w0.efficient = T,
+                 y = consump[2:nobs,1],
+                 X = consump[1:(nobs-1),],
+                 Z = consump[2:nobs,4])
1-step objective = 18.8505
```

The second line above is used to ensure that the weight matrix passed to GMM is symmetric. The restricted model is specified using the function iv.moments with $z_t = (1)$, and $\mathbf{x}_t = (\Delta c_{t-1}, \Delta y_{t-1}, r_{t-1}, 1)'$. The restricted fit is given by

```
> summary(gmm.fit.r)

Call:
GMM(start = start.vals, moments = iv.moments, method =
"iterative", w = s.ur, max.steps = 0, w0.efficient = T,
y = consump[2:nobs, 1], X = consump[1:(nobs - 1),],
Z = consump[2:nobs, 4])

Coefficients:
        Value Std.Error t value Pr(>|t|)
const  0.0209  0.0012   17.0687  0.0000

Test of Overidentification:
  J-stat Df P.value
```

```
 18.8505 3  0.0003

Optimization Info:
Number of Iterative Steps: 1
```

The restricted J-statistic is $J(\tilde{\boldsymbol{\delta}}_{\mathrm{R}}(\hat{\mathbf{S}}^{-1}), \hat{\mathbf{S}}^{-1}) = 18.8505$. The GMM-LR statistic is then

```
> gmm.lr = gmm.fit.r$objective - gmm.fit.iter$objective
> gmm.lr
[1] 16.99482
```

which is numerically identical to the Wald statistic computed earlier.

21.5.2 Testing Subsets of Orthogonality Conditions

Consider the linear GMM model (21.1) with instruments $\mathbf{x}_t = (\mathbf{x}_{1t}', \mathbf{x}_{2t}')'$ such that $\mathbf{x}_{1t}$ is $K_1 \times 1$ and $\mathbf{x}_{2t}$ is $K_2 \times 1$ with $K_1 \geq L$ and $K_1 + K_2 = K$. The instruments $\mathbf{x}_{1t}$ are assumed to be valid (i.e., $E[\mathbf{x}_{1t}\varepsilon_t] = 0$), whereas the instruments $\mathbf{x}_{2t}$ are suspected not to be valid (i.e., $E[\mathbf{x}_{2t}\varepsilon_t] \neq 0$). A procedure to test for the validity of $\mathbf{x}_{2t}$ due to Newey (1985) is as follows. First, estimate (21.1) by efficient GMM using the full set of instruments $\mathbf{x}_t$, giving

$$\hat{\boldsymbol{\delta}}(\hat{\mathbf{S}}_{\mathrm{Full}}^{-1}) = (\mathbf{S}_{xz}'\hat{\mathbf{S}}_{\mathrm{Full}}^{-1}\mathbf{S}_{xz})^{-1}\mathbf{S}_{xz}'\hat{\mathbf{S}}_{\mathrm{Full}}^{-1}\mathbf{S}_{xy}$$

where

$$\hat{\mathbf{S}}_{\mathrm{Full}} = \begin{bmatrix} \hat{\mathbf{S}}_{11,\mathrm{Full}} & \hat{\mathbf{S}}_{12,\mathrm{Full}} \\ \hat{\mathbf{S}}_{21,\mathrm{Full}} & \hat{\mathbf{S}}_{22,\mathrm{Full}} \end{bmatrix}$$

such that $\hat{\mathbf{S}}_{11,\mathrm{Full}}$ is $K_1 \times K_1$. Second, estimate (21.1) by efficient GMM using only the instruments $\mathbf{x}_{1t}$ and using the weight matrix $\hat{\mathbf{S}}_{11,\mathrm{Full}}^{-1}$ giving

$$\tilde{\boldsymbol{\delta}}(\hat{\mathbf{S}}_{11,\mathrm{Full}}^{-1}) = (\mathbf{S}_{x_1 z}'\hat{\mathbf{S}}_{11,\mathrm{Full}}^{-1}\mathbf{S}_{x_1 z})^{-1}\mathbf{S}_{x_1 z}'\hat{\mathbf{S}}_{11,\mathrm{Full}}^{-1}\mathbf{S}_{x_1 y}$$

Next, form the statistic[9]

$$C = J(\hat{\boldsymbol{\delta}}(\hat{\mathbf{S}}_{\mathrm{Full}}^{-1}), \hat{\mathbf{S}}_{\mathrm{Full}}^{-1}) - J(\tilde{\boldsymbol{\delta}}(\hat{\mathbf{S}}_{11,\mathrm{Full}}^{-1}), \hat{\mathbf{S}}_{11,\mathrm{Full}}^{-1}) \tag{21.29}$$

Under the null hypothesis that $E[\mathbf{x}_t\varepsilon_t] = \mathbf{0}$, the statistic C has a limiting chi-square distribution with $K - K_1$ degrees of freedom.

Example 145 *Testing the endogeneity of r_t in the consumption function*

Consider testing the hypothesis that r_t in (21.20) is exogenous ($E[r_t\varepsilon_t] = 0$). In this case, the full set of instruments is $\mathbf{x}_t = (\Delta c_{t-1}, \Delta y_{t-1}, r_{t-1}, 1, r_t)'$ and the reduced set is $\mathbf{x}_{1t} = (\Delta c_{t-1}, \Delta y_{t-1}, r_{t-1}, 1)'$. Efficient GMM estimation of the full model is achieved using

[9]The use of $\hat{\mathbf{S}}_{11,\mathrm{Full}}^{-1}$ guarantees that the C statistic is non-negative.

```
> start.vals = rep(0.1,3)
> names(start.vals) = c("GY","R3","const")
> gmm.fit.full = GMM(start.vals,iv.moments,
+                     method = "iterative", max.steps = 100,
+                     y = consump[2:nobs,"GC"],
+                     X = cbind(consump[1:(nobs-1),],
+                               consump[2:nobs,"R3"]),
+                     Z = consump[2:nobs,2:4])
```

The efficient weight matrix $\hat{\mathbf{S}}_{11,\text{Full}}^{-1}$ may be extracted using

```
> w11.full = solve(gmm.fit.full$weight.matrix[1:4,1:4])
> w11.full = (w11.full + t(w11.full))/2
```

Efficient GMM estimation using $\mathbf{x}_{1t}$ together with $\hat{\mathbf{S}}_{11,\text{Full}}^{-1}$ may be computed using

```
> start.vals = rep(0.1,3)
> names(start.vals) = c("GY","R3","const")
> gmm.fit.11 = GMM(start.vals,iv.moments,
+                   method = "iterative", max.steps = 0,
+                   w = w11.full, w0.efficient = T,
+                   y = consump[2:nobs,"GC"],
+                   X = consump[1:(nobs-1),],
+                   Z = consump[2:nobs,2:4])
```

The C statistic (21.29) for testing the exogeneity of r_t is then

```
> C.stat = gmm.fit.full$objective - gmm.fit.11$objective
> C.stat
[1] 0.01821106
```

Since $K = 5$ and $K_1 = 4$, C has an limiting chi-square distribution with one degree of freedom. The p-value for the test

```
> 1 - pchisq(C.stat,1)
[1] 0.8926527
```

indicates that r_t may be treated as exogenous in the consumption function.

21.5.3 Testing Instrument Relevance

In order to obtain consistent GMM estimates, the instruments $\mathbf{x}_t$ must be uncorrelated with the error term ε_t (valid instruments) and they must be correlated with the endogenous variables $\mathbf{z}_t$ (relevant instruments). The subset orthogonality tests of the previous subsection can be used to test instrument validity. This subsection discusses some simple *tests for instrument relevance.*

Instrument relevance is related to the rank condition (21.4). To see this, consider the simple GMM regression involving a single endogenous variable and a single instrument

$$\begin{aligned} y_t &= z_t\delta + \varepsilon_t \\ E[x_t\varepsilon_t] &= 0 \end{aligned}$$

The rank condition (21.4) reduces to $\text{rank}(\Sigma_{zx}) = 1$, which implies that $\Sigma_{zx} \neq 0$. Assuming that both z_t and x_t are demeaned, the rank condition can be restated as $\text{cov}(x, z) = \Sigma_{zx} \neq 0$. Hence, the rank condition will be satisfied as long as x is correlated with z. If there are K instruments $x_{1t}, \ldots, x_{Kt}$ but only one endogenous variable z_t, then the rank condition holds as long as $\text{cov}(x_k, z) \neq 0$ *for some* k. If $\text{cov}(x_k, z) \approx 0$ for all k, then the instruments are called *weak*.

Testing instrument relevance is important in practice because recent research (e.g., Stock and Wright (2000)) has shown that standard GMM procedures for estimation and inference may be highly misleading if instruments are weak. If instruments are found to be weak, then nonstandard methods of inference should be used for constucting confidence intervals and performing hypothesis tests. Stock, Wright, and Yogo (2002) gave a nice survey of the issues associated with using GMM in the presence of weak instruments and discussed the nonstandard inference procedures that should be used.

In the general linear GMM regression (21.1), the relevance of the set of instruments $\mathbf{x}_t$ for each endogenous variable in $\mathbf{z}_t$ can be tested as follows. First, let $\mathbf{z}_{1t}$ denote the $L_1 \times 1$ vector of nonconstant endogenous variables in $\mathbf{z}_t$, and let $\mathbf{x}_{1t}$ denote the $K_1 \times 1$ remaining deterministic or exogenous variables in $\mathbf{z}_t$ such that $\mathbf{z}_t = (\mathbf{z}'_{1t}, \mathbf{x}'_{1t})'$ and $L_1 + K_1 = L$. Similarly, define $\mathbf{x}_{2t}$ as the $K_2 \times 1$ vector of exogenous variables that are excluded from $\mathbf{z}_t$ so that $\mathbf{x}_t = (\mathbf{x}'_{1t}, \mathbf{x}'_{2t})'$ and $K_1 + K_2 = K$. What is important for the rank condition are the correlations between the endogenous variables in $\mathbf{z}_{1t}$ and the instruments in $\mathbf{x}_{2t}$. To measure these correlations and to test for instrument relevance, estimate by least squares the so-called first-stage regression

$$z_{1lt} = \mathbf{x}'_{1t}\boldsymbol{\pi}_{1l} + \mathbf{x}_{2t}\boldsymbol{\pi}_{2l} + v_{lt},\ l = 1, \ldots, L_1$$

for each endogenous variable in $\mathbf{z}_{1t}$. The t-ratios on the variables in $\mathbf{x}_{2t}$ can be used to assess the strength of the correlation between z_{1lt} and the variables in $\mathbf{x}_{2t}$. The F-statistic for testing $\boldsymbol{\pi}_{2l} = \mathbf{0}$ can be used to assess the joint relevance of $\mathbf{x}_{2t}$ for z_{1lt}.

Example 146 *Testing instrument relevance in the consumption function*

In the consumption function regression, $\mathbf{z}_{1t} = (\Delta y_t, r_t)'$, $x_{1t} = (1)$, and $\mathbf{x}_{2t} = (\Delta c_{t-1}, \Delta y_{t-1}, r_{t-1})'$. The first stage regressions for Δy_t and r_t may be computed simultaneously using the `S+FinMetrics` function `OLS` as follows:

```
> firstStage.fit = OLS(cbind(GY,R3) ~ tslag(GC) + tslag(GY)
+                      + tslag(R3), data = consump)
> class(firstStage.fit)
[1] "mOLS"
```

When a multivariate response is specified in the call to `OLS`, a regression is performed for each response variable and the returned object is of class `"mOLS"`. A summary of each first-stage regression is

```
> summary(firstStage.fit)
Response: GY

Call:
OLS(formula = cbind(GY, R3) ~tslag(GC) + tslag(GY) + tslag(
R3), data = consump)

Residuals:
     Min      1Q  Median      3Q     Max
 -0.0305 -0.0122 -0.0021  0.0112  0.0349

Coefficients:
              Value Std. Error t value Pr(>|t|)
(Intercept)  0.0067  0.0055     1.2323  0.2271
  tslag(GC)  1.2345  0.3955     3.1214  0.0039
  tslag(GY) -0.5226  0.2781    -1.8787  0.0697
  tslag(R3)  0.0847  0.1395     0.6069  0.5483

Regression Diagnostics:

         R-Squared 0.2791
Adjusted R-Squared 0.2093
Durbin-Watson Stat 1.8808

Residual Diagnostics:
              Stat P-Value
Jarque-Bera 0.6181 0.7341
  Ljung-Box 8.9289 0.8812

Residual standard error: 0.0163 on 31 degrees of freedom
F-statistic: 4 on 3 and 31 degrees of freedom, the p-value
is 0.01617

Response: R3

Call:
OLS(formula = cbind(GY, R3) ~tslag(GC) + tslag(GY) + tslag(
R3), data = consump)

Residuals:
     Min      1Q  Median      3Q     Max
 -0.0264 -0.0069  0.0021  0.0068  0.0436

Coefficients:
              Value Std. Error t value Pr(>|t|)
(Intercept)  0.0083  0.0044     1.8987  0.0669
  tslag(GC)  0.1645  0.3167     0.5192  0.6073
  tslag(GY) -0.4290  0.2228    -1.9259  0.0633
  tslag(R3)  0.8496  0.1117     7.6049  0.0000

Regression Diagnostics:

         R-Squared 0.6519
Adjusted R-Squared 0.6182
Durbin-Watson Stat 1.7470

Residual Diagnostics:
               Stat P-Value
Jarque-Bera 14.8961  0.0006
  Ljung-Box 20.7512  0.1450

Residual standard error: 0.01305 on 31 degrees of freedom
F-statistic: 19.35 on 3 and 31 degrees of freedom, the p-value
is 2.943e-007
```

In the first-stage regression for Δy_t, the t-ratios for Δc_{t-1} and Δy_{t-1} are significant at the 1% and 10% levels, respectively, indicating that these variables are correlated with Δy_t. The F-statistic for testing the joint significance of the variables in $\mathbf{x}_{2t}$ is 4, with a p-value of 0.0167 indicating that $\mathbf{x}_{2t}$ is relevant for Δy_t.

21.6 Nonlinear GMM

Nonlinear GMM estimation occurs when the K GMM moment conditions $g(\mathbf{w}_t, \boldsymbol{\theta})$ are nonlinear functions of the p model parameters $\boldsymbol{\theta}$. Depending on the model, the moment conditions $g(\mathbf{w}_t, \boldsymbol{\theta})$ may be $K \geq p$ nonlinear functions satisfying

$$E[g(\mathbf{w}_t, \boldsymbol{\theta}_0)] = \mathbf{0} \tag{21.30}$$

Alternatively, for a response variable y_t, L explanatory variables $\mathbf{z}_t$, and K instruments $\mathbf{x}_t$, the model may define a nonlinear error term ε_t

$$a(y_t, \mathbf{z}_t; \boldsymbol{\theta}_0) = \varepsilon_t$$

such that

$$E[\varepsilon_t] = E[a(y_t, \mathbf{z}_t; \boldsymbol{\theta}_0)] = 0$$

Given that $\mathbf{x}_t$ is orthogonal to ε_t, define $g(\mathbf{w}_t, \boldsymbol{\theta}_0) = \mathbf{x}_t\varepsilon_t = \mathbf{x}_t a(y_t, \mathbf{z}_t; \boldsymbol{\theta}_0)$ so that

$$E[g(\mathbf{w}_t, \boldsymbol{\theta}_0)] = E[\mathbf{x}_t\varepsilon_t] = E[\mathbf{x}_t a(y_t, \mathbf{z}_t; \boldsymbol{\theta}_0)] = \mathbf{0} \tag{21.31}$$

defines the GMM orthogonality conditions.

In general, the GMM moment equations (21.30) and (21.31) produce a system of K nonlinear equations in p unknowns. Identification of $\boldsymbol{\theta}_0$ requires that

$$\begin{aligned} E[g(\mathbf{w}_t, \boldsymbol{\theta}_0)] &= \mathbf{0} \\ E[g(\mathbf{w}_t, \boldsymbol{\theta})] &\neq \mathbf{0} \text{ for } \boldsymbol{\theta} \neq \boldsymbol{\theta}_0 \end{aligned}$$

and the $K \times p$ matrix

$$\mathbf{G} = E\left[\frac{\partial g(\mathbf{w}_t, \boldsymbol{\theta}_0)}{\partial \boldsymbol{\theta}'}\right] \tag{21.32}$$

has full column rank p. The sample moment condition for an arbitrary $\boldsymbol{\theta}$ is

$$g_n(\boldsymbol{\theta}) = n^{-1}\sum_{t=1}^{n} g(\mathbf{w}_t, \boldsymbol{\theta})$$

If $K = p$, then $\boldsymbol{\theta}_0$ is apparently just identified and the GMM objective function is

$$J(\boldsymbol{\theta}) = n g_n(\boldsymbol{\theta})' g_n(\boldsymbol{\theta})$$

which does not depend on a weight matrix. The corresponding GMM estimator is then

$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \ J(\boldsymbol{\theta})$$

If $K > p$, then $\boldsymbol{\theta}_0$ is apparently overidentified. Let $\hat{\mathbf{W}}$ denote a $K \times K$ symmetric and p.d. weight matrix, possibly dependent on the data, such that $\hat{\mathbf{W}} \xrightarrow{p} \mathbf{W}$ as $n \to \infty$ with $\mathbf{W}$ symmetric and p.d. Then, the GMM estimator of $\boldsymbol{\theta}_0$, denoted $\hat{\boldsymbol{\theta}}(\hat{\mathbf{W}})$, is defined as

$$\hat{\boldsymbol{\theta}}(\hat{\mathbf{W}}) = \arg\min_{\boldsymbol{\theta}} \ J(\boldsymbol{\theta}, \hat{\mathbf{W}}) = n g_n(\boldsymbol{\theta})' \hat{\mathbf{W}} g_n(\boldsymbol{\theta})$$

The efficient GMM estimator uses $\hat{\mathbf{W}} = \hat{\mathbf{S}}^{-1}$ such that $\hat{\mathbf{S}} \xrightarrow{p} \mathbf{S} = \text{avar}(\bar{\mathbf{g}})$. As with efficient GMM estimation of linear models, the efficient GMM estimator of nonlinear models may be computed using a two-step, iterated, or continuous updating estimator.

21.6.1 Asymptotic Properties

Under standard regularity conditions (see Hayashi, 2000, Chap. 7), it can be shown that

$$\hat{\boldsymbol{\theta}}(\hat{\mathbf{W}}) \xrightarrow{p} \boldsymbol{\theta}_0$$
$$\sqrt{n}\left(\hat{\boldsymbol{\theta}}(\hat{\mathbf{W}}) - \boldsymbol{\theta}_0\right) \xrightarrow{d} N(\mathbf{0}, \text{avar}(\hat{\boldsymbol{\theta}}(\hat{\mathbf{W}})))$$

where

$$\text{avar}(\hat{\boldsymbol{\theta}}(\hat{\mathbf{W}})) = (\mathbf{G}'\mathbf{W}\mathbf{G})^{-1}\mathbf{G}'\mathbf{W}\mathbf{S}\mathbf{W}\mathbf{G}(\mathbf{G}'\mathbf{W}\mathbf{G})^{-1} \tag{21.33}$$

and $\mathbf{G}$ is given by (21.32). If $\mathbf{W} = \mathbf{S}^{-1}$, then

$$\text{avar}(\hat{\boldsymbol{\theta}}(\hat{\mathbf{S}}^{-1})) = (\mathbf{G}'\mathbf{S}^{-1}\mathbf{G})^{-1} \tag{21.34}$$

Notice that with nonlinear GMM, the expression for $\text{avar}(\hat{\boldsymbol{\theta}}(\hat{\mathbf{W}}))$ is of the same form as in linear GMM except that $\boldsymbol{\Sigma}_{xz} = E[\mathbf{x}_t\mathbf{z}_t']$ is replaced by $\mathbf{G} = E\left[\frac{\partial g(\mathbf{w}_t,\boldsymbol{\theta}_0)}{\partial \boldsymbol{\theta}'}\right]$.

A consistent estimate of $\text{avar}(\hat{\boldsymbol{\theta}}(\mathbf{W}))$, denoted $\widehat{\text{avar}}(\hat{\boldsymbol{\theta}}(\hat{\mathbf{W}}))$, may be computed using

$$\widehat{\text{avar}}(\hat{\boldsymbol{\theta}}(\hat{\mathbf{W}})) = (\hat{\mathbf{G}}'\hat{\mathbf{W}}\hat{\mathbf{G}})^{-1}\hat{\mathbf{G}}'\hat{\mathbf{W}}\hat{\mathbf{S}}\hat{\mathbf{W}}\hat{\mathbf{G}}(\hat{\mathbf{G}}'\hat{\mathbf{W}}\hat{\mathbf{G}})^{-1} \tag{21.35}$$

where $\hat{\mathbf{S}}$ is a consistent estimate for $\mathbf{S} = \text{avar}(\bar{\mathbf{g}})$ and

$$\hat{\mathbf{G}} = \mathbf{G}_n(\hat{\boldsymbol{\theta}}(\hat{\mathbf{W}})) = n^{-1}\sum_{t=1}^{n}\frac{\partial g(\mathbf{w}_t, \hat{\boldsymbol{\theta}}(\hat{\mathbf{W}}))}{\partial \boldsymbol{\theta}'}$$

For the efficient GMM estimator, $\hat{\mathbf{W}} = \hat{\mathbf{S}}^{-1}$ and

$$\widehat{\text{avar}}(\hat{\boldsymbol{\theta}}(\hat{\mathbf{S}}^{-1})) = (\hat{\mathbf{G}}'\hat{\mathbf{S}}^{-1}\hat{\mathbf{G}})^{-1} \tag{21.36}$$

If $\{\mathbf{g}_t(\mathbf{w}_t, \boldsymbol{\theta}_0)\}$ is an ergodic stationary MDS with $E[\mathbf{g}_t(\mathbf{w}_t, \boldsymbol{\theta}_0)\mathbf{g}_t(\mathbf{w}_t, \boldsymbol{\theta}_0)']$ $= \mathbf{S}$, then a consistent estimator of $\mathbf{S}$ takes the form

$$n^{-1}\sum_{t=1}^{n}\mathbf{g}_t(\mathbf{w}_t, \hat{\boldsymbol{\theta}})\mathbf{g}_t(\mathbf{w}_t, \hat{\boldsymbol{\theta}})'$$

where $\hat{\boldsymbol{\theta}}$ is any consistent estimator of $\boldsymbol{\theta}_0$. If $\{\mathbf{g}_t(\mathbf{w}_t, \boldsymbol{\theta}_0)\}$ is a serially correlated ergodic stationary process, then

$$\mathbf{S} = \text{avar}(\bar{\mathbf{g}}) = \boldsymbol{\Gamma}_0 + 2\sum_{j=1}^{\infty}(\boldsymbol{\Gamma}_j + \boldsymbol{\Gamma}_j')$$

and the methods discussed in Section 21.3 may be used to consistently estimate $\mathbf{S}$.

21.6.2 Hypothesis Tests for Nonlinear Models

Most of the GMM test statistics discussed in the context of linear models have analogs in the context of nonlinear GMM. For example, the J-statistic for testing the validity of the K moment conditions (21.31) has the form (21.15) with the efficient GMM estimate $\hat{\boldsymbol{\theta}}(\hat{\mathbf{S}}^{-1})$ in place of $\hat{\boldsymbol{\delta}}(\hat{\mathbf{S}}^{-1})$. The asymptotic null distribution of the J-statistic is chi-squared with $K-p$ degrees of freedom. Wald statistics for testing linear and nonlinear restriction on $\boldsymbol{\theta}$ have the same form as (21.24) and (21.26), with $\hat{\boldsymbol{\theta}}(\hat{\mathbf{W}})$ in place of $\hat{\boldsymbol{\delta}}(\hat{\mathbf{W}})$. Similarly, GMM LR-type statistics for testing linear and nonlinear restriction on $\boldsymbol{\theta}$ have the form (21.28), with $\tilde{\boldsymbol{\theta}}(\hat{\mathbf{S}}^{-1})$ in place of $\tilde{\boldsymbol{\delta}}(\hat{\mathbf{S}}^{-1})$ and with $\hat{\boldsymbol{\theta}}(\hat{\mathbf{S}}^{-1})$ in place of $\hat{\boldsymbol{\delta}}(\hat{\mathbf{S}}^{-1})$. Testing the relevance of instruments in nonlinear models, however, is not as straightforward as it is in linear models. In nonlinear models, instruments are relevant if the $K \times p$ matrix $\mathbf{G}$ defined in (21.32) has full column rank p. Testing this condition is problematic because $\mathbf{G}$ depends on $\boldsymbol{\theta}_0$, which is unknown. See Wright (2003) for an approach that can be used to test for instrument relevance in nonlinear models.

21.7 Examples of Nonlinear Models

The following subsections give detailed examples of estimating nonlinear models using GMM.

21.7.1 Student's t Distribution

As in Hamilton (1994, Chap. 14), consider a random sample $y_1, \ldots, y_T$ that is drawn from a centered Student's t distribution with θ_0 degrees of freedom. The density of y_t has the form

$$f(y_t; \theta_0) = \frac{\Gamma[(\theta_0+1)/2]}{(\pi\theta_0)^{1/2}\Gamma(\theta_0/2)}[1 + (y_t^2/\theta_0)]^{-(\theta_0+1)/2}$$

where $\Gamma(\cdot)$ is the gamma function. The goal is to estimate the degrees of freedom parameter θ_0 by GMM using the moment conditions

$$\begin{aligned} E[y_t^2] &= \frac{\theta_0}{\theta_0 - 2} \\ E[y_t^4] &= \frac{3\theta_0^2}{(\theta_0-2)(\theta_0-4)} \end{aligned}$$

which require $\theta_0 > 4$. Let $\mathbf{w}_t = (y_t^2, y_t^4)'$ and define

$$\mathbf{g}(\mathbf{w}_t, \boldsymbol{\theta}) = \begin{pmatrix} y_t^2 - \theta/(\theta-2) \\ y_t^4 - 3\theta^2/(\theta-2)(\theta-4) \end{pmatrix} \qquad (21.37)$$

Then, $E[\mathbf{g}(\mathbf{w}_t,\theta_0)] = \mathbf{0}$ is the moment condition used for defining the GMM estimator for θ_0. Here, $K = 2$ and $p = 1$, so θ_0 is apparently overidentified. Using the sample moments

$$\mathbf{g}_n(\theta) = \frac{1}{n}\sum_{t=1}^{n}\mathbf{g}(\mathbf{w}_t,\theta) = \begin{pmatrix} \frac{1}{n}\sum_{t=1}^{n} y_t^2 - \theta/(\theta-2) \\ \frac{1}{n}\sum_{t=1}^{n} y_t^4 - 3\theta^2/(\theta-2)(\theta-4) \end{pmatrix}$$

the GMM objective function has the form

$$J(\theta) = n\mathbf{g}_n(\theta)'\hat{\mathbf{W}}\mathbf{g}_n(\theta)$$

where $\hat{\mathbf{W}}$ is a 2×2 p.d. and symmetric weight matrix, possibly dependent on the data, such that $\hat{\mathbf{W}}\xrightarrow{p}\mathbf{W}$. The efficient GMM estimator uses the weight matrix $\hat{\mathbf{S}}^{-1}$ such that $\hat{\mathbf{S}} \xrightarrow{p} \mathbf{S} = E[\mathbf{g}(\mathbf{w}_t,\theta_0)\mathbf{g}(\mathbf{w}_t,\theta_0)']$.

A random sample of $n = 250$ observations from a centered Student's t distribution with $\theta_0 = 10$ degrees of freedom may be generated using the S-PLUS function rt as follows:

```
> set.seed(123)
> y = rt(250,df = 10)
```

Basic summary statistics, computed using

```
> summaryStats(y)

Sample Quantiles:
    min      1Q  median     3Q   max
 -4.387 -0.6582 -0.0673 0.6886 3.924

Sample Moments:
     mean   std skewness kurtosis
 -0.03566 1.128  -0.3792    4.659

Number of Observations:  250
```

indicate that the data are roughly symmetric about zero and have thicker tails than the normal distribution.

An S-PLUS function to compute the moment condition (21.37) for $t = 1, \ldots, n$ is

```
t.moments <- function(parm,data=NULL) {
     # parm = df parameter
     # data = [y^2, y^4] is assumed to be a matrix
     m1 = parm/(parm - 2)
     m2 = 3*parm*parm/((parm - 2)*(parm - 4))
     t(t(data) - c(m1,m2))
}
```

The function `t.moments` has arguments `parm`, specifying the degrees of freedom parameter θ, and `data`, specifying an $n \times 2$ matrix with the tth row $\mathbf{w}_t = (y_t^2, y_t^4)'$.

To compute the iterated efficient GMM estimator of the degrees of freedom parameter θ from the simulated Student's t data use

```
> y = y - mean(y)
> t.data = cbind(y^2,y^4)
> start.vals = 15
> names(start.vals) = c("theta")
> t.gmm.iter = GMM(start.vals, t.moments,
+                  method = "iterative", max.steps = 100,
+                  data = t.data)
1-step objective = 0.471416
2-step objective = 0.302495
3-step objective = 0.302467
> summary(t.gmm.iter)

Call:
GMM(start = start.vals, moments = t.moments, method =
"iterative", max.steps = 100, data = data)

Coefficients:
       Value Std.Error t value Pr(>|t|)
theta 7.8150 1.1230    6.9592  0.0000

Test of Overidentification:
 J-stat Df P.value
 0.3025 1  0.5823

Optimization Info:
Number of Iterative Steps: 3
```

The iterated efficient GMM estimate of θ_0 is 7.8150, with an asymptotic standard error of 1.123. The small J-statistic indicates a correctly specified model.

21.7.2 MA(1) Model

Following Harris (1999), consider GMM estimation of the parameters in the moving average MA(1) model

$$\begin{gathered} y_t = \mu_0 + \varepsilon_t + \psi_0 \varepsilon_{t-1},\ t = 1, \ldots, n \\ \varepsilon_t \sim \text{iid } (0, \sigma_0^2),\ |\psi_0| < 1 \\ \boldsymbol{\theta}_0 = (\mu_0, \psi_0, \sigma_0^2)' \end{gathered}$$

Some population moment equations that can be used for GMM estimation are

$$\begin{aligned} E[y_t] &= \mu_0 \\ E[y_t^2] &= \mu_0^2 + \sigma_0^2(1+\psi_0^2) \\ E[y_t y_{t-1}] &= \mu_0^2 + \sigma_0^2\psi_0 \\ E[y_t y_{t-2}] &= \mu_0^2 \end{aligned}$$

Let $\mathbf{w}_t = (y_t, y_t^2, y_t y_{t-1}, y_t y_{t-2})'$ and define the moment vector

$$\mathbf{g}(\mathbf{w}_t, \boldsymbol{\theta}) = \begin{pmatrix} y_t - \mu \\ y_t^2 - \mu^2 - \sigma^2(1+\psi^2) \\ y_t y_{t-1} - \mu^2 - \sigma^2\psi \\ y_t y_{t-2} - \mu^2 \end{pmatrix} \tag{21.38}$$

Then,

$$E[\mathbf{g}(\mathbf{w}_t, \boldsymbol{\theta}_0)] = \mathbf{0}$$

is the population moment condition used for GMM estimation of the model parameters $\boldsymbol{\theta}_0$. The sample moments are

$$\mathbf{g}_n(\boldsymbol{\theta}) = \frac{1}{n-2}\sum_{t=3}^{n} g(\mathbf{w}_t, \boldsymbol{\theta}) = \begin{pmatrix} \sum_{t=3}^{n} y_t/(n-2) - \mu \\ \sum_{t=3}^{n} y_t^2/(n-2) - \mu^2 - \sigma^2(1+\psi^2) \\ \sum_{t=3}^{n} y_t y_{t-1}/(n-2) - \mu^2 - \sigma^2\psi \\ \sum_{t=3}^{n} y_t y_{t-2}/(n-2) - \mu^2 \end{pmatrix}$$

Since the number of moment conditions $K = 4$ is greater than the number of model parameters $p = 3$, $\boldsymbol{\theta}_0$ is apparently overidentified and the efficient GMM objective function has the form

$$J(\boldsymbol{\theta}) = (n-2)\cdot \mathbf{g}_n(\boldsymbol{\theta})'\hat{\mathbf{S}}^{-1}\mathbf{g}_n(\boldsymbol{\theta})$$

where $\hat{\mathbf{S}}$ is a consistent estimate of $\mathbf{S} = \text{avar}(\bar{\mathbf{g}}(\boldsymbol{\theta}_0))$. Notice that the process $\{\mathbf{g}(\mathbf{w}_t, \boldsymbol{\theta}_0)\}$ will be autocorrelated (at least at lag 1) since y_t follows an MA(1) process. As a result, an HAC-type estimator must be used to estimate $\mathbf{S}$.

Simulated MA(1) data with $\boldsymbol{\theta}_0 = (0, 0.5, 1)'$ and $n = 250$ is computed using the `S-PLUS` function `arima.sim`[10]:

```
> set.seed(123)
> ma1.sim = arima.sim(model = list(ma=-0.5),n=250)
```

These data along with the sample autocorrelation function (SACF) and sample partial autocorrelation function (SPACF) are illustrated in Figure 21.2. Summary statistics for the simulated data are

[10]Recall that the S-PLUS function `arima.sim` reverses the sign of the moving average parameter ψ.

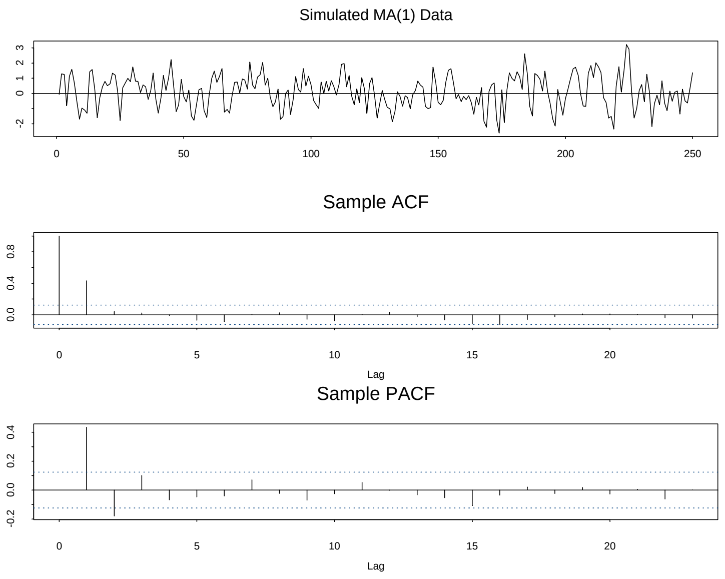


FIGURE 21.2. Simulated data, SACF and SPACF from MA(1) model with $\boldsymbol{\theta} = (0.5, 1)'$.

```
> summaryStats(ma1.sim)

Sample Quantiles:
    min      1Q median     3Q   max
 -2.606 -0.6466 0.1901 0.8755 3.221

Sample Moments:
   mean   std skewness kurtosis
 0.1126 1.071  -0.0624    2.634

Number of Observations:  250
```

An S-PLUS function to compute the moment conditions (21.38) is[11]

```
ma1.moments <- function(parm, data = NULL) {
  # parm = (mu,psi,sig2)'
  # data = (y(t),y(t)^2,y(t)*y(t-1),y(t)*y(t-2))
  m1 = parm[1]
  m2 = parm[1]^2 + parm[3]*(1 + parm[2]^2)
```

[11]In the function `ma1.moments`, the parameters are unrestricted. To force the moving average parameter to satisfy $|\psi| < 1$, use the logistic transformation $\psi = \exp(\gamma_1)/(1 + \exp(\gamma_1))$, and to force the variance parameter to be positive, use $\sigma^2 = \exp(\gamma_2)$.

```
  m3 = parm[1]^2 + parm[3]*parm[2]
  m4 = parm[1]^2
  t(t(data) - c(m1,m2,m3,m4))
}
```

The function `ma1.moments` has arguments `parm`, specifying the model parameters $\boldsymbol{\theta} = (\mu, \psi, \sigma^2)'$, and `data`, specifying an $(n-2) \times 4$ matrix with the tth row $\mathbf{w}_t = (y_t, y_t^2, yy_{t-1}, y_t y_{t-2})'$. The first five rows of $\mathbf{g}(\mathbf{w}_t, \boldsymbol{\theta}_0)$ are

```
> ma1.data = cbind(ma1.sim[3:nobs],ma1.sim[3:nobs]^2,
+                  ma1.sim[3:nobs]*ma1.sim[2:(nobs-1)],
+                  ma1.sim[3:nobs]*ma1.sim[1:(nobs-2)])
> start.vals = c(0,0.5,1)
> names(start.vals) = c("mu","psi","sig2")
> ma1.mom = ma1.moments(parm = start.vals, data = ma1.data)
> ma1.mom[1:5,]
         [,1]      [,2]     [,3]      [,4]
[1,]  1.24643  0.303579  1.10981 -0.071482
[2,] -0.80526 -0.601549 -1.50370 -1.040035
[3,]  1.13258  0.032744 -1.41203  1.411681
[4,]  1.58545  1.263659  1.29566 -1.276709
[5,]  0.67989 -0.787743  0.57794  0.770037
```

The sample average of $\mathbf{g}(\mathbf{w}_t, \boldsymbol{\theta}_0)$ is

```
> colMeans(ma1.mom)
[1]  0.10852675 -0.09134586  0.01198312  0.05919433
```

which is somewhat close to the population value $E[\mathbf{g}(\mathbf{w}_t, \boldsymbol{\theta}_0)] = \mathbf{0}$. The sample autocorrelations and cross-autocorrelations of $\mathbf{g}(\mathbf{w}_t, \boldsymbol{\theta}_0)$ are shown in Figure 21.3, which confirm the need for an HAC-type estimator for $\mathbf{S}$.

To estimate the MA(1) model by GMM with $\mathbf{S}$ estimated using a truncated (rectangular) kernel with bandwidth equal to one lag, use[12]:

```
> start.vals = c(0,0.5,1)
> names(start.vals) = c("mu","psi","sig2")
> ma1.gmm.trunc = GMM(start.vals, ma1.moments,
+        data = ma1.data,ts=T,
+        var.hac.control = var.hac.control(bandwidth = 1,
+        window = "truncated"))
1-step objective = 0.530132
2-step objective = 0.354946
3-step objective = 0.354926
```

The fitted results are

[12]Recall that $\hat{\mathbf{S}}$ computed with a truncated kernel is not guaranteed to be positive definite.

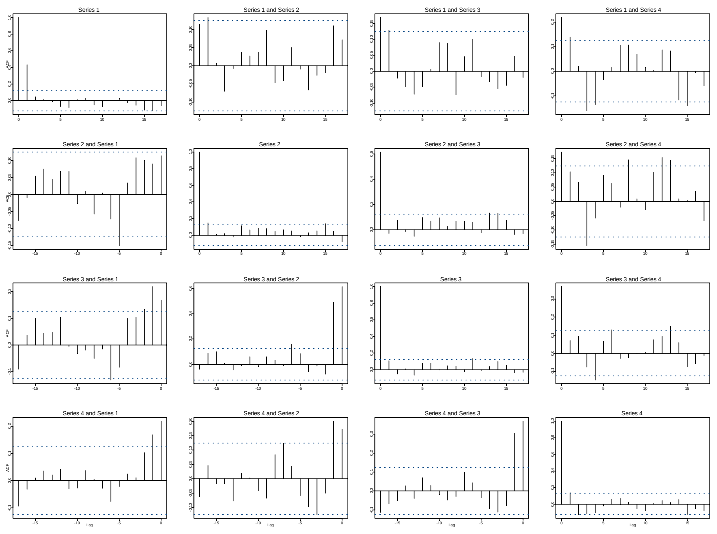


FIGURE 21.3. SACF of $\mathbf{g}(\mathbf{w}_t, \boldsymbol{\theta}_0)$ from the MA(1) model.

```
> summary(ma1.gmm.trunc)

Call:
GMM(start = start.vals, moments = ma1.moments, ts = T,
var.hac.control = var.hac.control(bandwidth = 1,
window = "truncated"), data = ma1.data)

Coefficients:
       Value Std.Error t value Pr(>|t|)
  mu  0.1018  0.0927    1.0980  0.2733
 psi  0.5471  0.0807    6.7788  0.0000
sig2  0.8671  0.0763   11.3581  0.0000

Test of Overidentification:
 J-stat Df P.value
 0.3549 1  0.5513

Optimization Info:
Number of Iterative Steps: 3
```

The GMM estimate of μ_0 is close to the sample mean, the estimate of ψ_0 is slightly larger than 0.5, and the estimate of σ_0^2 is slightly smaller than 1. The low J-statistic indicates a correctly specified model.

To illustrate the impact of model misspecification on the GMM estimation, consider fitting an MA(1) model by GMM to data simulated from an autoregression AR(1) model:

$$y_t - \mu_0 = \phi_0(y_{t-1} - \mu_0) + \varepsilon_t,\ \varepsilon_t \sim \text{iid } N(0, \sigma_0^2)$$

The AR(1) model has moments

$$\begin{aligned} E[y_t] &= \mu_0 \\ E[y_t^2] &= \mu_0^2 + \sigma_0^2/(1-\phi_0^2) \\ E[y_t y_{t-1}] &= \mu_0^2 + \phi\sigma_0^2/(1-\phi_0^2) \\ E[y_t y_{t-2}] &= \mu_0^2 + \phi^2\sigma_0^2/(1-\phi_0^2) \end{aligned}$$

Simulated AR(1) data with $\mu_0 = 0$, $\phi_0 = 0.5$, $\sigma_0^2 = 1$ and $n = 250$ is computed using the `S-PLUS` function `arima.sim`:

```
> set.seed(123)
> ar1.sim = arima.sim(model=list(ar=0.5),n=250)
```

The moment data required for GMM estimation of the MA(1) model are

```
> nobs = numRows(ar1.sim)
> ar1.data = cbind(ar1.sim[3:nobs],ar1.sim[3:nobs]^2,
+                  ar1.sim[3:nobs]*ar1.sim[2:(nobs-1)],
+                  ar1.sim[3:nobs]*ar1.sim[1:(nobs-2)])
```

The iterated efficient GMM estimates of the misspecified MA(1) model are

```
> ma1.gmm.ar1 = GMM(start.vals, ma1.moments,
+     data=ar1.data,ts = T,
+     var.hac.control = var.hac.control(bandwidth = 1,
+     window = "truncated"))
1-step objective = 25.6768
...
9-step objective = 13.0611
> summary(ma1.gmm.ar1, print.moments = T)

Call:
GMM(start = start.vals, moments = ma1.moments, ts = T,
var.hac.control = var.hac.control(bandwidth = 1,
window = "truncated"), data = ar1.data)

Coefficients:
        Value Std.Error  t value Pr(>|t|)
  mu  -0.0407   0.0909   -0.4478   0.6547
 psi   0.4199   0.0789    5.3207   0.0000
sig2   0.7840   0.0737   10.6391   0.0000
```

```
Test of Overidentification:
  J-stat Df P.value
 13.0611 1  0.0003

Optimization Info:
Number of Iterative Steps: 9

Normalized Moments:
           Moment Std.Error t value Pr(>|t|)
Moment 1   2.9166 0.8072    3.6131  0.0003
Moment 2   5.3807 1.4888    3.6140  0.0003
Moment 3   5.6389 1.5603    3.6140  0.0003
Moment 4   5.8329 1.6140    3.6140  0.0003
```

The GMM estimates of the MA(1) model parameters look reasonable. However, the large J-statistic correctly indicates a misspecified model. Setting the optional argument `print.moments=T` adds the normalized moment information to the summary output. Since the J-statistic is large, all of the normalized moments are significantly different from zero.

21.7.3 Euler Equation Asset Pricing Model

Following Hansen and Singleton (1982), a representative agent is assumed to choose an optimal consumption path by maximizing the present discounted value of lifetime utility from consumption

$$\max \sum_{t=1}^{\infty} E\left[\beta_0^t U(C_t)|I_t\right]$$

subject to the budget constraint

$$C_t + P_t Q_t \leq V_t Q_{t-1} + W_t$$

where I_t denotes the information available at time t, C_t denotes real consumption at t, W_t denotes real labor income at t, P_t denotes the price of a pure discount bond maturing at time $t+1$ that pays V_{t+1}, Q_t represents the quantity of bonds held at t, and β_0 represents a time discount factor. The first-order condition for the maximization problem may be represented as the conditional moment equation (Euler equation)

$$E\left[(1+R_{t+1})\beta_0 \frac{U'(C_{t+1})}{U'(C_t)}|I_t\right] - 1 = 0$$

where $1 + R_{t+1} = \frac{V_{t+1}}{V_t}$ is the gross return on the bond at time $t+1$. Assuming utility has the power form

$$U(C) = \frac{C^{1-\alpha_0}}{1-\alpha_0}$$

where α_0 represents the intertemporal rate of substitution (risk aversion), then

$$\frac{U'(C_{t+1})}{U'(C_t)} = \left(\frac{C_{t+1}}{C_t}\right)^{-\alpha_0}$$

and the conditional moment equation becomes

$$E\left[(1+R_{t+1})\beta_0\left(\frac{C_{t+1}}{C_t}\right)^{-\alpha_0}|I_t\right] - 1 = 0 \tag{21.39}$$

Define the nonlinear error term as

$$\begin{aligned}\varepsilon_{t+1} &= a(R_{t+1}, C_{t+1}/C_t; \alpha_0, \beta_0) = (1+R_{t+1})\beta_0\left(\frac{C_{t+1}}{C_t}\right)^{-\alpha_0} - 1 \\ &= a(\mathbf{z}_{t+1}, \boldsymbol{\theta}_0)\end{aligned}$$

with $\mathbf{z}_{t+1} = (R_{t+1}, C_{t+1}/C_t)'$ and $\boldsymbol{\theta}_0 = (\alpha_0, \beta_0)'$. Then, the conditional moment equation (21.39) may be represented as

$$E[\varepsilon_{t+1}|I_t] = E[a(\mathbf{z}_{t+1}, \boldsymbol{\theta}_0)|I_t] = 0$$

Since $\{\varepsilon_{t+1}, I_{t+1}\}$ is a MDS, potential instruments $\mathbf{x}_t$ include current and lagged values of the elements in $\mathbf{z}_t$ as well as a constant. For example, one could use

$$\mathbf{x}_t = (1, C_t/C_{t-1}, C_{t-1}/C_{t-2}, R_t, R_{t-1})'$$

Since $\mathbf{x}_t \subset I_t$, the conditional moment (21.39) implies that

$$E[\mathbf{x}_t\varepsilon_{t+1}|I_t] = E[\mathbf{x}_t a(\mathbf{z}_{t+1}, \boldsymbol{\theta}_0)|I_t] = \mathbf{0}$$

and by the law of total expectations,

$$E[\mathbf{x}_t\varepsilon_{t+1}] = \mathbf{0}$$

For the GMM estimation, define the nonlinear residual as

$$e_{t+1} = (1+R_{t+1})\beta\left(\frac{C_{t+1}}{C_t}\right)^{-\alpha} - 1$$

and form the vector of moments

$$\begin{aligned}g(\mathbf{w}_{t+1}, \boldsymbol{\theta}) &= \mathbf{x}_t e_{t+1} = \mathbf{x}_t a(\mathbf{z}_{t+1}, \boldsymbol{\theta}) \\ &= \begin{pmatrix} (1+R_{t+1})\beta\left(\frac{C_{t+1}}{C_t}\right)^{-\alpha} - 1 \\ (C_t/C_{t-1})\left((1+R_{t+1})\beta\left(\frac{C_{t+1}}{C_t}\right)^{-\alpha} - 1\right) \\ (C_{t-1}/C_{t-2})\left((1+R_{t+1})\beta\left(\frac{C_{t+1}}{C_t}\right)^{-\alpha} - 1\right) \\ R_t\left((1+R_{t+1})\beta\left(\frac{C_{t+1}}{C_t}\right)^{-\alpha} - 1\right) \\ R_{t-1}\left((1+R_{t+1})\beta\left(\frac{C_{t+1}}{C_t}\right)^{-\alpha} - 1\right) \end{pmatrix}\end{aligned} \tag{21.40}$$

In (21.40), there are $K = 5$ moment conditions to identify $L = 2$ model parameters giving $K - L = 3$ overidentifying restrictions. The GMM objective function is

$$J(\boldsymbol{\theta}, \hat{\mathbf{S}}^{-1}) = (n-2) \cdot g_n(\boldsymbol{\theta})' \hat{\mathbf{S}}^{-1} g_n(\boldsymbol{\theta})$$

where $\hat{\mathbf{S}}$ is a consistent estimate of $\mathbf{S} = \text{avar}(\bar{\mathbf{g}})$.

An S-PLUS function to compute the Euler moments (21.40) is

```
euler1.moments <- function(parm, data = NULL) {
   # parm = (beta,gamma)
   # data = (1+R(t+1),C(t+1)/C(t),1,C(t)/C(t-1),
   #         C(t-1)/C(t-2),R(t),R(t-1),...)
   ncol = numCols(data)
   euler = data[,1]*parm[1]*(data[,2]^(-parm[2])) - 1
   as.matrix(rep(euler,(ncol-2))*data[,3:ncol])
}
```

The function `euler1.moments` has arguments `parm`, specifying the model parameters $\boldsymbol{\theta} = (\beta, \alpha)'$, and `data`, specifying an $(n-2) \times 7$ matrix with the tth row $\mathbf{w}_{t+1} = (1 + R_{t+1}, C_{t+1}/C_t, 1, C_t/C_{t-1}, C_{t-1}/C_{t-2}, R_t, R_{t-1})'$.

Verbeek (2000, Chap. 5, Sec. 7) describes an extension of the model presented above that allows the individual to invest in J risky assets with returns $R_{j,t+1}$ ($j = 1, \ldots, J$), as well as a risk-free asset with certain return $R_{f,t+1}$. Assuming power utility and restricting attention to unconditional moments (i.e., using $\mathbf{x}_t = 1$), the Euler equations may be written as

$$\begin{aligned} E\left[(1 + R_{f,t+1})\beta_0 \left(\frac{C_{t+1}}{C_t}\right)^{-\alpha_0}\right] - 1 &= 0 \\ E\left[(R_{j,t+1} - R_{f,t+1})\beta_0 \left(\frac{C_{t+1}}{C_t}\right)^{-\alpha_0}\right] &= 0, \; j = 1, \ldots, J \end{aligned} \qquad (21.41)$$

Define the stochastic discount factor as

$$m_{t+1}(\boldsymbol{\theta}_0) = \beta_0 \left(\frac{C_{t+1}}{C_t}\right)^{-\alpha_0}$$

so that the risky assets satisfy

$$E[(R_{j,t+1} - R_{f,t+1}) \cdot m_{t+1}(\boldsymbol{\theta}_0)] = 0$$

Using the identity $E[xy] = \text{cov}(x, y) + E[x]E[y]$, the risk premium for the risky assets may be deduced as

$$E[R_{j,t+1} - R_{f,t+1}] = -\frac{\text{cov}(m_{t+1}(\boldsymbol{\theta}_0), R_{j,t+1} - R_{f,t+1})}{E[m_{t+1}(\boldsymbol{\theta}_0)]} \qquad (21.42)$$

If the asset pricing model is correct, then the right-hand side of (21.42) should explain the cross-sectional variation of expected returns across assets.

For the GMM estimation, define the $J+1$ vector of moments

$$g(\mathbf{w}_{t+1},\boldsymbol{\theta}) = \begin{pmatrix} (1+R_{f,t+1})\beta\left(\frac{C_{t+1}}{C_t}\right)^{-\alpha} - 1 \\ (R_{1,t+1}-R_{f,t+1})\beta\left(\frac{C_{t+1}}{C_t}\right)^{-\alpha} \\ \vdots \\ (R_{J,t+1}-R_{f,t+1})\beta\left(\frac{C_{t+1}}{C_t}\right)^{-\alpha} \end{pmatrix}$$

An S-PLUS function to compute the above moments is

```
euler2.moments <- function(parm, data = NULL) {
   # parm = (beta,gamma)
   # data = (C(t+1)/C(t),1+Rf(t+1),R1(t+1)-Rf(t+1),...,
   #         RJ(t+1)-Rf(t+1))
   ncol = numCols(data)
   sdf = parm[1]*data[,1]^(-parm[2])
   d1 = (sdf - 1)*data[,2]
   d2 = as.matrix(rep(sdf,(ncol-2))*data[,3:ncol])
   cbind(d1,d2)
}
```

The function `euler2.moments` has arguments `parm`, specifying the model parameters $\boldsymbol{\theta} = (\beta,\alpha)'$, and `data`, specifying an $n \times (J+1)$ matrix with the tth row $\mathbf{w}_{t+1} = (C_{t+1}/C_t, 1+R_{f,t+1}, R_{1,t+1}-R_{f,t+1},\ldots,R_{J,t+1}-R_{f,t+1})'$.

Following Verbeek (2000), the power utility asset pricing model is estimated using monthly data over the period February 1959 through November 1993.[13] Ten size-based portfolios from the Center for Research in Security Prices (CRSP) data base are used as the risky assets. That is, portfolio 1 contains the monthly returns on the smallest 10% of firms (by market capitalization) listed on the New York Stock Exchange, and portfolio 10 contains the returns on the largest 10% of firms. The risk-free asset is the monthly return on 3-month U.S. T-bills, and real consumption is measured by total U.S. personal consumption expenditures on nondurables and services. The data are available in the S+FinMetrics "timeSeries" object `pricing.ts`, which has variables

```
> colIds(pricing.ts)
 [1] "CONS" "R1"   "R2"   "R3"   "R4"   "R5"   "R6"   "R7"
 [9] "R8"   "R9"   "R10"  "RF"
```

The data to be passed to the function `euler2.moments` are constructed using

[13]The data are taken from Manro Verbeek's web page at `http://www.econ.kuleuven.ac.be/GME`.

```
> pricing.mat = as.matrix(seriesData(pricing.ts))
> ncol = numCols(pricing.mat)
> excessRet.mat = apply(pricing.mat[,2:(ncol-1)], 2,
+                       function(x,y){x-y},
+                       pricing.mat[,"RF"])
# data = (C(t+1)/C(t),1+Rf(t+1),R1(t+1)-Rf(t+1),...,
#         RJ(t+1)-Rf(t+1))
> euler.data = cbind(pricing.mat[,"CONS"],
+                    1 + pricing.mat[,"RF"],
+                    excessRet.mat)
```

The iterated efficient GMM estimator may be computed using

```
> start.vals = c(1,5)
> names(start.vals) = c("beta","alpha")
> euler.gmm.fit = GMM(start.vals, euler2.moments,
+                     method = "iterative",data = euler.data)
> summary(euler.gmm.fit)

Call:
GMM(start = start.vals, moments = euler2.moments, method =
"iterative", data = euler.data)

Coefficients:
        Value Std.Error t value Pr(>|t|)
 beta  0.8314  0.1170    7.1049  0.0000
alpha 57.4020 34.3021    1.6734  0.0950

Test of Overidentification:
 J-stat Df P.value
 5.6574 9  0.7737

Optimization Info:
Number of Iterative Steps: 9
```

The small J-statistic indicates a correctly specified model. The estimate of the time discount parameter β is economically reasonable and fairly precise, but the estimate of the risk-aversion parameter α is implausibly large and imprecise. The large estimate of α illustrates the well-known equity premium puzzle under which the high risk premia on equity assets is associated with extremely risk-averse investors.

The one-step inefficient GMM estimator with $\mathbf{W} = \mathbf{I}_{11}$ may be computed using

```
> euler.gmm.fit2 = GMM(start.vals, euler2.moments,
+                      method = "iterative", max.steps = 0,
+                      w = diag(11), w0.efficient = F,
+                      data = euler.data)
> summary(euler.gmm.fit2)

Call:
GMM(start = start.vals, moments = euler2.moments, method =
"iterative", w = diag(11), max.steps = 0, w0.efficient
= F, data = euler.data)

Coefficients:
        Value Std.Error t value Pr(>|t|)
 beta  0.7031  0.1446    4.8612  0.0000
alpha 91.4097 38.2102    2.3923  0.0172
```

The usual J-statistic is not valid for the one-step estimates so it is not reported. The one-step estimates are similar to the iterated efficient GMM estimates and have slightly larger estimated standard errors.

Cochrane (1996, 2001) recommended using the one-step GMM estimates to compute empirical pricing errors based on (21.42), since these estimates minimize such pricing errors by construction. The pricing errors can then be plotted to evaluate the economic significance of the proposed asset pricing model. The empirical pricing errors may be computed and plotted as follows. First, the historical average excess returns on the 10 portfolios are computed using

```
> excessRet.hat = colMeans(excessRet.mat)
```

Next, the average excess returns on the 10 portfolios predicted by the model, based on the one-step inefficient GMM estimates, may be computed using the function

```
predRet <- function(parm,data) {
   # parm = (beta,gamma)
   # data = (C(t+1)/C(t),1+Rf(t+1),R1(t+1)-Rf(t+1),...,
   #         RJ(t+1)-Rf(t+1))
   ncol = numCols(data)
   sdf = parm[1]*data[,1]^(-parm[2])
   tmp = function(x,y) { -var(x,y)/mean(y) }
   ans = apply(data[,3:ncol], 2, FUN = tmp,sdf)
   ans
}
> predRet.hat = predRet(coef(euler.gmm.fit2),euler.data)
```

Figure 21.4 shows the historical average excess returns and the predicted average excess returns along with a 45° line. If the model is correct, then all points should lie on the 45° line. The plot shows that the model underpredicts the small firm excess returns and overpredicts the large firm excess returns.

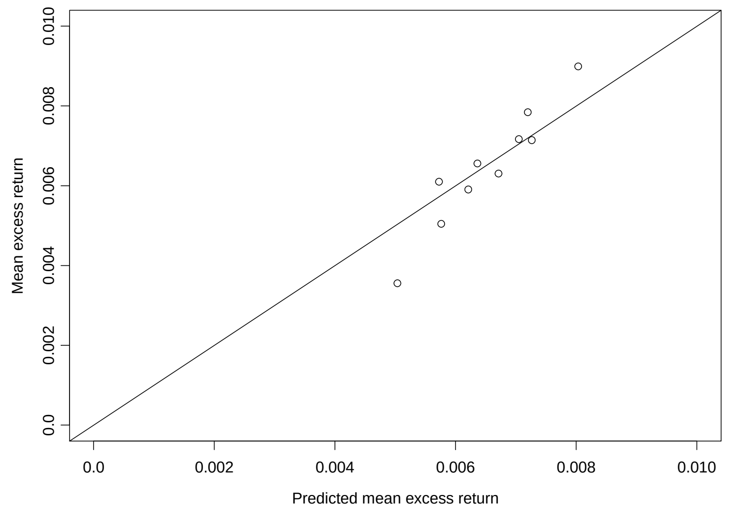


FIGURE 21.4. Monthly actual versus predicted mean excess returns of size-based portfolios.

21.7.4 Stochastic Volatility Model

The simple log-normal stochastic volatility (SV) model, due to Taylor (1986), is given by

$$\begin{aligned} y_t &= \sigma_t Z_t,\ t = 1, \ldots, n \qquad (21.43) \\ \ln \sigma_t^2 &= \omega_0 + \beta_0 \ln \sigma_{t-1}^2 + \sigma_{0,u} u_t \\ (Z_t, u_t)' &\sim \text{iid } N(\mathbf{0}, \mathbf{I}_2) \\ \boldsymbol{\theta}_0 &= (\omega_0, \beta_0, \sigma_{0,u})' \end{aligned}$$

For $0 < \beta_0 < 1$ and $\sigma_{0,u} \geq 0$, the series y_t is strictly stationary and ergodic, and unconditional moments of all orders exist. In the SV model, the series y_t is serially uncorrelated but dependency in the higher-order moments is induced by the serially correlated stochastic volatility term $\ln \sigma_t^2$. Shephard (1996) surveyed the use of autoregressive conditional heteroskedasticity (ARCH) and SV models in finance and noted some important advantages of SV models over ARCH models. In particular, their statistical properties are easier to find and understand, they generalize more easily to multivariate settings, and they have simpler continuous-time representations that can be used in contingent claims pricing.

Simulated data from (21.43) with $\omega_0 = -0.736$, $\beta_0 = 0.90$, $\sigma_{0,u} = 0.363$, and $n = 1000$ is illustrated in Figure 21.5. The data exhibit ARCH-like

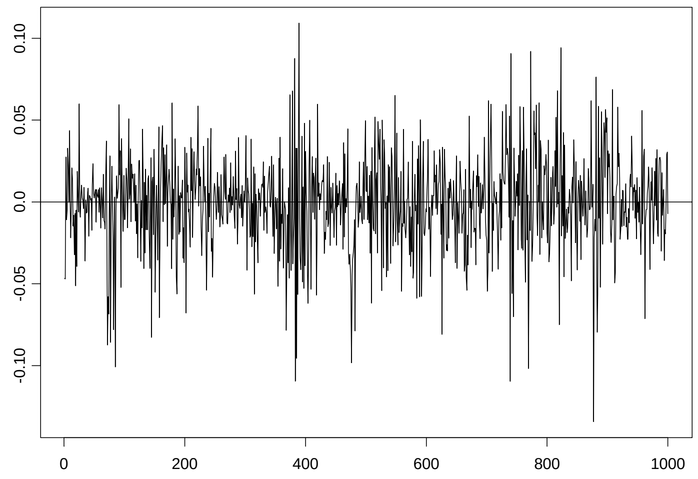


FIGURE 21.5. Simulated data from SV model (21.43) with $\omega_0 = -0.736, \beta_0 = 0.90$, and $\sigma_{0,u} = 0.363$.

features such as volatility clustering and fat tails. The data is created with the function `sv.as.gensim`, described in Chapter 22 on EMM estimation, using

```
# beta.t = log(beta/(1-beta))
> n.sim = 1000
> n.burn = 100
> nz = n.sim + n.burn
> set.seed(456)
> sv.as.aux = list(z = rnorm(nz),u = rnorm(nz))
> rho.as = c(-0.736,2.197225,0.363)
> names(rho.as) = c("alpha","beta.t","sigma.u")
> sv.as.sim = sv.as.gensim(rho = rho.as, n.sim = n.sim,
+             n.burn = n.burn, aux = sv.as.aux)
```

The GMM estimation of the SV model is surveyed in Andersen and Sorensen (1996). They recommended using moment conditions for GMM estimation based on lower-order moments of y_t, since higher-order moments tend to exhibit erratic finite sample behavior. They considered a GMM estimation based on (subsets) of 24 moments considered by Jacquier, Polson, and Rossi (1994). To describe these moment conditions, first define

$$\mu = \frac{\omega}{1-\beta}, \ \sigma^2 = \frac{\sigma_u^2}{1-\beta^2}$$

Then, the moment conditions, which follow from properties of the log-normal distribution and the Gaussian AR(1) model, are expressed as

$$\begin{aligned}
E[|y_t|] &= (2/\pi)^{1/2}E[\sigma_t] \\
E[y_t^2] &= E[\sigma_t^2] \\
E[|y_t^3|] &= 2\sqrt{2/\pi}E[\sigma_t^3] \\
E[y_t^4] &= 3E[\sigma_t^4] \\
E[|y_t y_{t-j}|] &= (2/\pi)E[\sigma_t\sigma_{t-j}], \ j = 1, \ldots, 10 \\
E[y_t^2 y_{t-j}^2] &= E[\sigma_t^2\sigma_{t-j}^2], \ j = 1, \ldots, 10
\end{aligned}$$

where for any positive integer j and positive constants r and s,

$$\begin{aligned}
E[\sigma_t^r] &= \exp\left(\frac{r\mu}{2} + \frac{r^2\sigma^2}{8}\right) \\
E[\sigma_t^r\sigma_{t-j}^s] &= E[\sigma_t^r]E[\sigma_t^s]\exp\left(\frac{rs\beta^j\sigma^2}{4}\right)
\end{aligned}$$

Let

$$\mathbf{w}_t = (|y_t|, y_t^2, |y_t^3|, y_t^4, |y_t y_{t-1}|, \ldots, |y_t y_{t-10}|, y_t^2 y_{t-1}^2, \ldots, y_t^2 y_{t-10}^2)'$$

and define the 24×1 vector

$$g(\mathbf{w}_t, \boldsymbol{\theta}) = \begin{pmatrix} |y_t| - (2/\pi)^{1/2}\exp\left(\frac{\mu}{2} + \frac{\sigma^2}{8}\right) \\ y_t^2 - \exp\left(\mu + \frac{\sigma^2}{2}\right) \\ \vdots \\ y_t^2 y_{t-10}^2 - \exp\left(\mu + \frac{\sigma^2}{2}\right)^2 \exp\left(\beta^{10}\sigma^2\right) \end{pmatrix} \tag{21.44}$$

Then, $E[g(\mathbf{w}_t, \boldsymbol{\theta}_0)] = \mathbf{0}$ is the population moment condition used for the GMM estimation of the model parameters $\boldsymbol{\theta}_0$. Since the elements of $\mathbf{w}_t$ are serially correlated, the efficient weight matrix $\mathbf{S} = \text{avar}(\bar{\mathbf{g}})$ must be estimated using an HAC estimator.

An `S-PLUS` function to compute the moment conditions (21.44) for $t = 1, \ldots, n$ is

```
sv.moments = function(parm, data = NULL)
{
  omega = parm[1]
  beta = parm[2]
  sigu = parm[3]
  mu = omega/(1-beta)
  sig2 = (sigu*sigu)/(1-beta*beta)
#
  E.sigma = c(sqrt(2/pi) * exp(mu/2 + sig2/8),
            exp(mu + sig2/2),
            2 * sqrt(2/pi) * exp(3*mu/2 + 9*sig2/8),
            3 * exp(2*mu + 2*sig2))
  E.sigma.c = c(2/pi * exp(2*(mu/2 + sig2/8)
              + beta^(1:10) * sig2/4),
              exp(2*(mu + sig2/2)
              + 4 * beta^(1:10) * sig2/4))
#
  t(t(data) - c(E.sigma, E.sigma.c))
}
```

The transformed simulated data to be passed to the function `sv.moments` for GMM estimation are created using

```
> sv.pow = cbind(abs(sv.as.sim), sv.as.sim^2,
+                abs(sv.as.sim)^3, sv.as.sim^4)
> sv.pow = sv.pow[-(1:10),]
> sv.cm = tslag(sv.as.sim, 1:10, trim=T) *
+               as.vector(sv.as.sim[-(1:10)])
> sv.data = cbind(sv.pow, abs(sv.cm), sv.cm^2)
```

The iterated efficient GMM estimator based on (21.44) may be computed using

```
> start.vals = c(0,0.5,0.5)
> names(start.vals) = c("omega","beta","sigu")
> sv.fit.1 = GMM(start.vals, sv.moments, method = "iterative",
+                ts = T, data = sv.data)
> summary(sv.fit.1)

Call:
GMM(start = start.vals, moments = sv.moments, method =
"iterative", ts = T, data = sv.data)

Coefficients:
         Value Std.Error  t value Pr(>|t|)
omega  -0.4628   0.3731   -1.2405   0.2151
 beta   0.9378   0.0502   18.6959   0.0000
 sigu   0.2233   0.0978    2.2841   0.0226

Test of Overidentification:
  J-stat Df P.value
 18.8761 21 0.5931

Optimization Info:
Number of Iterative Steps: 7
```

The high p-value for the J-statistic indicates that the 24 moment conditions are not rejected by the data. Consistent with the findings of Andersen and Sorensen, ω_0 is not estimated very precisely, whereas β_0 is estimated fairly precisely.

Now, consider estimation of the SV model (21.43) using the daily returns on the S&P 500 index over the period March 14, 1986 through June 30, 2003:

```
> SP500.ts = sp500.ts
> SP500.ts@data = as.matrix(seriesData(sp500.ts))
>
> SP500.pow = seriesMerge(abs(SP500.ts), SP500.ts^2,
+                         abs(SP500.ts)^3, SP500.ts^4)
> SP500.pow = SP500.pow[-(1:10),]@data
> SP500.cm = tslag(SP500.ts, 1:10, trim=T)@data *
+            as.vector(SP500.ts[-(1:10)]@data)
> SP500.data = cbind(SP500.pow, abs(SP500.cm), SP500.cm^2)
> colIds(SP500.data) = NULL
>
> # iterative GMM estimation of the SV model
> start.vals = c(0,0.5,0.5)
> names(start.vals) = c("omega","beta","sigu")
> sv.fit.SP500 = GMM(start.vals, sv.moments,
+                    method = "iterative", ts = T,
+                    data = SP500.data)
1-step objective = 2.10165e-7
...
9-step objective = 39.9342
> summary(sv.fit.SP500,print.moments=F)

Call:
GMM(start = start.vals, moments = sv.moments, method =
"iterative", ts = T, data = SP500.data)

Coefficients:
         Value Std.Error  t value Pr(>|t|)
omega  -0.1541   0.1758   -0.8767   0.3807
 beta   0.9838   0.0184   53.3692   0.0000
 sigu   0.1548   0.0908    1.7041   0.0884
```

```
Test of Overidentification:
  J-stat Df P.value
 39.9342 21 0.0076


Optimization Info:
Number of Iterative Steps: 9
```

The low p-value on the J-statistic indicates that the SV model (21.43) does not fit S&P 500 daily returns.

21.7.5 Interest Rate Diffusion Model

Chan, Karolyi, Longstaff, and Sanders (1992), hereafter CKLS, considered estimating the parameters of the continuous-time interest rate diffusion model

$$\begin{aligned} dr_t &= (\alpha_0 + \beta_0 r_t)\ dt + \sigma_0 r_t^{\gamma_0}\ dW_t \qquad (21.45)\\ \boldsymbol{\theta}_0 &= (\alpha_0, \beta_0, \sigma_0, \gamma_0)' \end{aligned}$$

using GMM. In (23.26), the drift function $(\alpha_0+\beta_0 r_t)dt$ may be reparameterized as $\kappa_0(\mu_0 - r_t)dt$, where $\alpha_0 = \kappa_0\mu_0$ and $\beta_0 = -\kappa_0$. The parameter μ_0 is the long-run mean, and the parameter κ_0 determines the speed of mean reversion. The model (23.26) encompasses a variety of models that have been proposed for the short-term interest rate. Simulated solutions from (23.26) based on Euler's method for various models using parameters estimated by CKLS are presented in Chapter 19.

CKLS derived moment conditions for GMM estimation of $\boldsymbol{\theta}_0$ from the Euler discretization

$$\begin{aligned} r_{t+\Delta t} - r_t &= (\alpha_0 + \beta_0 r_t)\Delta t + \sigma_0 r_t^{\gamma_0}\sqrt{\Delta t} z_{t+\Delta t}\\ E[z_{t+\Delta t}] &= 0,\ E[z_{t+\Delta t}^2] = 1 \end{aligned}$$

They defined the true model error as

$$\begin{aligned} \varepsilon_{t+\Delta t} &= a(r_{t+\Delta t} - r_t, r_t; \alpha_0, \beta_0, \sigma_0, \gamma_0) = (r_{t+\Delta t} - r_t) - (\alpha_0 + \beta_0 r_t)\Delta t\\ &= a(\mathbf{z}_{t+\Delta t}, \boldsymbol{\theta}_0) \end{aligned}$$

where $\mathbf{z}_{t+\Delta t} = (r_{t+\Delta t} - r_t, r_t)'$. Letting I_t represent information available at time t, the true error satisfies $E[\varepsilon_{t+\Delta t}|I_t] = 0$. Since $\{\varepsilon_{t+\Delta t}, I_{t+\Delta}\}$ is a MDS, potential instruments $\mathbf{x}_t$ include current and lagged values of the elements of $\mathbf{z}_t$ as well as a constant. As the basis for the GMM estimation, CKLS used $\mathbf{x}_t = (1, r_t)'$ as the instrument vector and deduced the following four conditional moments:

$$\begin{aligned} E[\varepsilon_{t+\Delta t}|I_t] &= 0,\ E[\varepsilon_{t+\Delta t}^2|I_t] = \sigma_0^2 r_t^{2\gamma_0}\Delta t\\ E[\varepsilon_{t+\Delta t} r_t|I_t] &= 0,\ E[\varepsilon_{t+\Delta t}^2 r_t|I_t] = \sigma_0^2 r_t^{2\gamma_0}\Delta t \cdot r_t \end{aligned}$$

For given values of α and β, define the nonlinear residual

$$e_{t+\Delta t} = (r_{t+\Delta t} - r_t) - (\alpha + \beta r_t)\Delta t$$

and for $\mathbf{w}_{t+\Delta t} = (r_{t+\Delta t} - r_t, r_t, r_t^2)'$, define the 4×1 vector of moments

$$g(\mathbf{w}_{t+\Delta t}, \boldsymbol{\theta}) = \begin{pmatrix} e_{t+\Delta t} \\ e_{t+\Delta t}^2 \end{pmatrix} \otimes \mathbf{x}_t = \begin{pmatrix} e_{t+\Delta t} \\ e_{t+\Delta t} r_t \\ e_{t+\Delta t}^2 - \sigma^2 r_t^{2\gamma} \Delta t \\ \left(e_{t+\Delta t}^2 - \sigma^2 r_t^{2\gamma} \Delta t\right) r_t \end{pmatrix}$$

Then, $E[g(\mathbf{w}_{t+\Delta t}, \boldsymbol{\theta}_0)] = \mathbf{0}$ gives the GMM estimating equation. Even though $\{\varepsilon_t, I_t\}$ is a MDS, the moment vector $g(\mathbf{w}_t, \boldsymbol{\theta}_0)$ is likely to be autocorrelated since it contains ε_t^2. For the most general specification (23.26), $K = L$ and the model is just identified. For all submodels of (23.26) listed in Table 20.3 from Chapter 19, $K > L$, so that they are overidentified.

An S-PLUS function to compute the CKLS moments is

```
ckls.moments <- function(parm, data = NULL,dt = 1/12) {
   # parm = (alpha,beta,sigma,gamma)'
   # data = [r(t+dt)-r(t),r(t)]
   # dt = discretization step
   e.hat = as.vector(data[,1] -
          (parm[1] + parm[2]*data[,2])*dt)
   m2 = e.hat*as.vector(data[,2])
   m3 = e.hat^2 - dt*parm[3]*parm[3]*
        (as.vector(data[,2])^(2*parm[4]))
   m4 = m3*data[,2]
   cbind(e.hat,m2,m3,m4)
}
```

The function `ckls.moments` has arguments `parm`, specifying the model parameters $\boldsymbol{\theta} = (\alpha, \beta, \sigma, \gamma)'$, `data`, specifying an $n \times 2$ matrix with tth row $(r_{t+\Delta t} - r_t, r_t)$, and `dt`, specifying the discretization increment.

The GMM estimation is performed on (23.26) using monthly observations on the U.S. 30-day T-bill rate over the period June 1964 through November 1989. The same data are analyzed in Cliff (2003), and are illustrated in Figure 21.6.

The data to be passed to the function `ckls.moments` are constructed using

```
> data.ckls.ts = seriesMerge(diff(ckls.ts), tslag(ckls.ts))
> colIds(data.ckls.ts)[1] = "RF.diff"
> data.ckls = as.matrix(seriesData(data.ckls.ts))
```

The summary statistics for Δr_t and r_{t-1}

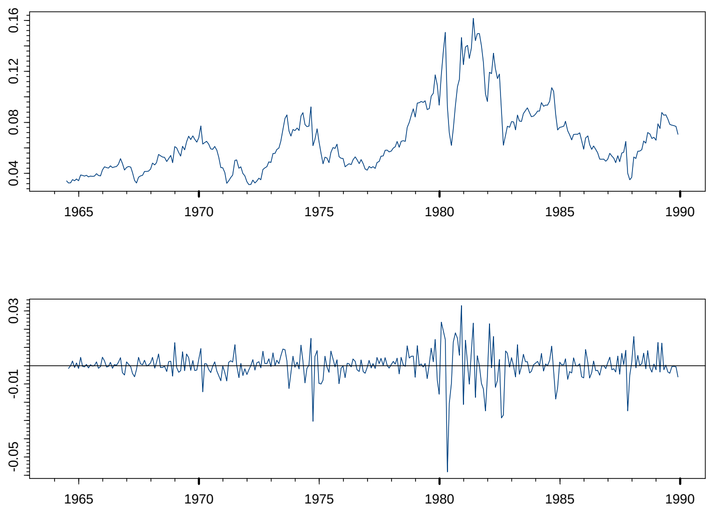


FIGURE 21.6. Monthly observations on the U.S. 30-day T-bill rate and its first difference.

```
> summaryStats(data.ckls)

Sample Quantiles:
              min       1Q median      3Q     max
RF.diff -0.05813 -0.00267 0.0002 0.00324 0.03285
RF.lag1  0.03127  0.04756 0.0603 0.07883 0.16150

Sample Moments:
            mean      std skewness kurtosis
RF.diff 0.00012 0.008187   -1.347   13.354
RF.lag1 0.06659 0.026375    1.206    4.332

Number of Observations:  305
```

are very close to those presented in Table II of CKLS (1992).

First, consider GMM estimation of the most general model (23.26). Sensible starting values for α and σ may be determined using the following approximations:

$$\begin{aligned}\mu &= \frac{\alpha}{-\beta} \approx E[r_t] \\ \sigma^2 &\approx \frac{\text{var}(r_{t+\Delta t} - r_t)}{\text{var}(r_t)\Delta t}\end{aligned}$$

The GMM estimation of $\boldsymbol{\theta}$ may be performed using

```
> start.vals = c(0.06,-0.5,1,1)
> names(start.vals) = c("alpha","beta","sigma","gamma")
> gmm.ckls = GMM(start.vals,ckls.moments,ts = T,
+                data = data.ckls,dt = 1/12)
> summary(gmm.ckls)

Call:
GMM(start = start.vals, moments = ckls.moments, ts = T, data
= data.ckls, dt = 1/12)

Coefficients:
        Value Std.Error t value Pr(>|t|)
alpha  0.0419  0.0193    2.1710  0.0307
 beta -0.6077  0.3454   -1.7592  0.0796
sigma  1.3337  1.0004    1.3331  0.1835
gamma  1.5081  0.2900    5.2010  0.0000

Test of Overidentification:
model is just-identified

Optimization Info:
 Number of Iterations: 14
 Convergence: absolute function convergence
```

In the call to `GMM`, the parameter `dt` is set to 1/12 because the data are annual rates sampled monthly. Since the model is just identified, the J-statistic is zero and the estimates do not depend on a weight matrix. The results are similar to those obtained by CKLS. In particular, the estimate of γ is roughly 1.5 and is highly significant. The estimates of the long-run mean and speed of adjustment are

```
> theta.hat = coef(gmm.ckls)
> theta.hat["alpha"]/-theta.hat["beta"]
      alpha
 0.06895576
> -theta.hat["beta"]
      beta
 0.6076583
```

The GMM estimates for the restricted models in Table 3 of Chapter 20 may be computed in a similar fashion by modifying the function `ckls.moments`. For example, to estimate the CIR SR model, use

```
cir.moments <- function(parm, data = NULL, dt = 1/12) {
   # parm = (alpha,beta,sigma)'
   # data = [r(t+dt)-r(t),r(t)]
   # dt = discretization step
   e.hat = as.vector(data[,1] -
          (parm[1] + parm[2]*data[,2])*dt)
   m2 = e.hat*as.vector(data[,2])
   m3 = e.hat^2 - dt*parm[3]*parm[3]*
        (as.vector(data[,2]))
   m4 = m3*data[,2]
   cbind(e.hat,m2,m3,m4)
}
> start.vals = c(0.06,-0.5,1)
> names(start.vals) = c("alpha","beta","sigma")
> gmm.cir = GMM(start.vals, cir.moments, ts = T,
+               data = data.ckls, dt = 1/12)

> summary(gmm.cir,print.moments = F)

Call:
GMM(start = start.vals, moments = cir.moments, ts = T, data =
data.ckls, dt = 1/12)

Coefficients:
         Value Std.Error  t value Pr(>|t|)
alpha   0.0204   0.0166    1.2258   0.2212
 beta  -0.2407   0.3023   -0.7963   0.4265
sigma   0.0841   0.0066   12.7661   0.0000

Test of Overidentification:
 J-stat Df P.value
 3.7977 1  0.0513

Optimization Info:
Number of Iterative Steps: 13
```

The low p-value on the J-statistic indicates a potentially misspecified model.

21.8 References

Andersen, T.G. and B.E. Sorensen (1996). "GMM Estimation of a Stochastic Volatility Model: A Monte Carlo Study," *Journal of Business and Economic Statistics*, 14, 328-352.

Andrews, D.W. (1991). "Heteroskedasticity and Autocorrelation Consistent Covariance Matrix Estimation," *Econometrica*, 59, 817-858.

Andrews, D.W. and C.J. Monahan (1992). "An Improved Heteroskedasticity and Autocorrelation Consistent Covariance Matrix Estimator," *Econometrica*, 60, 953-966.

Campbell, J.Y., A.W. Lo and A.C. MacKinlay (1997). *The Econometrics of Financial Markets.* Princeton University Press, Princeton, NJ.

Campbell, J. and G. Mankiw (1990). "Permanent Income, Current Income and Consumption," *Journal of Business and Economic Statistics*, 8, 265-279.

Chan, K.C., G.A. Karolyi, F.A. Longstaff and A.B. Sanders (1992). "An Empirical Comparison of Alternative Models of the Term Structure of Interest Rates", *Journal of Finance*, 47, 1209-1227.

Cliff, M.T. (2003). "GMM and MINZ Program Libraries for MATLAB," unpublished manuscript, Krannert Graduate School of Management, Purdue University.

Cochrane, J.H. (1996). "A Cross-Sectional Test of an Investment-Based Asset Pricing Model," *Journal of Political Economy*, 104, 572-561.

Cochrane, J.H. (2001). *Asset Pricing.* Princeton University Press, Princeton, NJ.

Davidson, R. and J. MacKinnon (2004). *Econometric Theory and Methods.* Oxford University Press, Oxford.

den Haan, W.J. and A. Levin (1997). "A Practioner's Guide to Robust Covariance Matrix Estimation," in G. Maddala and C. Rao (eds.), *Handbook of Statistics, Volume 15*, Elsevier, Amsterdam, pp. 309-327.

Ferson, W.E. (1995). "Theory and Empirical Testing of Asset Pricing Models," in R.A. Jarrow, V. Maksimovic and W.T. Ziemba (eds.), *Handbooks in OR & MS, Volume 9, Finance.* Elsevier Science B.V., Amsterdam.

Greene, W.H. (1994). *Econometric Analysis.* Prentice Hall, Englewood Cliffs, NJ.

Hall, R.E. (1978). "Stochastic Implications of the Life Cycle-Permanent Income Hypothesis: Theory and Evidence," *Journal of Political Economy*, 86(6), 971-987.

Hall, A. (2005). *Generalized Method of Moments.* Oxford University Press, Oxford.

Hamilton, J.D. (1994). *Time Series Analysis.* Princeton University Press, Princeton, NJ.

Hansen, L.P. (1982). "Large Sample Properties of Generalized Method of Moments Estimators," *Econometrica*, 50, 1029-1054.

Hansen, L.P. and K.J. Singleton (1982). "Generalized Instrumental Variables Estimation of Nonlinear Rational Expectations Models," *Econometrica*, 50, 1269-1286.

Hansen, L.P., J. Heaton and A. Yaron (1996). "Finite-Sample Properties of Some Alternative GMM Estimators," *Journal of Business and Economic Statistics*, 14, 262-280.

Harris, D. (1999). "GMM Estimation of Time Series Models," in L. Mátyás (ed.), *Generalized Method of Moments*, Cambridge University Press, Cambridge.

Hayashi, F. (2000). *Econometrics.* Princeton University Press, Princeton, NJ.

Imbens, G.W. (2002). "Generalized Method of Moments and Empirical Likelihood," *Journal of Business and Economic Statistics*, 20, 493-506.

Jacquier, E., N.G. Polson and P.E. Rossi (1994). "Bayesian Analysis of Stochastic Volatility Models," *Journal of Business and Economic Statistics*, 12, 371-389.

Jagannathan, R. and G. Skoulakis(2002). "Generalized Method of Moments: Applications in Finance," *Journal of Business and Economic Statistics*, 20, 470-482.

James, J. and N. Webber (2000). *Interest Rate Models.* John Wiley & Sons, Chichester, England.

Newey, W. (1985). "Generalized Method of Moments Specification Testing," *Journal of Econometrics*, 29, 229-256.

Newey, W. and R. Smith (2004). "Higher Order Properties of GMM and Generalized Empirical Likelihood," *Econometrica*, 72, 219-255.

Newey, W. and K. West (1987). "Hypothesis Testing with Efficient Method of Moments," *International Economic Review*, 28, 777-787.

Newey, W. and K. West (1994). "Automatic Lag Selection for Covariance Matrix Estimation," *Review of Economic Studies*, 61, 631-653.

Ogaki, M. (1992). “Generalized Method of Moments: Econometric Applications”, in G. Maddala, C. Rao and H. Vinod (eds.), *Handbook of Statistics, Volume 11: Econometrics*, North-Holland, Amsterdam.

Ruud, P. A. (2000). *An Introduction to Classical Econometric Theory.* Oxford University Press, Oxford.

Sargan, D. (1958). “The Estimation of Economic Relationships Using Instrumental Variables,” *Econometrica*, 26, 393-415.

Shephard, N. G. (1996). “Statistical Aspects of ARCH and Stochastic Volatility,” in D.R. Cox, D.V. Hinkley and O.E. Barndorff-Nielsen (eds.), *Time Series Models in Econometrics, Finance and Other Fields.* Chapman & Hall, London.

Stock, J.H. and J.H. Wright (2000). “GMM with Weak Identification,” *Econometrica*, 68, 1055-96.

Stock, J.H., J.H. Wright and M. Yogo (2002). “A Survey of Weak Instruments and Weak Identification in Generalized Method of Moments,” *Journal of Business and Economic Statistics*, 20, 518-529.

Taylor, S.J. (1986). *Modelling Financial Time Series.* John Wiley & Sons, Chichester, England.

Verbeek, M. (2000). *A Guide to Modern Econometrics.* John Wiley & Sons, Chichester, England.

White, H. (1980). “A Heteroskedasticity-Consistent Covariance Matrix Estimator and a Direct Test for Heteroskedasticity,” *Econometrica*, 48, 817-838.

Wooldrige, J. (2002). *Introduction to Econometrics, 2nd edition.* South-Western, Cincinnati, OH.

Wright, J. (2003). “Detecting Lack of Identification in GMM,” *Econometric Theory*, 19, 322-330.

22 Seminonparametric Conditional Density Models

22.1 Introduction

For a stationary multiple time series, the one-step-ahead conditional density represents the underlying stochastic process. The conditional density incorporates all information about various characteristics of the time series, including conditional heteroskedasticity, non-normality, time irreversibility, and other forms of nonlinearity. These properties are now widely considered to be important features of many time series processes. Since the conditional density completely characterizes a stochastic process, it is thus naturally viewed as the fundamental statistical object of interest.

SNP is a **s**emi**n**on**p**arametric method, based on an expansion in Hermite functions, for estimation of the one-step-ahead conditional density. The method was first proposed by Gallant and Tauchen (1989) in connection with an asset pricing application. It consists of applying classical *parametric* estimation and inference procedures to models derived from *nonparametric* truncated series expansions. Particularly, estimation of SNP models entails using a standard maximum likelihood procedure together with a model selection strategy that determines the appropriate degree of the nonparametric expansion. The general form of the SNP model is a nonlinear, non-Gaussian multivariate generalized autoregressive conditional heteroskedasticity (GARCH) model. Under reasonable regularity conditions, the estimator is consistent and efficient for the unknown one-step-ahead conditional density.

As a pure time series model, the SNP model extends the standard vector autoregressive (VAR) model described in Chapter 11 to allow for non-Gaussian multivariate GARCH-type error structures. As a result, they provide a semiparametric alternative to the multivariate GARCH models described in Chapter 13. The SNP models can also be used as an alternative to the copula methodology discussed in Chapter 19 for the general modeling of multivariate distributions. An important use of SNP models is to act as a score generator, or auxiliary model, for *efficient method of moments* (EMM) estimation. The EMM estimation of general discrete-time and continuous-time dynamic time series models based on SNP auxiliary models is described in the next chapter.

This chapter provides a general overview of the SNP methodology, and introduces a comprehensive set of `S-PLUS` functions for the estimation and analysis of SNP models that are based on the FORTRAN code of Gallant and Tauchen (2001). The exposition of the SNP methodology borrows heavily from Gallant and Tauchen (2001, 2001b, and 2002). The outline of the chapter is as follows. Section 22.2 gives a general overview of the SNP methodology and the `S-PLUS` functions for fitting SNP models. Section 22.3 illustrates the use of the `S+FinMetrics` function `SNP` for estimating a variety of univariate and bivariate SNP models. Section 22.4 describes the estimation of SNP models using model selection criteria. Section 22.5 discusses various types of diagnostic analysis of SNP models, including residual diagnostics and simulation diagnostics. Prediction from SNP models is covered in Section 22.6, and useful data transformations are covered in Section 22.7. Finally, Section 22.8 presents several in-depth examples of fitting SNP models to financial time series.

22.2 Overview of SNP Methodology

Consider a stationary and ergodic multivariate time series $\{\mathbf{y}_t\}_{t=-L+1}^{T}$, where each $\mathbf{y}_t$ is a vector of length M. Assume that $\mathbf{y}_t$ is Markovian (i.e., the conditional distribution of $\mathbf{y}_t$ given the entire past depends only on a finite number L of lagged values of $\mathbf{y}_t$)[1]. Denote these lagged values by the state vector

$$\mathbf{x}_{t-1} = (\mathbf{y}'_{t-1}, \ldots, \mathbf{y}'_{t-L})'$$

which is a vector of length $M \cdot L$. For simplicity, the time subscripts are often suppressed and $\mathbf{y}$ and $\mathbf{x}$ are used to denote the contemporaneous value and lagged state vector, respectively. With these conventions, the stationary density of the dynamic system under consideration can be written $p(\mathbf{x}, \mathbf{y}|\boldsymbol{\rho})$

[1]This section is taken from Gallant and Tauchen (2001b) by permission of the authors.

and its transition density as

$$p(\mathbf{y}|\mathbf{x},\boldsymbol{\rho}) = \frac{p(\mathbf{x},\mathbf{y}|\boldsymbol{\rho})}{\int p(\mathbf{x},\mathbf{y}|\boldsymbol{\rho})\,d\mathbf{x}} \tag{22.1}$$

If one expands $\sqrt{p(\mathbf{x},\mathbf{y}\,|\,\boldsymbol{\rho}^o)}$ in a Hermite series and derives the transition density of the truncated expansion, then one obtains a transition density $f_K(\mathbf{y}_t\,|\,\mathbf{x}_{t-1},\boldsymbol{\theta})$ that has the form of a location-scale transform

$$\mathbf{y}_t = \mathbf{R}\mathbf{z}_t + \boldsymbol{\mu}_{x_{t-1}}$$

of an innovation $\mathbf{z}_t$ (Gallant, Hsieh, and Tauchen, 1991). The density function of this innovation is

$$h_K(\mathbf{z}|\mathbf{x}) = \frac{[\mathcal{P}(\mathbf{z},\mathbf{x})]^2\phi(\mathbf{z})}{\int[\mathcal{P}(\mathbf{u},\mathbf{x})]^2\phi(\mathbf{u})\,d\mathbf{u}} \tag{22.2}$$

where $\mathcal{P}(\mathbf{z},\mathbf{x})$ is a polynomial in $(\mathbf{z},\mathbf{x})$ of degree K and $\phi(\mathbf{z})$ denotes the multivariate normal density function with dimension M, zero mean vector zero, and identity variance-covariance matrix.

It proves convenient to express the polynomial $\mathcal{P}(\mathbf{z},\mathbf{x})$ in a rectangular expansion

$$\mathcal{P}(\mathbf{z},\mathbf{x}) = \sum_{\boldsymbol{\alpha}=0}^{K_z}\left(\sum_{\boldsymbol{\beta}=0}^{K_x}\mathbf{a}_{\boldsymbol{\beta}\boldsymbol{\alpha}}\mathbf{x}^{\boldsymbol{\beta}}\right)\mathbf{z}^{\boldsymbol{\alpha}} \tag{22.3}$$

where $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are multi-indexes of maximal degrees K_z and K_x, respectively, and $K = K_z + K_x$. Because $[\mathcal{P}(\mathbf{z},\mathbf{x})]^2/\int[\mathcal{P}(\mathbf{u},\mathbf{x})]^2\phi(\mathbf{u})d\mathbf{u}$ is a homogeneous function of the coefficients of the polynomial $\mathcal{P}(\mathbf{z},\mathbf{x})$, $\mathcal{P}(\mathbf{z},\mathbf{x})$ can only be determined to within a scalar multiple. To achieve a unique representation, the constant term a_{00} of the polynomial $\mathcal{P}(\mathbf{z},\mathbf{x})$ is put to one. With this normalization, $h_K(\mathbf{z}|\mathbf{x})$ has the interpretation of a series expansion whose leading term is the normal density $\phi(\mathbf{z})$ and whose higher-order terms induce departures from normality.

The location function is linear

$$\boldsymbol{\mu}_x = \mathbf{b}_0 + \mathbf{B}\mathbf{x}_{t-1} \tag{22.4}$$

where $\mathbf{b}_0$ is a vector and $\mathbf{B}$ is a matrix.

It proves advantageous in applications to allow the scale $\mathbf{R}$ of the location-scale transformation $\mathbf{y} = \mathbf{R}\mathbf{z} + \boldsymbol{\mu}_x$ to depend on $\mathbf{x}$ because it reduces the degree K_x required to achieve an adequate approximation to the transition density $p(\mathbf{y}|\mathbf{x},\boldsymbol{\rho}^o)$. With this, the location-scale transformation becomes

$$\mathbf{y} = \mathbf{R}_x\mathbf{z} + \boldsymbol{\mu}_x \tag{22.5}$$

where $\mathbf{R}_x$ is an upper triangular matrix that depends on $\mathbf{x}$. The two choices of $\mathbf{R}_x$ that have given good results in applications are an autoregressive conditional heteroskedasticity (ARCH)-like moving average specification and a GARCH-like autoregressive-moving average (ARMA) specification.

For an ARCH specification, let $\mathbf{R}_{x_{t-1}}$ be a linear function of the absolute values of the elements of the vectors $\mathbf{y}_{t-L_r} - \boldsymbol{\mu}_{x_{t-1-L_r}}$ through $\mathbf{y}_{t-1} - \boldsymbol{\mu}_{x_{t-2}}$; namely,

$$\text{vech}(\mathbf{R}_{x_{t-1}}) = \boldsymbol{\rho}_0 + \sum_{i=1}^{L_r} \mathbf{P}_{(i)} |\mathbf{y}_{t-1-L_r+i} - \boldsymbol{\mu}_{x_{t-2-Lr+i}}|$$

where vech($\mathbf{R}$) denotes a vector of length $M(M+1)/2$ containing the elements of the upper triangle of $\mathbf{R}$, ρ_0 is a vector of length $M(M+1)/2$, $\mathbf{P}_{(1)}$ through $\mathbf{P}_{(Lr)}$ are $M(M+1)/2 \times M$ matrices, and $|\mathbf{y} - \boldsymbol{\mu}|$ denotes a vector containing the absolute values of $\mathbf{y} - \boldsymbol{\mu}$. The classical ARCH (Engle, 1982) has

$$\boldsymbol{\Sigma}_{x_{t-1}} = \mathbf{R}_{x_{t-1}} \mathbf{R}'_{x_{t-1}}$$

depending on a linear function of squared lagged residuals. The SNP version of ARCH is more akin to the suggestions of Taylor (1986) and Davidian and Carroll (1987).

Since the absolute value function is not differentiable, $|u|$ is approximated in the above formula for $\mathbf{R}_x$ by the twice continuously differentiable function

$$a(u) = \begin{cases} (|100u| - \pi/2 + 1)/100, & |100u| \geq \pi/2 \\ (1 - \cos(100u))/100, & |100u| < \pi/2 \end{cases}$$

The above scale factor 100 represents a compromise. Small values, such as 3, improve the stability of the computations but then $a(\cdot)$ does not approximate $|\cdot|$ well.

For a GARCH specification, let

$$\begin{aligned} \text{vech}(\mathbf{R}_{x_{t-1}}) \quad &= \quad \boldsymbol{\rho}_0 + \sum_{i=1}^{L_r} \mathbf{P}_{(i)} |\mathbf{y}_{t-1-L_r+i} - \boldsymbol{\mu}_{x_{t-2-Lr+i}}| \qquad (22.6) \\ &\quad + \sum_{i=1}^{L_g} \text{diag}(\mathbf{G}_{(i)}) \mathbf{R}_{x_{t-2-L_g+i}} \end{aligned}$$

where $\mathbf{G}_{(1)}$ through $\mathbf{G}_{(L_g)}$ are vectors of length $M(M+1)/2$.

The classical GARCH (Bollerslev, 1986) has $\boldsymbol{\Sigma}_{x_{t-1}}$ expressed in terms of squared lagged residuals and lagged values of $\boldsymbol{\Sigma}_{x_{t-1}}$. As with the SNP variant of ARCH, the SNP version of GARCH is expressed in terms of the absolute value of lagged residuals and standard deviations.

Note that when $L_g > 0$, the SNP model is not Markovian and that one must know both $\mathbf{x}_{t-1}$ and $\mathbf{R}_{x_{t-2-L_g}}$ through $\mathbf{R}_{x_{t-2}}$ to move forward to the value for $\mathbf{y}_t$. Thus, $\mathbf{x}_{t-1}$ and $\mathbf{R}_{x_{t-2-L_g}}$ through $\mathbf{R}_{x_{t-2}}$ represent the state of the system at time $t-1$ and must be retained in order to evaluate the SNP conditional density of $\mathbf{y}_t$ or to iterate the SNP model forward by simulation. If one wants to compute the derivatives of the SNP density with respect to model parameters, one must retain the derivatives of $\mathbf{R}_{x_{t-2-L_g}}$ through $\mathbf{R}_{x_{t-2}}$ with respect to model parameters as well.

The change of variable formula applied to the location-scale transform (22.5) and innovation density (22.2) yields the SNP density

$$f_K(\mathbf{y} \mid \mathbf{x}, \boldsymbol{\theta}) = \frac{h_K\big[\mathbf{R}_x^{-1}(\mathbf{y} - \boldsymbol{\mu}_x) \mid \mathbf{x}\big]}{\det(\mathbf{R}_x)} \tag{22.7}$$

Hereafter, the lag lengths appearing in the various components of the expansion will be distinguished. The number of lags in $\boldsymbol{\mu}_x$ is denoted L_u, the number of lags in $\mathbf{R}_x$ is $L_u + L_r$, and the number of lags in the $\mathbf{x}$ part of the polynomial, $\mathcal{P}(\mathbf{z}, \mathbf{x})$, is L_p. The maximum lag is defined as $L = \max(L_u, L_u + L_r, L_p)$.

Large values of M can generate a large number of interactions (cross-product terms) for even modest settings of degree K_z, and similarly for $M \cdot L_p$ and K_x. Accordingly, two additional tuning parameters, I_z and I_x, are introduced to represent filtering out of these high-order interactions. $I_z = 0$ means that no interactions are suppressed, $I_z = 1$ means that the highest-order interactions are suppressed, namely those of degree K_z. In general, a positive I_z means that all interactions of order larger than $K_z - I_z$ are suppressed; similarly for $K_x - I_x$.

In summary, L_u, L_g, and L_r determine the location-scale transformation $\mathbf{y} = \mathbf{R}_x \mathbf{z}_t + \boldsymbol{\mu}_x$ and, hence, determine the nature of the leading term of the expansion. The number of lags in the location function $\boldsymbol{\mu}_x$ is L_u and the number of lags in the scale function $\mathbf{R}_x$ is $L_u + L_r$. The number of lags that go into the $\mathbf{x}$ part of the polynomial $\mathcal{P}(\mathbf{z}, \mathbf{x})$ is L_p. The parameters K_z, K_x, I_z, and I_x determine the degree of $\mathcal{P}(\mathbf{z}, \mathbf{x})$ and, hence, the nature of the innovation process $\{\mathbf{z}_t\}$.

22.3 Estimating SNP Models in S+FinMetrics

S+FinMetrics contains a variety of functions for estimation and analysis of SNP models. Table 22.1 gives a brief summary of these functions. The main functions are based on the public domain FORTRAN code for the estimation of SNP models described in Gallant and Tauchen (2001).

The function SNP is used to estimate univariate and multivariate SNP models. The arguments expected by SNP are

```
> args(SNP)
function(data, model = SNP.model(), n.drop = NULL, control =
SNP.control(), coef = NULL, est = NULL, iAr = 0, iArch
= 0, iGarch = 0, iLagP = 0, iZPoly = 0, iXPoly = 0,
```

Function	Description
`SNP`	Fit SNP model using quasi-maximum likelihood estimation
`SNP.model`	Create SNP model list
`SNP.control`	Create list of control parameters for SNP model and estimation
`expand`	Expand fitted SNP model to larger model
`SNP.auto`	Automatically find best fitting SNP model by minimizing Bayesian information criterion (BIC)
`SNP.density`	Plot SNP conditional transition density

TABLE 22.1. `S+FinMetrics` functions for analysis of SNP models

```
iTrimZ = 0, iTrimX = 0, LStransform = NULL, save.data
= F, trace = F)
```

The main arguments are `data`, which may be a univariate or multivariate rectangular data object, `model`, which is a list describing the SNP model to be fit, and `control`, which is a list containing various parameters that control aspects of the SNP model and optimization algorithm. The other arguments will be explained in the examples below.

The function `SNP.model` creates a list containing components that specify the type of SNP model to be fit. The arguments of `SNP.model` are

```
> args(SNP.model)
function(ar = 0, arch = 0, garch = 0, lagP = 1, zPoly = 0,
xPoly = 0, trimZ = 0, trimX = 0)
```

The form of a fitted SNP model is determined by a number of tuning parameters. These parameters and the corresponding arguments to the function `SNP.model` are summarized in Table 22.2. When describing a particular SNP model, it is convenient to refer to the model as a string of tuning parameters: $L_u L_g L_r L_p K_z I_z K_x I_x$. For example, a semiparametric VAR(1) with $K_z = 4$ is denoted 10014000 and a nonlinear nonparametric VAR(3)-GARCH(1,1) model with $K_z = 4$, $K_x = 2$ excluding all interactions and allowing two lags of x_{t-1} in $\mathcal{P}(\mathbf{z}_t, \mathbf{x}_{t-1})$ is denoted 31124321.

Table 22.3 provides a useful taxonomy of SNP models, defined by putting certain restrictions on the tuning parameters. The `print` and `summary` methods of `"SNP"` objects reflect this taxonomy.

The function `SNP.control` creates a list with parameters that control restrictions on the SNP model and set options for the optimization algorithm. The default arguments to `SNP.control` are

```
> args(SNP.control)
function(initial.itmax = 15, final.itmax = 385, tol = 1e-005,
 xTransform = "none", inflection = NULL, diagS = T, darch = T,
 rgarch = F, dvec = T, n.start = 0, seed = 72674,
```

Parameter	`SNP.model` **argument**	**Description**
L_u	`ar`	VAR lag length
L_g	`garch`	GARCH lag length
L_r	`arch`	ARCH lag length
L_p	`lagP`	Lags of $\mathbf{x}_{t-1}$ in $\mathcal{P}(\mathbf{z}_t, \mathbf{x}_{t-1})$
K_z	`zPoly`	Order of $\mathbf{z}_t$ in $\mathcal{P}(\mathbf{z}_t, \mathbf{x}_{t-1})$
I_z	`trimZ`	Index of $\mathbf{z}_t$ interactions in $\mathcal{P}(\mathbf{z}_t, \mathbf{x}_{t-1})$
K_x	`xPoly`	Order of $\mathbf{x}_{t-1}$ in $\mathcal{P}(\mathbf{z}_t, \mathbf{x}_{t-1})$
I_x	`trimX`	Index of $\mathbf{x}_{t-1}$ interactions in $\mathcal{P}(\mathbf{z}_t, \mathbf{x}_{t-1})$

TABLE 22.2. SNP tunning parameters

Parameter Setting	**Characterization of** $\{\mathbf{y}_t\}$
$L_u = 0, L_g = 0, L_r = 0, L_p \geq 0, K_z = 0, K_x = 0$	iid Gaussian
$L_u > 0, L_g = 0, L_r = 0, L_p \geq 0, K_z = 0, K_x = 0$	Gaussian VAR
$L_u > 0, L_g = 0, L_r = 0, L_p \geq 0, K_z > 0, K_x = 0$	Semiparametric VAR
$L_u \geq 0, L_g = 0, L_r > 0, L_p \geq 0, K_z = 0, K_x = 0$	Gaussian ARCH
$L_u \geq 0, L_g = 0, L_r > 0, L_p \geq 0, K_z > 0, K_x = 0$	Semiparametric ARCH
$L_u \geq 0, L_g > 0, L_r > 0, L_p \geq 0, K_z = 0, K_x = 0$	Gaussian GARCH
$L_u \geq 0, L_g > 0, L_r > 0, L_p \geq 0, K_z > 0, K_x = 0$	Semiparametric GARCH
$L_u \geq 0, L_g \geq 0, L_r \geq 0, L_p \geq 0, K_z > 0, K_x > 0$	Nonlinear nonparametric

TABLE 22.3. Taxonomy of SNP Models

```
fOld = 0, fNew = 0, eps0 = 1e-005, version = 8.9)
```

These control parameters are explained in the examples in the following subsections.

The following subsections describe in detail the SNP methodology for describing the conditional transition density of a stationary and ergodic time series, and the use of the S+FinMetrics function SNP for estimating the SNP density. The analysis starts with a simple Gaussian VAR model. This model is then generalized to a semiparametric VAR model. Finally, conditional heterogeneity is introduced into the SNP model. In each subsection, examples of using the SNP function to estimate univariate and bivariate SNP models are given.

22.3.1 Example Data

The data for the following examples are in the "timeSeries" object dmRet.dat, which contains weekly observations on three variables: the percentage changes in the German mark and U.S. dollar (DM/US) exchange

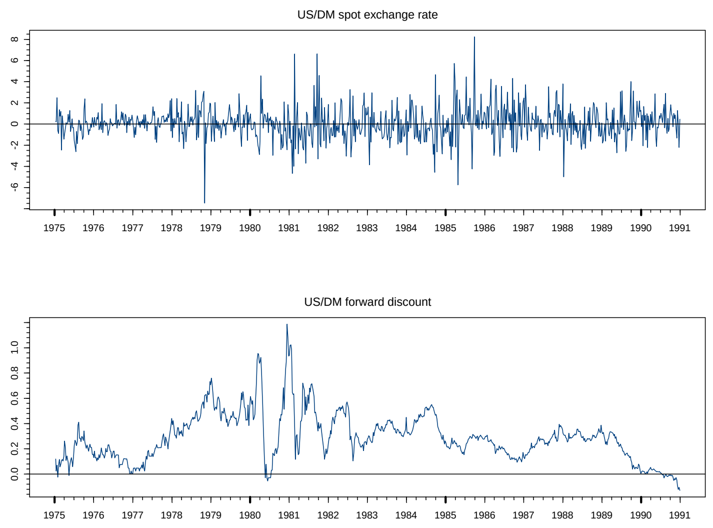


FIGURE 22.1. Weekly observations on US/DM spot exchange rate and 30-day forward discount.

rate; the difference between the 30-day forward rate and the spot exchange rate; and the U.S. 30-day Treasury bill rate. The data cover the period January 10, 1975 to December 28, 1990. This data set has been analyzed by Bansal, Gallant, Hussey, and Tauchen (1994), and Gallant and Tauchen (2001) used it to illustrate their FORTRAN code for fitting SNP models.[2]

The three variables in `dmRet.dat` are

```
> colIds(dmRet.dat)
[1] "exch"    "forward" "tbill"
```

which give the spot returns, the forward discounts, and the U.S. Treasury bill rates, respectively. Figure 22.1 shows the log returns on the spot exchange rate and the forward discount, which will be used in the examples. The sample autocorrelation and cross-autocorrelation functions for the two series are given in Figure 22.2. The spot returns are nearly uncorrelated, whereas the forward discounts are highly persistent. Some sample descriptive statistics, from the S+FinMetrics function `summaryStats`, are

```
> summaryStats(dmRet.dat[,c("exch","forward")])
```

[2]The data are taken from the file `dm_fri.dat` available from `ftp://ftp.econ.duke.edu/pub/get/data/`

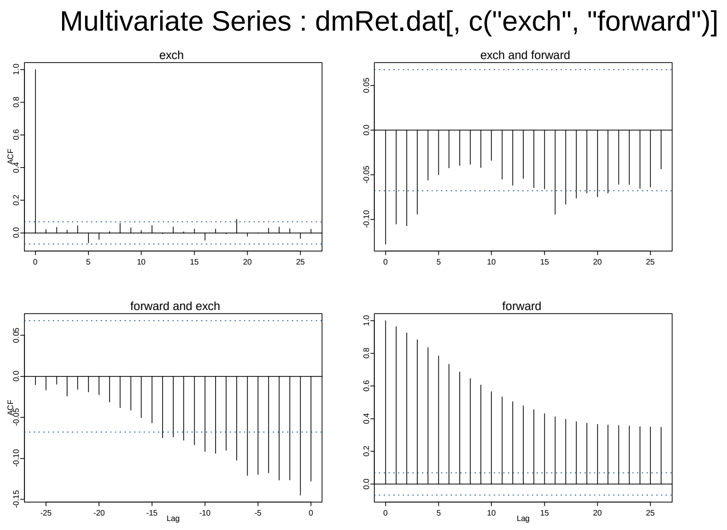


FIGURE 22.2. Sample autocorrelation function (SACF) for weekly spot return and forward discount data.

```
Sample Quantiles:
            min      1Q   median     3Q   max
   exch -7.4622 -0.6977 0.006332 0.8175 8.225
forward -0.1291  0.1619 0.279385 0.3955 1.187

Sample Moments:
           mean    std skewness kurtosis
   exch 0.05516 1.4964   0.3367    6.172
forward 0.29111 0.1911   0.8574    4.672

Number of Observations:  834
```

The distribution of spot returns is nearly symmetric but has fatter tails than the normal distribution. The distribution of the forward discount is slightly positively skewed and has fatter tails than the normal.

22.3.2 Markovian Time Series and the Gaussian Vector Autoregression Model

As in Section 22.2, consider an M-dimensional stationary and ergodic Markovian multivariate time series $\{\mathbf{y}_t\}_{t=-L+1}^{T}$ with state vector $\mathbf{x}_{t-1} = (\mathbf{y}'_{t-1}, \ldots, \mathbf{y}'_{t-L})'$. The likelihood function conditional on the first L observations can be written as

$$\mathcal{L} = \prod_{t=1}^{T} f(\mathbf{y}_t|\mathbf{x}_{t-1}, \boldsymbol{\theta}) \tag{22.8}$$

where $f(\cdot)$ is the density function of $\mathbf{y}_t$ conditional on $\mathbf{x}_{t-1}$ and $\boldsymbol{\theta}$ consists of the parameters of the density function.

A traditional approach to modeling the Markovian multiple time series $\mathbf{y}_t$ uses the *Gaussian VAR* model as described in Chapter 11. For a Gaussian VAR model, $\mathbf{y}_t$ is assumed to be of the form

$$\begin{aligned} \mathbf{y}_t &= \boldsymbol{\mu}_0 + \mathbf{B}\mathbf{x}_{t-1} + \mathbf{R}\mathbf{z}_t, \\ &= \boldsymbol{\mu}_{x_{t-1}} + \mathbf{R}\mathbf{z}_t \end{aligned} \tag{22.9}$$

where $\boldsymbol{\mu}_0$ is an $M \times 1$ vector, $\mathbf{B}$ is an $M \times ML$ matrix, $\boldsymbol{\mu}_{x_{t-1}} = \boldsymbol{\mu}_0 + \mathbf{B}\mathbf{x}_{t-1}$, $\mathbf{R}$ is an upper triangular matrix, and $\mathbf{z}_t$ follows an M-dimensional Gaussian distribution with zero mean and identity covariance matrix. Note that since there are L lagged values in $\mathbf{x}_{t-1}$, this represents an Lth order VAR (VAR(L) model). The term $\boldsymbol{\mu}_{x_{t-1}}$ is referred to as the *conditional mean equation.*

By using the location-scale transformation

$$\mathbf{z}_t = \mathbf{R}^{-1}(\mathbf{y}_t - \boldsymbol{\mu}_0 - \mathbf{B}\mathbf{x}_{t-1}) = \mathbf{R}^{-1}(\mathbf{y}_t - \boldsymbol{\mu}_{x_{t-1}}) \tag{22.10}$$

it is easy to show that for the Gaussian VAR model, the one-step conditional density of $\mathbf{y}_t$ is given by

$$\begin{aligned} f(\mathbf{y}_t|\mathbf{x}_{t-1}, \boldsymbol{\theta}) &= \frac{1}{|\det(\mathbf{R})|} N(\mathbf{R}^{-1}(\mathbf{y}_t - \boldsymbol{\mu}_0 - \mathbf{B}\mathbf{x}_{t-1}); \mathbf{0}, \mathbf{I}_M) \\ &= N(\mathbf{y}_t; \boldsymbol{\mu}_{x_{t-1}}, \mathbf{R} \cdot \mathbf{R}') \end{aligned} \tag{22.11}$$

where $N(\mathbf{z}; \boldsymbol{\mu}, \boldsymbol{\Omega})$ denotes the density function of a multivariate Gaussian random vector $\mathbf{z}$ with mean $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Omega}$. Note that for the Gaussian VAR model, the model parameter $\boldsymbol{\theta}$ consists of $\boldsymbol{\mu}_0$, $\mathbf{B}$, and $\mathbf{R}$ (since $\boldsymbol{\Omega} = \mathbf{R} \cdot \mathbf{R}'$). Lütkepohl (1990) showed that maximum likelihood estimates of the Gaussian VAR parameters may be obtained using multivariate regression.

Example 147 *Gaussian VAR for univariate exchange rate data*

Consider fitting a Gaussian VAR(1) model to the log returns on the US/DM spot exchange rate, Δs_t. For a univariate time series y_t, a VAR(1) model reduces to a simple first-order autoregressive, AR(1), model. This is in the form (22.9) with $x_{t-1} = y_{t-1}$ and

$$\begin{aligned} \mu_{x_{t-1}} &= \mu_0 + b y_{t-1} \\ R &= \sigma = \sqrt{\mathrm{var}(y_t|x_{t-1})} \end{aligned}$$

To fit a Gaussian AR(1) model to the spot rate return, call the SNP function using the "exch" column of dmRet.dat:

```
> fit.10010000 = SNP(dmRet.dat[,"exch"],model=SNP.model(ar=1),
+                    n.drop=14)
```

The model specification is accomplished by calling the function SNP.model and passing the result as a model argument to SNP. The ar=1 argument to SNP.model determines the order of the VAR model, and the optional argument n.drop=14 specifies setting aside 14 observations as presample observations and starting the estimation from the 15th observation.

The returned object fit.10010000 is of class "SNP":

```
> class(fit.10010000)
[1] "SNP"
```

Typing the name of an object at the command line automatically invokes the print method for the corresponding class and prints out the function call that generated the object, the estimated model parameters, along with some commonly used information criteria[3]:

```
> fit.10010000

Call:
SNP(data = dmRet.dat[, "exch"], model = SNP.model(ar = 1),
    n.drop = 14)

Model: Gaussian VAR

Conditional Mean Coefficients:
     mu  ar(1)
 0.0005 0.0222

Conditional Variance Coefficients:
  sigma
 1.0025

Information Criteria:
    BIC     HQ    AIC    logL
 1.4337 1.4284 1.4251 -1165.6

Convergence Type:
both x and relative function convergence
```

[3]If the default settings are used to invoke the SNP function, the SNP estimates of a Gaussian VAR model are not the same as traditional VAR estimates because of a location-scale transform employed on the data.

```
number of iterations: 2
```

The conditional mean coefficients are the estimates of the coefficients in $\mu_{x_{t-1}}$, which include $\hat{\mu}_0 = 0.005$ and $\hat{b} = 0.0222$. The conditional variance coefficients are the estimates of the coefficients in $\mathbf{R}$, which in this case is just $\hat{\sigma} = 1.0025$.

To obtain standard errors of the estimated parameters, use the generic `summary` function, which automatically invokes the `summary` method for the corresponding class:

```
> summary(fit.10010000)

Call:
SNP(data = dmRet.dat[, "exch"], model = SNP.model(ar = 1),
    n.drop = 14)

Model: Gaussian VAR

Conditional Mean Coefficients:
               mu  ar(1)
     coef 0.0005 0.0222
(std.err) 0.0356 0.0270
 (t.stat) 0.0143 0.8245


Conditional Variance Coefficients:
             sigma
     coef  1.0025
(std.err)  0.0158
 (t.stat) 63.2994


Information Criteria:
    BIC     HQ    AIC    logL
 1.4337 1.4284 1.4251 -1165.6

Convergence Type:
both x and relative function convergence
number of iterations: 2
```

Given the sample ACF of the spot rate return, it is not surprising that the autoregressive coefficient $\hat{b}$ is not significantly different from zero.

Example 148 *Gaussian VAR for bivariate exchange rate data*

Now consider fitting a bivariate Gaussian VAR(1) model to the spot rate return and the forward discount. Here, $y_t = (\Delta s_t, fd_t)'$ and the VAR model

(22.9) has $\mathbf{x}_{t-1} = \mathbf{y}_{t-1} = (\Delta s_{t-1}, fd_{t-1})'$ with

$$\begin{aligned}
\boldsymbol{\mu}_{x_{t-1}} &= \boldsymbol{\mu}_0 + \mathbf{B}\mathbf{y}_{t-1} \\
&= \begin{pmatrix} \mu_{0,1} \\ \mu_{0,2} \end{pmatrix} + \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} \begin{pmatrix} \Delta s_{t-1} \\ fd_{t-1} \end{pmatrix} \\
\mathbf{R} &= \begin{pmatrix} r_{11} & r_{21} \\ 0 & r_{22} \end{pmatrix}, \ \mathbf{R}\mathbf{R}' = \boldsymbol{\Omega} = \mathrm{var}(\mathbf{y}_t|\mathbf{x}_{t-1})
\end{aligned}$$

To fit the bivariate VAR(1) to the exchange rate data, use[4]

```
> bfit.10010000 = SNP(dmRet.dat[,1:2],model=SNP.model(ar=1),
+                     n.drop=14)
> bfit.10010000

Call:
SNP(data = dmRet.dat[, 1:2], model = SNP.model(ar = 1),
    n.drop = 14)

Model: Gaussian VAR

Conditional Mean Coefficients:
            exch forward
      mu  0.0025 -0.0016
var(1;1)  0.0087 -0.1065
var(1;2) -0.0201  0.9671

Conditional Variance Coefficients:
 sigma11 sigma12 sigma22
   0.992 -0.0994  0.2541

Information Criteria:
    BIC     HQ    AIC      logL
 1.4965 1.4806 1.4706 -1196.927

Convergence Type:
both x and relative function convergence
number of iterations: 2
```

The first row of the conditional mean coefficients are the intercept estimates $\hat{\boldsymbol{\mu}}_0$. The second row, labeled `var(1;1)`, gives the estimates on the lag-

[4]To match the output of the `S+FinMetrics` function `VAR`, the location-scale transformation must be turned off. To do this, in the call to `SNP` set the optional argument `LStransform=list(mu=c(0,0),cstat=diag(2),pstat=diag(2))`. Note: the arrangement of the VAR coefficients printed from an `"SNP"` object is different than the arrangement of the coefficients printed from a `"VAR"` object.

1 variables for the spot return equation (first row of $\hat{\mathbf{B}}$). The third row, labeled `var(1;2)`, gives the estimates on the lag-1 variables for the forward discount equation (second row of $\hat{\mathbf{B}}$). The estimated equations are then

$$\begin{aligned}\Delta s_t &= 0.0025 + 0.0087\Delta s_{t-1} - 0.1065 fd_{t-1} \\ fd_t &= -0.0016 - 0.0201\Delta s_{t-1} + 0.9671 fd_{t-1}\end{aligned}$$

The conditional variance coefficients give the estimates of the elements of the upper triangular matrix $\mathbf{R}$:

$$\hat{\mathbf{R}} = \begin{pmatrix} 0.992 & -0.0994 \\ 0 & 0.2541 \end{pmatrix}$$

22.3.3 Hermite Expansion and the Semiparametric VAR

In practice, especially for economic and financial time series modeling, the standardized residuals $\mathbf{z}_t$ are rarely Gaussian. To model different characteristics usually observed in empirical data, like fat tails or asymmetry, one needs a good general-purpose density $f(\mathbf{y}_t|\mathbf{x}_{t-1}, \boldsymbol{\theta})$ that can accommodate any shape, especially the Gaussian as it is most likely *a priori*. The SNP model proposed by Gallant and Tauchen (1989) provides such a general-purpose approximation.

The SNP model is based on an expansion in Hermite functions. Using a Hermite expansion as a general approximation to a density function has a long established tradition in statistics; examples are Gram-Charlier and Edgeworth expansions. The form of a Hermite expansion is a multivariate polynomial in the standardized residuals $\mathbf{z}_t$ multiplied by the standard Gaussian density; that is,

$$f(\mathbf{y}_t|\mathbf{x}_{t-1}, \boldsymbol{\theta}) \propto [\mathcal{P}(\mathbf{z}_t)]^2 N(\mathbf{y}_t; \boldsymbol{\mu}_{x_{t-1}}, \boldsymbol{\Omega}) \tag{22.12}$$

where $\boldsymbol{\Omega} = \mathbf{R} \cdot \mathbf{R}'$ and $\mathcal{P}(\mathbf{z}_t)$ denotes a multivariate polynomial of degree K_z.[5] The polynomial $\mathcal{P}(\mathbf{z}_t)$ has the form

$$\mathcal{P}(\mathbf{z}_t) = \sum_{|\boldsymbol{\alpha}|=0}^{K_z} a_{\boldsymbol{\alpha}} \mathbf{z}_t^{\boldsymbol{\alpha}}$$

with $\boldsymbol{\alpha}$ denoting the non-negative multi-index $(\alpha_1, \ldots, \alpha_M)$, and

$$\begin{aligned}|\boldsymbol{\alpha}| &= \alpha_1 + \cdots + \alpha_M, \\ \mathbf{z}_t^{\boldsymbol{\alpha}} &= (z_{t,1})^{\alpha_1} \cdot \cdots \cdot (z_{t,M})^{\alpha_M}\end{aligned}$$

[5]The constant of proportionality in (22.12) is $1/\int [\mathcal{P}(\mathbf{s})]^2 \phi(\mathbf{s}) d\mathbf{s}$, where $\phi(\mathbf{s})$ denotes the density of a multivariate normal random variable with zero mean and identity covariance matrix.

For example, let $K_z = 2$ and $M = 2$. Then,[6]

$$\begin{aligned}
\mathbf{z} &= (z_1, z_2)',\ \boldsymbol{\alpha} = (\alpha_1, \alpha_2)',\ a_{\boldsymbol{\alpha}} = a_{(\alpha_1,\alpha_2)} \\
|\boldsymbol{\alpha}| &= 0 \Rightarrow \boldsymbol{\alpha} = (0,0)' \\
|\boldsymbol{\alpha}| &= 1 \Rightarrow \boldsymbol{\alpha} = (1,0)', (0,1)' \\
|\boldsymbol{\alpha}| &= 2 \Rightarrow \boldsymbol{\alpha} = (1,1)', (2,0)', (0,2)'
\end{aligned}$$

and

$$\begin{aligned}
\mathcal{P}(\mathbf{z}) &= 1 + a_{(1,0)} z_1 + a_{(0,1)} z_2 \\
&\quad + a_{(1,1)} z_1 z_2 + a_{(2,0)} z_1^2 + a_{(0,2)} z_2^2
\end{aligned}$$

The order K_z of the Hermite polynomial expansion acts like a smoothness parameter in nonparametric analysis. When K_z is set to zero, the SNP density $f(\mathbf{y}_t|\mathbf{x}_{t-1}, \boldsymbol{\theta})$ reduces to the Gaussian VAR density function in (22.11). As a result, the SNP density in (22.12) nests the Gaussian VAR as a special case. In general, the SNP density (22.12) is referred to as the *semiparametric VAR*. When K_z is positive, the semiparametric VAR density can accommodate arbitrary shape departures from the Gaussian VAR as long as K_z is large enough. The shape modifications achieved by the semiparametric VAR are rich enough to accurately approximate densities from a large class that includes densities with fat, t-like tails, densities with tails that are thinner than Gaussian, skewed densities, and even multimodal densities. See Gallant and Tauchen (1999) for details.

The parameter vector $\boldsymbol{\theta}$ of the semiparametric VAR density consists of the coefficients $a_{\boldsymbol{\alpha}}$ of the polynomial $\mathcal{P}(\mathbf{z})$, plus $\boldsymbol{\mu}_0$, $\mathbf{B}$, and $\mathbf{R}$. They can be estimated using a quasi-maximum likelihood procedure, by minimizing

$$s_n(\boldsymbol{\theta}) = -\frac{1}{n} \sum_{t=1}^{T} \log f(\mathbf{y}_t|\mathbf{x}_{t-1}, \boldsymbol{\theta}) \tag{22.13}$$

using nonlinear regression techniques. Under regularity conditions, Gallant and Nychka (1987) showed that if the number of parameters in $\boldsymbol{\theta}$ grows with the sample size, the true density and various features of it, such as derivatives and moments, are estimated consistently. Fenton and Gallant (1996a, 1996b) proved the efficiency of SNP estimates and other properties.

Example 149 *Semi-parametric VAR for univariate exchange rate data*

Fitting a semiparametric VAR model using the `SNP` function requires specifying the order of the Hermite polynomial K_z in the specification of

[6]However, since the density function in (22.12) can be shown to be a homogeneous function of the coefficients of the polynomial $\mathcal{P}(\mathbf{z}_t)$, the coefficients a_α can only be determined to within a scalar multiple. Thus, to achieve model identification, the constant term of the polynomial part is always set to 1.

the SNP model. The optional argument `zPoly` of the function `SNP.model` corresponds to K_z, and setting this to a non-negative integer specifies a semiparametric VAR model. For example, to estimate a semiparametric AR(1) model with $K_z = 4$ to the spot return data, use[7]

```
> fit.10014000 = SNP(dmRet.dat[,"exch"],model=SNP.model(ar=1,
                     zPoly=4), n.drop=14)
> summary(fit.10014000)

Call:
SNP(data = dmRet.dat[, "exch"], model = SNP.model(ar = 1,
    zPoly = 4), n.drop = 14)

Model: Semiparametric VAR

Hermite Polynomial Coefficients:
             z^0     z^1     z^2     z^3     z^4
     coef  1.0000 -0.0214 -0.1812  0.0110  0.0235
(std.err)          0.0795  0.0226  0.0102  0.0032
 (t.stat)         -0.2699 -8.0043  1.0703  7.3975


Conditional Mean Coefficients:
               mu   ar(1)
     coef -0.0287  0.0460
(std.err)  0.1151  0.0283
 (t.stat) -0.2493  1.6245


Conditional Variance Coefficients:
            sigma
     coef  1.0458
(std.err)  0.0376
 (t.stat) 27.8252


Information Criteria:
   BIC     HQ    AIC      logL
 1.399 1.3866 1.3789 -1123.668

Convergence Type:
relative function convergence
```

[7]Gallant and Tauchen (2001, p. 15) state "Experience has taught us never to consider a value of $K_z < 4$."

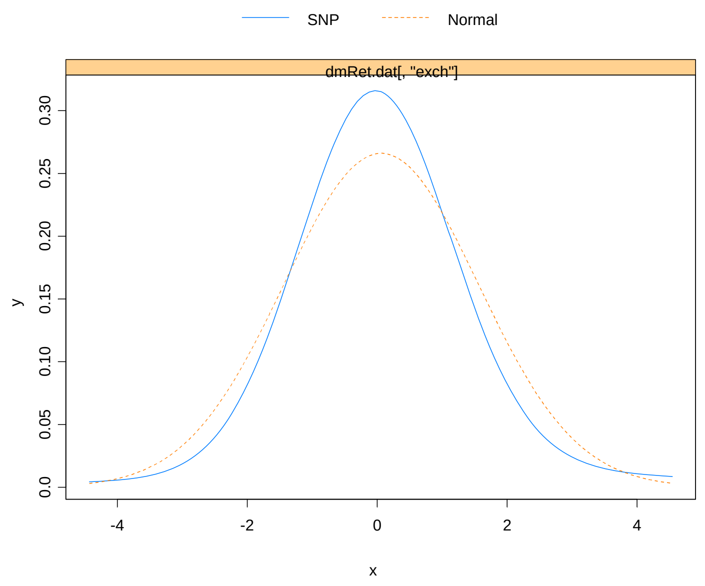


FIGURE 22.3. Estimated conditional density from semiparametric AR(1) for weekly spot returns.

```
number of iterations: 12
```

The estimated coefficient values for $a_{\boldsymbol{\alpha}}$ of $\mathcal{P}(z)$ are given under the table labeled `Hermite Polynomial Coefficients`. The estimate of $\mathcal{P}(z)$ has the form

$$\hat{\mathcal{P}}(z) = 1 - 0.0214 \cdot z - 0.1812 \cdot z^2 + 0.0110 \cdot z^3 + 0.0235 \cdot z^4$$

The estimated coefficients on the even powers of z are highly significant and indicate a conditional density with tails fatter than the Gaussian distribution. The coefficients on the odd powers of z are not significantly differently from zero and suggest that the conditional density is symmetric. To be sure, the function `SNP.density` can be used to compute and plot the estimated SNP conditional density (22.12):

```
> SNP.density(fit.10014000)
```

The estimated density, shown in Figure 22.3, is computed with $\mathbf{x}_{t-1}$ set to its sample mean value. Also shown is a normal density with the same mean and variance as (22.12).

Given that the Hermite coefficients on z_1 and z_3 are insignificant, it may be of interest to re-estimate the semiparametric AR(1) with these coefficients restricted to zero. The optional argument `est` determines which coefficients of the SNP model are fixed at their initial values and which are estimated. The argument est is a vector of logical values with length equal to coef, the vector of starting values. If an element of est is TRUE, then the corresponding element of coef is estimated freely; if an element of est is FALSE, then the corresponding element of coef is fixed during the estimation. To re-estimate the semiparametric AR(1) with the coefficients on z_1 and z_3 equal to zero, the starting values for these coefficients must be set to zero and the corresponding elements of the vector est must be set to FALSE[8]:

```
> coef.start = fit.10014000$coef
> est.start = fit.10014000$est
> coef.start[c(2,4)] = 0
> est.start[c(2,4)] = F
> fitr.10014000 = SNP(dmRet.dat[,"exch"],model=SNP.model(ar=1,
+                     zPoly=4), n.drop=14, coef=coef.start,
+                     est=est.start)
> fitr.10014000

Call:
SNP(data = dmRet.dat[, "exch"], model = SNP.model(ar = 1,
    zPoly = 4), n.drop = 14, coef = coef.start,
    est = est.start)

Model: Semiparametric VAR

Hermite Polynomial Coefficients:
 z^0 z^1     z^2 z^3    z^4
   1   0 -0.1819   0 0.0236

Conditional Mean Coefficients:
      mu ar(1)
 -0.0244 0.051

Conditional Variance Coefficients:
  sigma
 1.0481

Information Criteria:
    BIC     HQ    AIC     logL
 1.3927 1.3839 1.3784 -1125.25

Convergence Type:
```

8Currently, the names of the SNP model coefficients are not attached to the elements of coef. However, these names can be determined using the coef() extractor function.

```
relative function convergence
number of iterations: 4
```

Example 150 *Semiparametric VAR for bivariate exchange rate data*

With multivariate time series data, the semiparametric VAR can become complicated and include many terms if K_z is large. To illustrate, consider fitting a semiparametric VAR(1) model to the bivariate exchange rate data with $K_z = 4$:

```
>  bfit.10014000 = SNP(dmRet.dat[,1:2],model=SNP.model(ar=1,
+                      zPoly=4), n.drop=14)
> summary(bfit.10014000)

Call:
SNP(data = dmRet.dat[, 1:2], model = SNP.model(ar = 1,
zPoly = 4), n.drop = 14)

Model: Semiparametric VAR

Hermite Polynomial Coefficients:
              z^00     z^01     z^10     z^02     z^11
     coef   1.0000   0.0967   0.0908  -0.3057   0.0489
(std.err)            0.0533   0.0708   0.0196   0.0355
 (t.stat)            1.8140   1.2821 -15.6251   1.3766


              z^20     z^03     z^12     z^21     z^30
     coef  -0.1703  -0.0025   0.0228  -0.0054   0.0002
(std.err)   0.0210   0.0096   0.0118   0.0130   0.0099
 (t.stat)  -8.1056  -0.2604   1.9325  -0.4126   0.0221


              z^04     z^13     z^22     z^31     z^40
     coef   0.0393  -0.0271   0.0066   0.0001   0.0211
(std.err)   0.0034   0.0070   0.0082   0.0089   0.0031
 (t.stat)  11.5137  -3.8679   0.8103   0.0084   6.7813


Conditional Mean Coefficients:
               exch   forward
       mu   -0.2372   -0.0334
(std.err)    0.1041    0.0139
 (t.stat)   -2.2772   -2.4017

 var(1;1)    0.0086   -0.1501
```

```
(std.err)    0.0283    0.0331
 (t.stat)    0.3048   -4.5398

 var(1;2)   -0.0022    0.9786
(std.err)    0.0053    0.0056
 (t.stat)   -0.4124  173.8433


Conditional Variance Coefficients:
          sigma11 sigma12 sigma22
     coef  1.0376  0.0068  0.2389
(std.err)  0.0374  0.0410  0.0033
 (t.stat) 27.7294  0.1659 72.3597


Information Criteria:
    BIC     HQ    AIC      logL
 1.3014 1.2607 1.2354 -990.0173

Convergence Type:
relative function convergence
number of iterations: 63
```

In the printout of the Hermite coefficients, the indices in the superscripts of the variable `z` correspond to the indices $\boldsymbol{\alpha} = (\alpha_1, \alpha_2)$ of the coefficients in the polynomial $\mathcal{P}(\mathbf{z})$. For example, the indices `01` and `10` correspond to $a_{(0,1)}$ and $a_{(1,0)}$, which are the coefficients on z_2 and z_1 in $\mathcal{P}(\mathbf{z})$, respectively. With $K_z = 4$, the estimated value of $\mathcal{P}(\mathbf{z})$ has the form

$$\begin{aligned}\hat{\mathcal{P}}(z) \quad = \quad & 1 + 0.0908 \cdot z_1 + 0.0967 \cdot z_2 \\ & +0.0489 \cdot z_1 z_2 - 0.1703 \cdot z_1^2 - 0.3057 \cdot z_2^2 \\ & -0.0054 \cdot z_1^2 z_2 + 0.0228 \cdot z_1 z_2^2 + 0.0002 \cdot z_1^3 - 0.0025 \cdot z_2^3 \\ & +0.0001 \cdot z_1^3 z_2 + 0.0066 \cdot z_1^2 z_2^2 - 0.0271 \cdot z_1 z_2^3 + 0.0211 \cdot z_1^4 \\ & +0.0393 \cdot z_2^4\end{aligned}$$

Notice that only the even powers of z_1 and z_2 are highly significant in the bivariate semiparametric VAR(1).

The marginal conditional densities for the spot return and the forward discount may be computed and plotted using

```
> SNP.density(bfit.10014000)
```

These densities are illustrated in Figure 22.4.

For multivariate models, one may want to eliminate interaction terms (e.g., $z_1 z_2$, $z_1^2 z_2$, $z_2 z_1^2$, etc.) in the polynomial $P(z)$ to reduce the number of estimated parameters and to improve the stability of the fit. By default,

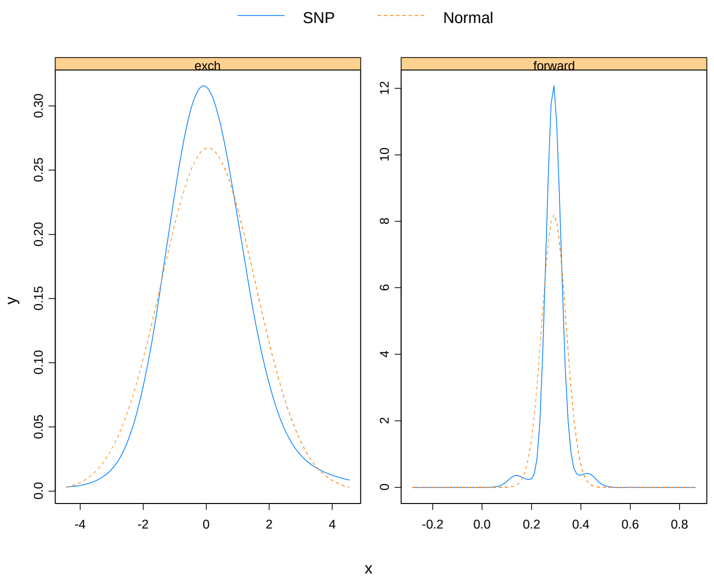


FIGURE 22.4. Estimated marginal conditional densities from bivariate semiparametric VAR(1) model for weekly spot returns and forward discounts.

all possible interactions are allowed. To suppress the highest-order interactions, set the optional `SNP.model` argument `trimZ=1`. This will suppress all interaction terms of order K_z. For example,

```
> bfit.10014100 = SNP(dmRet.dat[,1:2],model=SNP.model(ar=1,
+                     zPoly=4, trimZ=1), n.drop=14)
> bfit.10014100

Call:
SNP(data = dmRet.dat[, 1:2], model = SNP.model(ar = 1,
    zPoly= 4, trimZ = 1), n.drop = 14)

Model: Semiparametric VAR

Hermite Polynomial Coefficients:
 z^00   z^01   z^10    z^02   z^11    z^20    z^03   z^12
    1 0.0671 0.0843 -0.2938 0.0398 -0.1417 -0.0026 0.0046

    z^21   z^30   z^04   z^40
 -0.0166 0.0019 0.0414 0.0202
...
```

With `zPoly=4` and `trimZ=1`, the interaction terms $z_1 z_2^3$, $z_1^2 z_2^2$, and $z_1^3 z_2$ are suppressed. The component `idxZ` of the fitted `"SNP"` object indicates which terms are included in $\mathcal{P}(\mathbf{z})$:

```
> bfit.10014100$idxZ
      [,1] [,2]
 [1,]    0    0
 [2,]    0    1
 [3,]    1    0
 [4,]    0    2
 [5,]    1    1
 [6,]    2    0
 [7,]    0    3
 [8,]    1    2
 [9,]    2    1
[10,]    3    0
[11,]    0    4
[12,]    4    0
```

In general, all interaction terms of degree higher than `zPoly-trimZ` will be supressed. To filter out all interactions, set `trimZ` to `zPoly-1`:

```
> bfit.10014300 = SNP(dmRet.dat[,1:2],model=SNP.model(ar=1,
+                       zPoly=4, trimZ=3), n.drop=14)
> bfit.10014300

Call:
SNP(data = dmRet.dat[, 1:2], model = SNP.model(ar = 1, zPoly
= 4, trimZ = 3), n.drop = 14)

Model: Semiparametric VAR

Hermite Polynomial Coefficients:
 z^00   z^01   z^10    z^02    z^20    z^03   z^30   z^04
    1 0.0344 0.0322 -0.2884 -0.1289 -0.0008 0.0064 0.0407

   z^40
 0.0194
...
```

22.3.4 Conditional Heterogeneity

Although a semiparametric VAR density can accurately approximate densities with arbitrary shape departures from the Gaussian VAR, the approximated shape is constant with respect to variations in $\mathbf{x}_{t-1}$; that is, the semiparametric VAR density is conditionally homogeneous, because only the first moment of the density depends on $\mathbf{x}_{t-1}$. However, for economic and financial time series data, it is usually found that the underlying stochastic processes are conditionally heterogeneous, or conditionally heteroskedastic.

The SNP methodology allows one to model the conditional heterogeneity in multiple time series using two approaches, which can be used either separately or jointly. The first approach is to let the Hermite polynomial coefficients $a_{\boldsymbol{\alpha}}$ be a multivariate polynomial of order K_x in $\mathbf{x}_{t-1}$:

$$a_{\boldsymbol{\alpha}}(\mathbf{x}_{t-1}) = \sum_{|\boldsymbol{\beta}|=0}^{K_x} a_{\boldsymbol{\alpha}\boldsymbol{\beta}} \mathbf{x}_{t-1}^{\boldsymbol{\beta}}$$

with $\boldsymbol{\beta}$ denoting the non-negative multi-index $(\beta_1, \ldots, \beta_{ML})$, and

$$\begin{aligned} |\boldsymbol{\beta}| &= \beta_1 + \cdots + \beta_M, \\ \mathbf{x}_t^{\boldsymbol{\beta}} &= (x_{t,1})^{\beta_1} \times \cdots \times (x_{t,ML})^{\beta_{ML}} \end{aligned}$$

The Hermite polynomial becomes

$$P(\mathbf{z}_t, \mathbf{x}_{t-1}) = \sum_{|\boldsymbol{\alpha}|=0}^{K_z} \sum_{|\boldsymbol{\beta}|=0}^{K_x} a_{\boldsymbol{\alpha}\boldsymbol{\beta}} \mathbf{x}_{t-1}^{\boldsymbol{\beta}} \mathbf{z}_t^{\boldsymbol{\alpha}}$$

For example, let $M = 1$, $K_z = 2$, $K_x = 1$, and $x_{t-1} = y_{t-1}$. Then,

$$\mathcal{P}(z_t, x_{t-1}) = (1 + a_{0,1} y_{t-1}) + (a_{1,0} + a_{1,1} y_{t-1}) z_t + (a_{2,0} + a_{2,1} y_{t-1}) z_t^2$$

which shows that the coefficients on the powers of z are linear functions of the conditioning variable y_{t-1}. If $K_x = 2$, then the coefficients will contain linear and quadratic terms in y_{t-1}.

The SNP density function becomes

$$f(\mathbf{y}_t | \mathbf{x}_{t-1}, \boldsymbol{\theta}) \propto [\mathcal{P}(\mathbf{z}_t, \mathbf{x}_{t-1})]^2 N(\mathbf{y}_t; \boldsymbol{\mu}_{x_{t-1}}, \boldsymbol{\Omega}) \qquad (22.14)$$

where $\mathcal{P}(\mathbf{z}_t, \mathbf{x}_{t-1})$ is now a multivariate polynomial of degree $K_z + K_x$. It is referred to as the *nonlinear nonparametric VAR model*. When K_x is set to zero, the above density reduces to the semiparametric VAR density in (22.12) and thus nests semiparametric VAR as a special case. When K_x is positive, the above density can approximate any form of conditional heterogeneity in principle, because the shape and thus all moments of the density will depend on $\mathbf{x}_{t-1}$.

In the polynomial $\mathcal{P}(\mathbf{z}_t, \mathbf{x}_{t-1})$, the number of terms associated with each power of $\mathbf{z}_t$ is a function of the number of elements in $\mathbf{x}_{t-1}$. If $\mathbf{x}_{t-1}$ contains many lags of $\mathbf{y}_t$ then the number of terms in $\mathcal{P}(\mathbf{z}_t, \mathbf{x}_{t-1})$ can become very large. In applications, it may be advantageous to restrict the number of lags, L_p, of $\mathbf{x}_{t-1}$ to a small number. In addition, if K_z is large, there will be many interaction terms between the elements of $\mathbf{z}_t$. Similarly, if K_x is large there will be many interaction terms between the elements of $\mathbf{x}_{t-1}$. In applications, it may be desirable to limit the number of these interactions. The following example illustrates how to do this.

Example 151 *Nonlinear nonparametric VAR for univariate exchange rate data*

To fit a nonlinear nonparametric AR(1) model to the spot return data with $K_z = 4$ and $K_x = 1$, use

```
> fit.10014010 = SNP(dmRet.dat[,"exch"],model=SNP.model(ar=1,
+                     zPoly=4, xPoly=1), n.drop=14)
> fit.10014010

Call:
SNP(data = dmRet.dat[, "exch"], model = SNP.model(ar = 1,
    zPoly = 4, xPoly = 1), n.drop = 14)

Model: Nonlinear Nonparametric

Hermite Polynomial Coefficients:
        x^0     x^1
z^0  1.0000 -0.2118
z^1  0.0054 -0.0508
z^2 -0.1939  0.0443
z^3  0.0081 -0.0031
z^4  0.0230 -0.0039

Conditional Mean Coefficients:
     mu  ar(1)
 -0.037 0.2047

Conditional Variance Coefficients:
  sigma
 1.0899

Information Criteria:
    BIC     HQ    AIC      logL
 1.4175 1.3963 1.3831 -1122.127

Convergence Type:
relative function convergence
number of iterations: 36
```

With $K_z = 4$ and $K_x = 1$, the printout of the Hermite coefficients is a 5×2 table giving the polynomial coefficients $a_{\alpha\beta}$. The first row gives the coefficients on z^0, the second row gives the coefficients on z^1, and so on.

Since $x_{t-1} = y_{t-1}$, the estimated polynomial $P(z_t, x_{t-1})$ has the form[9]

$$\begin{aligned} P(z_t, y_{t-1}) &= (1 - 0.2118y_{t-1}) + (0.0054 - 0.0508y_{t-1})\, z_t \\ &\quad + (-0.1939 + 0.0443y_{t-1})\, z_t^2 + (0.0081 - 0.0031y_{t-1})\, z_t^3 \\ &\quad + (0.0230 - 0.0039y_{t-1})\, z_t^4 \end{aligned}$$

If xPoly is not zero, the optional argument lagP to the SNP.model function can be used to set the number of lagged values in $\mathbf{x}_{t-1}$ to go into the polynomial $P(\mathbf{z}_t, \mathbf{x}_{t-1})$. By default, lagP is set to 1 so that $\mathbf{x}_{t-1}$ only contains $\mathbf{y}_{t-1}$. To re-estimate the nonlinear nonparametric AR(1) with two lags of y_t in $\mathbf{x}_{t-1}$, use

```
> fit.10024010 = SNP(dmRet.dat[,"exch"],
+                 model=SNP.model(ar=1,zPoly=4,xPoly=1,lagP=2),
+                 n.drop=14)
> fit.10024010

Call:
SNP(data = dmRet.dat[, "exch"], model = SNP.model(ar = 1,
    zPoly = 4,xPoly = 1, lagP = 2), n.drop = 14)

Model: Nonlinear Nonparametric

Hermite Polynomial Coefficients:
       x^00    x^01    x^10
z^0  1.0000 -0.1381  0.1434
z^1  0.0094  0.0030  0.0480
z^2 -0.1956  0.0389 -0.0369
z^3  0.0073 -0.0049 -0.0074
z^4  0.0231 -0.0034  0.0021
...
```

Example 152 *Nonlinear nonparametric VAR for bivariate exchange rate data*

To fit a bivariate nonlinear nonparametric VAR(1) to the exchange rate data with $K_z = 4$, $K_x = 1$, and $L_p = 1$, use

```
> bfit.10014010 = SNP(dmRet.dat[,1:2],model=SNP.model(ar=1,
+                       zPoly=4,xPoly=1), n.drop=14)
> bfit.10014010

Call:
SNP(data = dmRet.dat[, 1:2], model = SNP.model(ar = 1,
```

[9]Keep in mind that the constant term of the polynomial is always set to one.

```
    zPoly = 4, xPoly = 1), n.drop = 14)

Model: Nonlinear Nonparametric

Hermite Polynomial Coefficients:
        x^00    x^01    x^10
z^00  1.0000 -0.1985 -0.1009
z^01  0.0605  0.0041  0.1063
z^10  0.0279 -0.0566  0.0374
z^02 -0.2736  0.0598  0.0269
z^11  0.0656 -0.0198 -0.0149
z^20 -0.1472  0.0331  0.0258
z^03 -0.0035 -0.0004 -0.0317
z^12  0.0138  0.0061 -0.0123
z^21 -0.0103 -0.0002 -0.0002
z^30  0.0072  0.0064 -0.0066
z^04  0.0338 -0.0001 -0.0042
z^13 -0.0130 -0.0013  0.0156
z^22 -0.0027  0.0132  0.0002
z^31 -0.0002  0.0021 -0.0005
z^40  0.0196 -0.0026  0.0004

Conditional Mean Coefficients:
            exch forward
      mu -0.1195 -0.0222
var(1;1)  0.0015 -0.0799
var(1;2) -0.0224  0.9721

Conditional Variance Coefficients:
 sigma11 sigma12 sigma22
  0.9789 -0.0948  0.2373

Information Criteria:
    BIC     HQ    AIC      logL
 1.3763 1.2826 1.2242 -950.8071

Convergence Type:
relative function convergence
number of iterations: 113
```

The fitted SNP model has many parameters: 6 conditional mean parameters, 3 conditional variance parameters, and 44 Hermite polynomial coefficients. The form of the estimated multivariate polynomial $\hat{\mathcal{P}}(\mathbf{z}_t, \mathbf{x}_{t-1})$ is rather complicated:

$$\begin{aligned}\hat{\mathcal{P}}(\mathbf{z}_t, \mathbf{x}_{t-1}) \;&=\; (1 - 0.1985\Delta s_{t-1} - 0.1009fd_{t-1}) \\ &\quad + (0.0279 - 0.0566\Delta s_{t-1} + 0.0374fd_{t-1})\, z_{1t} \\ &\quad + (0.0605 + 0.0041\Delta s_{t-1} + 0.1063fd_{t-1})\, z_{2t} \\ &\quad + \cdots + \\ &\quad + (0.0338 - 0.0001\Delta s_{t-1} - 0.0042fd_{t-1})\, z_{2t}^4\end{aligned}$$

In general, many of the coefficients in $\hat{\mathcal{P}}(\mathbf{z}_t, \mathbf{x}_{t-1})$ may be insignificant. In particular, many of the interaction terms between the elements of $\mathbf{z}_t$ and $\mathbf{x}_{t-1}$ may be insignificant. When zPoly>0 and xPoly>0, the number of interaction terms allowed in the Hermite polynomial is controlled by the SNP.model optional arguments trimZ and trimX. By default, trimZ=0 and trimX=0, which allows all interactions. Setting trimZ=1 will exclude interactions between the elements of $\mathbf{z}_t$ at the highest order (K_z); setting trimX=1 will exclude interactions between the elements of $\mathbf{x}_{t-1}$ at the highest order (K_x). To exclude all interaction terms, set trimZ to zPoly-1 and trimX to xPoly-1. For example, to fit a bivariate nonlinear nonparametric VAR(1) model with $K_z = 4$, $K_x = 2$ and no interaction terms, use

```
> bfit.10014321 = SNP(dmRet.dat[,1:2],
+ model=SNP.model(ar=1,zPoly=4,xPoly=2,trimZ=3,trimX=1),
+ n.drop=14)
> bfit.10014321

Call:
SNP(data = dmRet.dat[, 1:2], model = SNP.model(ar = 1,
    zPoly = 4, xPoly = 2, trimZ = 3, trimX = 1),
    n.drop = 14)

Model: Nonlinear Nonparametric

Hermite Polynomial Coefficients:
        x^00    x^01    x^10    x^02    x^20
z^00  1.0000 -0.2433 -0.4283  0.0797  0.1209
z^01  0.0359  0.0868  0.0554 -0.0240 -0.0494
z^10  0.0530  0.0260  0.0245  0.0030 -0.0163
z^02 -0.2807  0.0860  0.1495 -0.0213 -0.0360
z^20 -0.1627  0.0690  0.0364 -0.0157  0.0180
z^03 -0.0001 -0.0138 -0.0317  0.0033  0.0130
z^30  0.0090 -0.0012 -0.0003 -0.0029 -0.0007
z^04  0.0340 -0.0040 -0.0205  0.0090  0.0021
z^40  0.0173 -0.0036 -0.0017 -0.0023  0.0015
...
```

22.3.5 ARCH/GARCH Leading Term

In practice, in order to model conditional heteroskedasticity often observed in financial time series data, the order K_x of the polynomial in $\mathbf{x}_{t-1}$ can get quite large. To keep K_x small when the data are markedly conditionally heteroskedastic, the SNP methodology allows for a second approach based on using a Gaussian ARCH model or Gaussian GARCH model as the leading term in the Hermite expansion, rather than a Gaussian VAR model.

The class of ARCH/GARCH models, discussed in Chapters 7 and 13, are very successful at modeling conditional heteroskedasticity in macroeconomic and financial time series data; for example, see Engle (1995). To use the Gaussian ARCH as the leading term for the SNP method, Gallant, Hsieh, and Tauchen (1991) suggested modeling the upper triangular matrix $\mathbf{R}$ as a linear function of the absolute values of the lagged elements of $\mathbf{y}_t - \boldsymbol{\mu}_{x_{t-1}}$:

$$\text{vech}(\mathbf{R}_{x_{t-1}}) = \boldsymbol{\rho}_0 + \sum_{i=1}^{L_r} \mathbf{P}_i |\mathbf{y}_{t-1-L_r+i} - \boldsymbol{\mu}_{x_{t-2-L_r+i}}| \tag{22.15}$$

where vech($\mathbf{R}$) denotes a vector of length $M(M+1)/2$ containing the elements of the upper triangle of $\mathbf{R}$, $\boldsymbol{\rho}_0$ is a vector of length $M(M+1)/2$, and $\mathbf{P}_i$ is an $M(M+1)/2 \times M$ matrix for $i = 1, \ldots, L_r$. The expression (22.15), and its extensions, is referred to as the *conditional variance equation.*

Equation (22.15) represents the SNP version of the L_r-th order ARCH model, or ARCH(L_r) model. However, the classical ARCH model has $\mathbf{R}_{x_{t-1}}$ depending on a linear function of squared lagged residuals instead of absolute values. The SNP version of ARCH is more akin to the suggestions of Taylor (1986) and Davidian and Carroll (1987). In practice, this version has been shown to perform better than the classical ARCH model, for example, see Schwert (1990) and Ding, Granger, and Engle (1993).

The Gaussian ARCH leading term can usually be further improved by adding a Gaussian GARCH leading term:

$$\begin{aligned} \text{vech}(\mathbf{R}_{x_{t-1}}) &= \boldsymbol{\rho}_0 + \sum_{i=1}^{L_r} \mathbf{P}_i |\mathbf{y}_{t-1-L_r+i} - \boldsymbol{\mu}_{x_{t-2-L_r+i}}| \\ &\quad + \sum_{i=1}^{L_g} \text{diag}(\mathbf{G}_i)\text{vech}(\mathbf{R}_{x_{t-2-L_g+i}}) \end{aligned} \tag{22.16}$$

where $\mathbf{G}_i$ is a vector of length $M(M+1)/2$ for $i = 1, \ldots, L_g$. This equation represents a GARCH model with order (L_r, L_g), where L_r gives the ARCH order and L_g gives the GARCH order.

In summary, a Hermite polynomial with coefficients modeled as a polynomial in terms of $\mathbf{x}_{t-1}$, which is referred to as "state dependence," and a Gaussian ARCH/GARCH leading term can be used either separately or jointly to model conditional heterogeneity.[10] The resulting SNP density has the following form:

$$f(\mathbf{y}_t|\mathbf{x}_{t-1},\boldsymbol{\theta}) \propto \mathcal{P}(\mathbf{z}_t,\mathbf{x}_{t-1})^2 N(\mathbf{y}_t|\boldsymbol{\mu}_{x_{t-1}},\boldsymbol{\Omega}_{x_{t-1}})$$

where $\boldsymbol{\Omega}_{x_{t-1}} = \mathbf{R}_{x_{t-1}} \cdot \mathbf{R}'_{x_{t-1}}$. When K_z (and thus K_x) is set to zero, the SNP density becomes a *Gaussian GARCH model*; when K_z is positive and K_x is zero, the SNP density becomes a *semiparametric GARCH model* similar to Engle and González-Rivera (1991); when both K_z and K_x are positive, the resulting nonparametric density also allows state dependence. In general, the model parameter $\boldsymbol{\theta}$ contains the GARCH coefficients in addition to the coefficients $a_{\boldsymbol{\alpha}\boldsymbol{\beta}}$ of the Hermite polynomial, plus $\boldsymbol{\mu}_0$ and $\mathbf{B}$.

In practice GARCH processes have been very successful in modeling the conditional heteroskedasticity in univariate time series. However, for a multivariate time series, the GARCH structure requires many parameters in the conditional variance equation. In addition, it may be difficult to obtain convergence of the maximum likelihood estimate, and if convergence occurs, the resulting estimates may be sensitive to starting values. Experience has shown that the GARCH(1,1) process is a useful starting point for analysis. To improve the performance of SNP models with a GARCH structure and to ease the computational burden for multiple time series modeling, Gallant and Tauchen (2001) recommended imposing two simplifying restrictions: (1) restrict the ARCH coefficient matrix so that the diagonal elements of $\mathbf{R}_{x_{t-1}}$ only depend on the corresponding elements of $|\mathbf{y}_{t-1-L_r+i} - \boldsymbol{\mu}_{x_{t-2-L_r+i}}|$ and (2) restrict the elements of each GARCH coefficient matrix, $\mathbf{G}_i$, to be the same.

In the S+FinMetrics implementation of the SNP methodology, a *diagonal VEC multivariate GARCH model* is also available. This model for the conditional variance, discussed in Section 3 of Chapter 13, has the form

$$\begin{aligned}\text{vech}(\boldsymbol{\Omega}_{x_{t-1}}) &= \boldsymbol{\rho}_0 + \sum_{i=1}^{L_r} \text{diag}(\mathbf{P}_i)\text{vech}\left(\boldsymbol{\varepsilon}_{t-i}\boldsymbol{\varepsilon}'_{t-i}\right) \qquad (22.17)\\ &\quad + \sum_{i=1}^{L_g} \text{diag}(\mathbf{G}_i)\text{vech}(\boldsymbol{\Omega}_{x_{t-2-L_g+i}})\end{aligned}$$

where $\boldsymbol{\varepsilon}_t = \mathbf{y}_t - \boldsymbol{\mu}_{\mathbf{x}_{t-1}}$.

Example 153 *Gaussian ARCH/GARCH model for univariate exchange rate data*

[10]However, when the Gaussian GARCH leading term is used (i.e., when $L_g > 0$) the SNP model is no longer Markovian, because one must know lagged values of $\mathbf{R}_{\mathbf{x}_{t-1}}$ as well as $\mathbf{x}_{t-1}$ to iterate forward for $\mathbf{y}_t$.

To model conditional heteroskedasticity in spot returns with a GARCH(1,1) leading term, use

```
>  fit.11110000 = SNP(dmRet.dat[,"exch"],model=SNP.model(ar=1,
+                       arch=1,garch=1), n.drop=14)
```

The returned object `fit.11110000` represents the fit of an AR(1)-GARCH(1,1) model, where the optional arguments `arch` and `garch` to the `SNP.model` function specify the ARCH order L_r and GARCH order L_g, respectively. Since the model is univariate, the conditional variance equation (22.16) has the form

$$\sigma_t = \rho_0 + p_1|y_{t-1} - \mu_{x_{t-2}}| + g_1\sigma_{t-1}$$

where $\mu_{x_{t-2}} = \mu_0 - by_{t-2}$. A summary of the estimated model is

```
> summary(fit.11110000)

Call:
SNP(data = dmRet.dat[, "exch"], model = SNP.model(ar = 1,
    arch = 1, garch = 1), n.drop = 14)

Model: Gaussian GARCH

Conditional Mean Coefficients:
              mu  ar(1)
     coef 0.0180 0.0911
(std.err) 0.0259 0.0333
 (t.stat) 0.6929 2.7351


Conditional Variance Coefficients:
               s0 arch(1) garch(1)
     coef  0.0783  0.1881  0.7822
(std.err)  0.0162  0.0229  0.0289
 (t.stat)  4.8174  8.2272 27.0207


Information Criteria:
    BIC     HQ    AIC      logL
 1.3635 1.3546 1.3491 -1101.283

Convergence Type:
relative function convergence
number of iterations: 25
```

The estimated ARCH and GARCH coefficients are $p_1 = 0.1881$ and $g_1 = 0.7822$, respectively, and are highly significant.[11]

A semiparametric GARCH model, or state dependence model, can also be fitted by setting `zPoly` or `xPoly` to a non-negative integer as shown in earlier examples.

Example 154 *Gaussian ARCH/GARCH model for bivariate exchange rate data*

For the bivariate exchange rate data, consider fitting a VAR(1)-GARCH(1,1) using the commands

```
> bfit.11110000.r2 = SNP(dmRet.dat[,1:2],
+                    model=SNP.model(ar=1,arch=1,garch=1),
+                    control=SNP.control(dvec=F),
+                    n.drop=14)
```

The `SNP.control` optional argument `dvec=F` turns off the default diagonal VEC structure. The fitted model coefficients are

```
> bfit.11110000

Call:
SNP(data = dmRet.dat[, 1:2], model = SNP.model(ar = 1,
    arch = 1, garch = 1), n.drop = 14, control =
    SNP.control(darch = F, rgarch = F, dvec = F))

Model: Gaussian GARCH

Conditional Mean Coefficients:
            exch forward
      mu -0.0139 -0.0104
var(1;1)  0.0795 -0.0836
var(1;2) -0.0058  0.9946

Conditional Variance Coefficients:
              R11     R12     R22
       s0  0.0967  0.0312  0.0152
arch(1;1)  0.2066 -0.0504 -0.0065
arch(1;2) -0.0350 -0.2094  0.2656
 garch(1)  0.7546  0.0002  0.7662

Information Criteria:
    BIC     HQ    AIC      logL
```

[11]Since the absolute value is used in the ARCH specification, the usual condition $p_1 + g_1 < 1$ does not determine stationarity of the model.

```
 1.1378 1.1059 1.0861 -872.6075

Convergence Type:
relative function convergence
number of iterations: 47
```

The ARCH and GARCH coefficients are given in the Conditional Variance Coefficients table. The form of the estimated conditional variance equation (22.16) has coefficient matrices

$$\hat{\boldsymbol{\rho}}_0 = \begin{pmatrix} 0.0967 \\ 0.0312 \\ 0.0152 \end{pmatrix},\ \hat{\mathbf{P}}_1 = \begin{pmatrix} 0.2066 & -0.0350 \\ -0.0504 & -0.2094 \\ -0.0065 & 0.2656 \end{pmatrix},\ \hat{\mathbf{G}}_1 = \begin{pmatrix} 0.7546 \\ 0.0002 \\ 0.7662 \end{pmatrix}$$

To estimate the model imposing a diagonal ARCH structure on $\mathbf{P}_1$, set the `SNP.control` optional argument `darch=T`, and to estimate the model imposing common GARCH parameters in $\mathbf{G}_1$, set the optional argument `rgarch=T`:

```
> bfit.11110000.r2 = SNP(dmRet.dat[,1:2],
+      model=SNP.model(ar=1,arch=1,garch=1),
+      control=SNP.control(darch=T,rgarch=T,dvec=F),
+      n.drop=14)
> bfit.11110000.r2

Call:
SNP(data = dmRet.dat[, 1:2], model = SNP.model(ar = 1,
    arch = 1, garch = 1),n.drop = 14, control =
    SNP.control(darch = T, rgarch = T, dvec = F))

Model: Gaussian GARCH

Conditional Mean Coefficients:
            exch forward
      mu -0.0079 -0.0086
var(1;1)  0.0712 -0.0853
var(1;2) -0.0084  0.9967

Conditional Variance Coefficients:
             R11     R12    R22
       s0 0.0856 -0.0076 0.0109
arch(1;1) 0.2034  0.0000 0.0000
arch(1;2) 0.0000  0.0000 0.2689
 garch(1) 0.7637  0.7637 0.7637

Information Criteria:
    BIC     HQ   AIC      logL
 1.1195 1.0982 1.085 -877.7125

Convergence Type:
relative function convergence
number of iterations: 37
```

In the diagonal ARCH with restricted GARCH coefficients, the estimated coefficient matrices for the conditional variance equation are

$$\hat{\boldsymbol{\rho}}_0 = \begin{pmatrix} 0.0856 \\ -0.0076 \\ 0.0109 \end{pmatrix}, \ \hat{\mathbf{P}}_1 = \begin{pmatrix} 0.2034 & 0 \\ 0 & 0 \\ 0 & 0.2689 \end{pmatrix}, \ \hat{\mathbf{G}}_1 = \begin{pmatrix} 0.7637 \\ 0.7637 \\ 0.7637 \end{pmatrix}$$

To estimate a VAR-DVEC(1,1) model, which is the default multivariate GARCH model, use

```
> bfit.11110000.dvec = SNP(dmRet.dat[,1:2],
+      model=SNP.model(ar=1,arch=1,garch=1),
+      control=SNP.control(dvec=T),
+      n.drop=14)
> bfit.11110000.dvec

Call:
SNP(data = dmRet.dat[, 1:2], model = SNP.model(ar = 1,
    arch = 1, garch = 1), n.drop = 14,
    control = SNP.control(dvec = T))

Model: Gaussian GARCH

Conditional Mean Coefficients:
            exch forward
      mu -0.0055 -0.0082
var(1;1)  0.0352 -0.0828
var(1;2) -0.0052  0.9946

Conditional Variance Coefficients:
            R11    R12    R22
      s0 0.2894 0.0339 0.0218
 arch(1) 0.3222 0.3364 0.3653
garch(1) 0.8440 0.0470 0.9292

Information Criteria:
    BIC     HQ    AIC      logL
 1.0902 1.0636 1.0471 -843.6081
```

```
Convergence Type:
relative function convergence
number of iterations: 112
```

Common GARCH coefficients may be imposed by setting `rgarch=T`.

For fitting SNP models with a multivariate GARCH structure, Gallant and Tauchen (2001) recommended starting with a highly restricted specification to obtain a stable fit and then moving on to more general processes.

22.4 SNP Model Selection

The examples in the previous sections showed how to fit different SNP models using the `S+FinMetrics` function `SNP`. It was seen that the SNP model can become increasingly more complex by setting different model parameters, such as `ar`, `arch`, `garch`, `zPoly`, and `xPoly`, to large values. Due to the nested structure of SNP models, model selection can be conducted using conventional information criteria, such as the Akaike Information Criterion (AIC), the Bayesian Information Criterion (BIC), or the Hannan-Quinn (HQ) criterion. Model selection strategies for determining the best fitting SNP models are discussed in Fenton and Gallant (1996a, 1996b), Coppejans and Gallant (2002), and Gallant and Tauchen (2001). The general consensus from these articles is that minimizing the BIC

$$\mathrm{BIC}(p_{\boldsymbol{\theta}}) = s_n(\boldsymbol{\theta}) + \frac{p_{\boldsymbol{\theta}}}{2n}\ln(n)$$

where $s_n(\boldsymbol{\theta})$ is the average log-likelihood (22.13) and $p_{\boldsymbol{\theta}}$ denotes the number of parameters in the SNP model, performs best.

The model selection strategy entails moving upward along an expansion path. For univariate models, Gallant and Tauchen (2001) suggest the search order:

[1] Determine best VAR order L_u.

[2] Determine best ARCH and GARCH orders L_r and L_g.

[3] Determine best z polynomial order K_z (start at $K_z = 4$).

[4] Determine the best x polynomial order K_x.

The fitted SNP models become more richly parameterized at each level along the path. At each step of the expansion, Gallant and Tauchen recommended using the previously fit (smaller) model coefficients as starting values for the currently fit model. In addition, for each SNP model fit, they recommended that a random restart procedure of the optimizer be implemented to avoid getting stuck at potential local minima. The function SNP implements the random restart procedure advocated by Gallant and Tauchen, and is discussed in detail below.

The suggested search procedure is not an exhaustive search of all possible models and may not produce the model that globally minimizes BIC. Furthermore, Fenton and Gallant (1996a) found that the strategy of minimizing BIC often produced an overly conservative model. In particular, BIC had a tendency to select models with $K_x = 0$ in situations in which one would expect $K_x > 0$. As a result, Gallant and Tauchen recommended that a battery of residual and graphical diagnostics be run on any tentative model. Section 22.5 discusses SNP diagnostics and provides some examples.

Model selection issues in multivariate SNP models have not been studied as closely as model selection in univariate models. The main complications are with the form of the multivariate ARCH/GARCH model and the form of the multivariate Hermite polynomial $\mathcal{P}(\mathbf{z}_t, \mathbf{x}_{t-1})$. Multivariate ARCH/GARCH models are notoriously unstable. As a result, Gallant and Tauchen (2001) advocated first fitting a highly restricted specification (e.g., GARCH(1,1) with common GARCH coefficients or DVEC-GARCH(1,1)) to obtain a stable fit and then moving on to more general processes. With respect to the specification of $\mathcal{P}(\mathbf{z}_t, \mathbf{x}_{t-1})$, Gallant and Tauchen recommended starting with models that do not allow any interactions (i.e., setting trimZ to zPoly-1 and trimX to xPoly-1) and then using BIC to determine if any interactions need to be included.

In S+FinMetrics, the functions expand and SNP.auto are provided to facilitate model selection strategies to determine the best fitting SNP model. The function expand allows one to expand an existing SNP model toward a larger model, and the function SNP.auto implements Gallant and Tauchen's suggested model selection strategy for determining the best fitting univariate SNP model. These functions are described in the following subsections.

22.4.1 Random Restarts

The function SNP implements the random restart procedure described in Gallant and Tauchen (2001, Sec. 3). The procedure is best described through an example. Consider fitting an AR(1)-ARCH(3) SNP model to the spot return data using SNP. First, fit a simple AR(1) model using

```
> fit.10010000 = SNP(dmRet.dat[,"exch"],model=SNP.model(ar=1),
+ n.drop=14)
> fit.10010000

Call:
SNP(data = dmRet.dat[, "exch"], model = SNP.model(ar = 1),
    n.drop = 14)
```

```
Model: Gaussian VAR

Conditional Mean Coefficients:
     mu  ar(1)
 0.0005 0.0222

Conditional Variance Coefficients:
  sigma
 1.0025

Information Criteria:
    BIC     HQ    AIC    logL
 1.4337 1.4284 1.4251 -1165.6

Convergence Type:
both x and relative function convergence
number of iterations: 2
```

Since the quasilikelihood for an AR(1) model is a quadratic function of $\boldsymbol{\theta}$, random restarts are not needed to check for local minima.

Next, consider fitting an AR(1)-ARCH(3) without random restarts using the coefficients from the AR(1) as starting values:

```
> fit.10310000 = SNP(dmRet.dat[,"exch"],model=SNP.model(ar=1),
+ iArch=3,coef=fit.10010000$coef,n.drop=14)
> fit.10310000

Call:
SNP(data = dmRet.dat[, "exch"], model = SNP.model(ar = 1),
    n.drop = 14, coef = fit.10010000$coef, iArch = 3)

Model: Gaussian ARCH

Conditional Mean Coefficients:
    mu  ar(1)
 0.008 0.1362

Conditional Variance Coefficients:
     s0 arch(1) arch(2) arch(3)
 0.5059  0.2655  0.1764  0.2152

Information Criteria:
    BIC     HQ    AIC      logL
 1.3735 1.3629 1.3563 -1106.182

Convergence Type:
```

```
relative function convergence
number of iterations: 9
```

The optional argument `iArch=3` tells the `SNP` function to increment the ARCH component of the initial model specified in the call to `SNP.model` by 3. When one or more of the optional increment arguments (`iAr`, `iArch`, `iGarch`, `iZpoly`, `iXpoly`, `itrimZ`, `itrimX`) to `SNP` are utilized, the model specified in the call to `SNP.model` is called the *initial model*, and the model fit is called the *expanded model*. By setting `coef=fit.10010000$coef`, the `SNP` function uses the AR(1) model coefficients as starting values for the coefficients in the initial model and sets the starting values for the additional ARCH(3) coefficients to zero. The coefficients for the initial model are referred to as the "old" coefficients, and additional coefficients specified by increment arguments (e.g., `iArch=3`) are referred to as the "new" coefficients.[12]

The same fit to the AR(1)-ARCH(3) may be obtained using

```
> fit.10310000b = SNP(dmRet.dat[,"exch"],model=SNP.model(ar=1,
+                         arch=3), n.drop=14)
```

The reason for this is that, by default, the `SNP` function uses a VAR(p) model (where p is specified by the `ar` component of `SNP.model`) to generate starting values for any specified SNP model. As a result, if the starting value and model increment arguments are not specified in the call to `SNP`, then the "old" coefficients are the coefficients to the VAR(p) and the "new" coefficients are the remaining parameters of the specified SNP model.

Now, consider refitting the AR(1)-ARCH(3) using Gallant and Tauchen's random restart procedure. The restart procedure involves refitting the AR(1)-ARCH(3) model using randomly perturbed starting values for both the old (initial model) coefficients and the new (expanded model) coefficients. The user has control over how the old and new coefficients are perturbed, as well as how many random perturbations are performed. The random perturbations of the start values are generated using the scheme

$$\begin{array}{rcl} \rho_i^{\text{old}} & \rightarrow & \rho_i^{\text{old}} \cdot (1 + u \cdot \text{tweak}^{\text{old}}) \\ \rho_i^{\text{new}} & \rightarrow & \rho_i^{\text{new}} \cdot (1 + u \cdot \text{tweak}^{\text{new}}) \end{array} \tag{22.18}$$

where u is a random variable distributed as $U[-1, 1]$, and tweak$^{\text{old}}$ and tweak$^{\text{new}}$ are so-called tweak constants. For each random restart, the starting values are perturbed as in (22.18), and the SNP model optimization is run for a specified number of iterations to explore the surface of the SNP objective function (22.13). At convergence (or last iteration, whichever comes first), the SNP objective function value is recorded, and the restart

[12]This example is given to illustrate how the `SNP` function distinguishes between old and new parameters. In practice, the user would not call `SNP` with the increment arguments. Instead, the user would call the `expand` function discussed in the next subsection.

procedure is repeated. Of all the randomly restarted fits, the best fit is determined as the one with the lowest SNP objective function value. If necessary, this fit is then iterated to convergence.

In the function `SNP`, the random restart procedure is controlled through the `SNP.control` optional arguments `fOld`, `fNew`, `n.start`, `initial.itmax`, `final.itmax`, and `seed`. The arguments `fOld` and `fNew` determine the tweak constants $\text{tweak}^{\text{old}}$ and $\text{tweak}^{\text{new}}$, respectively, and `n.start` determines the number of random restarts. The argument `initial.itmax` sets the maximum number of iterations for each random-restart, and `final.itmax` sets the maximum number of iterations for the best of the randomly restarted models. The `seed` argument sets the random number seed used to generate u.

For example, to refit the AR(1)-ARCH(3) using 25 random restarts with $\text{tweak}^{\text{old}} = \text{tweak}^{\text{new}} = 0.1$, use

```
> fit.10310000.restart = SNP(dmRet.dat[,"exch"],
+     model=SNP.model(ar=1,arch=3), n.drop=14,
+     control=SNP.control(fOld=0.1,fNew=0.1,n.start=25),
+     trace=T)
random start 1, objective = 1.349
random start 2, objective = 1.349
...
random start 25, objective = 1.349
Iteration No. 1, objective = 1.349
```

The optional argument `trace=T` allows the progress of the restart procedure to be monitored on screen.[13] For each random restart the SNP objective function value (22.13) is printed. After the best fit is determined, it is iterated to convergence. In the above output, it appears that all of the randomly restarted fits are very close.

The above example only utilized one tweak constant value for the old and new parameters. In general, Gallant and Tauchen recommended that a variety of tweak constants of various magnitudes be used to thoroughly explore the surface of the SNP objective function. To accommodate this recommendation, the arguments `fOld`, `fNew`, and `n.start` can be vectors. For example, to implement a wave of random restarts using the collection of tweak constants specified by Gallant and Tauchen (2001, p. 27), use

```
> fOld = c(0, 1e-5, 0, -1e-5, 1e-5, 1e-5, 1e-4, 0, -1e-4,
+          1e-4, 1e-4, 1e-3, 0, -1e-3, 1e-3, 1e-3, 1e-5, 0,
+          -1e-2, 1e-2, 1e-2, 1e-1, 0, -1e-1, 1e-1, 1e-1,
+          1e0, 0, -1e0, 1e0, 1e0)
```

[13]On Windows systems, the trace information may be printed to the screen all at once at the end of the restart procedure. This occurs because of the way screen output is buffered in the Windows environment.

```
> fNew = c(0, 0, 1e-5, 1e-5, -1e-5, 1e-5, 0, 1e-4, 1e-4,
+          -1e-4, 1e-4, 0, 1e-3, 1e-3, -1e-3, 1e-3, 0,
+          1e-2, 1e-2, -1e-2, 1e-2, 0, 1e-1, 1e-1, -1e-1,
+          1e-1, 0, 1e0, 1e0, -1e0, 1e0)
> n.start = c(0,rep(25,30))
```

The vectors `fOld` and `fNew` specify 31 different tweak constant combination. The vector `n.start`, which is the same length as `fOld` and `fNew`, specifies how many random restarts are to be performed for each combination of tweak constants. Notice that the first set of tweak constants are zero and there are no random restarts. This corresponds to the SNP fit without a random restart. To estimate the SNP model using the above list of restart parameters use

```
> fit.10310000.mruns = SNP(dmRet.dat[,"exch"],
+     model=SNP.model(ar=1, arch=3),
+     control = SNP.control(n.start = n.start,
+     seed = 011667, fOld = fOld, fNew = fNew),
+     trace = T)

Run 1 , 0 starts, fOld = 0 fNew = 0
Iteration No. 1, objective = 1.52188
Iteration No. 2, objective = 1.38215
...
Iteration No. 9, objective = 1.34811

Run 2 , 25 starts, fOld = 1e-005 fNew = 0
random start 1, objective = 1.34811
random start 2, objective = 1.34811
...
Run 31 , 25 starts, fOld = 1 fNew = 1
random start 1, objective = 1.34811
random start 2, objective = 1.34811
...
random start 25, objective = 1.48451
Iteration No. 1, objective = 1.34811
```

With the above restart specification, the SNP model is fit $1+30\times 25 = 751$ times.[14] The best fitting model is given by

```
> fit.10310000.mruns

Call:
SNP(data = dmRet.dat[, "exch"], model = SNP.model(ar = 1,
```

[14]Estimation of these 751 models takes about 20 s on a 1.6-GHz Pentium M under Windows XP Pro.

```
    arch = 3), control = SNP.control(n.start = n.start,
    seed = 11667, fOld = fOld, fNew = fNew), trace = F)

Minimum BIC (out of 31 runs) occurred for control parameters
 fOld fNew n.start
    1    0      25

Model: Gaussian ARCH

Conditional Mean Coefficients:
     mu  ar(1)
 0.0059 0.1336

Conditional Variance Coefficients:
     s0 arch(1) arch(2) arch(3)
 0.5078  0.2631  0.1707   0.218

Information Criteria:
    BIC     HQ    AIC      logL
 1.3724 1.3619 1.3553 -1118.931

Convergence Type:
relative function convergence
number of iterations: 1
```

With multiple runs of random restarts, the `print` method displays the optimal tweak constants (`fOld=1`, `fNew=0`). The fit information from each run is contained in the component `runs`, which is a list with components

```
> names(fit.10310000.mruns$runs)
[1] "nruns"  "best"   "detail" "coef"
```

The best fit occurs for run 27,

```
> fit.10310000.mruns$runs$best
[1] 27
```

and the information for this run is

```
> fit.10310000.mruns$runs$detail[27,]
 fOld fNew n.start      BIC       HQ      AIC      logL
    1    0      25 1.372405 1.361883 1.355339 -1118.931
```

22.4.2 The `expand` Function

The function `expand` is designed to allow a user to easily expand an existing SNP model toward a larger model. The arguments expected by `expand` are

```
> args(expand)
function(x, ar = 0, arch = 0, garch = 0, lagP = 0, zPoly = 0,
xPoly = 0, trimZ = 0, trimX = 0, control = NULL, trace
= F)
```

The argument `x` denotes a fitted `"SNP"` object from which the model expansion is to take place. The arguments `ar`, `arch`, `garch`, `lagP`, `zPoly`, `xPoly`, `trimZ`, and `trimX` give the desired increments to the SNP tuning parameters of the initial SNP model that define the new, expanded SNP model. For example, to expand from a Gaussian AR(3)-ARCH(1) to a semiparametric AR(4)-ARCH(4) with $K_z = 4$ the increment specification would be `ar=1`, `arch=3`, and `zPoly=4`. The `expand` function calls the `SNP` function using the coefficients of the supplied `"SNP"` object as starting values, and it expands the SNP model according to the specified increments in a manner similar to that described in the previous subsection. By default, the `expand` function uses the `control` information from the supplied `"SNP"` object. The result of `expand` is an object of class `"SNP"`.

To illustrate, consider expanding the initial Gaussian AR(1) SNP model fit to the spot return data to a Gaussian AR(1)-ARCH(3) model. The Gaussian AR(1) model is represented by the `"SNP"` object `fit.10010000`. To expand this model to an AR(1)-ARCH(3), use

```
> fit.10310000 = expand(fit.10010000,arch=3)
> class(fit.10310000)
[1] "SNP"
> fit.10310000

Call:
SNP(data = dmRet.dat[, "exch"], model = list(ar = 1, arch = 0,
 garch = 0, lagP = 1, zPoly = 0, xPoly = 0, trimZ = 0,
 trimX = 0), n.drop = 14,coef = c(1., 0.000508918280817768,
0.0222361334180125, 1.00252754954042), est = c(0, 1, 1, 1),
 iArch = 3, trace = F)

Model: Gaussian ARCH

Conditional Mean Coefficients:
    mu  ar(1)
 0.008 0.1362

Conditional Variance Coefficients:
     s0 arch(1) arch(2) arch(3)
 0.5059  0.2655  0.1764  0.2152

Information Criteria:
    BIC     HQ    AIC      logL
 1.3735 1.3629 1.3563 -1106.182

Convergence Type:
relative function convergence
number of iterations: 9
```

The result of `expand` is an object of class `"SNP"` representing the expanded SNP fit. In the `Call` information, notice that the initial model is specified as an AR(1) and that the starting values for the initial model (old) parameters are set to the estimated coefficients from the AR(1) model. The expansion to the AR(1)-ARCH(3) is indicated through the increment argument `iArch=3`.

When expanding a SNP fit, Gallant and Tauchen (2001) stressed that random restarts be used to avoid getting stuck at local minima. When using the `expand` function, a random restart procedure may be specified in two ways. The first way is to specify a random restart procedure for the initial SNP model. This restart information, contained in the `control` component of the `"SNP"` object, will then be passed on to the `expand` function and used. The second way is to directly specify the control information passed to the `expand` function. For example, in the `"SNP"` object `fit.10010000` representing the AR(1) fit, there is no restart information in the `control` component:

```
> fit.10010000$control$fold
[1] 0
> fit.10010000$control$fnew
[1] 0
> fit.10010000$control$nstart
[1] 0
```

The vectors of restart information from the previous section may be inserted into the `control` component as follows:

```
> control.new = fit.10010000$control
> control.new$fold = fOld
> control.new$fnew = fNew
> control.new$nstart = n.start
```

The expanded model fit with random restarts may be obtained using

```
> fit.10310000.mruns = expand(fit.10010000,arch=3,
+ control=control.new,trace=T)
```

The `expand` function may be used to conduct a specific-to-general model selection strategy for determining the best fitting SNP model. For each expanded model, the model selection criteria AIC, BIC, and HQ may be used to select the best fitting model. This information is available in the component `obj` of an `"SNP"` object. For example, a comparison of the model selection criteria for the AR(1) and AR(1)-ARCH(3) models may be obtained using

```
> fit.10010000$obj - fit.10310000.mruns$obj
       BIC         HQ        AIC      logL
 0.0613315 0.06654345 0.06978232 -46.66811
```

The AR(1)-ARCH(3) is preferred to the AR(1) using all of the model selection criteria.

22.4.3 The `SNP.auto` Function

The function `SNP.auto` implements Gallant and Tauchen's (2001) suggested model selection strategy for determining the best fitting univariate SNP model by minimizing the BIC. The arguments expected by `SNP.auto` are

```
> args(SNP.auto)
function(data, n.drop = NULL, control = SNP.control(), trace
= T, arMax = 4, zPolyMax = 8, xPolyMax = 4, lagPMax
= 4)
```

The first four arguments are the same as those expected by the `SNP` function. The remaining arguments, `arMax`, `zPolyMax`, `xPolyMax`, and `lagPMax`, correspond to the maximum orders of L_u, K_z, K_x, and L_p, respectively, for the prospective SNP models. The search for the best fitting SNP model follows a restricted specific-to-general model expansion path based on the specified maximum orders of a subset of SNP model parameters. Each model is fit to the same span of data and the BIC is computed.[15] The initial SNP model is a simple Gaussian location model. The SNP model is first expanded (using the `expand` function with increments of 1) toward a Gaussian VAR model with maximum lag determined by `arMax`. Then, the Gaussian VAR model is expanded to a GARCH(1,1) model. Next, the $\mathbf{z}_t$ part of the polynomial $\mathcal{P}(\mathbf{z}_t, \mathbf{x}_{t-1})$ is sequentially expanded up to the maximum order set by `zPolyMax` and then the $\mathbf{x}_{t-1}$ part of the polynomial $\mathcal{P}(\mathbf{z}_t, \mathbf{x}_{t-1})$ is sequentially expanded up to the maximum order set by `xPolyMax`. Finally, the number of lags that go into the $\mathbf{x}_{t-1}$ part of the polynomial $\mathcal{P}(\mathbf{z}_t, \mathbf{x}_{t-1})$ is sequentially expanded up to the maximum order set by `lagPMax`. The SNP model with the lowest value of BIC is returned as an object of class `"SNP"`.

To illustrate, consider using `SNP.auto` with the default parameters to find the best SNP model preferred by BIC for the spot return data:

```
> fit.auto = SNP.auto(dmRet.dat[,"exch"], n.drop=14)
```

[15]The user must set the argument `n.drop` so that sufficient presample observations are available to accommodate the maximum lags specified by `arMax` and `lagPMax`.

```
Initializing using a Gaussian model ...
Expanding the order of VAR: 1234
Expanding toward GARCH model ...
Expanding the order of z-polynomial: 12345678
Expanding the order of x-polynomial: 1234
> class(fit.auto)
[1] "SNP"
```

By default, SNP.auto displays information about the order of the expansion path on screen. The best fitting model is a semiparametric GARCH(1,1):

```
> fit.auto

Call:
SNP(data = dmRet.dat[, "exch"], model = list(ar = 0, arch = 1,
 garch = 1,lagP = 1, zPoly = 3, xPoly = 0, trimZ = 0,
 trimX = 0), n.drop = 14, coef = c(1., -0.13265782318379,
 0.00224280476347416, 0.0189674305327692, 0.122336259918009,
 0.0624215245663479, 0.171626674286156, 0.808804423426396),
 est = c(0, 1, 1, 1, 1, 1,1, 1), iZPoly = 1, trace = F)

Model: Semiparametric GARCH

Hermite Polynomial Coefficients:
 z^0    z^1     z^2   z^3    z^4
   1 -0.097 -0.1068 0.018 0.0227

Conditional Mean Coefficients:
     mu
 0.0618

Conditional Variance Coefficients:
    s0 arch(1) garch(1)
 0.037  0.1515   0.8343

Information Criteria:
    BIC     HQ    AIC      logL
 1.3437 1.3295 1.3207 -1074.972

Convergence Type:
relative function convergence
number of iterations: 16
```

By default, SNP.auto does not utilize random restarts. To incorporate random restarts at each stage of the expansion path, the user needs to supply restart information in the control argument to SNP.auto. For example, to repeat the above estimation using Gallant and Tauchen's suggested restart information, use[16]

```
> fOld = c(0, 1e-5, 0, -1e-5, 1e-5, 1e-5, 1e-4, 0, -1e-4,
+          1e-4, 1e-4, 1e-3, 0, -1e-3, 1e-3, 1e-3, 1e-5, 0,
+          -1e-2, 1e-2, 1e-2, 1e-1, 0, -1e-1, 1e-1, 1e-1,
+          1e0, 0, -1e0, 1e0, 1e0)
> fNew = c(0, 0, 1e-5, 1e-5, -1e-5, 1e-5, 0, 1e-4, 1e-4,
+          -1e-4, 1e-4, 0, 1e-3, 1e-3, -1e-3, 1e-3, 0,
+          1e-2, 1e-2, -1e-2, 1e-2, 0, 1e-1, 1e-1, -1e-1,
+          1e-1, 0, 1e0, 1e0, -1e0, 1e0)
> n.start = c(0,rep(25,30))
> fit.auto.restart = SNP.auto(dmRet.dat[,"exch"],n.drop=14,
+          control = SNP.control(n.start = n.start,
+          seed = 011667, fOld = fOld, fNew = fNew))
```

The search procedure employed by `SNP.auto` is not an exhaustive search of all possible models and may not produce the model that globally minimizes BIC. As mentioned earlier, the BIC has a tendency to select models with $K_x = 0$ in situations in which one would expect $K_x > 0$. As a result, it is recommend that a battery of residual and graphical diagnostics be run on any tentative model to confirm the adequacy of the model. The next section describes how to do this.

The function `SNP.auto` can be used on multivariate data, but it is not recommended to do so. The default search procedure may not produce satisfactory results for a variety of reasons. In particular, the default DVEC-GARCH(1,1) model imposes no restrictions on the ARCH/GARCH parameters. Moreover, the default values for `trimZ` and `trimX` allow all interactions among the elements of $\mathbf{z}_t$ and $\mathbf{x}_{t-1}$, respectively, in the Hermite polynomial $\mathcal{P}(\mathbf{z}_t, \mathbf{x}_{t-1})$.

22.5 SNP Model Diagnostics

Although model selection criteria may be used to determine the apparent best fitting SNP model, it is still important to perform diagnostic checks on the fitted model to verify that the chosen model is adequate and appropriate. This is particularly important when the fitted SNP model is to be used as a score generator (auxiliary model) for EMM estimation. In this context, for the EMM estimation to work well, it is imperative that the SNP density capture all of the salient features (correlation structure, conditional heteroskedasticity, asymmetry, fat tails, etc.) of the observed data. Two types of diagnostic check should be routinely conducted: residual

[16]Since each model in the expansion is restarted 751 times, this may take a while!

analysis, and SNP model simulation. The S+FinMetrics implementation of SNP provides convenient method functions to facilitate this diagnostic analysis.

22.5.1 Residual Analysis

Table 22.4 shows the generic *extractor functions* for "SNP" objects available in S+FinMetrics. To illustrate these functions, consider the Gaussian AR(1)-GARCH(1,1) SNP object fit.1111000 computed from the spot return data. To extract the estimated coefficients, use

```
> coef(fit.11110000)

Conditional Mean Coefficients:
    mu  ar(1)
 0.018 0.0911

Conditional Variance Coefficients:
     s0 arch(1) garch(1)
 0.0783  0.1881   0.7822
```

Note that S-PLUS allows "lazy evaluation" so that coef() may be used instead of coefficients().[17] To return the residuals, use

```
> resid.11110000 = residuals(fit.11110000)
> resid.11110000[1:3]
  Positions        exch
 04/18/1975  0.09765122
 04/25/1975 -0.10085499
 05/02/1975  0.04326546
```

Since underlying data, dmRet.dat, is a "timeSeries" object, the residuals are returned as a "timeSeries" object. The standardized residuals (residuals divided by an estimate of the conditional standard deviation) may also be extracted by passing the optional argument standardized:

```
> resid.11110000.std = residuals(fit.11110000,stardard=T)
```

The estimated conditional standard deviations may be extracted directly using the extractor function sigma.t.SNP:

```
> sig.11110000 = sigma.t.SNP(fit.1111000)
```

A comparison of these estimated conditional standard deviations with a simple nine-period moving average of absolute log returns, computed using

[17]Similarly, fitted() may be used instead of fitted.values() and resid() may be used instead of residuals().

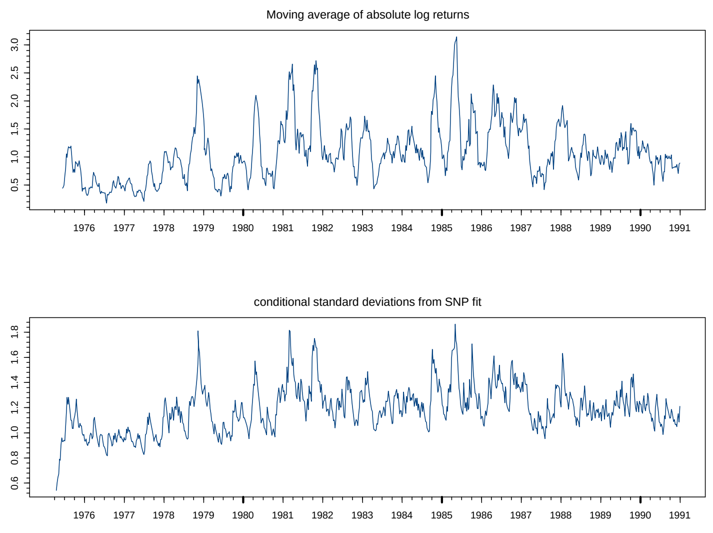


FIGURE 22.5. Nine-period moving average of absolute log returns and estimated conditional standard deviations from the Gaussian AR(1)-GARCH(1,1) SNP model.

Function	**Description**
`coefficients`	Extract the estimated coefficients
`residuals`	Extract the residuals
`fitted.values`	Extract the fitted values
`sigma.t.SNP`	Extract the estimated conditional standard deviations

TABLE 22.4. SNP extractor functions

```
> par(mfrow=c(2,1))
> plot(SMA(abs(dmRet.dat[15:834,"exch"])),
+      main="Moving average of absolute log returns",
+      reference.grid=F)
> plot(sqrt(sig.11110000),
+      main="conditional standard deviations from SNP fit",
+      reference.grid=F)
> par(mfrow=c(1,1))
```

is shown in Figure 22.5.

For multivariate models, the `sigma.t.SNP` extracts the estimated conditional standard deviations associated with each series. To extract conditional covariances, set the optional argument `cov=T`.

If the fitted SNP model adequately describes the conditional density of the observed data, then the standardized residuals should resemble a Gaussian white noise process. A variety of diagnostic functions for testing if a series behaves like Gaussian white noise are available in S-PLUS and S+FinMetrics. Chapter 3 reviews some of these functions.

Graphical diagnostics (correlograms, QQ-plots, histograms, etc.) may be conducted on the extracted residuals from an "SNP" object. For example, to plot the standardized residuals from the AR(1) SNP model for the spot return, use

```
> plot(residuals(fit.11110000,stardard=T),reference.grid=F)
> abline(h=0)
```

Alternatively, the plot method for "SNP" objects may be used to automatically generate a number of useful diagnostic plots for residual analysis. To display a menu of available plots, type

```
> plot(fit.11110000)

Make a plot selection (or 0 to exit):

1: plot: All
2: plot: Response and Fitted Values
3: plot: SNP Marginal Density
4: plot: Std. Residuals
5: plot: Normal QQplot of Std. Residuals
6: plot: ACF of Std. Residuals
7: plot: PACF of Std. Residuals
8: plot: ACF of Squared Std. Residuals
9: plot: PACF of Squared Std. Residuals
Selection:
```

For example, selecting 4 gives a trellis plot of the standardized residuals shown in Figure 22.6.

To create plots directly without using the menu, set the optional argument ask=F. For example, to create an ACF plot of the squared residuals, use[18]

```
> plot(fit.11110000,ask=F,which.plots=7)
```

This plot, displayed in Figure 22.7, shows that the Gaussian AR(1)-GARCH(1,1) has captured the autocorrelation in the conditional heteroskedasticity present in the spot return data.

For multivariate data, the diagnostic plots produced by the plot method adjust accordingly. To illustrate, consider the bivariate Gaussian VAR(1)

[18]Note that the number associated with the plot is one less than the number displayed in the menu of plot choices.

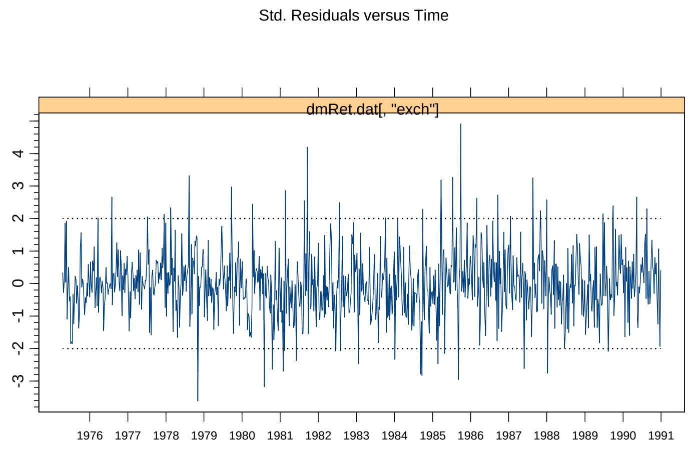


FIGURE 22.6. Standardized residuals from a Gaussian AR(1) SNP model for weekly spot returns.

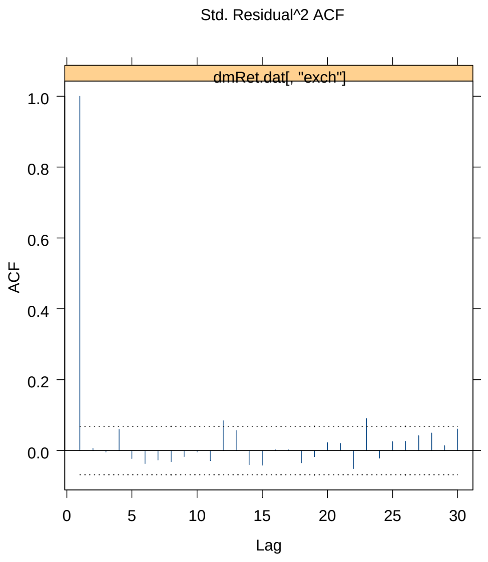


FIGURE 22.7. SACF of standardized squared residuals from a Gaussian AR(1) SNP model fit to weekly spot returns.

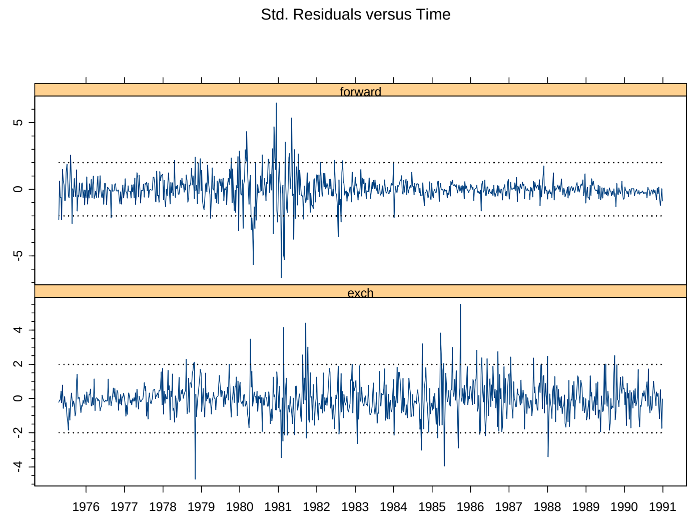


FIGURE 22.8. Standardized residuals from a bivariate Gaussian VAR(1) SNP model fit to weekly spot returns and forward discounts.

SNP model fit to the exchange rate data. Figures 22.8 and 22.9 show the plots of the standardized residuals and correlograms for the squared standardized residuals produced using

```
> plot(bfit.10010000,ask=F,which.plots=c(3,7))
```

22.5.2 Simulation

In addition to examining the residuals from a fitted SNP model, one should also examine the properties of simulated data from a fitted SNP model. If the SNP density truly captures all of the features of the observed data, then simulated data from the model should also reflect these features. Further, as emphasized in Andersen and Lund (1997), if the fitted SNP model is to be used in conjunction with EMM, it is important to check the dynamic stability of the SNP model. For complicated SNP models, it may be nearly impossible to determine if the model is dynamically stable by examining the estimated coefficients. A simple way to check dynamic stability is to generate long simulations from the fitted model and observe if these simulations become explosive.

Once an SNP model has been fit, simulated data of length equal to the observed data from the fitted model can be generated using the generic

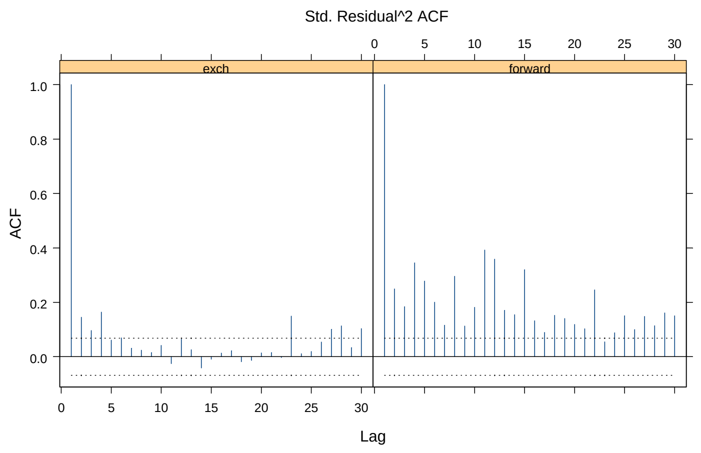


FIGURE 22.9. SACF of squared standardized residuals from a bivariate Gaussian VAR(1) SNP model fit to weekly spot returns and forward discounts.

`simulate` function. For example, to simulate from the Gaussian AR(1) SNP model fit to the spot return data, use

```
> sim.10010000 = simulate(fit.10010000)
```

The simulated return data along with the actual return data are illustrated in the top two panels of Figure 22.10. Clearly, the data simulated from the Gaussian AR(1) does not reflect the marked conditional heteroskedasticity in the actual return data. This can be further verified by comparing the correlograms of the squared returns for the two series. In contrast, the bottom panel of Figure 22.10 shows that simulations from a semiparametric AR(1)-GARCH(1,1) mimic the actual data quite well.

22.6 Prediction from an SNP Model

Once an SNP model has been fit, h-step-ahead predictions, along with estimated confidence bands, may be computed using the generic `predict` function. For example, to compute h-step-ahead predictions ($h = 1, \ldots, 10$) from the Gaussian AR(1)-GARCH(1,1) model for the spot return, use

```
> predict.11110000 = predict(fit.11110000,n.predict=10)
> class(predict.11110000)
```

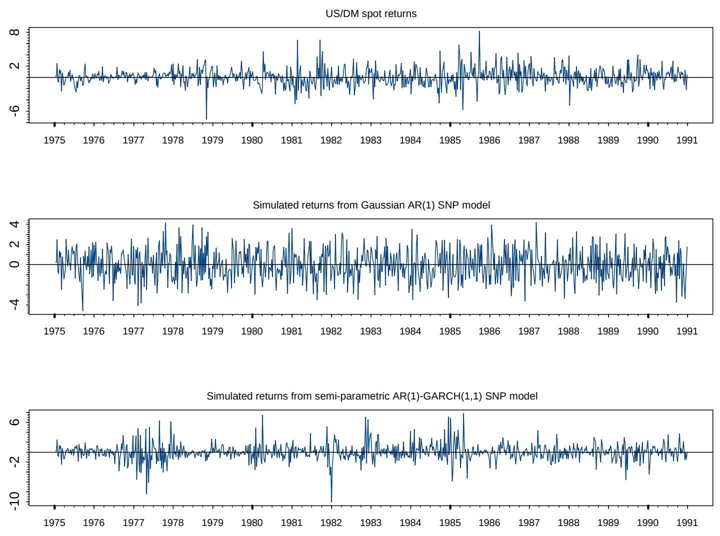


FIGURE 22.10. Actual returns and simulated returns from a Gaussian AR(1) SNP model and semiparametric AR(1)-GARCH(1,1) SNP model.

```
[1] "forecast"
```

The object returned by `predict` is of class `"forecast"`, which has `print`, `summary`, and `plot` methods.[19] The `summary` method shows the forecasts along with estimated standard errors:

```
> summary(predict.11110000)

Predicted Values with Standard Errors:

              prediction std.err
 1-step-ahead 0.0870     0.1260
 2-step-ahead 0.0863     0.1338
...
10-step-ahead 0.0882     0.1495
```

The `plot` method allows the forecasts and user specified confidence bands to be plotted along with the original data. For example, to plot the spot return forecasts appended to the actual data along with 95% confidence bands use

[19]Objects of class `"forecast"` are defined and used in `S+FinMetrics`. See the online help for `forecast.object`.

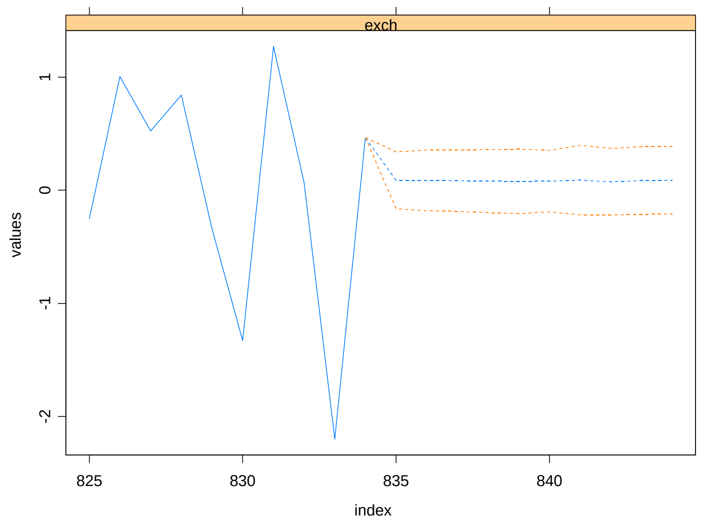


FIGURE 22.11. h-step-ahead forecasts from a Gaussian AR(1)-GARCH(1,1) SNP model.

```
> plot(predict.11110000,dmRet.dat[,"exch"],n.old=10,width=2)
```

The resulting plot is illustrated in Figure 22.11. The optional arguments `n.old=10` and `width=2` specifies that the forecasts are appended to the 10 most recent observations and that the confidence band has width equal to two times the estimated forecast standard error.

22.7 Data Transformations

A number of data transformations may be employed to improve the stability of the fitted SNP models. These transformations are described in the following subsections. The `S-PLUS` code is structured so that the transformations are transparent to the user and all input and output is in the units of the untransformed data.

22.7.1 Centering and Scaling Transformation

To improve the stability of computation, the original data for SNP estimation are centered and scaled to have mean zero and identity covariance matrix. This location-scale transform is accomplished by computing estimates of the unconditional mean and variance of the original data:

$$\bar{\mathbf{y}} = 1/T \sum_{t=1}^{T} \mathbf{y}_t$$

$$\mathbf{S} = 1/T \sum_{t=1}^{T} (\mathbf{y}_t - \bar{\mathbf{y}})(\mathbf{y}_t - \bar{\mathbf{y}})'$$

The SNP model is then estimated using the standardized data:

$$\tilde{\mathbf{y}}_t = \mathbf{S}^{-1/2}(\mathbf{y}_t - \bar{\mathbf{y}}) \tag{22.19}$$

where $\mathbf{S}^{-1/2}$ denotes the Cholesky factor of $\mathbf{S}^{-1}$. To aid interpretation of results with multivariate data, it is often useful to make $\mathbf{S}$ diagonal before the standardization (22.19) is employed. Because of the location-scale transform (22.10) employed in SNP estimation itself, the statistical properties of the SNP estimates are not affected by the standardization (22.19). When predictions, simulations, or other information about $\mathbf{y}_t$ is produced from the SNP estimates, the location-scale transformation is reversed so that the desired quantities relate to the actual data.

Sometimes it may be necessary to use external estimates of $\bar{\mathbf{y}}$ and $\mathbf{S}$ instead of the defaults to standardize the data or it may not be necessary to standardize the data. Within the `SNP` function, the standardization (22.19) is controlled using the optional list argument `LStransform`. The components of `LStransform` are `mu`, `cstat`, and `pstat`, which correspond to $\bar{\mathbf{y}}$, $\mathbf{S}^{1/2}$, and $\mathbf{S}^{-1/2}$, respectively. By default, `LStransform=NULL` and the standardization (22.19) is performed with $\mathbf{S}$ diagonal. To perform the standardization with $\mathbf{S}$ nondiagonal, set the `SNP.control` optional argument `diagS=F`. To turn off the standardization, call `SNP` with `LStransform=list(mu=0, cstat=diag(M), pstat=diag(M))`, where `M` is the number of columns in the `data` argument.

For example, to estimate an AR(1) model for the spot return data with no standardization, use

```
> fit.ar1 = SNP(dmRet.dat[,"exch"], model=SNP.model(ar=1),
+           n.drop=14,LStransform=list(mu=0,cstat=1,pstat=1))
> coef(fit.ar1)

Conditional Mean Coefficients:
     mu  ar(1)
 0.0547 0.0222
```

The above result agrees with an AR(1) model estimated using the `S+FinMetrics` function `OLS`:

```
> fit.OLS = OLS(exch~tslag(exch), data=dmRet.dat[14:834,])
> coef(fit.OLS)
```

```
 (Intercept) tslag(exch)
   0.0546925  0.02223613
```

To estimate a bivariate VAR(1) for the exchange rate data with no standardization, use

```
> fit.var1 = SNP(dmRet.dat[,1:2],model=SNP.model(ar=1),
+            n.drop=14,
+            LStransform=list(mu=c(0,0),cstat=diag(2),
+                              pstat=diag(2)))
> coef(fit.var1)

Conditional Mean Coefficients:
            exch forward
      mu  0.3012  0.0094
var(1;1)  0.0087 -0.8338
var(1;2) -0.0026  0.9671
```

The VAR(1) fit using the `S+FinMetrics` function `VAR` is

```
> fit.VAR = VAR(cbind(exch,forward)~ar(1),
+               data=dmRet.dat[14:834,])
> print(coef(fit.VAR), digits=2)
                exch forward
 (Intercept)  0.3012  0.0094
   exch.lag1  0.0087 -0.0026
forward.lag1 -0.8338  0.9671
```

Notice that the coefficients are arranged differently in the VAR output.

22.7.2 Transformations to Deal with Heavy Tailed Data

Time series data, particularly financial market data, often contain extreme or outlying observations. This is not necessarily a problem when the extreme value is considered as a value for $\mathbf{y}_t$, because it just fattens the tails of the estimated conditional density. However, once it becomes a lag and passes into $\mathbf{x}_{t-1}$, the optimization algorithm can use an extreme value in $\mathbf{x}_{t-1}$ to fit an element of $\mathbf{y}_t$ nearly exactly, thereby reducing the corresponding conditional variance to near zero and inflating the likelihood. This problem is endemic to procedures that adjust variance on the basis of observed explanatory variables.

The `SNP` function allows two methods to alleviate the problem of extreme values in $\mathbf{y}_t$. First, for heavy-tailed, mean-reverting data such as daily stock market returns, the following *logistic transformation* of the variables in $\mathbf{x}_{t-1}$ may be employed:

$$\hat{x}_i = 4\sigma_{tr}\frac{\exp\{x_i/\sigma_{tr}\}}{1+\exp\{x_i/\sigma_{tr}\}} - 2\sigma_{tr} \quad i = 1,\ldots,M \cdot L \tag{22.20}$$

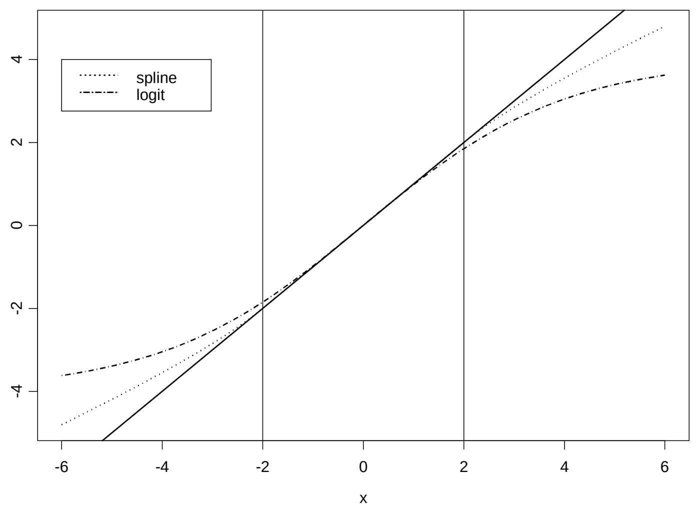


FIGURE 22.12. Logistic and spline transformations with $\sigma_{tr} = \pm 2$.

where x_i denotes an element of $\mathbf{x}_{t-1}$. This transformation has a negligible effect on values of x_i between $-\sigma_{tr}$ and σ_{tr}, but progressively compresses values that exceed $\pm\sigma_{tr}$. Second, for highly persistent data (such as interest rates) that have a strong ARCH/GARCH component, it can happen that an estimated SNP density will generate explosive simulations. This creates problems if a fitted SNP model object is to be used as an auxiliary model (score) for the EMM estimation. To deal with this problem, the `SNP` function allows the following spline transformation of $\mathbf{x}_{t-1}$ as an alternative to the logistic transformation:

$$\hat{x}_i = \begin{cases} \frac{1}{2}[x_i - \sigma_{tr} - \ln(1 - x_i - \sigma_{tr})] & \text{for} \quad x_i < -\sigma_{tr} \\ x_i & \text{for} \quad -\sigma_{tr} < x_i < \sigma_{tr} \\ \frac{1}{2}[x_i + \sigma_{tr} + \ln(1 - \sigma_{tr} + x_i)] & \text{for} \quad \sigma_{tr} < x_i \end{cases} \tag{22.21}$$

Figure 22.12 illustrates the logistic and spline transformations. Because the logistic and spline transformations only affect $\mathbf{x}$, not $\mathbf{y}$, the asymptotic properties of SNP estimator is not affected.

To use the logistic transformation in SNP estimation, set the optional argument `xTransform` of `SNP.control` to `"logistic"`. For example, to estimate a semiparametric AR(1) model for the spot return with a logistic transformation, use

```
> fit.10014000.logit = SNP(dmRet.dat[,"exch"],
+               model=SNP.model(ar=1,zPoly=4), n.drop=14,
+              control=SNP.control(xTransform="logistic"))
```

Since the original data are usually standardized before applying the SNP estimation, the inflection point σ_{tr} of the logistic transformation is set to two by default. To change the default value of σ_{tr}, pass the optional argument `inflection` to `SNP.control`:

```
> fit.10014000.logit2 = SNP(dmRet.dat[,"exch"],
+              model=SNP.model(ar=1,zPoly=4), n.drop=14,
+              control=SNP.control(xTransform = "logistic",
+              inflection=2.5))
```

To use the spline transformation, set the optional argument `xTransform` of `SNP.control` to `"spline"`. By default, the inflection point σ_{tr} for the spline transformation is set to 4. To change the default value of the inflection point, simply set the optional argument `inflection` to the desired value. To illustrate the importance of the spline transformation for highly persistent data with a strong ARCH/GARCH component, consider fitting an AR(1)-ARCH(4) model to the weekly 3-month U.S T-bill data in the `"timeSeries"` object `tbill.dat` with and without a spline transformation:

```
> tb3mo = tbill.dat[,"tb3mo"]
> fit.204100 = SNP(tbill.dat[,"tb3mo"],model=SNP.model(ar=2,
+                  arch=4), n.drop=6)
> fit.204100.s = SNP(tbill.dat[,"tb3mo"],model=SNP.model(ar=2,
+                    arch=4), n.drop=6,
+                    control=SNP.control(xTransform="spline",
+                                        inflection=4))
```

Simulations from the two fits, along with the T-bill data, are given in Figure 22.13. Notice that the untransformed fit generates explosive simulations, whereas the spline transformed fit does not.

22.7.3 Transformation to Deal with Small SNP Density Values

If the fitted SNP model is used in conjunction with the EMM estimation, it may happen that the estimated SNP density is smaller than the smallest value that the FORTRAN `DLOG` function can evaluate at simulated values generated by the `EMM` function. To avoid potential numerical problems, the SNP density may be replaced by

$$f^*(y|x,\theta) = \frac{\left\{\mathcal{P}^2[\mathbf{R}_x^{-1}(y-\mu_x),x] + \varepsilon_0\right\} n_M(y|\mu_x,\Omega_x)}{\int [P(s,x)]^2\phi(s)ds + \varepsilon_0}$$

where the user sets the value of ε_0. The default value suggested by Gallant and Tauchen (2001) is $\varepsilon_0 = 0.001$. This value is set by the `SNP.control` optional argument `eps0`.

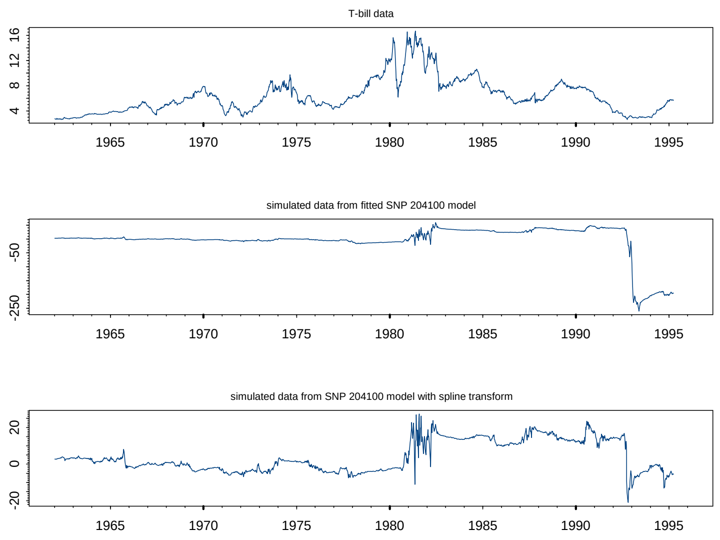


FIGURE 22.13. Weekly 3-month U.S. T-bill rates and simulations from a SNP AR(1)-ARCH(4) model without and with spline transformation.

22.8 Examples

The following subsections illustrate the process of fitting an SNP model to some common univariate financial time series. The fitting process makes use of the model selection strategy and residual diagnostics detailed in the previous sections. The final models will be used as score generators (auxiliary models) for the EMM examples in the next chapter.[20]

22.8.1 SNP Models for Daily Returns on Microsoft Stock

Data

The data are daily continuously compounded returns, based on closing prices, on Microsoft stock over the period March 14, 1986 through June 30, 2003. There are 4365 observations. Similar data, ending in February 2001, was analyzed in Gallant and Tauchen (2002). Figure 22.14 shows the data along with the sample ACF of the squared returns, and Figure 22.15 shows four distribution summary plots. Some summary statistics, computed using the `S+FinMetrics` function `summaryStats`, are

[20]The `"SNP"` objects for these models are included with `S+FinMetrics` 2.0.

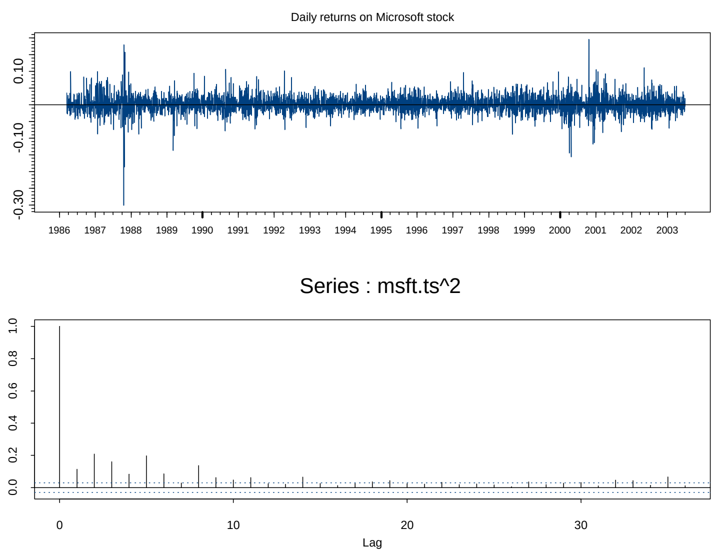


FIGURE 22.14. Daily log returns on Microsoft and SACF of squared returns.

```
> summaryStats(msft.ts)

Sample Quantiles:
     min       1Q median      3Q    max
 -0.3012 -0.01274      0 0.01567 0.1957

Sample Moments:
   mean     std skewness kurtosis
 0.0016 0.02526  -0.2457    11.66

Number of Observations:  4365
```

From Figures 22.14 and 22.15, it is clear that the returns exhibit considerably conditional heteroskedasticity as well as a few large outliers. The conditional transition density is likely to be non-Gaussian but symmetric.

SNP Model Selection

For heavy-tailed mean-reverting financial return data, it is recommended to use the logistic transformation (22.20) when fitting the SNP models. In the model selection strategy with financial return data, it is recommended to initially increment K_z from 0 to 4. Financial return data have thick tailed densities relative to the Gaussian, and only a quartic term in K_z can capture this. The Hermite polynomial $\mathcal{P}(z)$ has to "move" the mass

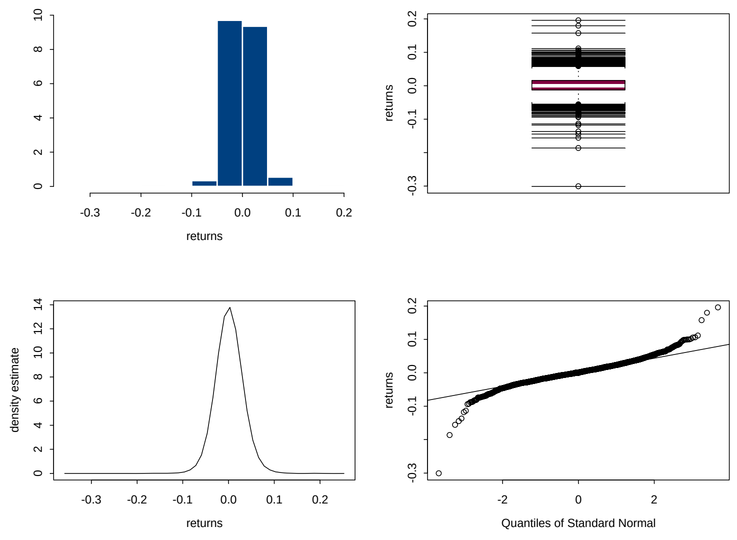


FIGURE 22.15. Distribution diagnostics for daily log returns on Microsoft.

of the Gaussian density around. It has to increase the mass around zero, depress the mass somewhat on either side of zero, and then increase the mass in the tails by going to $\pm\infty$. Linear, quadratics, and cubics in $\mathcal{P}(z)$ cannot do this, but a quartic can. In fitting SNP models, it is possible that incrementing K_z by 1 starting from $K_z = 0$ and moving up to $K_z = 4$ produces inferior fits. The reason is that a model with $K_z = 1$ is not a good model for the density of the return data, and the fit may involve an area of the parameter space that is far away from SNP models with larger values of K_z. It is not recommended to fit SNP models with $K_z > 8$. For these models, $\mathcal{P}(z)$ generally only captures little wiggles and the nonlinear fits can become somewhat unstable. If the density is roughly symmetric, it is preferred to increase K_z from 4 in increments of 2, as the even powers of $P(z)$ are more important.

Based on the above discussion, the following model selection strategy is entertained to find the best fitting SNP model for the MSFT daily returns:

$$\begin{aligned}
10010000 &\rightarrow 11110000 \\
&\rightarrow 11114000 \\
&\rightarrow 11116000 \rightarrow 11116010 \rightarrow 11116020 \\
&\rightarrow 11118000 \\
&\rightarrow 11216000
\end{aligned}$$

The initial Gaussian AR(1) SNP model is fit using the random restart parameters

```
> fOld = c(0, 1e-5, 0, -1e-5, 1e-5, 1e-5, 1e-4, 0, -1e-4,
+          1e-4, 1e-4, 1e-3, 0, -1e-3, 1e-3, 1e-3, 1e-5, 0,
+          -1e-2, 1e-2, 1e-2, 1e-1, 0, -1e-1, 1e-1, 1e-1,
+          1e0, 0, -1e0, 1e0, 1e0)
> fNew = c(0, 0, 1e-5, 1e-5, -1e-5, 1e-5, 0, 1e-4, 1e-4,
+          -1e-4, 1e-4, 0, 1e-3, 1e-3, -1e-3, 1e-3, 0,
+          1e-2, 1e-2, -1e-2, 1e-2, 0, 1e-1, 1e-1, -1e-1,
+          1e-1, 0, 1e0, 1e0, -1e0, 1e0)
> n.start = c(0,rep(25,30))
> fit.msft.10010000 = SNP(msft.ts, model=SNP.model(ar=1),
+          control = SNP.control(xTransform="logistic",
+                                n.start=n.start,fOld=fOld,
+                                fNew=fNew),
+          n.drop=14, trace=T)
```

Although it is not necessary to use random restarts with the initial model, it will become essential to use them for more complicated models. The remaining models in the model selection strategy are fit using the `expand` function as follows:

```
> fit.msft.11110000 = expand(fit.msft.10010000, arch=1,
+                            garch=1, trace=T)
> fit.msft.11114000 = expand(fit.msft.11110000, zPoly=4)
> fit.msft.11116000 = expand(fit.msft.11114000, zPoly=2)
> fit.msft.11116010 = expand(fit.msft.11116000, xPoly=1)
> fit.msft.11116020 = expand(fit.msft.11116010, xPoly=1)
> fit.msft.11216000 = expand(fit.msft.11116000, arch=1)
> fit.msft.12116000 = expand(fit.msft.11116000, garch=1)
> fit.msft.11118000 = expand(fit.msft.11116000, zPoly=2)
```

The BIC values for the fitted SNP models are summarized in Table 22.5. According to the BIC, the best fitting SNP models are the semiparametric GARCH(1,1) with $K_z = 6$. The same specification was chosen by Gallant and Tauchen (2002). A summary of this SNP fit is

```
> summary(fit.msft.11116000)
Model: Semiparametric GARCH

Hermite Polynomial Coefficients:
               z^0      z^1      z^2      z^3      z^4
     coef   1.0000  -0.0189  -0.3336   0.0138   0.0558
(std.err)            0.0302   0.0172   0.0097   0.0043
 (t.stat)           -0.6254 -19.4181   1.4262  12.9964
```

SNP Model	L_u	L_g	L_r	L_p	K_z	I_z	K_x	I_x	**BIC**
10010000	1	0	0	1	0	0	0	0	1.4216
11110000	1	1	1	1	0	0	0	0	1.3651
11114000	1	1	1	1	4	0	0	0	1.3394
11116000	**1**	**1**	**1**	**1**	**6**	**0**	**0**	**0**	**1.3377**
11116s00	**1**	**1**	**1**	**1**	**6**	**0**	**0**	**0**	**1.3354**
11118000	1	1	1	1	8	0	0	0	1.3390
11116010	1	1	1	1	6	0	1	0	1.3434
11116020	1	1	1	1	6	0	2	0	1.3487
11216000	1	1	2	1	6	0	0	0	1.3387
12116000	1	2	1	1	6	0	0	0	1.3383

TABLE 22.5. BIC values for SNP models fit to Microsoft returns

```
               z^5      z^6
     coef  -0.0012  -0.0025
(std.err)   0.0008   0.0003
 (t.stat)  -1.5228  -8.7763

Conditional Mean Coefficients:
               mu   ar(1)
     coef -0.0020  0.0186
(std.err)  0.0318  0.0159
 (t.stat) -0.0612  1.1660

Conditional Variance Coefficients:
               s0 arch(1) garch(1)
     coef  0.0679  0.1630  0.8615
(std.err)  0.0073  0.0135  0.0099
 (t.stat)  9.2548 12.1146 86.9439
```

Notice that only the coefficients on the even powers of z in the Hermite polynomial $\mathcal{P}(z)$ are statistically different from zero. This suggests refitting the 11116000 model with the added restriction that the coefficients on the odd powers of z are equal to zero

```
> fit.msft.11116s00 = SNP(data = msft.ts,
+          model = SNP.model(ar=1, arch=1, garch=1, zPoly=6),
+          control = SNP.control(xTransform="logistic",
+                                n.start=n.start,
+                                fOld=fOld, fNew=fNew),
+          n.drop = 14, trace=T,
+          coef = c(1,0,0,0,0,0,0,0,0,1,0,0),
+          est = c(0,0,1,0,1,0,1,1,1,1,1,1))
```

This restricted model, denoted 11116s00, gives the overall lowest BIC value.

The function `SNP.auto` may also be used to find the best fitting SNP model:

```
> fit.msft.auto = SNP.auto(msft.ts, n.drop=14,
+                 control = SNP.control(xTransform="logistic",
+                 n.start = n.start, seed = 011667,
+                 fOld = fOld, fNew = fNew))
```

Here, the best fitting model is a semiparametric GARCH(1,1) with $K_z = 7$, 01117000. It has a BIC value equal to 1.3373, which is slightly lower than the 11116000 model but slightly higher than the restricted 11116000 model.

SNP Model Diagnostics

From the analysis in the previous section, the SNP 11116000 and 11116s00 models are determined to be the best fitting SNP models for the daily returns on Microsoft stock. The estimated SNP density for the 11116000 model, compared to a normal density, is shown in Figure 22.16. As expected, the SNP density is roughly symmetric and has fatter tails than the normal. The sample ACF and PACF plots of the standardized residuals and squared standardized residuals (not shown) indicate no remaining serial correlation.

Simulations from the SNP 11116000 model are computed using

```
> sim.msft.11116000 = simulate(fit.msft.11116000)
```

The simulated data, shown in Figure 22.17, mimics the actual data quite well.

22.8.2 SNP Models for Daily Returns on the S&P 500 Index

Data

The data consist of daily continuously compounded returns, based on closing prices, on the S&P 500 index over the period March 14, 1986 through June 30, 2003 (same time span as the data for the Microsoft returns). Similar data were analyzed in Gallant and Tauchen (2002). Figure 22.18 shows the data along with the sample ACF of the squared returns, and Figure 22.19 shows four distribution summary plots. Sample summary statistics are

```
> summaryStats(sp500.ts)

Sample Quantiles:
     min        1Q    median       3Q     max
 -0.2047 -0.004686 0.0004678 0.005827 0.09099
Sample Moments:
      mean     std skewness kurtosis
```

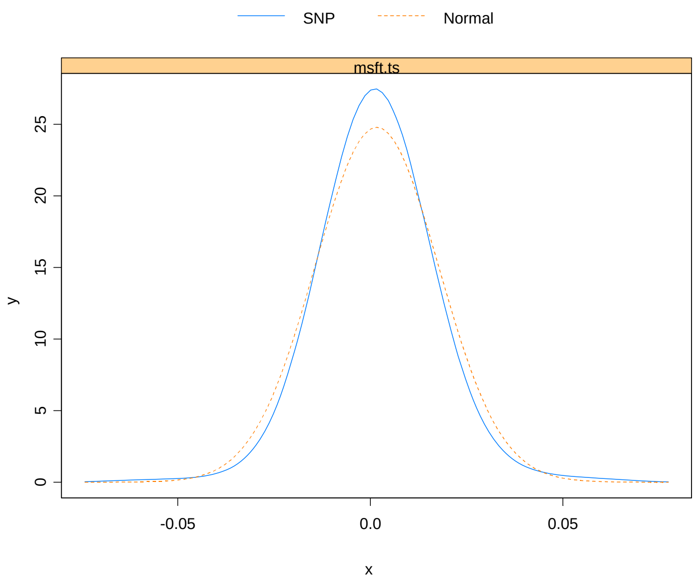


FIGURE 22.16. Estimated conditional density from the SNP 11116000 model fit.

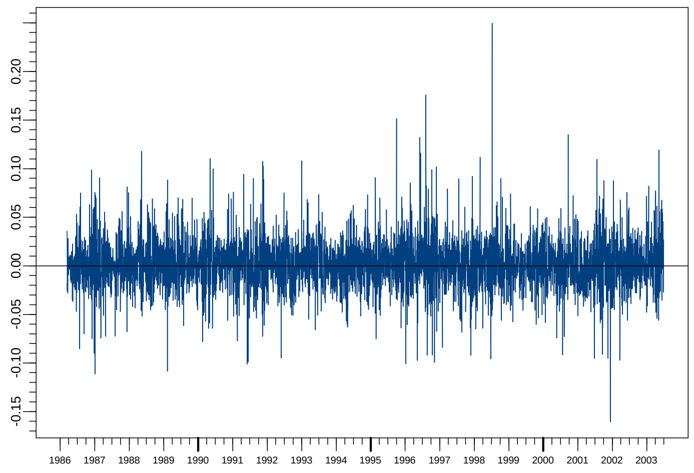


FIGURE 22.17. Simulated returns from the SNP 11116000 model fit.

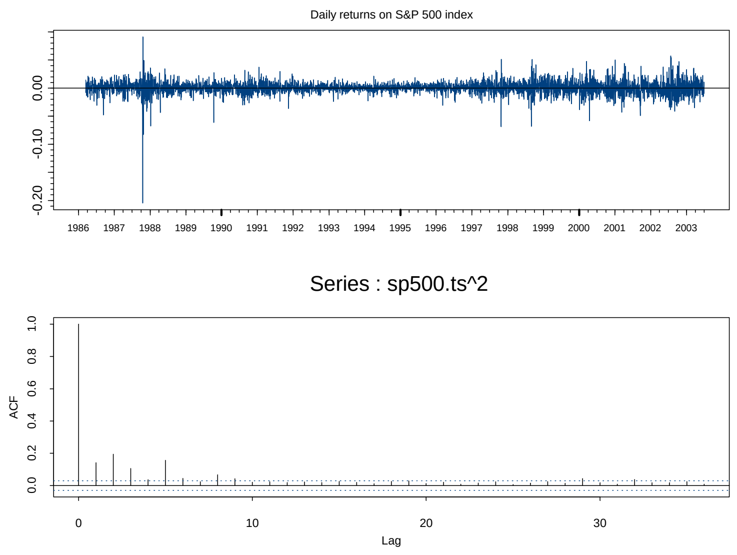


FIGURE 22.18. Daily log returns on the S&P 500 index and SACF of squared returns.

```
 0.0003916 0.01124   -1.486    32.59
Number of Observations:  4365
```

The S&P 500 returns appear to be highly non-normal, with negative skewness and very large kurtosis, and display considerable conditional heteroskedasticity.

SNP Model Selection

A model selection strategy similar to that used for the Microsoft returns is followed. However, the S&P 500 index is expected to display richer dynamics than any single stock. The manual search strategy employed is

$$\begin{aligned}10010000 &\rightarrow 11110000\\ &\rightarrow 11114000\\ &\rightarrow 11116000\\ &\rightarrow 11118000 \rightarrow 11118010 \rightarrow 11118020\end{aligned}$$

Table 22.6 reports the BIC values for the above models using the `S+FinMetrics` functions `SNP` and `expand` with random restarts.

The BIC preferred model is the semiparametric GARCH(1,1) model with $K_z = 8$, 11118000. A summary of the fit is

```
> summary(fit.sp500.11118000)
```

```
Model: Semiparametric GARCH

Hermite Polynomial Coefficients:
               z^0      z^1      z^2      z^3      z^4
     coef   1.0000  -0.0938  -0.4103   0.0356   0.0925
(std.err)            0.0317   0.0239   0.0187   0.0098
 (t.stat)           -2.9543 -17.1575   1.8996   9.4081

               z^5      z^6      z^7      z^8
     coef  -0.0049  -0.0082   0.0001   0.0002
(std.err)   0.0036   0.0014   0.0002   0.0001
 (t.stat)  -1.3710  -5.7050   0.5784   3.8843

Conditional Mean Coefficients:
              mu  ar(1)
     coef 0.1032 0.0202
(std.err) 0.0232 0.0162
 (t.stat) 4.4547 1.2451

Conditional Variance Coefficients:
                s0  arch(1) garch(1)
     coef   0.0122   0.1620   0.9105
(std.err)   0.0032   0.0072   0.0049
 (t.stat)   3.8641  22.6131 184.9564

Information Criteria:
    BIC     HQ    AIC      logL
 1.2426 1.2365 1.2331 -5352.208
```

The best fitting model using `SNP.auto`

```
> fit.sp500.auto = SNP.auto(sp500.ts,n.drop=14,
+       control = SNP.control(xTransform="logistic",
+       n.start = n.start, seed = 011667,
+       fOld = fOld, fNew = fNew))
```

is an SNP 01118000 model, which is very similar to the 11118000 model but has a slightly higher BIC value:

```
>  fit.sp500.auto$obj[,"BIC"]
1.243061
```

Gallant and Tauchen (2002) reported the results from a specification search and suggested the SNP models 20b14000 and 20b14010, where b represents $L_r = 11$. When these models are fit to the updated S&P 500 data, using an expansion path starting from a Gaussian AR(2) with ran-

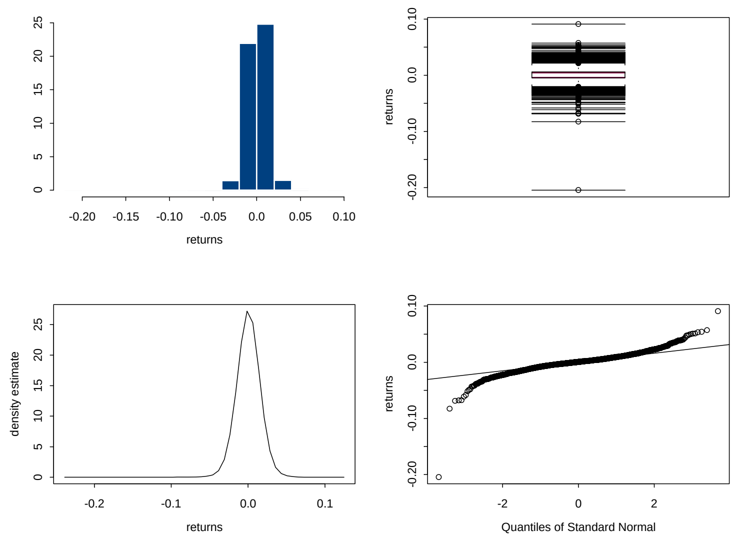


FIGURE 22.19. Distribution summary from daily log returns on the S&P 500 index.

SNP Model	L_u	L_g	L_r	L_p	K_z	I_z	K_x	I_x	**BIC**
10010000	1	0	1	1	0	0	0	0	1.4218
11110000	1	1	1	1	0	0	0	0	1.2798
11114000	1	1	1	1	4	0	0	0	1.2492
11116000	1	1	1	1	6	0	0	0	1.2479
11118000	**1**	**1**	**1**	**1**	**8**	**0**	**0**	**0**	**1.2426**
11118010	1	1	1	1	8	0	1	0	1.2465
11118020	1	1	1	1	8	0	2	0	1.2481

TABLE 22.6. BIC values for SNP models fit to the SP 500 returns.

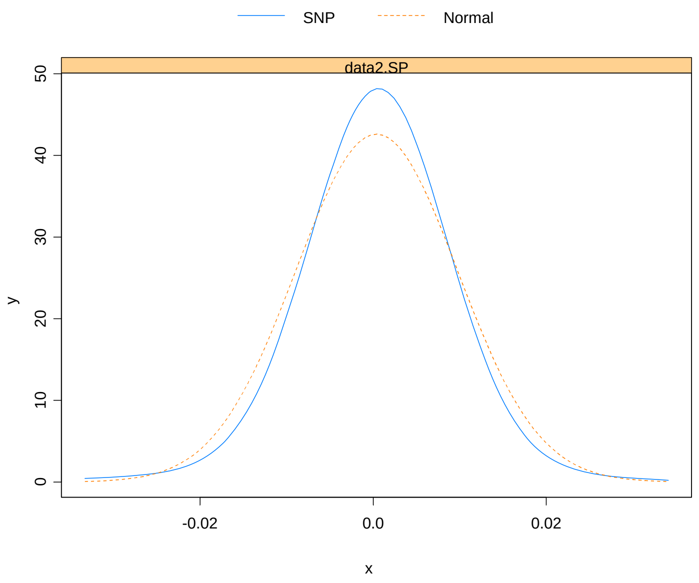


FIGURE 22.20. Estimated conditional density from the SNP 11118000 model fit.

dom restarts, the BIC values are 1.2597 and 1.2598, respectively. The SNP 11118000 model still dominates in terms of BIC.

SNP Model Diagnostics

From the analysis in the last subsection, the SNP 11118000 model is determined to be the best fitting SNP model for the daily returns on the S&P 500 index. The estimated SNP density, shown in Figure 22.20, is roughly symmetric and has fatter tails than the normal. Simulated data from the fitted 11118000 model, along with the SACF of the squared data, are depicted in Figure 22.21. These plots confirm that the fitted 11118000 model adequately captures the salient features of the S&P 500 daily returns.

22.8.3 SNP Models for Weekly 3-Month U.S. T-Bill Rates

Data

The data are weekly (Friday) observations on the 3-month U.S. Treasury Bill rate (annualized rates times 100) over the period January 5, 1962 through March 31, 1995. This data has been analyzed by Gallant and Tauchen (2002) to illustrate the EMM estimation of a two factor interest

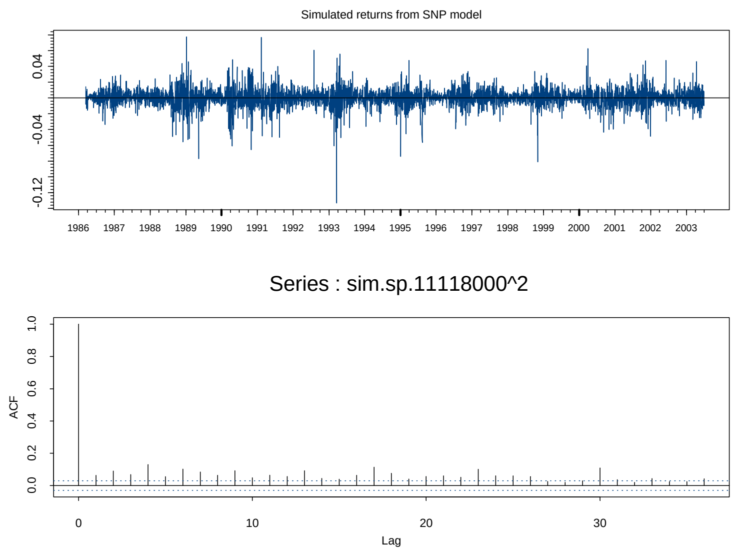


FIGURE 22.21. Simulated returns and SACF of squared returns from the SNP 11118000 model fit.

rate diffusion model[21]. The data, obtained from `ftp://ftp.duke.econ`, are in the first column of the `S+FinMetrics` `"timeSeries"` object `tbill.dat`

```
> colIds(tbill.dat)
[1] "tb3mo"  "tb12mo" "tb10yr"
```

and are labeled `"tb3mo"`. The data and sample ACF are displayed in Figure 22.22. These plots show that interest rates are highly persistent and that the volatility of rates increases with the level of rates. A simple AR(1) model is often used to characterize the conditional mean. Summary statistics of the residuals from a fitted AR(1) to the interest rate data are

```
> resid.ar1 = residuals(OLS(tb3mo~tslag(tb3mo),
                            data=tbill.dat, na.rm=T))
> summaryStats(resid.ar1)

Sample Quantiles:
    min       1Q    median      3Q   max
 -2.156 -0.07646 -0.006411 0.08537 1.852
```

[21]These data are similar to the data used by Andersen and Lund (1997). They used weekly (Wednesday) observations on the 3-month U.S. T-bill over the period 1954 to 1995.

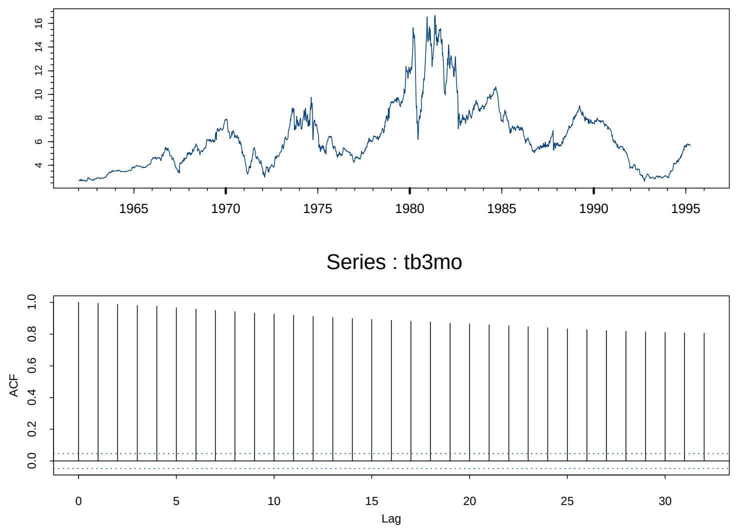


FIGURE 22.22. Weekly observations on U.S. 3-month T-bill rate and SACF.

```
Sample Moments:
       mean    std skewness kurtosis
 1.465e-015 0.2667   -0.925    16.89

Number of Observations:  1734
```

These statistics and the distribution summary plots in Figure 22.23, show that the innovation to the one-step-ahead conditional density based on an AR(1) model of interest rates is highly non-normal.

SNP Model Selection

Andersen and Lund (1997), Tauchen (1997), McManus and Watt (1999), Jensen (2001), and Gallant and Tauchen (2002) fitted SNP models to weekly interest rates. For highly persistent data like interest rates, it is recommended to use the spline transformation (22.20) when fitting the SNP models. In addition, as stressed by Andersen and Lund (1997), Gallant and Long (1997), and Tauchen (1997, 1998), for interest rate data it is important to check the dynamic stability of any candidate SNP model that is to be used as a score generator for the EMM estimation. Dynamic stability may be checked visually by plotting long simulations from the fitted SNP model.

SNP Model	L_u	L_g	L_r	L_p	K_z	I_z	K_x	I_x	**BIC**
				GARCH Models					
11110000	1	1	1	1	0	0	0	0	-1.4071
11114000	1	1	1	1	4	0	0	0	-1.4524
11116000	1	1	1	1	6	0	0	0	-1.4482
11118000	**1**	**1**	**1**	**1**	**8**	**0**	**0**	**0**	**-1.4603**
11118010	1	1	1	1	8	0	1	0	-1.4474
11118020	1	1	1	1	8	0	2	0	-1.4320
				ARCH Models					
10110000	1	0	1	1	0	0	0	0	-1.1700
10210000	1	0	2	1	0	0	0	0	-1.2261
10310000	1	0	3	1	0	0	0	0	-1.3104
10410000	1	0	4	1	0	0	0	0	-1.3406
10510000	1	0	5	1	0	0	1	0	-1.3554
10610000	1	0	6	1	0	0	2	0	-1.3622
10710000	1	0	7	1	0	0	0	0	-1.3631
10810000	1	0	8	1	0	0	0	0	-1.3678
10814000	1	0	8	1	4	0	0	0	-1.4217
10815000	1	0	8	1	5	0	0	0	-1.4196
10816000	1	0	8	1	6	0	0	0	-1.4174
10817000	1	0	8	1	7	0	0	0	-1.4234
10818000	**1**	**0**	**8**	**1**	**8**	**0**	**0**	**0**	**-1.4289**
10818010	1	0	8	1	8	0	1	0	-1.4197
10818020	1	0	8	1	8	0	2	0	-1.4044

TABLE 22.7. BIC values for SNP models fit to weekly U.S. T-bill rates

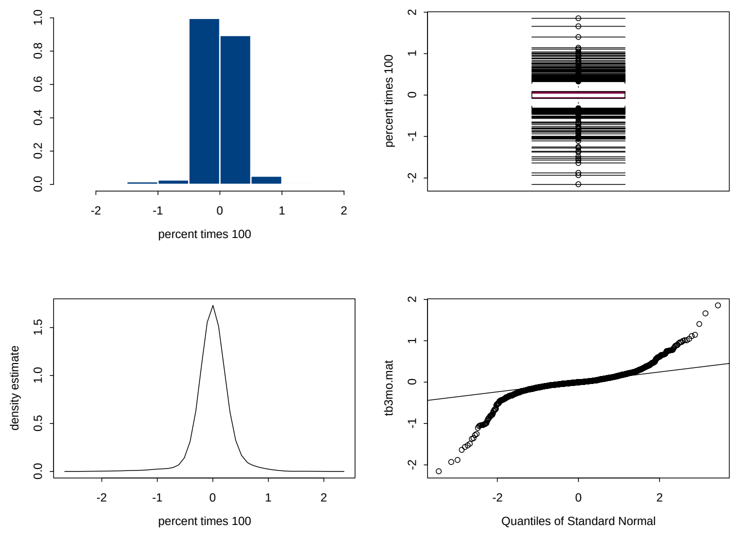


FIGURE 22.23. Distribution summary of weekly returns on U.S. 3-month T-bill rate.

Two types of manual model selection strategy are followed. In the first strategy, an expansion path based on a GARCH(1,1) leading term is used. This strategy produces the overall BIC minimizing SNP model. In the second strategy, following Gallant and Tauchen (2002), only pure ARCH models with $L_r \leq 4$ are considered. In both cases, the best fitting initial Gaussian AR model has $L_u = 5$, which seems rather high. In fact, for both strategies, SNP models with $L_u = 1$ lead to better overall fits as measured by BIC.

The BIC values for a subset of the models fit with the `S+FinMetrics` functions `SNP` and `expand` using the above strategies are given in Table 22.7. For the first strategy, the best fitting model is a semiparametric AR(1)-GARCH(1,1) with $K_z = 8$, 11118000. For the second strategy, the best fitting model is a semiparametric AR(1)-ARCH(8) with $K_z = 8$, 10818000. Gallant and Tauchen (2002) reported their preferred dynamically stable SNP model as a nonlinear nonparametric AR(1)-ARCH(4), 10414010. The BIC values for these models are -1.4031 and -1.4113, respectively. Finally, the BIC minimizing model found using `SNP.auto` is a semiparametric AR(5)-GARCH(1,1) with $K_z = 8$. The BIC for this model is -1.4594, which is slightly higher than the BIC for the 11118000 model. A summary of the SNP 11118000 model fit is

```
> summary(fit.tb3mo.11118000)
```

```
Model: Semiparametric GARCH

Hermite Polynomial Coefficients:
              z^0     z^1      z^2     z^3     z^4
     coef  1.0000  0.0440  -0.4300 -0.0091  0.1200
(std.err)          0.0369   0.0355  0.0204  0.0139
 (t.stat)          1.1933 -12.1030 -0.4445  8.6598

              z^5     z^6      z^7     z^8
     coef  0.0014 -0.0117  -0.0001  0.0004
(std.err)  0.0033  0.0016   0.0002  0.0001
 (t.stat)  0.4359 -7.1523  -0.3864  6.9470

Conditional Mean Coefficients:
                mu    ar(1)
     coef  -0.0034   0.9960
(std.err)   0.0017   0.0013
 (t.stat)  -1.9517 753.3579

Conditional Variance Coefficients:
              s0 arch(1) garch(1)
     coef  0.0010  0.2095  0.8842
(std.err)  0.0001  0.0201  0.0093
 (t.stat)  7.0564 10.3975 94.7220

Information Criteria:
     BIC      HQ     AIC    logL
 -1.4603 -1.4732 -1.4808 2561.52

Convergence Type:
relative function convergence
number of iterations: 59
```

SNP Model Diagnostics

Tauchen (1997) and Gallant and Tauchen (2002) stressed that the BIC minimizing SNP model may not be the best choice for weekly interest rate data, especially if the SNP model is to be used as an auxiliary model for the EMM estimation. The above analysis identifies four potential SNP models: 11118000, 10818000, 51118000, and 10414010. The standard residual diagnostics are similar for these models. All of the models except the 5111800 model show slight autocorrelation in the standardized residuals. Figure 22.24 shows simulated values and estimated conditional volatilities

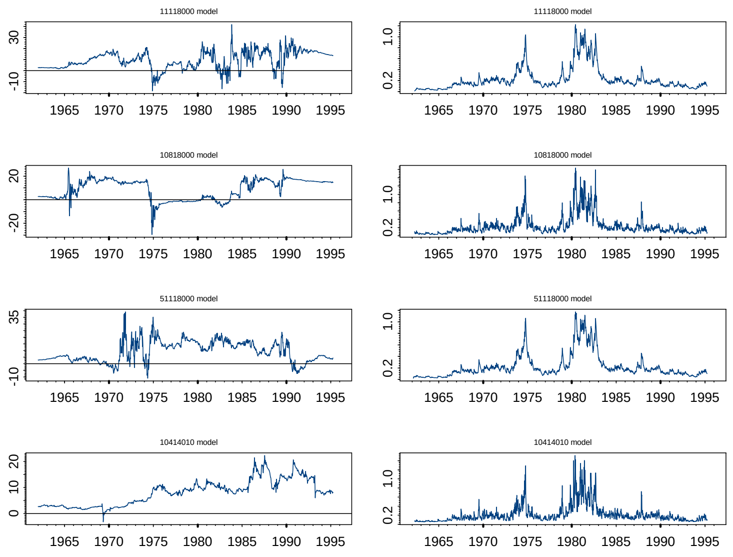


FIGURE 22.24. Simulations (left column) and estimated conditional volatility (right column) from fitted SNP models for weekly U.S. T-bill rates.

from the four models. All models appear to be dynamically stable and have similar fitted conditional volatilities.

22.9 References

ANDERSEN, T.G. AND J. LUND (1997). "Estimating Continuous-Time Stochastic Volatility Models of the Short-Term Interest Rate," *Journal of Econometrics*, 77, 343-377.

BANSAL, R., A.R. GALLANT, R. HUSSEY AND G. TAUCHEN (1994). "Nonparametric Estimation of Structural Models for High-Frequency Currency Market Data," *Journal of Econometrics*, 66, 251-287.

BOLLERSLEV, T. (1986). "Generalized Autoregressive Conditional Heteroskedasticity," *Journal of Econometrics*, 31, 307-327.

COPPEJANS, M. AND A.R. GALLANT (2002). "Cross-Validated SNP Density Estimates," *Journal of Econometrics*, 110, 27-65.

DAVIDIAN, M. AND R.J. CARROLL (1987). "Variance Function Estimation," *Journal of the American Statistical Association*, 82, 1079-1091.

Ding, Z., C.W.J. Granger and R.F. Engle (1993). “A Long Memory Property of Stock Market Returns and a New Model,” *Journal of Empirical Finance*, 1, 83-106.

Engle, R.F. (1982). “Autoregressive Conditional Heteroskedasticity With Estimates of the Variance of U.K. Inflation,” *Econometrica*, 50, 987-1008.

Engle, R.F. (1995). *ARCH: Selected Readings.* Oxford University Press, Oxford.

Engle, R.F. and G. González-Rivera (1991). “Semiparametric ARCH Models,” *Journal of Business and Economic Statistics*, 9(4), 345-359.

Fenton, V.M. and A.R. Gallant (1996a). “Convergence Rates of SNP Density Estimators,” *Econometrica*, 64, 719-727.

Fenton, V.M. and A.R. Gallant (1996b). “Qualitative and Asymptotic Performance of SNP Density Estimators,” *Journal of Econometrics*, 74, 77-118.

Gallant, A.R. and J.R. Long (1997). “Estimating Stochastic Differential Equations Efficiently by Minimum Chi-Square,” *Biometrika*, 84, 125-141.

Gallant, A.R. and D.W. Nychka (1987). “Seminonparametric Maximum Likelihood Estimation,” *Econometrica*, 55, 363-390.

Gallant, A.R., D.A. Hsieh and G. Tauchen (1991). “On Fitting a Recalcitrant Series: The Pound/Dollar Exchange Rate, 1974-83,” in W.A. Barnett, J. Powell and G. Tauchen (eds.), *Nonparametric and Semiparametric Methods in Econometrics and Statistics.* Cambridge University Press, Cambridge.

Gallant, A.R. and G. Tauchen (1989). “Seminonparametric Estimation of Conditionally Constrained Heterogeneous Processes: Asset Pricing Applications,” *Econometrica*, 57, 1091-1120.

Gallant, A.R. and G. Tauchen (1999). “The Relative Efficiency of Method of Moments Estimators,” *Journal of Econometrics*, 92, 149-172.

Gallant, A.R. and G. Tauchen (2001). “SNP: A Program for Nonparametric Time Series Analysis, Version 8.8, User’s Guide,” Working Paper, University of North Carolina at Chapel Hill.

Gallant, A.R. and G. Tauchen (2001b). “Efficient Method of Moments,” unpublished manuscript, Department of Economics, University of North Carolina.

GALLANT, A.R. AND G. TAUCHEN (2002). "Simulated Score Methods and Indirect Inference for Continuous-time Models," forthcoming in Y. AIT-SAHALIA, L.P. HANSEN AND J. SCHIENKMAN (eds.), *Handbook of Financial Econometrics*. North-Holland, Amsterdam.

JENSEN, M.B. (2001). "Efficient Method of Moments Estimation of the Longstaff and Schwartz Interest Rate Model," unpublished manuscript, Department of Business Studies, Aalborg University, Denmark.

LÜTKEPOHL, H. (1990). *Introduction to Multiple Time Series Analysis.* Springer-Verlag, New York.

MCMANUS, D. AND D. WATT (1999). "Estimating One Factor Models of Short-Term Interest Rates," Working Paper 99-18, Bank of Canada, Ottawa, Canada.

SCHWERT, W. (1990). "Stock Volatility and the Crash of '87," *Review of Financial Studies*, 3(1), 77-102.

TAUCHEN, G. (1997). "New Minimum Chi-Square Methods in Empirical Finance," in *Advances in Econometrics, Seventh World Congress*, D. Kreps and K. Wallis (eds.), Cambridge University Press, 279-317.

TAUCHEN, G. (1998). "The Objective Function of Simulation Estimators Near the Boundry of the Unstable Region of the Parameter Space," *Review of Economics and Statistics*, 80, 389-398.

TAYLOR, S. (1986). *Modelling Financial Time Series.* John Wiley & Sons, New York.

23
Efficient Method of Moments

23.1 Introduction

Dynamic nonlinear models that have unobserved variables pervade science.[1] Most often they arise from dynamic systems described by a system of deterministic or stochastic differential equations in which the state vector is partially observed. For example, in epidemiology, the SEIR model determines those susceptible, exposed, infected, and recovered from a disease, whereas usually data are from case reports that report only those infected (Olsen and Schaffer, 1990). Other examples are continuous- and discrete-time stochastic volatility models of speculative markets from finance (Ghysels, Harvey, and Renault, 1996), general equilibrium models from economics (Gennotte and Marsh, 1993), and compartment models from pharmacokinetics (Mallet, Mentré, Steimer, and Lokiec, 1988).

Standard statistical methods, both classical and Bayesian, are usually not applicable in these situations either because it is not practicable to obtain the likelihood for the entire state vector or because the integration required to eliminate unobservables from the likelihood is infeasible. On a case-by-case basis, statistical methods are often available. However, the purpose here is to describe methods that are generally applicable.

Although determining the likelihood of a nonlinear dynamic system that has unobserved variables is often infeasible, simulating the evolution of the

[1]Sections 1 and 2 are based on Gallant and Tauchen (2001) by permission of the authors.

state vector is often quite practicable. The *efficient method of moments* (EMM) methodology described in this chapter relies on this.

Briefly, the steps involved in EMM are as follows: Summarize the data by using quasi-maximum-likelihood to project the observed data onto a transition density that is a close approximation to the true data generating process. This transition density is called the *auxiliary model* and its score is called the *score generator* for EMM. The SNP model described in the previous chapter provides a convenient general-purpose auxiliary model in this connection. Once a score generator is in hand, given a parameter setting for the system, one may use *simulation* to evaluate the expected value of the score under the stationary density of the system and compute a chi-squared, or generalized method of moments (GMM)-type, criterion function. A nonlinear optimizer is used to find the parameter setting that minimizes the criterion.

If the auxiliary model encompasses the true data generating process, then quasi maximum likelihood estimates become sufficient statistics and EMM is fully efficient (Gallant and Tauchen, 1996). If the auxiliary model is a close approximation to the data generating process, then one can expect the efficiency of EMM to be close to that of maximum likelihood (Gallant and Long, 1997; Tauchen, 1997). Because EMM is a GMM-type estimator, diagnostic tests are available to assess system adequacy as well as are graphics that suggest reasons for failure.

Due to the fundamental importance of diffusion models for stock price dynamics, there has been significant progress recently using alternative simulation strategies; see Brandt and Santa-Clara (1999), Durham and Gallant (2002), Elerian, Chib, and Shephard (2001), and the references therein. Despite this recent progress, the alternatives to the simulation methods discussed here are not as general purpose and are limited in their ability to deal with latent variables.

This chapter is organized as follows. Section 23.2 gives an overview of the EMM methodology based on Gallant and Tauchen (2001). Section 23.3 describes the implementation of EMM in `S+FinMetrics`. Section 23.4 gives detailed examples of EMM estimation of moving average models for time series, discrete-time stochastic volatility models for asset returns, and continuous-time models for interest rates.

Efficient method of moments has found several recent applications, and the following is a selected list mainly from finance. Andersen and Lund (1997), Dai and Singleton (2000), and Ahn, Dittmar, and Gallant (2002) use the method for interest rate applications. Liu (2000), Andersen, Benzoni, and Lund (2002), and Chernov, Gallant, Ghysels, and Tauchen (2003) use it to estimate stochastic volatility models for stock prices with such complications as long memory and jumps. Chung and Tauchen (2001) use it to estimate and test target zone models of exchange rates. Jiang and van der Sluis (2000) use it to price options. Valderrama (2001) employs it for a macroeconomic analysis and Nagypal (2001) employs it in a labor economics application. An excellent survey of EMM and related methods is given in Gallant and Tauchen (2005). The `S+FinMetrics` implementation of EMM described in this chapter is based on the public-domain FORTRAN code developed by Ronald Gallant and George Tauchen and summarized in Gallant and Tauchen (2002).[2]

23.2 An Overview of the EMM Methodology

The following subsections, based on Gallant and Tauchen (2001), give a general overview of the EMM methodology, and draw connections with the GMM and maximum likelihood (ML) methodologies. The EMM methodology is first motivated by the problem of estimating a continuous-time stochastic volatility model for short-term interest rates. Minimum chi-square estimators are then defined, and their efficiency properties are compared. The SNP model is introduced next as a general-purpose auxiliary model, and then the steps of the EMM methodology are described.

23.2.1 Continuous-Time Stochastic Volatility Model for Interest Rates

The data used to illustrate ideas are weekly observations on the 3-month U.S. T-bill rate, from January 1962 through March 1995, yielding 1,735 raw observations.[3] Figure 23.1 is a plot of the data. The series $\{y_t\}$ is the annualized return on the 3-month T-bill.

The finance literature normally treats interest rates as a diffusion, usually expressed as a system of stochastic differential equations. The data shown in Figure 23.1 would thus be regarded as having resulted from discretely sampling a diffusion.

An example is the following two-factor mean-reverting continuous-time stochastic volatility model:

$$\begin{aligned} dX_{1t} &= \kappa_1(\mu - X_{1t})dt + \exp(X_{2t}/2)X_{1t}^{\gamma}dW_{1t},\ \gamma > 0 \qquad (23.1) \\ dX_{2t} &= \kappa_2(\alpha - X_{2t})dt + \xi dW_{2t} \end{aligned}$$

where X_{1t} is observable at time t and represents the instantaneous short-term interest rate. The second component X_{2t} is not observable and represents a factor that affects the volatility of the process. The variables dW_{1t} and dW_{2t} are increments to standard Brownian motion, as discussed in Chapter 20. Versions of this model without an unobservable volatility factor

$$dX_t = \kappa(\mu - X_t)dt + \sigma X_t^{\gamma} dW_t,\ \gamma > 0$$

[2]The code is available at `ftp.econ.duke.edu` in directory `pub/get/emm`.

[3]These data were analyzed in the previous chapter.

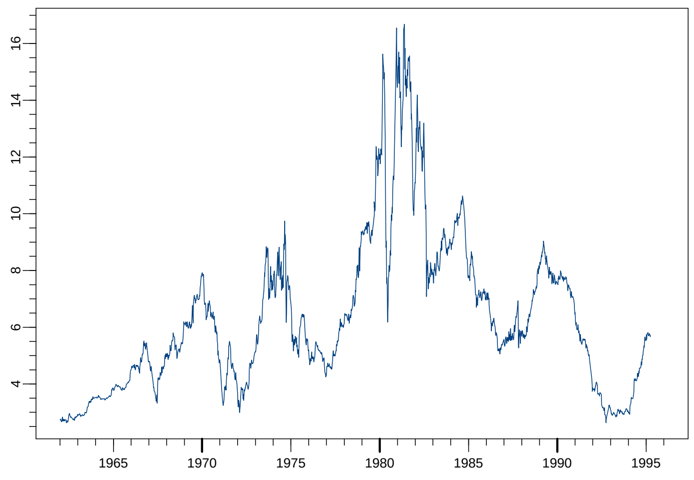


FIGURE 23.1. Weekly observations on 3-month U.S. T-bill rate.

have a long history in financial economics.

In matrix notation, the system (23.1) is

$$d\mathbf{X}_t = \mathbf{A}(\mathbf{X}_t)\,dt + \mathbf{B}(\mathbf{X}_t)\,d\mathbf{W}_t \qquad 0 \le t < \infty \tag{23.2}$$

where

$$\begin{aligned}\mathbf{X}_t &= \begin{pmatrix} X_{1t} \\ X_{2t} \end{pmatrix}, \quad \mathbf{A}(\mathbf{X}) = \begin{pmatrix} \kappa_1(\mu - X_{1t}) \\ \kappa_2(\alpha - X_{2t}) \end{pmatrix}, \\ \mathbf{W}_t &= \begin{pmatrix} W_{1t} \\ W_{2t} \end{pmatrix}, \quad \mathbf{B}(\mathbf{X}) = \begin{pmatrix} \exp(X_{2t}/2)X_{1t}^{\gamma} & 0 \\ 0 & \xi \end{pmatrix}\end{aligned}$$

The process is discretely sampled so that the data available for analysis are

$$y_t = 100 * (X_{1,t} - X_{1,t-1}), \qquad t = 1, 2, ... \tag{23.3}$$

which corresponds directly to the interest rate series plotted in Figure 23.1. The parameters of the system are

$$\boldsymbol{\rho} = (\kappa_1,\ \mu,\ \gamma,\ \kappa_2,\ \alpha,\ \xi) \tag{23.4}$$

As is well known since Lo (1988), the likelihood of the observed process y_t given $\boldsymbol{\rho}$ under the system dynamics

$$d\mathbf{X}_t = \mathbf{A}(\mathbf{X}_t)\,dt + \mathbf{B}(\mathbf{X}_t)\,d\mathbf{W}_t \tag{23.5}$$

is not readily available in closed form. This aspect of the problem motivates the *method of moments* estimation and, in particular, *simulated method of moments* as in Ingram and Lee (1991) and Duffie and Singleton (1993) and the essentially equivalent *indirect inference* method proposed by Gourieroux, Monfort, and Renault (1993) and Smith (1993).

The system dynamics (23.5) suggest that, given a value for X_0, one could simulate an increment $X_\Delta - X_0$ from the process $\{X_t : 0 \le t < \infty\}$ for a small time value Δ as follows: Generate two independent normal $(0, \Delta^2)$ variates x_1 and x_2, simulate the Brownian motion increment $\mathbf{W}_\Delta - \mathbf{W}_0$ by putting

$$\mathbf{W}_\Delta - \mathbf{W}_0 = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

and simulate the increment $X_\Delta - X_0$ by putting

$$\mathbf{X}_\Delta - \mathbf{X}_0 = \mathbf{A}(\mathbf{X}_0)\Delta + \mathbf{B}(\mathbf{X}_0)\left(\mathbf{W}_\Delta - \mathbf{W}_0\right)$$

To simulate a value for X_t, sum over the increments:

$$\begin{aligned} \mathbf{X}_t - \mathbf{X}_0 &= \sum_{i=1}^{t/\Delta} \left(\mathbf{X}_{i\Delta} - \mathbf{X}_{(i-1)\Delta}\right) \qquad (23.6) \\ &= \sum_{i=1}^{t/\Delta} \mathbf{A}(\mathbf{X}_{(i-1)\Delta})\Delta + \sum_{i=1}^{t/\Delta} \mathbf{B}(\mathbf{X}_{(i-1)\Delta})\left(\mathbf{W}_{i\Delta} - \mathbf{W}_{(i-1)\Delta}\right) \end{aligned}$$

In passing, it is noted that under regularity conditions (Karatzas and Shreve, 1991), as Δ decreases, the random variable $\sum_{i=1}^{t/\Delta} \mathbf{B}(X_{(i-1)\Delta})(\mathbf{W}_{i\Delta} - \mathbf{W}_{(i-1)\Delta})$ converges in mean square to a random variable that has many properties of an integral and is therefore usually denoted as $\int_0^t \mathbf{B}(X_s)\, d\mathbf{W}_s$. Similarly, the Reimann sum $\sum_{i=1}^{t/\Delta} \mathbf{A}(X_{(i-1)\Delta})\Delta$ converges to $\int_0^t \mathbf{A}(X_s)\, ds$. The process $\{X_t : 0 \le t < \infty\}$ is interpreted as the solution to the integral equation

$$\mathbf{X}_t - \mathbf{X}_0 = \int_0^t \mathbf{A}(\mathbf{X}_s)\, ds + \int_0^t \mathbf{B}(\mathbf{X}_s)\, d\mathbf{W}_s$$

which exists under smoothness and growth conditions on the functions $\mathbf{A}(X)$ and $\mathbf{B}(X)$ (Karatzas and Shreve, 1991).

The important feature of this example, and of all the applications that are considered in this chapter, is that it can be simulated; that is, given a value for the parameters of the model, it is straightforward to generate a simulation $\{\hat{y}_t\}_{t=1}^N$ of arbitrary length N.

The simulation scheme (23.6) is known as an *Euler scheme*. This scheme and more sophisticated simulation schemes are discussed in Chapter 20, and in Kloeden and Platen (1992).

If the model (23.1) is stationary at a given value for $\boldsymbol{\rho}$, then a time-invariant stationary density

$$p(y_{-L}, \ldots, y_{-1}, y_0 | \boldsymbol{\rho}) \qquad (23.7)$$

exists for any stretch $(y_{t-L}, \ldots, y_{t-1}, y_t)$ of obsevables. If, in addition, (23.1) is ergodic at $\boldsymbol{\rho}$, then the expectation of a time-invariant (nonlinear) function $g(y_{-L}, \ldots, y_{-1}, y_0)$ with respect to (23.7)

$$E_\rho\left[g\right] = \int \cdots \int g(y_{-L}, \ldots, y_0) p(y_{-L}, \ldots, y_0 | \boldsymbol{\rho})\, dy_{-L} \cdots dy_0$$

can be approximated as accurately as desired by averaging over a simulation, namely

$$E_\rho\left[g\right] \doteq \frac{1}{N} \sum_{t=1}^{N} g(\hat{y}_{t-L}, \ldots, \hat{y}_{t-1}, \hat{y}_t)$$

Throughout, it is presumed that the requisite initial lags for this computation are primed via draws from the stationary distribution, which is usually accomplished by letting the system run long enough for transients to die out. For examples such as that above, one typically uses values of Δ on the order of 1/7 per week or 1/24 per day, a burn-in period of 1000 to 5000, and values of N on the order of 50,000 to 100,000.

The ability to compute $E_\rho[g]$ for given $\boldsymbol{\rho}$ and arbitrary $g(y_{-L}, \ldots, y_{-1}, y_0)$ means that model parameters can be computed by the method of moments or minimum chi-squared, as discussed in the next subsection.

23.2.2 Minimum Chi-Squared Estimators

In general, nonlinear systems are considered that have the features of the stochastic volatility model (23.1) just described. Specifically, (i) for a parameter vector $\boldsymbol{\rho}$ in a parameter space R, the random variables determined by the system have a stationary density

$$p(y_{-L}, \ldots, y_{-1}, y_0 | \boldsymbol{\rho}) \tag{23.8}$$

for every stretch $(y_{t-L}, \ldots y_t)$ and (ii) for $\boldsymbol{\rho} \in R$, the system is easily simulated so that expectations

$$E_\rho[g] = \int \cdots \int g(y_{-L}, \ldots, y_0) p(y_{-L}, \ldots, y_0 \mid \boldsymbol{\rho})\, dy_{-L} \cdots dy_0 \tag{23.9}$$

can be approximated as accurately as desired by averaging over a long simulation

$$E_\rho[g] \doteq \frac{1}{N} \sum_{t=1}^{N} g(\hat{y}_{t-L}, \ldots, \hat{y}_{t-1}, \hat{y}_t) \tag{23.10}$$

Henceforth, $\{y_t\}$ will be used to denote the stochastic process determined by the system, $\{\hat{y}_t\}_{t=1}^N$ to denote a simulation from the system, $\{\tilde{y}_t\}_{t=1}^n$ to denote data presumed to have been generated by the system, and $(y_{-L}, \ldots, y_{-1}, y_0)$ to denote function arguments and dummy variables of integration.

The true value of the parameter vector of the system (23.8) is denoted by $\boldsymbol{\rho}^o$.

A *classical method of moments* estimator $\hat{\boldsymbol{\rho}}_n$ of $\boldsymbol{\rho}^o$ is implemented by (i) setting forth a moment function, such as

$$\tilde{\psi}_c(y_{-L}, \ldots, y_{-1}, y_0) = \begin{pmatrix} y_0 & - & \tilde{\mu}_1 \\ y_0^2 & - & \tilde{\mu}_2 \\ & \vdots & \\ y_0^k & - & \tilde{\mu}_k \\ y_{-1}y_0 & - & \tilde{\gamma}(1) \\ y_{-2}y_0 & - & \tilde{\gamma}(2) \\ & \vdots & \\ y_{-L}y_0 & - & \tilde{\gamma}(L) \end{pmatrix} \tag{23.11}$$

where $\tilde{\mu}_j = \frac{1}{n}\sum_{t=1}^{n} \tilde{y}_t^j$, $\tilde{\gamma}(h) = \frac{1}{n}\sum_{t=1+h}^{n} \tilde{y}_t \tilde{y}_{t-h}$, (ii) computing the moment equations

$$\mathbf{m}_n(\boldsymbol{\rho}) = E_\rho[\boldsymbol{\psi}_c] = \int \cdots \int \tilde{\boldsymbol{\psi}}_c(y_{-L}, \ldots, y_0) p(y_{-L}, \ldots, y_0 \,|\, \rho)\, dy_{-L} \cdots dy_0$$

and (iii) attempting to solve the estimating equations

$$\mathbf{m}_n(\boldsymbol{\rho}) = \mathbf{0}$$

for the system parameters $\boldsymbol{\rho}$. If a solution $\hat{\boldsymbol{\rho}}_n$ can be found, then that solution is the *method of moments estimate* of the system parameters. As indicated earlier, the moment equations will usually have to be computed by generating a long simulation $\{\hat{y}_t\}_{t=-L}^{N}$ from the system at parameter setting $\boldsymbol{\rho}$ and then averaging over the simulation:

$$\mathbf{m}_n(\boldsymbol{\rho}) \doteq \frac{1}{N} \sum_{t=1}^{N} \tilde{\boldsymbol{\psi}}_c(\hat{y}_{t-L}, \ldots, \hat{y}_{t-1}, \hat{y}_t)$$

If there are multiple roots of the estimating equations $\mathbf{m}_n(\boldsymbol{\rho}) = \mathbf{0}$, a particular solution can be selected as the estimate using methods discussed in Heyde and Morton (1998).

In the event that there is no solution to the estimating equations because, for instance, the dimension of $\boldsymbol{\psi}_c$ is larger than the dimension of $\boldsymbol{\rho}$ (so that there are more equations than unknowns), then one must resort to *minimum chi-squared estimation* (Neyman and Pearson, 1928) as adapted to dynamic models (*GMM estimation*) by Hansen (1982). The minimum chi-squared, or GMM, estimator is obtained by using a nonlinear optimizer to minimize a quadratic form in the moment equations. Specifically,

$$\hat{\boldsymbol{\rho}}_n = \arg\min_{\rho} \; \mathbf{m}_n(\boldsymbol{\rho})' \tilde{\mathbf{S}}^{-1} \mathbf{m}_n(\boldsymbol{\rho})$$

where the matrix $\tilde{\mathbf{S}}$ appearing in the quadratic form is a heteroskedasticity and autocorrelation consistent (HAC) estimate of the variance of $\sqrt{n}\mathbf{m}_n(\boldsymbol{\rho})$ and may be computed using the methods described in Section 21.3 of Chapter 21.

If y_t is multivariate, that is,

$$\mathbf{y}_t = (y_{1,t}, \ldots, y_{M,t})'$$

then, instead of (23.11), the vector $\boldsymbol{\psi}_c$ is comprised of the elements

$$\prod_{i=1}^{M} (y_{i,0})^{\alpha_i} - \tilde{\mu}_\alpha$$

$$(y_{i,0})(y_{j,-h}) - \gamma_{ij}(h)$$

where

$$\begin{aligned}
\boldsymbol{\alpha} &= (\alpha_1, \ldots, \alpha_M)', \quad \alpha_i \geq 0, \quad 0 < \sum_{i=1}^{M} \alpha_i \leq K \\
\tilde{\mu}_\alpha &= \frac{1}{n} \sum_{t=1}^{n} \prod_{i=1}^{M} (\tilde{y}_{i,t})^{\alpha_i} \\
\gamma_{ij}(h) &= \frac{1}{n} \sum_{t=1+h}^{n} \tilde{y}_{i,t}\tilde{y}_{j,t-h}, \quad 0 \leq i \leq j \leq M, \quad 0 \leq h \leq L
\end{aligned}$$

The use of method of moments together with simulation to estimate the parameters of dynamic models with unobserved variables has been proposed by Ingram and Lee (1991), Duffie and Singleton (1993), Gourieroux, Monfort, and Renault (1993), Smith (1993), and others. The particular methods discussed next are due to Gallant and Tauchen (1996).

23.2.3 Efficiency Considerations

The previous subsection described a minimum chi-squared (GMM) estimation strategy based on the moment function $\tilde{\boldsymbol{\psi}}_c$ that can be used to estimate system parameters $\boldsymbol{\rho}$. There are two open questions with regard to this estimator: What is the best choice of the moment function $\tilde{\boldsymbol{\psi}}$? How many moments should be included in $\tilde{\boldsymbol{\psi}}$?

These questions will be considered in the simplest case where the random variables defined by the system (23.8) generate univariate independently and identically distributed random variables $\{y_t\}$ with density $p(y|\boldsymbol{\rho})$. The ideas for the general case of a multivariate, non-Markovian, stationary system are the same, but the algebra is far more complicated (Gallant and Long, 1997). Nothing essential is lost by considering the simplest case.

Consider three moment functions $\tilde{\boldsymbol{\psi}}_{c,n}$, $\tilde{\boldsymbol{\psi}}_{p,n}$, and $\tilde{\boldsymbol{\psi}}_{f,n}$ that correspond to the *Classical Method of Moments* (CMM), *Maximum Likelihood* (ML), and *Efficient Method of Moments* (EMM), respectively, defined as follows:

$$\tilde{\boldsymbol{\psi}}_{c,n}(y) = \begin{pmatrix} y - \frac{1}{n}\sum_{i=1}^{n} \tilde{y}_i \\ y^2 - \frac{1}{n}\sum_{i=1}^{n} (\tilde{y}_i)^2 \\ \vdots \\ y^K - \frac{1}{n}\sum_{i=1}^{n} (\tilde{y}_i)^K \end{pmatrix}$$

$$\tilde{\boldsymbol{\psi}}_{p,n}(y) = \frac{\partial}{\partial \rho} \log p(y|\tilde{\rho}_n)$$

$$\tilde{\boldsymbol{\psi}}_{f,n}(y) = \frac{\partial}{\partial \theta} \log f(y|\tilde{\theta}_n)$$

where the exponent K that appears in $\tilde{\boldsymbol{\psi}}_{c,n}(y)$ is the degree of the largest moment used in a method of moments application, the function $f(y|\boldsymbol{\theta})$ that appears in $\tilde{\boldsymbol{\psi}}_{f,n}(y)$ is a density that closely approximates the true data generating process in a sense made precise later, and the statistics $\tilde{\boldsymbol{\rho}}_n$ and $\tilde{\boldsymbol{\theta}}_n$ that appear in $\tilde{\boldsymbol{\psi}}_{p,n}(y)$ and $\tilde{\boldsymbol{\psi}}_{f,n}(y)$ are

$$\tilde{\boldsymbol{\rho}}_n = \arg\max_{\rho} \frac{1}{n}\sum_{i=1}^{n} \log p(\tilde{y}_i|\boldsymbol{\rho}),$$

$$\tilde{\boldsymbol{\theta}}_n = \arg\max_{\theta} \frac{1}{n}\sum_{i=1}^{n} \log f(\tilde{y}_i|\boldsymbol{\theta});$$

where $\boldsymbol{\rho}$ is of length p_ρ and $\boldsymbol{\theta}$ is of length $p_\theta \geq p_\rho$.

Note that each of the moment functions $\tilde{\boldsymbol{\psi}}_{p,n}$, $\tilde{\boldsymbol{\psi}}_{c,n}$, and $\tilde{\boldsymbol{\psi}}_{f,n}$ is in the null space of the expectation operator corresponding to the empirical distribution of the data, denoted as $E_{\tilde{F}_n}$; that is, $E_{\tilde{F}_n}\left[\tilde{\boldsymbol{\psi}}_{p,n}\right] = E_{\tilde{F}_n}\left[\tilde{\boldsymbol{\psi}}_{c,n}\right] = E_{\tilde{F}_n}\left[\tilde{\boldsymbol{\psi}}_{f,n}\right] = \mathbf{0}$. The method of moments is basically an attempt to do the same for the model $p(y|\boldsymbol{\rho})$; that is, the method of moments attempts to find a $\boldsymbol{\rho}$ that puts one of these moment functions, denoted generically as $\tilde{\boldsymbol{\psi}}_n$, in the null space of the expectation operator E_ρ corresponding to $p(y|\boldsymbol{\rho})$.

In addition to computing $\tilde{\boldsymbol{\psi}}_n$, one computes

$$\tilde{\mathbf{S}} = E_{\tilde{F}_n}\left[\tilde{\boldsymbol{\psi}}_n \tilde{\boldsymbol{\psi}}_n'\right]$$

Once $\tilde{\boldsymbol{\psi}}_n$ and $\tilde{\mathbf{S}}$ have been computed, the data are summarized and what is referred to as "the projection step" is finished.

For the estimation, define

$$\mathbf{m}_n(\boldsymbol{\rho}) = E_\rho\left[\tilde{\boldsymbol{\psi}}_n\right]$$

If the dimensions of $\boldsymbol{\rho}$ and $\tilde{\boldsymbol{\psi}}_n(y)$ are the same, then usually the equations $\mathbf{m}_n(\boldsymbol{\rho}) = \mathbf{0}$ can be solved to obtain an estimator $\hat{\boldsymbol{\rho}}_n$. For $\tilde{\boldsymbol{\psi}}_{p,n}$, the solution is the *maximum likelihood estimator* (Gauss, 1816; Fisher, 1912). For $\tilde{\boldsymbol{\psi}}_{c,n}$ with $K = p_\rho$, it is the *classical method of moments estimator* (Pearson, 1894). For $\tilde{\boldsymbol{\psi}}_{c,n}$ with $K > p_\rho$, no solution exists and the moment functions $\tilde{\boldsymbol{\psi}}_{c,n}$ are those of minimum chi-squared or generalized method of moments (Neyman and Pearson, 1928; Hansen, 1982) as customarily implemented.

As just noted, when $K > p_\rho$, then $\tilde{\boldsymbol{\psi}}_n$ cannot be placed in the null space of the operator E_ρ for any $\boldsymbol{\rho}$, because the equations $\mathbf{m}_n(\boldsymbol{\rho}) = \mathbf{0}$ have no solution. In this case, the minimum chi-squared estimator relies on the fact that under standard regularity conditions (Gallant and Tauchen, 1996) and choices of $\tilde{\boldsymbol{\psi}}_n$ similar to the above, there is a function $\boldsymbol{\psi}^o$ such that

$$\begin{aligned}
\lim_{n\to\infty} \tilde{\boldsymbol{\psi}}_n(y) &= \boldsymbol{\psi}^o(y) \quad \text{a.s.} \\
\lim_{n\to\infty} \tilde{\mathbf{S}} &= E_{\rho^o}\left[\boldsymbol{\psi}^o \boldsymbol{\psi}^{o\prime}\right] \quad \text{a.s.} \\
\sqrt{n}\,\mathbf{m}_n(\boldsymbol{\rho}^o) &\xrightarrow{d} N\left(\mathbf{0},\, E_{\rho^o}\left[\boldsymbol{\psi}^o \boldsymbol{\psi}^{o\prime}\right]\right)
\end{aligned}$$

where E_{ρ^o} denotes expectation taken with respect to $p(y|\boldsymbol{\rho}^o)$. For the three choices $\tilde{\boldsymbol{\psi}}_{p,n}$, $\tilde{\boldsymbol{\psi}}_{c,n}$, and $\tilde{\boldsymbol{\psi}}_{f,n}$ of $\boldsymbol{\psi}_n(y)$ above, the functions $\boldsymbol{\psi}^o_p$, $\boldsymbol{\psi}^o_c$, and $\boldsymbol{\psi}^o_f$ given by this result are

$$\boldsymbol{\psi}^o_c(y) = \begin{pmatrix} y - E_{\rho^o}[y] \\ y^2 - E_{\rho^o}[y^2] \\ \vdots \\ y^K - E_{\rho^o}[y^K] \end{pmatrix}$$

$$\boldsymbol{\psi}^o_p(y) = \frac{\partial}{\partial \rho} \log p(y|\boldsymbol{\rho}^o)$$

and

$$\boldsymbol{\psi}^o_f(y) = \frac{\partial}{\partial \theta} \log f(y|\boldsymbol{\theta}^o)$$

where

$$\boldsymbol{\theta}^o = \arg\max_{\theta} : E_{\rho^o}[\log f(\cdot|\boldsymbol{\theta})]$$

With these results in hand, $\boldsymbol{\rho}$ may be estimated by minimum chi-squared (GMM), viz.,

$$\hat{\boldsymbol{\rho}}_n = \arg\min_{\rho} : \mathbf{m}_n(\boldsymbol{\rho})'\,\tilde{\mathbf{S}}^{-1}\mathbf{m}_n(\boldsymbol{\rho})$$

and

$$\sqrt{n}(\hat{\boldsymbol{\rho}}_n - \boldsymbol{\rho}^o) \xrightarrow{d} N\left(\mathbf{0},\ (\mathbf{C}^o)^{-1}\right)$$

where

$$\mathbf{C}^o = E_{\rho^o}\left[\boldsymbol{\psi}^o_p\,\boldsymbol{\psi}^{o\prime}\right] E_{\rho^o}\left[\boldsymbol{\psi}^o\,\boldsymbol{\psi}^{o\prime}\right]^{-1} E_{\rho^o}\left[\boldsymbol{\psi}^o\,\boldsymbol{\psi}^{o\prime}_p\right]$$

Note that for any nonzero $\mathbf{a} \in \mathcal{R}^{p_\rho}$,

$$\min_{\mathbf{b}} E_{\rho^o}\left[\mathbf{a}'\boldsymbol{\psi}_p^o - \boldsymbol{\psi}^{o\prime}\mathbf{b}\right]^2 = E_{\rho^o}\left(\mathbf{a}'\boldsymbol{\psi}_p^o\right)^2 - \mathbf{a}'\mathbf{C}^o\mathbf{a} \geq 0 \tag{23.12}$$

Expression (23.12) implies that $\mathbf{a}'\mathbf{C}^o\mathbf{a}$ cannot exceed $E_{\rho^o}\left[\mathbf{a}'\boldsymbol{\psi}_p^o\right]^2 = \mathbf{a}'E_{\rho^o}\left[\boldsymbol{\psi}_p^o\,\boldsymbol{\psi}_p^{o\prime}\right]\mathbf{a}$ and therefore the best achievable asymptotic variance of the estimator $\hat{\boldsymbol{\rho}}_n$ is $\mathcal{I}_p^o = E_{\rho^o}\left[\boldsymbol{\psi}_p^o\,\boldsymbol{\psi}_p^{o\prime}\right]^{-1}$, which is the variance of the maximum likelihood estimator of $\boldsymbol{\rho}$. It is also apparent from (23.12) that if $\{\boldsymbol{\psi}_i^o\}_{i=1}^{\infty}$ spans the $L_{2,p}$ probability space $L_{2,p} = \{g : E_{\rho^o}\left[g^2\right] < \infty\}$ and $\boldsymbol{\psi}^o = (\psi_1^o, \ldots, \psi_K^o)$, then $\hat{\boldsymbol{\rho}}_n$ has good efficiency relative to the maximum likelihood estimator for large K. The polynomials span $L_{2,p}$ if $p(y|\boldsymbol{\rho})$ has a moment generating function (Gallant, 1980). Therefore, one might expect good asymptotic efficiency from $\tilde{\boldsymbol{\psi}}_{c,n}$ for large K.

Rather than just spanning $L_{2,p}$, EMM requires, in addition, that the moment functions actually be the score vector $\boldsymbol{\psi}_{f,n}(y)$ of some density $f(y|\tilde{\boldsymbol{\theta}}_n)$ that closely approximates $p(y|\boldsymbol{\rho}^o)$. Possible choices of $f(y|\tilde{\boldsymbol{\theta}}_n)$ are discussed in Gallant and Tauchen (1996). Of them, one commonly used in applications is the *SNP density*, which was proposed by Gallant and Nychka (1987) in a form suited to cross-sectional applications and by Gallant and Tauchen (1989) in a form suited to time series applications.

The SNP density, discussed in detail in the previous chapter, is obtained by expanding the square root of an innovation density $h(z)$ in a Hermite expansion

$$\sqrt{h(z)} = \sum_{i=0}^{\infty} \theta_i z^i \sqrt{\phi(z)}$$

where $\phi(\mathbf{z})$ denotes the standard normal density function. Because the Hermite functions are dense in L_2 (Lebesque) and $\sqrt{h(z)}$ is an L_2 function, this expansion must exist. The truncated density is

$$h_K(z) = \frac{\mathcal{P}_K^2(z)\phi(z)}{\int \mathcal{P}_K^2(u)\phi(u)\,du}$$

where

$$\mathcal{P}_K(z) = \sum_{i=0}^{K} \theta_i z^i$$

and the renormalization is necessary so that the density $h_K(z)$ integrates to 1. The location-scale transformation $y = \sigma z + \mu$ completes the definition of the SNP density

$$f_K(y|\boldsymbol{\theta}) = \frac{1}{\sigma}\,h_K\left(\frac{y-\mu}{\sigma}\right) \tag{23.13}$$

with $\boldsymbol{\theta} = (\mu, \sigma, \theta_0, \ldots, \theta_K)$. Gallant and Long (1997) have shown that

$$\boldsymbol{\psi}_f^o(y) = \frac{\partial}{\partial \boldsymbol{\theta}} \log f_K(y|\boldsymbol{\theta}^o)$$

with

$$\boldsymbol{\theta}^o = \arg\max_{\boldsymbol{\theta}} : E_{\rho^o}\left[\log f_K(\cdot|\boldsymbol{\theta})\right]$$

spans $L_{2,p}$.

Although a spanning argument can be used to show that high efficiency obtains for large K, it gives no indication as to what might be the best choice of moment functions with which to span $L_{2,p}$. Moreover, if $\boldsymbol{\psi}_p$ is in the span of $\boldsymbol{\psi}^o$ for some finite K, then full efficiency obtains at once (Gallant and Tauchen, 1996). For instance, the score of the normal density is in the span of both $\tilde{\boldsymbol{\psi}}_{c,n}$ and $\tilde{\boldsymbol{\psi}}_{f,n}$ for $K \geq 2$. These considerations seem to rule out any hope of general results showing that one moment function should be better than another.

With general results unattainable, the best one can do is compare efficiencies over a class of densities designed to stress test an estimator and over some densities thought to be representative of situations likely to be encountered in practice to see if any conclusions seem to be indicated. Comparisons using Monte Carlo methods are reported by Andersen, Chung, and Sorensen (1999), Chumacero (1997), Ng and Michaelides (2000), and van der Sluis (1999). Overall, their work supports the conjecture that EMM is more efficient than GMM in representative applications at typical sample sizes.

Analytical comparisons are possible for the independently and identically distributed case and are reported in Gallant and Tauchen (1999). Their measure of efficiency is the volume of a confidence region on the parameters of the density $p(y|\boldsymbol{\rho})$ computed using the asymptotic distribution of $\hat{\boldsymbol{\rho}}_n$. This region has the form $\{\boldsymbol{\rho} : (\boldsymbol{\rho} - \boldsymbol{\rho}^o)'(\mathbf{C}^o)^{-1}(\boldsymbol{\rho} - \boldsymbol{\rho}^o) \leq \mathcal{X}_d^2/n\}$ with volume

$$\frac{2\pi^{d/2}(\mathcal{X}_d^2/n)^d}{d\Gamma(d/2)\det(\mathbf{C}^o)}$$

where $\mathcal{X}_d^2$ denotes a critical value of the chi-squared distribution on d degrees of freedom. As small volumes are to be preferred and the region $\{\boldsymbol{\rho} : (\boldsymbol{\rho} - \boldsymbol{\rho}^o)'\mathcal{I}_p^o(\boldsymbol{\rho} - \boldsymbol{\rho}^o) \leq \mathcal{X}_d^2/n\}$ has the smallest achievable volume,

$$\text{RE} = \frac{\det(\mathbf{C}^o)}{\det(\mathcal{I}_p^o)}$$

is a measure of relative efficiency. Over a large collection of densities thought to represent typical applications, their computations support the conclusion that EMM dominates GMM. Moreover, their computations indicate that once $f_K(\cdot|\boldsymbol{\theta}^o)$ begins to approximate $p(\cdot|\boldsymbol{\rho}^o)$ accurately, the efficiency of the EMM estimator begins to increase rapidly.

The second question to address is how many moments to include in the moment function $\boldsymbol{\psi}_f$. As the computations in Gallant and Tauchen (1999) suggest, the answer is as many as is required for f to well approximate p. The natural conclusion is that one should use standard statistical model selection criteria to determine f. This approach has a distinct advantage over the use of $\boldsymbol{\psi}_c$ in that there seems to be no objective statistical criterion for determining the number of moments to include in $\boldsymbol{\psi}_c$.

23.2.4 A General Purpose Auxiliary Model

As indicated in the previous subsection, the best choice of a moment function $\boldsymbol{\psi}$ to implement a simulated method of moments is the score of an auxiliary model that closely approximates the density of the data. The SNP methodology of Gallant and Tauchen (1989) extended to the time series context serves as a general-purpose auxiliary model or score generator for use with the EMM estimation of general dynamic systems. In this context, the observed data are assumed to be a stationary and ergodic Markovian multivariate time series $\{\mathbf{y}_t\}_{t=-L+1}^{T}$, where each $\mathbf{y}_t$ is a vector of length M. Since $\mathbf{y}_t$ is Markovian, the conditional distribution of $\mathbf{y}_t$ given the entire past depends only on a finite number L of lagged values of $\mathbf{y}_t$ denoted $\mathbf{x}_{t-1} = (\mathbf{y}'_{t-1}, \ldots, \mathbf{y}'_{t-L})'$. The SNP model is then used to approximate the transition density $p(\mathbf{y}_t|\mathbf{x}_{t-1}, \boldsymbol{\rho}^o)$. The SNP methodology and its implementation in `S+FinMetrics` is described in the previous chapter.

23.2.5 The Projection Step

The best choice of a moment function to implement simulated method of moments estimation of general dynamic systems is the score of an auxiliary model $f(\mathbf{y}|\mathbf{x}, \boldsymbol{\theta})$ that closely approximates the transition density implied by the system, where the parameter vector $\boldsymbol{\theta}$ of the auxiliary model is evaluated at its *quasi-maximum-likelihood* estimate $\tilde{\boldsymbol{\theta}}_n$; that is, the best choice has the form

$$\tilde{\boldsymbol{\psi}}_f(\mathbf{x}, \mathbf{y}) = \frac{\partial}{\partial \boldsymbol{\theta}} \log f(\mathbf{y}|\mathbf{x}, \tilde{\boldsymbol{\theta}}_n) \tag{23.14}$$

where

$$\begin{aligned} \tilde{\boldsymbol{\theta}}_n &= \arg\min_{\theta}\ s_n(\boldsymbol{\theta}) \qquad (23.15) \\ s_n(\boldsymbol{\theta}) &= -\frac{1}{n}\sum_{t=1}^{n} \log f(\tilde{\mathbf{y}}_t|\tilde{\mathbf{x}}_{t-1}, \boldsymbol{\theta}) \end{aligned}$$

A considerable advantage of closely approximating the transition density of the system is that the computational formula for the weighting matrix $\tilde{\mathbf{S}}$ for the chi-squared (GMM) estimator simplifies to

$$\tilde{\mathbf{S}} = \frac{1}{n}\sum_{t=1}^{n} \tilde{\boldsymbol{\psi}}_f(\tilde{\mathbf{x}}_{t-1}, \tilde{\mathbf{y}}_t)\tilde{\boldsymbol{\psi}}'_f(\tilde{\mathbf{x}}_{t-1}, \tilde{\mathbf{y}}_t) \tag{23.16}$$

because the covariance terms of the HAC estimator can be neglected when the auxiliary model closely approximates the transition density of the system in a sense made precise by Gallant and Long (1997).

23.2.6 The Estimation Step

The objectives are (i) to estimate $\boldsymbol{\rho}$, (ii) test the hypothesis that the dynamic system under consideration generated the observed data $\{\tilde{\mathbf{y}}_t\}_{t=1}^n$, and (iii) provide diagnostics that indicate how a rejected system should be modified to better describe the distribution of the observable process $\{\mathbf{y}_t\}$.

The EMM Estimator

It is presumed that the data have been summarized in the projection step, as described in the previous subsection, and that a moment function of the form

$$\tilde{\boldsymbol{\psi}}_f(\mathbf{x},\mathbf{y}) = \frac{\partial}{\partial\boldsymbol{\theta}} \log f(\mathbf{y} \,|\, \mathbf{x}, \tilde{\boldsymbol{\theta}}_n)$$

and a weighting matrix

$$\tilde{\mathbf{S}} = \frac{1}{n}\sum_{t=1}^{n}\left[\frac{\partial}{\partial\boldsymbol{\theta}} \log f(\tilde{\mathbf{y}}_t \,|\, \tilde{\mathbf{x}}_{t-1}, \tilde{\boldsymbol{\theta}}_n)\right]\left[\frac{\partial}{\partial\theta} \log f(\tilde{\mathbf{y}}_t \,|\, \tilde{\mathbf{x}}_{t-1}, \tilde{\boldsymbol{\theta}}_n)\right]'$$

are available from the projection step. Here, it is assumed that $f(\mathbf{y} \,|\, \mathbf{x}, \tilde{\boldsymbol{\theta}}_n)$ closely approximates $p(\mathbf{y} \,|\, \mathbf{x}, \boldsymbol{\rho}^o)$. If not, the weighting matrix given by a HAC estimator must be used. If the SNP density $f_K(\mathbf{y} \,|\, \mathbf{x}, \boldsymbol{\theta})$ is used as the auxiliary model with tuning parameters selected by the Bayesian Information Criterion (BIC), $\tilde{\mathbf{S}}$ as computed above will be adequate (Gallant and Long, 1997; Gallant and Tauchen, 1999; Coppejans and Gallant, 2002).

Here, it is explicitly indicated that the dependence on n of the moment equations $\mathbf{m}_n(\boldsymbol{\rho})$ enters through the quasi-maximum-likelihood estimate $\tilde{\boldsymbol{\theta}}_n$ by writing $\mathbf{m}(\boldsymbol{\rho}, \tilde{\boldsymbol{\theta}}_n)$ for $\mathbf{m}_n(\boldsymbol{\rho})$, where

$$\mathbf{m}(\boldsymbol{\rho}, \boldsymbol{\theta}) = E_\rho\left[\frac{\partial}{\partial\boldsymbol{\theta}} \log f(\mathbf{y} \,|\, \mathbf{x}, \boldsymbol{\theta})\right]$$

Recall that the moment equations of the minimum chi-squared procedure are computed by averaging over a long simulation:

$$\mathbf{m}(\boldsymbol{\rho}, \tilde{\boldsymbol{\theta}}_n) \doteq \frac{1}{N}\sum_{t=1}^{N}\frac{\partial}{\partial\boldsymbol{\theta}} \log f(\hat{\mathbf{y}}_t \,|\, \hat{\mathbf{x}}_{t-1}, \tilde{\boldsymbol{\theta}}_n)$$

The *EMM estimator* is

$$\hat{\boldsymbol{\rho}}_n = \arg\min_{\boldsymbol{\rho}}\ \mathbf{m}(\boldsymbol{\rho}, \tilde{\boldsymbol{\theta}}_n)'\tilde{\mathbf{S}}^{-1}\mathbf{m}(\boldsymbol{\rho}, \tilde{\boldsymbol{\theta}}_n)$$

Asymptotic Properties

The asymptotics of the estimator are as follows. If $\boldsymbol{\rho}^o$ denotes the true value of $\boldsymbol{\rho}$ and $\boldsymbol{\theta}^o$ is an isolated solution of the moment equations $\mathbf{m}(\boldsymbol{\rho}^o, \boldsymbol{\theta}) = \mathbf{0}$, then under regularity conditions (Gallant and Tauchen, 1996; Gallant and Long, 1997),

$$\lim_{n\to\infty} \hat{\boldsymbol{\rho}}_n = \boldsymbol{\rho}^o \quad \text{a.s.}$$
$$\sqrt{n}(\hat{\boldsymbol{\rho}}_n - \boldsymbol{\rho}^o) \xrightarrow{d} N\left\{\mathbf{0}, [\mathbf{M}^{o\prime}\mathcal{I}^{o-1}\mathbf{M}^o]^{-1}\right\} \tag{23.17}$$
$$\lim_{n\to\infty} \hat{\mathbf{M}}_n = \mathbf{M}^o \quad \text{a.s}$$
$$\lim_{n\to\infty} \tilde{\mathbf{S}} = \mathcal{I}^o \quad \text{a.s.}$$

where $\hat{\mathbf{M}}_n = \mathbf{M}(\hat{\boldsymbol{\rho}}_n, \tilde{\boldsymbol{\theta}}_n)$, $\mathbf{M}^o = \mathbf{M}(\boldsymbol{\rho}^o, \boldsymbol{\theta}^o)$, $\mathbf{M}(\boldsymbol{\rho}, \boldsymbol{\theta}) = (\partial/\partial\boldsymbol{\rho}')\mathbf{m}(\boldsymbol{\rho}, \boldsymbol{\theta})$, and

$$\mathcal{I}^o = E_{\rho^o}\left[\frac{\partial}{\partial\boldsymbol{\theta}} \log f(\mathbf{y}_0 \,|\, \mathbf{x}_{-1}, \boldsymbol{\theta}^o)\right]\left[\frac{\partial}{\partial\boldsymbol{\theta}} \log f(\mathbf{y}_0 \,|\, \mathbf{x}_{-1}, \boldsymbol{\theta}^o)\right]'$$

Under the null hypothesis that $p(y_{-L}, \ldots, y_0 \,|\, \boldsymbol{\rho})$ is the correct model, the *EMM J-statistic*

$$J_0 = n\,\mathbf{m}(\hat{\boldsymbol{\rho}}_n, \tilde{\boldsymbol{\theta}}_n)'\tilde{\mathbf{S}}^{-1}\mathbf{m}(\hat{\boldsymbol{\rho}}_n, \tilde{\boldsymbol{\theta}}_n) \tag{23.18}$$

is asymptotically chi-squared on $p_\theta - p_\rho$ degrees of freedom.

Hypothesis Tests

Under the null hypothesis that $h(\boldsymbol{\rho}^o) = \mathbf{0}$, where h maps R into $\Re^q$, the *EMM likelihood ratio (LR)-type statistic*

$$\text{LR}_{\text{EMM}} = n\left[\mathbf{m}(\breve{\boldsymbol{\rho}}_n, \tilde{\boldsymbol{\theta}}_n)'\tilde{\mathbf{S}}^{-1}\mathbf{m}(\breve{\boldsymbol{\rho}}_n, \tilde{\boldsymbol{\theta}}_n) - \mathbf{m}(\hat{\boldsymbol{\rho}}_n, \tilde{\boldsymbol{\theta}}_n)'\tilde{\mathbf{S}}^{-1}\mathbf{m}(\hat{\boldsymbol{\rho}}_n, \tilde{\boldsymbol{\theta}}_n)\right] \tag{23.19}$$

is asymptotically chi-squared on q degrees of freedom where

$$\breve{\boldsymbol{\rho}}_n = \arg\min_{\rho} \mathbf{m}(\boldsymbol{\rho}, \tilde{\boldsymbol{\theta}}_n)'\tilde{\mathbf{S}}^{-1}\,\mathbf{m}(\boldsymbol{\rho}, \tilde{\boldsymbol{\theta}}_n) \text{ s.t. } h(\boldsymbol{\rho}) = \mathbf{0}$$

A Wald confidence interval on an element ρ_i of $\boldsymbol{\rho}$ can by constructed in the usual way from an asymptotic standard error $\sqrt{\hat{\sigma}_{ii}}$. A standard error may be obtained by computing the Jacobian $\mathbf{M}_n(\boldsymbol{\rho}, \boldsymbol{\theta})$ numerically and taking the estimated asymptotic variance $\hat{\sigma}_{ii}$ to be the ith diagonal element of $\hat{\boldsymbol{\Sigma}} = (1/n)[\hat{\mathbf{M}}_n'\tilde{\mathbf{S}}^{-1}\hat{\mathbf{M}}_n]^{-1}$. These intervals, which are symmetric, are somewhat misleading because they do not reflect the rapid increase in the EMM objective function $s_n(\boldsymbol{\rho}) = \mathbf{m}(\boldsymbol{\rho}, \tilde{\boldsymbol{\theta}}_n)'\tilde{\mathbf{S}}^{-1}\mathbf{m}(\boldsymbol{\rho}, \tilde{\boldsymbol{\theta}}_n)$ when ρ_i approaches a value for which the system under consideration is explosive. Confidence intervals obtained by inverting the criterion difference test LR_{EMM} do reflect this phenomenon and are therefore more useful. To invert the test one puts in the interval those ρ_i^* for which $\mathrm{LR}_{\mathrm{EMM}}$ for the hypothesis $\rho_i^o = \rho_i^*$ is less than the critical point of a chi-squared on one degree of freedom. To avoid re-optimization one may use the approximation

$$\breve{\boldsymbol{\rho}}_n = \hat{\boldsymbol{\rho}}_n + \frac{\rho_i^* - \hat{\rho}_{in}}{\hat{\sigma}_{ii}} \hat{\boldsymbol{\Sigma}}_{(i)}$$

in the formula for $\mathrm{LR}_{\mathrm{EMM}}$ where $\hat{\boldsymbol{\Sigma}}_{(i)}$ is the ith column of $\hat{\boldsymbol{\Sigma}}$. The above remarks should only be taken to imply that confidence intervals obtained by inverting the criterion difference test have more desirable structural characteristics than those obtained by inverting the Wald test and not that they have more accurate coverage probabilities.

Diagnostics

When J_0 exceeds the chi-squared critical point, diagnostics that suggest improvements to the system are desirable. Because

$$\sqrt{n}\,\mathbf{m}(\hat{\boldsymbol{\rho}}_n, \tilde{\boldsymbol{\theta}}_n) \xrightarrow{d} N\left\{\mathbf{0}, \mathcal{I}^o - \mathbf{M}^o[\mathbf{M}^{o\prime}(\mathcal{I}^o)^{-1}\mathbf{M}^o]^{-1}\mathbf{M}^{o\prime}\right\} \tag{23.20}$$

inspection of the t-ratios

$$T_n = \mathbf{S}_n^{-1}\sqrt{n}\,\mathbf{m}(\hat{\boldsymbol{\rho}}_n, \tilde{\boldsymbol{\theta}}_n) \tag{23.21}$$

where $\mathbf{S}_n = \left(\mathrm{diag}\{\tilde{\mathbf{S}} - \hat{\mathbf{M}}_n[\hat{\mathbf{M}}_n'\tilde{\mathbf{S}}^{-1}\hat{\mathbf{M}}_n]^{-1}\hat{\mathbf{M}}_n'\}\right)^{1/2}$, can suggest reasons for failure. Different elements of the score correspond to different characteristics of the data and large t-ratios reveal those characteristics that are not well approximated.

23.3 EMM Estimation in S+FinMetrics

A general class of discrete-time and continuous-time dynamic time series models may be estimated by EMM using the S+FinMetrics function EMM. The arguments expected by EMM are

```
> args(EMM)
function(score, coef = NULL, appcode = NULL, ui, ur, control
= EMM.control(), est = NULL, save.sim = F, trace = T,
gensim.fn = NULL, gensim.language = "C", gensim.aux =
NULL, save.intermediate = F)
```

These arguments are summarized in Table 23.1. The main arguments are score, which is an object of class "SNP" representing the fit of the auxiliary model to the observed data, coef, which is a vector containing starting

Argument	Description
`score`	Object of class "SNP" representing the auxiliary model fit.
`coef`	Starting values for the model coefficients.
`appcode`	Integer identifying the simulation function built into EMM.
`ui`	Vector of integer parameters to the simulation function.
`ur`	Vector of real parameters to the simulation function.
`control`	List of EMM control parameters. Values set using `EMM.control`.
`est`	Vector of logical values indicating which coefficients are held fixed.
`save.sim`	If `TRUE`, simulated values at final iteration are saved.
`trace`	If `TRUE`, iteration count and objective are printed to screen.
`gensim.fn`	The simulation function. If `NULL`, then value is specified by appcode.
`gensim.language`	The language of the simulation function: C, FORTRAN or SPLUS.
`gensim.aux`	An `S-PLUS` object that will be passed to the simulation function.
`save.intermediate`	If `TRUE`, intermediate results are saved.

TABLE 23.1. Arguments to function `EMM`

values for the model coefficients to be estimated, and `gensim.fn`, which is a function to create simulations from the structural model given a set of parameters. The `score` object is usually created by a call to the `S-PLUS` function `SNP`. See the previous chapter for details on the creation of "`SNP`" objects. The function specified by `gensim.fn` is called repeatedly by the `EMM` function to generate simulated data as a function of a set of model parameters. For EMM to work correctly, the same random numbers must be used to generate simulations for different values of the model parameters. The simulation function may be defined as an `S-PLUS` function or a C or FORTRAN function.[4] If an `S-PLUS` function is supplied, then the user must set `gensim.language="SPLUS"`. Details on creating an `S-PLUS` simulation function for use with the function `EMM` are given in the next section. The function `EMM` creates an object of class "`EMM`" for which there are `print`, `summary`, and `plot` methods, and extractor functions `coef` and `vcov`.

[4]See the online help for `EMM` for details on specifying a C or FORTRAN function.

The `control` component of `EMM` is a list containing parameters that control aspects of the EMM estimation. Defaults for these parameters are set by the function `EMM.control`:

```
> args(EMM.control)
function(initial.itmax = 10, final.itmax = 200, tol = 1e-005,
n.burn = 100, n.sim = 10000, n.start = 0, tweak = 0,
siglev = 0.95, sfac1 = 1, sfac2 = 2, sfdrho = 1e-006,
seed = 61883, optimize = T, cf.interval = T, std.err
= T, scale = 1)
```

These components are summarized in Table 23.2. The most important components are `n.burn` and `n.sim`, which specify the number of burn-in and simulation values, respectively, for the simulation function specified by the argument `gensim.fn`. In complicated nonlinear models for which good starting values are hard to obtain, it is recommended that the random restart option for the optimizer be utilized. This feature allows the user to restart the EMM optimization at random perturbations of the initial values in order to explore the surface of the EMM objective function and avoid getting stuck at possible local minima. The component `initial.itmax` sets the maximum number of iterations for each random restart, and `final.itmax` determines the maximum number of iterations in the final estimation. The number of random restarts is determined by `n.start`, which may be a scalar or a vector and must have the same length as `tweak`. If `n.start` is a vector, then the ith element of `n.start` determines how many random restarts are performed for the corresponding value in the ith element of `tweak`. If `n.start=0`, then `tweak` is ignored. The component `tweak` contains tweak constants for random restarts of the optimizer. The starting value for each coefficient, ρ_i, is perturbed such that ρ_i gets transformed to $(1+u\times$ `tweak`$)\times\rho_i$, where u is a random draw from the uniform distribution on $(-1,1)$.

23.3.1 Simulator Functions

The user is required to supply a function to generate simulations from the model under study as a function of the model parameters. This function may be an `S-PLUS`, C, or FORTRAN function. The following subsections describe how to create these functions.

`S-PLUS` Simulator Functions

The easiest way to use EMM is with a user-written `S-PLUS` simulator function. The simulator function supplied to the `gensim.fn` argument of `EMM` must have the form

```
my.gensim = function(rho, n.sim, n.var, n.burn, aux) {
... }
```

Component	Description
`initial.itmax`	Maximum number of iterations used for each random restart.
`final.itmax`	Maximum number of iterations used for final model
`tol`	convergence tolerance for optimization.
`n.burn`	Number of burn-in values to discard from beginning of simulation.
`n.sim`	Number of simulated values for each series to be simulated.
`n.start`	Number of random restarts of the optimizer.
`tweak`	Tweak constants for random restarts of the optimizer.
`siglevel`	Significance level for confidence intervals.
`sfac1`	Scale factor for confidence intervals.
`sfac2`	Scale factor for confidence intervals.
`sfdrho`	Perturbation to rho for numerical derivatives.
`seed`	Random number seed used for random restarts.
`optimize`	If `TRUE`, perform optimization.
`cf.interval`	If `TRUE`, compute confidence intervals by inverting LR_{EMM} -statistic.
`std.err`	If `TRUE`, compute Wald standard errors.
`scale`	scale factor used by the `S-PLUS` optimizer `nlminb`.

TABLE 23.2. Arguments to `EMM.control`

where `rho` gives the model parameters, `n.sim` specifies the number of simulated observations, `n.var` sets the number of simulated variables, `n.burn` gives the number of burn-in observations, `aux` is an `S-PLUS` structure containing any auxiliary components required by the simulation function, and `...` represents the body of the function. The function `my.gensim` must return a numeric vector of length `n.sim*n.var`, containing the simulated values.[5]

In most applications, the structural model to be estimated is a discrete-time or continuous-time dynamic time series model. For computational efficiency, it is extremely important that the simulation function for the structural model be fast. For discrete-time models, it is recommended that the user utilize existing `S-PLUS` or `S+FinMetrics` simulation functions within the body of the function specified by the `gensim.fn` argument since these functions rely on underlying C or FORTRAN code to speed computations. A listing of the simulation functions available in `S-PLUS` and `S+FinMetrics` is given in Table 23.3. The examples in the following sections illustrate the

[5]If simulations for a multivariate time series are required, then the function still returns a numeric vector. The multivariate simulations must be ordered such that the first `n.var` elements correspond to the first simulated values of all the series, the second `n.var` elements correspond to the second simulated values of all the series, an so on.

Simulation Function	**Source**	**Description**
`arima.fracdiff.sim`	`S-PLUS`	Simulate from fractional ARIMA model.
`arima.sim`	`S-PLUS`	Simulate from ARIMA model.
`filter`	`S-PLUS`	Compute convolution or recursive filter.
`simulate.FARIMA`	`S+FinMetrics`	Simulate from fractional ARIMA model.
`simulate.garch`	`S+FinMetrics`	Simulate from univariate garch model.
`simulate.mgarch`	`S+FinMetrics`	Simulate from multivariate garch model.
`simulate.SNP`	`S+FinMetrics`	Simulate from SNP models.
`simulate.VAR`	`S+FinMetrics`	Simulate from vector autoregressive model.
`SsfSim`	`S+FinMetrics`	Simulate from linear state space model.
`atsm.ssfsim`	`S+FinMetrics`	Simulate from affine term structure state space model.
`simulate.SVOL`	`S+FinMetrics`	Simulate from discrete-time stochastic volatility model.
`ssfSimMS`	`S+FinMetrics`	Simulate from Markov switching state space model.

TABLE 23.3. `S-PLUS` and `S+FinMetrics` simulation functions

creation of user-specified simulation functions for discrete-time dynamic models utilizing the functions from Table 23.3.

For continuous-time dynamic time series models, simulations are often generated as the solution of a system of stochastic differential equations (SDEs) of the form

$$d\mathbf{X}_t = \mathbf{a}(\mathbf{X}_t, t)dt + \mathbf{b}(\mathbf{X}_t, t)d\mathbf{W}_t \tag{23.22}$$

where $\mathbf{X}_t$ is a univariate or multivariate stochastic process, $\mathbf{a}(\mathbf{X}_t, t)$ is the drift function, $\mathbf{b}(\mathbf{X}_t, t)$ is the diffusion function, and $\mathbf{W}_t$ is a standard Brownian motion (Wiener) process. This SDE may be univariate or multivariate. For most applications, the SDE solution must be approximated using numerical methods such as Euler's method. `S+FinMetrics` contains several functions for numerically simulating the solution to univariate and multivariate SDEs, and the main functions are listed in Table 23.4. Chapter 20 describes the solution methods underlying these functions in detail and provides many examples describing how to use the functions. The four p-code functions are particularly handy since they allow `S-PLUS` functions

Simulation function	Description
`euler1d.pcode.gensim`	Solve univariate SDE using Euler's method
`euler.pcode.gensim`	Solve SDE using Euler's method
`strong1.pcode.gensim`	Solve SDE using Strong order 1 scheme
`weak2.pcode.gensim`	Solve SDE using Weak order 2 scheme
`CIR.gensim`	Euler's method for Cox-Ingersoll-Ross square root diffusion
`IRD.gensim`	Euler's method for two-factor interest rate diffusion
`OU.gensim`	Exact solution for Ornstein-Uhlenbeck (or Vasicek) SDE

TABLE 23.4. Simulation functions for continuous-time SDEs

to be used for specifying arbitrary drift and diffusion functions $\mathbf{a}(\mathbf{X}_t, t)$ and $\mathbf{b}(\mathbf{X}_t, t)$, respectively. These p-code functions are heavily utilized in the examples in the following sections.

C or FORTRAN Simulator Functions

The p-code functions listed in Table 23.4 for simulating the solutions to the SDE (23.22) are flexible and easy to use, but they generally run two to five times slower than compiled C code. These functions, however, utilize underlying C code. If computational speed is important, the user may wish to write their own simulator functions in C or FORTRAN that call the C code functions for implementing the SDE solution methods.

23.3.2 SNP Auxiliary Model Estimation

The auxiliary model, or score generator, used to summarize the observed data is created using the `S+FinMetrics` function `SNP`. The previous chapter describes the `SNP` and associated functions and gives several examples of fitting SNP models to common financial time series. The fitted SNP models from these examples are used as score generators for some of the examples of EMM estimation in the following sections.

23.4 Examples

The following subsections describe the estimation of some common univariate and multivariate time series models for financial data using the function `EMM`. The examples cover the estimation of moving average models, discrete-time and continuous-time stochastic volatility models for equity returns, and continuous-time stochastic volatility models for interest rates. Some of the examples utilize the financial time series described and analyzed in the preceding chapter, and the reader is referred to that chapter for the details of fitting the SNP auxiliary models. The examples illustrate the creation of S-PLUS simulation functions for use with EMM, and describe in detail the many options for controlling the EMM estimation. The examples also illustrate aspects of statistical inference and model diagnostics associated with EMM estimation.

23.4.1 MA(1) Model

Following Chumacero (1997), consider the first-order moving average, MA(1), model

$$\begin{aligned} y_t &= \mu_0 + \varepsilon_t + \psi_0 \varepsilon_{t-1},\ t = 1, \ldots, n \\ &\quad \varepsilon_t \sim \text{iid } N(0, \sigma_0^2) \\ \boldsymbol{\rho}_0 &= (\mu_0, \psi_0, \sigma_0^2)' = \text{true value} \end{aligned} \tag{23.23}$$

The MA(1) model is a convenient example for illustrating the mechanics of EMM because it is easy to simulate, there is a simple auxiliary model based on a pth order autoregressive, AR(p), model, and the method of moments and maximum likelihood estimators are easy to compute. The GMM estimation of (23.23) was discussed in Chapter 21.

Given the model parameters $\boldsymbol{\rho}$, it is straightforward to simulate values for y_t. In S-PLUS, the function `arima.sim` may be used to efficiently generate simulated values. For $\boldsymbol{\rho}_0 = (0, 0.5, 1)'$ and $n = 250$, simulated values may be created using

```
> set.seed(123)
> ma1.sim = arima.sim(model=list(ma=-0.5),n=250)
```

The simulated y_t are illustrated in Chapter 21, Figure 21.2, along with the sample autocorrelation function (SACF) and sample partial autocorrelation function (SPACF).

Auxiliary Model

If $|\psi| < 1$, then the MA(1) is invertible and has the AR(∞) representation

$$\begin{aligned} y_t &= c + \sum_{j=1}^{\infty} \phi_j y_{t-j} + \varepsilon_t \\ \phi_j &= (-1) \cdot (-\psi)^j,\ c = \mu/(1+\psi) \end{aligned}$$

A feasible auxiliary model is an AR(p) model

$$\begin{aligned} y_t &= c + \phi_1 y_{t-1} + \cdots + \phi_p y_{t-p} + u_t \\ &\quad u_t \sim \text{iid } N(0, \sigma_u^2) \\ \boldsymbol{\theta} &= (c, \phi_1, \ldots, \phi_p, \sigma_u^2)' \end{aligned}$$

with p sufficiently large to adequately capture the dynamics of the MA(1) model. The log-density for the auxiliary model is

$$\begin{aligned}\ln f(y_t|\mathbf{x}_{t-1},\boldsymbol{\theta}) &= -\frac{1}{2}\ln(2\pi)-\frac{1}{2}\ln(\sigma_u^2)\\ &\quad -\frac{1}{2\sigma_u^2}(y_t-c-\phi_1 y_{t-1}-\cdots-\phi_p y_{t-p})^2\end{aligned}$$

where $\mathbf{x}_{t-1}=(y_{t-1},\ldots,y_{t-p})'$. If $p\to\infty$ at rate $n^{1/3}$ then $AR(p)$ will encompass the MA(1) data generating process as $n\to\infty$.

Quasi-Maximum-Likelihood Estimation

Given a sample of observed data $\{\tilde{y}_t\}_{t=1}^n$ based on $\boldsymbol{\rho}_0$, the quasi-maximum-likelihood estimation of the auxiliary model parameters $\boldsymbol{\theta}$ based on the conditional log-likelihood solves

$$\tilde{\boldsymbol{\theta}}_n=\arg\min_{\boldsymbol{\theta}}\ s_n(\boldsymbol{\theta})=-\frac{1}{n}\sum_{t=1}^{n}\ln f(\tilde{y}_t|\tilde{\mathbf{x}}_{t-1},\boldsymbol{\theta})$$

The sample score vector $s_n(\tilde{\boldsymbol{\theta}}_n)$ has $p+2$ elements

$$\begin{aligned}&\frac{-1}{n\tilde{\sigma}_u^2}\sum_{t=1}^{n}(\tilde{y}_t-\tilde{c}-\tilde{\phi}_1\tilde{y}_{t-1}-\cdots-\tilde{\phi}_p\tilde{y}_{t-p})\\ &\frac{-1}{n\tilde{\sigma}_u^2}\sum_{t=1}^{n}(\tilde{y}_t-\tilde{c}-\tilde{\phi}_1\tilde{y}_{t-1}-\cdots-\tilde{\phi}_p\tilde{y}_{t-p})\tilde{y}_{t-1}\\ &\vdots\\ &\frac{-1}{n\tilde{\sigma}_u^2}\sum_{t=1}^{n}(\tilde{y}_t-\tilde{c}-\tilde{\phi}_1\tilde{y}_{t-1}-\cdots-\tilde{\phi}_p\tilde{y}_{t-p})\tilde{y}_{t-p}\\ &\frac{1}{2\tilde{\sigma}^2}[n+\tilde{\sigma}^{-2}]\sum_{t=1}^{n}(\tilde{y}_t-\tilde{c}-\tilde{\phi}_1\tilde{y}_{t-1}-\cdots-\tilde{\phi}_p\tilde{y}_{t-p})\end{aligned}$$

and by construction, $s_n(\tilde{\boldsymbol{\theta}}_n)=\mathbf{0}$.

Estimating the Auxiliary Model Using `SNP`

For the EMM estimation in `S-PLUS`, the auxiliary model must be estimated using the function `SNP`. The `SNP` function estimates Gallant and Tauchen's general-purpose seminonparametric auxiliary model for stationary multivariate time series, as described in the previous chapter, and can be used to estimate a wide class of auxiliary models, including the AR(p) model. The object returned by `SNP` contains the information necessary to compute the analytic score required for computing the EMM objective function.

For illustrative purposes, let the auxiliary model for the MA(1) data be a Gaussian AR(3) model. This model has a 5×1 parameter vector $\boldsymbol{\theta} = (\mu, \phi_1, \phi_2, \phi_3, \sigma_u^2)'$ and may be estimated using SNP as follows:

```
> ar3.fit = SNP(data=ma1.sim, model=SNP.model(ar=3))
> class(ar3.fit)
[1] "SNP"
> summary(ar3.fit)

Call:
SNP(data = ma1.sim, model = SNP.model(ar = 3))

Model: Gaussian VAR

Conditional Mean Coefficients:
               mu   ar(1)   ar(2)   ar(3)
     coef -0.0045  0.5312 -0.2328  0.1007
(std.err)  0.0565  0.0637  0.0690  0.0659
 (t.stat) -0.0788  8.3444 -3.3763  1.5280


Conditional Variance Coefficients:
            sigma
     coef  0.8828
(std.err)  0.0462
 (t.stat) 19.1253


Information Criteria:
  BIC     HQ    AIC      logL
 1.35 1.3288 1.3145 -319.6804

Convergence Type:
both x and relative function convergence
number of iterations: 1
```

For successful estimation using EMM, it is vitally important that the auxiliary model provide a good fit to the observed data and be able to replicate the important dynamic features of the data. For the AR(3) auxiliary model, the estimated values for ϕ_1, ϕ_2, and ϕ_3 are all statistically different from zero and have values that are consistent with $\phi_j = -(-0.5)^j$ for $j = 1, 2, 3$. In addition, the estimated value of σ_u is close to $\sigma_\varepsilon = 1$. Graphical residual diagnostics, produced using

```
> plot(ar3.fit, which.plots=3:6)
```

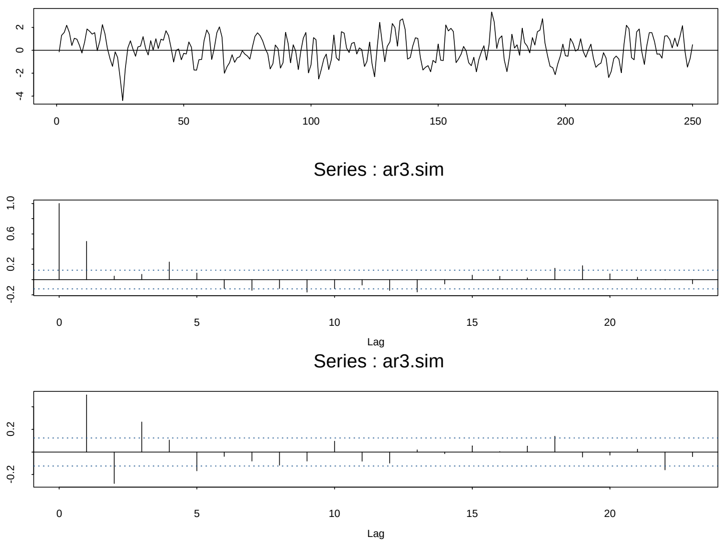


FIGURE 23.2. Simulated data, SACF and PACF from the AR(3) auxiliary model fit to simulated data from the MA(1) model.

indicate that the AR(3) has adequately captured the dynamics in the data. To be sure, simulated data from the AR(3) auxiliary model should qualitatively match the characteristics of the MA(1) data. Simulations from the fitted SNP model of length equal to the actual data may be computed using the generic `simulate` method function

```
> ar3.sim = simulate(ar3.fit)
```

Figure 23.2 shows the simulated data along with the SACF and SPACF. These plots verify that the auxiliary model adequately mimics the MA(1) model.

EMM Algorithm

For a fixed $\boldsymbol{\rho}$, let $\{\hat{y}_t(\boldsymbol{\rho})\}_{t=1}^N$ denote a simulated sample of size $N >> n$ from the MA(1) model. The length, N, of the simulation should, in general, be much larger than the sample size, n, of the observed data. The sample score of the auxiliary model evaluated using the simulated data is

$$\mathbf{m}(\boldsymbol{\rho}, \tilde{\boldsymbol{\theta}}_n) = \frac{1}{N} \sum_{t=1}^{N} \frac{\partial}{\partial \boldsymbol{\theta}} \ln f(\hat{y}_t(\boldsymbol{\rho}) | \hat{\mathbf{x}}_{t-1}, \tilde{\boldsymbol{\theta}}_n)$$

which, for the AR(p) auxiliary model, has elements

$$\begin{aligned}
&\frac{1}{N}\sum_{t=1}^{N}(\hat{y}_t(\boldsymbol{\rho}) - \tilde{c} - \tilde{\phi}_1\hat{y}_{t-1}(\boldsymbol{\rho}) - \cdots - \tilde{\phi}_p\hat{y}_{t-p}(\boldsymbol{\rho}))\\
&\frac{1}{N}\sum_{t=1}^{N}(\hat{y}_t(\boldsymbol{\rho}) - \tilde{c} - \tilde{\phi}_1\hat{y}_{t-1}(\boldsymbol{\rho}) - \cdots - \tilde{\phi}_p\hat{y}_{t-p}(\boldsymbol{\rho}))\hat{y}_{t-1}(\boldsymbol{\rho})\\
&\vdots\\
&\frac{1}{N}\sum_{t=1}^{N}(\hat{y}_t(\boldsymbol{\rho}) - \tilde{c} - \tilde{\phi}_1\hat{y}_{t-1}(\boldsymbol{\rho}) - \cdots - \tilde{\phi}_p\hat{y}_{t-p}(\boldsymbol{\rho}))\hat{y}_{t-p}(\boldsymbol{\rho})\\
&-\frac{1}{2\tilde{\sigma}^2}[N + \tilde{\sigma}^{-2}]\sum_{t=1}^{N}(\hat{y}_t(\boldsymbol{\rho}) - \tilde{c} - \tilde{\phi}_1\hat{y}_{t-1}(\boldsymbol{\rho}) - \cdots - \tilde{\phi}_p\hat{y}_{t-p}(\boldsymbol{\rho}))
\end{aligned}$$

If $\boldsymbol{\rho} = \boldsymbol{\rho}_0$, then

$$\mathbf{m}(\boldsymbol{\rho}_0, \tilde{\boldsymbol{\theta}}_n) \approx \mathbf{0}$$

whereas if $\boldsymbol{\rho} \neq \boldsymbol{\rho}_0$, then

$$\mathbf{m}(\boldsymbol{\rho}, \tilde{\boldsymbol{\theta}}_n) \neq \mathbf{0}$$

The EMM algorithm attempts to find the value $\hat{\boldsymbol{\rho}}_n$ such that $\mathbf{m}(\hat{\boldsymbol{\rho}}_n, \tilde{\boldsymbol{\theta}}_n)$ is as close to zero as possible. Since $\boldsymbol{\theta}$ is $(p+2)\times 1$ and $\boldsymbol{\rho}$ is (3×1), it is not possible to find $\hat{\boldsymbol{\rho}}_n$ such that $\mathbf{m}(\hat{\boldsymbol{\rho}}_n, \tilde{\boldsymbol{\theta}}_n) = \mathbf{0}$. Instead, $\boldsymbol{\rho}$ is estimated by minimizing the GMM-type objective function:

$$\hat{\boldsymbol{\rho}}_n = \arg\min_{\boldsymbol{\rho}}\ \mathbf{m}(\boldsymbol{\rho}, \tilde{\boldsymbol{\theta}}_n)'\tilde{\mathbf{S}}^{-1}\mathbf{m}(\boldsymbol{\rho}, \tilde{\boldsymbol{\theta}}_n)$$

where $\tilde{\mathbf{S}}$ is a consistent estimate of avar$(\sqrt{n}\mathbf{m}(\boldsymbol{\rho}_0, \tilde{\boldsymbol{\theta}}_n))$.

The Simulator Function

An example of a simulation function for the MA(1) model in a form for use with the function `EMM` is

```
MA1.gensim = function(rho, n.sim, n.var=1, n.burn,
                      aux=MA1.aux())
{
  # rho = (mu,psi,sigma2)'
  # aux is a list with components
  # z = vector of N(0,1) values of length (n.sim + n.burn)
  nz = n.burn + n.sim
  if(is.null(z <- aux$z))
     stop("aux$z must be supplied")
  if(length(z) != nz)
     stop("aux$z must be of length",nz," if provided")
  ans = rho[1]+arima.sim(model = list(ma=-rho[2]),
         innov = z[(n.burn+1):nz]*sqrt(rho[3]),
         start.innov = z[1:n.burn]*sqrt(rho[3]))
  as.numeric(ans)
}
```

In the body of MA1.gensim, the S-PLUS function arima.sim is called to efficiently simulate from the MA(1) model. The function is specified such that the random numbers used to generate the simulation are computed in advance and passed to the simulator through the optional aux list component. This allows MA1.gensim to use the same random numbers to generate simulations for different values of rho.

EMM Estimation

Once the auxiliary model has been fit using the SNP function and the user has written the simulator function, the structural model (MA(1) model) may be estimated by EMM using the EMM function. For example, commands to estimate the MA(1) model using EMM with $N = 10,000$ simulations and 10 burn-in values are

```
> set.seed(345)
> nsim = 10000
> nburn = 10
> MA1.aux = list(z = rnorm(nsim+nburn))
> start.vals = c(0,0.5,1)
> names(start.vals) = c("mu","theta","sigma2")
> EMM.MA1.fit = EMM(ar3.fit,coef = start.vals,
+         control = EMM.control(n.sim = nsim, n.burn = nburn),
+         gensim.fn = "MA1.gensim", gensim.language = "SPLUS",
+         gensim.aux = MA1.aux, save.sim = T)
Iteration No. 1, objective = 114.452
...
Iteration No. 12, objective = 0.0963164
> class(EMM.MA1.fit)
[1] "EMM"
```

In the call to EMM, the simulator function is specified by the arguments gensim.fn="MA1.gensim" and gensim.language="SPLUS". The control argument sets the number of burn-in and simulation values. The random numbers, which are fixed across iterations in the optimization, are contained in the user-created list variable MA1.aux and are passed to the simulator within EMM using gensim.aux="MA1.gensim". The dimension of the vectors in MA1.aux must match the dimension n.burn+n.sim determined by the corresponding arguments passed to the function EMM.control. By setting save.sim=T, the simulated observations evaluated at the estimated value of $\boldsymbol{\rho}$, $\hat{y}_t(\hat{\boldsymbol{\rho}})$, are added as the component `smdat` of the returned "`EMM`" object.

The function `EMM` returns an object of class "`EMM`" for which there are `print`, `summary`, and `plot` methods. Typing the name of an "`EMM`" object invokes the `print` method

```
> EMM.MA1.fit
Call:
EMM(score = ar3.fit, coef = start.vals, control = EMM.control(
n.sim = nsim, n.burn = nburn), save.sim = T, gensim.fn
= "MA1.gensim", gensim.language = "SPLUS", gensim.aux
= MA1.aux)

Coefficients:
        Value Std. Error      95% Conf. Int.
    mu 0.0985    0.09960 -0.09606     0.2955
   psi 0.5267    0.06473  0.41302     0.6689
sigma2 0.9024    0.07060  0.76378     1.0403

Final optimization:
  Convergence: relative function convergence
  Iterations: 12
  Score at final iteration: 0.09632
```

The estimated coefficients are close to their true values, but have somewhat large standard errors. The 95% confidence intervals are based on inverting the $\mathrm{LR}_{\mathrm{EMM}}$ statistic for testing $\rho_i = \rho_i^o$. To compute the usual Wald-type 95% confidence intervals, $\hat{\rho}_i \pm 1.96 \cdot \widehat{\mathrm{SE}}(\hat{\rho}_i)$, set the optional argument `cf.interval=F` in `EMM.control`. The final value of the score is the normalized EMM objective function, and its small value indicates that the overidentifying conditions are not rejected by the data.

The `summary` method provides some useful diagnostics for assessing the EMM fit:

```
> summary(EMM.MA1.fit)
Call:
EMM(score = ar3.fit, coef = start.vals, control = EMM.control(
n.sim = nsim, n.burn = nburn), save.sim = T, gensim.fn
= "MA1.gensim", gensim.language = "SPLUS", gensim.aux
= MA1.aux)

Coefficients:
        Value Std. Error      95% Conf. Int.
    mu 0.0985    0.09960 -0.09606     0.2955
   psi 0.5267    0.06473  0.41302     0.6689
sigma2 0.9024    0.07060  0.76378     1.0403
```

```
Final optimization:
  Convergence: relative function convergence
  Iterations: 12
  EMM objective at final iteration: 0.09632
  P-Value:  0.953  on  2  degrees of freedom

Score Information:
      Mean Score Std. Error  t-ratio Adj. Std. Error t-ratio
   mu   0.005372      1.133 0.004743         0.09656 0.05564
ar(3)   0.235363      1.174 0.200504         1.16387 0.20222
ar(2)   0.354856      1.209 0.293554         1.14360 0.31030
ar(1)   0.219727      1.105 0.198858         0.84555 0.25986
sigma   0.093842      1.427 0.065739         0.36389 0.25789
```

First, the p-value for the EMM objective function based on the limiting chi-square distribution is given. The degrees of freedom for the chi-square distribution is equal to the number of parameters in the auxiliary SNP model minus the number of parameters in the structural model: $5 - 3 = 2$. Here, the very large p-value of 0.9558 supports the validity of the overidentifying conditions. Second, the individual normalized score values (23.20) along with their (unadjusted and adjusted) asymptotic standard errors and t-statistics (23.21) are given.[6] In a correctly specified model, the EMM objective function should not be significantly large and the absolute value of the score t-ratios should be less than 2. If the model is misspecified in some way and the auxiliary model adequately describes the observed data, then the EMM objective should be significantly large, and one or more of the score t-ratios should be large reflecting a characteristic of the auxiliary model that is not being captured by the structural model. For the MA(1) model, the EMM objective function is not significantly large and all of the score t-ratios are less than 2 indicating a well specified-model.

The plot method gives a barchart of the unadjusted or adjusted score t-ratios. For example, to display the adjusted score t-ratios, use

```
> plot(EMM.MA1.fit)
```

The resulting plot is shown in Figure 23.3.

The simulated observations at the estimated value of $\boldsymbol{\rho}$, $\hat{y}_t(\hat{\boldsymbol{\rho}})$, are in the component `smdat`. These values should mimic the properties of the observed data if the estimated structural model is correctly specified.

[6]The t-ratios (23.21) are called the adjusted t-ratios. The unadjusted t-ratios use $\tilde{\mathbf{S}}$ instead of $\mathbf{S}_n$ in (23.21). The unadjusted t-ratios are easier to compute but are biased downward relative to 2.0.

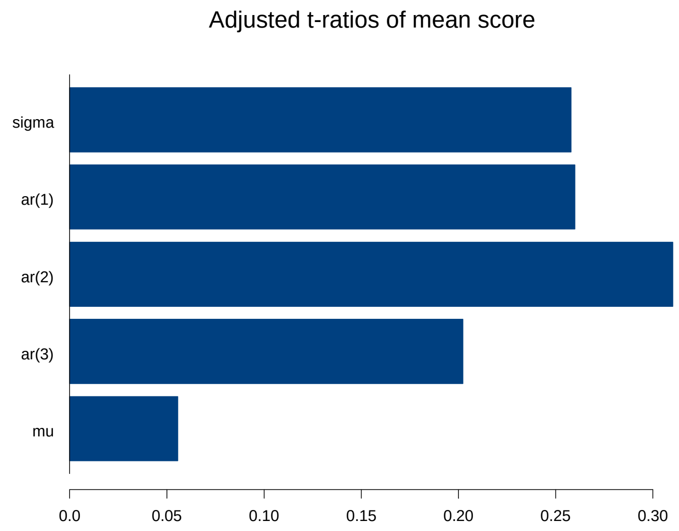


FIGURE 23.3. Adjusted score t-ratios from the EMM estimation of an MA(1) model using a Gaussian AR(3) SNP model.

GMM and ML Estimation

The GMM estimation of the MA(1) model (23.23) using the moments

$$\begin{aligned} E[y_t] &= \mu_0 \\ E[y_t^2] &= \mu_0^2 + \sigma_0^2(1+\psi_0^2) \\ E[y_t y_{t-1}] &= \mu_0^2 + \sigma_0^2\psi_0 \\ E[y_t y_{t-2}] &= \mu_0^2 \end{aligned}$$

is discussed in Chapter 21. The GMM estimates of $\boldsymbol{\rho}$ for the simulated data in `ma1.sim`, based on an HAC estimate of the asymptotic variance of the moments with a truncated kernel, are repeated here:

```
> summary(ma1.gmm.trunc)

Call:
GMM(start = start.vals, moments = ma1.moments, ts = T,
var.hac.control = var.hac.control(bandwidth = 1,
window = "truncated"), data = ma1.data)

Coefficients:
       Value Std.Error t value Pr(>|t|)
  mu  0.1018  0.0927    1.0980  0.2733
```

```
 psi  0.5471  0.0807    6.7788  0.0000
sig2  0.8671  0.0763   11.3581  0.0000

Test of Overidentification:
 J-stat Df P.value
 0.3549 1  0.5513

Optimization Info:
Number of Iterative Steps: 3
```

The GMM estimates are close to the EMM estimates. The standard errors for the EMM estimates of ψ and σ^2 are slightly smaller than the standard errors of the corresponding GMM estimates. This efficiency gain reflects the fact that EMM uses the efficient moments based on the score of the auxiliary model, whereas GMM uses a set of arbitrary moments. EMM also benefits from not having to use an HAC estimator for the asymptotic variance of the moments.

The conditional maximum likelihood estimation of the MA(1) model (23.23) may be performed using the S-PLUS function arima.mle as follows:

```
> mle.fit = arima.mle(ma1.sim,model=list(ma=-0.5),
+                     xreg=rep(1, length(ma1.sim)))
> mle.fit
Call: arima.mle(x = ma1.sim, model = list(ma = -0.5),
xreg = rep(1, length(ma1.sim)))
Method:  Maximum Likelihood
Model :  0 0 1

Coefficients:
MA : -0.52262

Variance-Covariance Matrix:
            ma(1)
ma(1) 0.002907486
Coeffficients for regressor(s): rep(1, length(ma1.sim))
[1] 0.11416

Optimizer has  converged
Convergence Type: relative function convergence
AIC: 683.51018
> mle.fit$sigma2
[1] 0.8859344
```

The estimates of μ, ψ, and σ^2 are 0.5226, 0.1142, and 0.8859, respectively. These estimates are close to the GMM and EMM estimates. The S-PLUS function arima.mle only computes an estimate of the coefficient covariance matrix for the autoregressive moving average (ARMA) parameters. The estimated standard error for ψ is

```
> sqrt(mle.fit$var.coef)
           ma(1)
ma(1) 0.05392111
```

which is slightly smaller than the EMM standard error. Here we see that the EMM estimates are comparable to the ML estimates. As the sample size increases and the number of AR terms in the SNP model increases, the difference between the EMM and ML estimates will approach zero.

23.4.2 Discrete-Time Stochastic Volatility Models

This subsection describes the EMM estimation of some discrete-time stochastic volatility (SV) models used for modeling the daily continuously compounded returns on the S&P 500 index. The first model is an elementary and empirically unrealistic SV model used by Gallant and Tauchen (2002), hereafter GT, to illustrate some of the workings and properties of EMM estimation and inference. The second model is a more realistic SV model with AR(1) dynamics considered by Andersen and Sorensen (1996) and Andersen, Chung, and Sorensen (1999). The third model is a general SV model with AR(p) dynamics that allows for leverage effects considered by Gallant, Hsieh, and Tauchen (1997). The analysis of the SV models in this section is in the spirit of the analysis done in Gallant, Hsieh, and Tauchen (1997) and GT.

Data

The data consist of daily continuously compounded returns, based on closing prices, on the S&P 500 index over the period March 14, 1986 through June 30, 2003. These data were extensively analyzed in the previous chapter. Similar data were analyzed in GT. Figure 22.18 of Chapter 22 shows the data along with the sample ACF of the squared returns. Sample summary statistics are

```
> summaryStats(sp500.ts)

Sample Quantiles:
     min        1Q    median       3Q     max
 -0.2047 -0.004686 0.0004678 0.005827 0.09099

Sample Moments:
      mean     std skewness kurtosis
 0.0003916 0.01124   -1.486    32.59

Number of Observations:  4365
```

The S&P 500 returns appear to be highly non-normal, with negative skewness and very large kurtosis, and display considerable conditional heteroskedasticity.

SNP Auxiliary Model Fit

The details of finding the best fitting SNP model for the S&P 500 returns are described in the previous chapter. There it was found that a non-Gaussian AR(1)-GARCH(1,1) model, 11118000, adequately describes the data. The auxiliary model information is contained in the "`SNP`" object `fit.sp500.11118000`, which is provided with the `S+FinMetrics` module. For illustrative purposes, the EMM estimation using the simplistic non-Gaussian AR(1) with $K_z = 4$ will also be considered. This fit is available in the "`SNP`" object `fit.sp500.10014000`.

Gallant and Tauchen (2002)

To illustrate the workings of EMM, GT considered the very simple discrete-time SV model

$$\begin{aligned} y_t - \mu &= c(y_{t-1} - \mu) + \sigma \exp(w_t) z_t \\ w_t - \mu_w &= \sigma_w z_{wt} \end{aligned} \tag{23.24}$$

where μ, c, σ, μ_w, and σ_w are parameters and $\{z_t\}$ and $\{z_{wt}\}$ are mutually independent iid Gaussian processes with mean zero and unit variance. GT impose the normalization $\mu_w = -\sigma_w^2$, which implies that $\text{var}(y_t|y_{t-1}) = \sigma^2$. The model allows for serially correlated returns, but no serial correlation in the unobserved log volatility w_t. This model is too simplistic for actual data. Nonetheless, its estimation by EMM provides important insights to the proper use of the EMM methodology.

Simulator Function

An `S-PLUS` function to efficiently simulate from the SV model (23.24) that can be used in conjunction with the function `EMM` is

```
sv.gt.gensim = function(rho,n.sim,n.var=1,
                        n.burn,aux=sv.gt.aux)
{
# rho = (mu,c,sigma,sigma.w)
# aux is a list with components
# z = vector of N(0,1) values of length (n.sim + n.burn)
# zw = vector N(0,1) values of length (n.sim + n.burn)
   nz = n.burn + n.sim
   sv = rho[3]*exp(-rho[4]*rho[4] + rho[4]*aux$zw)*aux$z
   y = arima.sim(model = list(ar=rho[2]),
            innov = sv[(n.burn+1):nz],
```

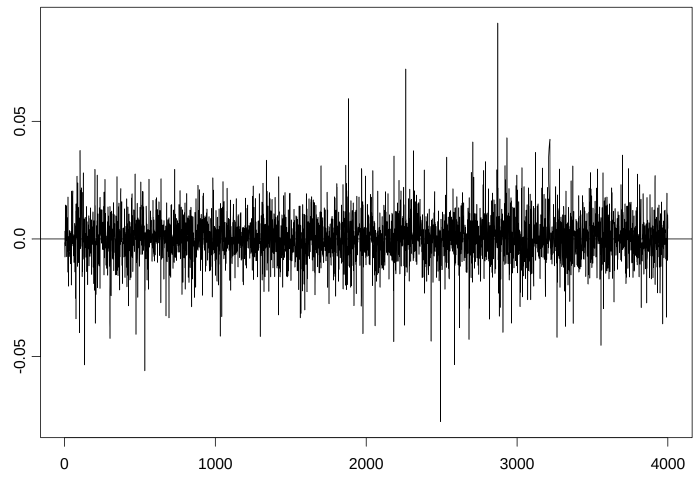


FIGURE 23.4. Simulated data from a discrete-time SV model.

```
            start.innov = sv[1:n.burn])
   y = y + rho[1]
   as.numeric(y)
}
```

Notice that the S-PLUS function `arima.sim` is used to efficiently simulate from an AR(1) process. Also notice that the function `sv.gt.gensim` is set up so that the user must pass two vectors of random numbers as components in the list `sv.gt.aux`.

To generate a simulated sample with $N = 4000$, with 100 burn-in values, calibrated to match the S&P 500 returns, use

```
> n.sim = 4000
> n.burn = 100
> nz = n.sim + n.burn
> set.seed(123)
> sv.gt.aux = list(z=rnorm(nz), zw=rnorm(nz))
> rho.gt = c(0.0004, 0.01, 0.01, 0.5)
> sv.gt.sim = sv.gt.gensim(rho = rho.gt, n.sim = n.sim,
+                          n.burn = n.burn, aux = sv.gt.aux)
```

The simulated sample is illustrated in Figure 23.4. The simple SV model (23.24) cannot produce the large outliers or volatility clustering observed in the actual data.

EMM Estimation

As in GT, first consider the EMM estimation of the simple SV model (23.24) for the S&P 500 returns using the poorly fitting auxiliary model represented by the "`SNP`" object `fit.sp500.10014000`. The EMM estimation with $N = 10,000$ simulations and 100 burn-in values may be performed using

```
> n.sim = 10000
> n.burn = 100
> nz = n.sim + n.burn
> set.seed(456)
> sv.gt.aux = list(z=rnorm(nz), zw=rnorm(nz))
> rho.gt = c(0.0004, 0.01, 0.01, 0.5)
> names(rho.gt) = c("mu","c","sigma","sigma.w")
> emm.svgt.fit1 = EMM(fit.sp500.10014000, coef = rho.gt,
+                     EMM.control(n.burn=n.burn, n.sim=n.sim),
+                     gensim.fn = "sv.gt.gensim",
+                     gensim.language = "SPLUS",
+                     gensim.aux = sv.gt.aux)
Iteration No. 1, objective = 2.0701e+010
...
Iteration No. 18, objective = 4.66991
```

The EMM estimation converges, and the small objective value indicates that the data, as summarized through the 10014000 auxiliary model, are consistent with the structural model (23.24). A summary of the EMM fit is

```
> summary(emm.svgt.fit1)
Call:
EMM(score = fit.sp500.10014000, coef = rho.gt, appcode =
EMM.control(n.burn = n.burn, n.sim = n.sim), gensim.fn
= "sv.gt.gensim", gensim.language = "SPLUS",
gensim.aux = sv.gt.aux)

Coefficients:
            Value Std. Error       95% Conf. Int.
     mu 0.0006102  0.0001488  0.000318  0.0009016
      c 0.0111869  0.0157008 -0.019764  0.0417771
  sigma 0.0104773  0.0002044  0.010078  0.0108867
sigma.w 0.4411473  0.0261296  0.388516  0.4929767

Final optimization:
  Convergence: relative function convergence
  Iterations: 18
  EMM objective at final iteration: 4.67
  P-Value:  0.1976  on  3  degrees of freedom
```

```
Score Information:
      Mean Score Std. Error  t-ratio Adj. Std. Error t-ratio
  z^1    -0.3076      2.436 -0.12631          1.4234 -0.2161
  z^2    -0.2058      3.953 -0.05206          0.5299 -0.3884
  z^3     3.0758      9.393  0.32745          8.5520  0.3597
  z^4     3.6977     22.354  0.16541          3.1628  1.1691
   mu    -0.3931      1.118 -0.35160          0.2239 -1.7557
ar(1)     0.9893      1.262  0.78408          0.6526  1.5160
sigma     4.4093      3.315  1.33009          3.0670  1.4377
```

The autoregressive coefficient is not statistically different from zero, but the variance of the stochastic volatility term is highly significant. The p-value of 0.1976 on the EMM objective function indicates that the overidentifying moment conditions from the 10014000 scores are not rejected by the data. As a result, the absolute value of the t-ratios on the individual score elements are all less than 2. The results indicate that the simple SV model fits the data, as summarized through the 10014000 auxiliary model.

Next, consider fitting the simple SV model (23.24) using the best fitting SNP 11118000 auxiliary model with $N = 10,000$ simulations and 100 burn-in values:

```
> emm.svgt.fit = EMM(fit.sp500.11118000, coef=rho.gt,
+                    EMM.control(n.burn=n.burn, n.sim=n.sim),
+                    gensim.fn = "sv.gt.gensim",
+                    gensim.language = "SPLUS",
+                    gensim.aux = sv.gt.aux)
Iteration No. 1, objective = 3.5508e+008
...
Iteration No. 44, objective = 156.49
```

The EMM estimation converges with a high objective value, indicating that the data now reject the structural model (23.24). This is confirmed by examining the EMM fit:

```
> summary(emm.svgt.fit)
Call:
EMM(score = fit.sp500.11118000, coef = rho.gt,
appcode = EMM.control(n.burn = n.burn, n.sim = n.sim),
 gensim.fn = "sv.gt.gensim",
 gensim.language = "SPLUS", gensim.aux = sv.gt.aux)

Coefficients:
            Value Std. Error       95% Conf. Int.
     mu 0.0005499 0.00009724 0.0003601  0.0006583
      c 0.0267012 0.01025450 0.0052093  0.0480410
  sigma 0.0128051 0.00059129 0.0120651  0.0135576
```

```
sigma.w 0.4470297 0.00786893 0.4392415  0.4575820

Final optimization:
  Convergence: relative function convergence
  Iterations: 44
  EMM objective at final iteration: 156.5
  P-Value:  0  on  9  degrees of freedom

Score Information:
         Mean Score Std. Error t-ratio Adj. Std. Error t-ratio
     z^1    -4.5158     2.4735 -1.8257          2.1519 -2.0986
     z^2   -27.4201     3.7849 -7.2446          3.3053 -8.2958
     z^3    -4.7150     8.6945 -0.5423          8.1304 -0.5799
     z^4  -157.5280    21.7408 -7.2457         20.0389 -7.8611
     z^5   124.1809    65.0210  1.9099         60.8025  2.0424
     z^6  -934.3530   213.5628 -4.3751        200.8732 -4.6515
     z^7  2537.7699   760.7266  3.3360        726.0131  3.4955
     z^8 -6559.4648  2856.7967 -2.2961       2758.2166 -2.3782
      mu    -2.4675     1.4805 -1.6666          0.8574 -2.8779
   ar(1)     0.5342     0.9803  0.5449          0.3660  1.4595
      s0   -42.1830    14.6437 -2.8806         13.9578 -3.0222
 arch(1)   -13.6875     8.0812 -1.6938          6.6419 -2.0608
garch(1)   -42.1351    15.6225 -2.6971         12.7673 -3.3003
```

The zero p-value on the EMM objective function, based on the chi-square distribution with 9 degrees of freedom, leads to a rejection of the overidentifying moment conditions. The large t-ratios on the individual score elements associated with the Hermite polynomial elements and the GARCH elements shows that the simple SV model cannot capture the fat tails and volatility clustering that exists in the data, as summarized by the 11118000 auxiliary model. This example highlights the importance of using an auxiliary model that captures all of the important features of the data when estimating and evaluating a structural model by EMM.

Andersen, Chung, and Sorensen (1999)

The SV model considered by Andersen and Sorensen (1997), Chumacero (1997), and Andersen, Chung, and Sorensen (1999) is of the form

$$\begin{aligned} y_t &= \sigma_t z_t \\ \ln \sigma_t^2 &= \omega + \beta \ln \sigma_{t-1}^2 + \sigma_u u_t \end{aligned} \tag{23.25}$$

where (z_t, u_t) is iid $N(\mathbf{0}, \mathbf{I}_2)$. This model was estimated by GMM using 24 moments in Chapter 21. The return series y_t is assumed to be demeaned. For $-1 < \beta < 1$ and $\sigma_u > 0$, the return series, y_t, is strictly stationary and ergodic, and unconditional moments of any order exist. Let $w_t = \ln \sigma_t^2$ so that $\sigma_t = \exp(w_t/2)$. Then, (23.25) may be rewritten as

$$\begin{aligned} y_t &= \exp(w_t/2) \cdot z_t \\ w_t &= \omega + \beta w_{t-1} + \sigma_u u_t \end{aligned}$$

which is in a form similar to the model (23.24). The SV model (23.25) allows log-volatility to be serially correlated, which produces autoregressive conditional heteroskedasticity (ARCH)-like features in the data. See Shephard (1996) and Ghysels, Harvey, and Renault (1996) for a detailed comparison of ARCH and SV models.

Simulator Function

An S-PLUS function to generate simulations from the SV model (23.25) is

```
sv.as.gensim = function(rho, n.sim, n.var=1, n.burn,
                         aux = sv.as.aux)
{
# rho = (alpha, beta.t, sigma.u)
# aux is a list with components
# z = vector of N(0,1) values of length (n.sim + n.burn)
# u = vector N(0,1) values of length (n.sim + n.burn)
   nz = n.burn + n.sim
   beta = exp(rho[2])/(1 + exp(rho[2]))
   mu = rho[1]/(1-beta)
   w = mu + arima.sim(model = list(ar=beta),
            innov = aux$u[(n.burn + 1):nz]*rho[3],
            start.innov = aux$u[1:n.burn]*rho[3])
   y = exp(w/2)*aux$z[(n.burn + 1):nz]
   as.numeric(y)
}
```

To impose stationarity on the log-volatility process, the logistic transformation is used to define the autoregressive parameter β from the unrestricted parameter β^*:

$$\beta = \frac{\exp(\beta^*)}{1+\exp(\beta^*)}, \quad -\infty < \beta^* < \infty$$

The logistic transformation restricts β to the interval $(0, 1)$. This restriction is reasonable since negative values of β are not empirically relevant for asset returns. The unconditional mean of the log-volatility process is $\mu = \omega/(1-\beta)$.

Figure 23.5 shows a simulated sample of size $N = 4000$ from (23.25) using the parameterization $\alpha = -0.147, \beta = 0.98$ and $\sigma_u = 0.166$ taken from Andersen, Chung, and Sorensen (1999) (Monte Carlo design II). The parameters are calibrated to match typical daily return data. The simulation is generated using

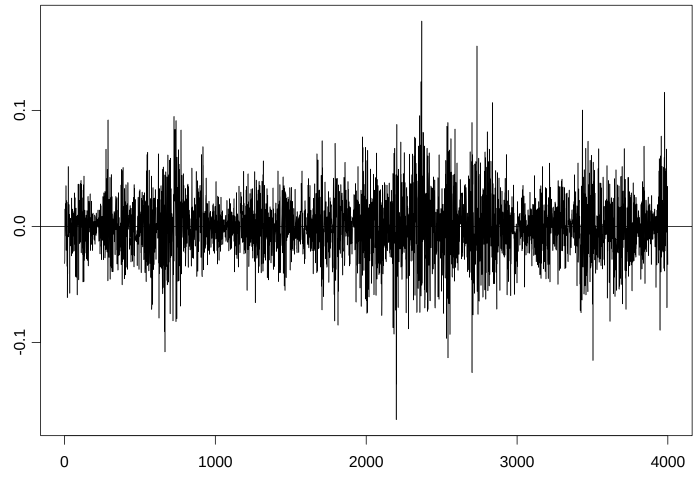


FIGURE 23.5. Simulated data from the Andersen-Sorensen SV model with $\alpha = 0.736, \beta = 0.90$, and $\sigma_u = 0.363$.

```
> n.sim = 4000
> n.burn = 100
> nz = n.sim + n.burn
> set.seed(456)
> sv.as2.aux = list(z=rnorm(nz), u=rnorm(nz))
> rho.as2 = c(-0.147, 3.89182, 0.166)
> sv.as2.sim = sv.as.gensim(rho = rho.as, n.sim = n.sim,
+                              n.burn = n.burn, aux = sv.as.aux)
```

The autocorrelated log-volatility process generates ARCH-like volatility clustering in the simulated data.

EMM Estimation on Simulated Data

Andersen, Chung, and Sorensen (1999) studied the EMM estimation of (23.25) using an extensive Monte Carlo study. They found that EMM performs substantially better than GMM and comparably to direct likelihood-based inference procedures. For samples of size 1000 or less they found that a simple Gaussian GARCH(1,1) SNP model was a good choice for a score generator. Only for much larger samples did they find that adding Hermite polynomial terms to the SNP model improved efficiency. Regarding inference, they found that the EMM objective function test for overidentifying restriction was remarkably reliable in contrast to the analogous GMM objective function test (Hansen's J-test).

The estimation of (23.25) using the simulated data in `sv.as2.sim` is based on the following Gaussian GARCH(1,1) SNP model:

```
> fOld = c(0,1e-5, 0, -1e-5, 1e-5, 1e-5, 1e-4, 0, -1e-4,
          1e-4, 1e-4, 1e-3, 0, -1e-3, 1e-3, 1e-3, 1e-2,
          0, -1e-2, 1e-2, 1e-2, 1e-1, 0, -1e-1, 1e-1,
          1e-1, 1e0, 0, -1e0, 1e0, 1e0)
> fNew = c(0,0, 1e-5, 1e-5, -1e-5, 1e-5, 0, 1e-4, 1e-4,
          -1e-4, 1e-4, 0, 1e-3, 1e-3, -1e-3, 1e-3, 0,
          1e-2, 1e-2, -1e-2, 1e-2, 0, 1e-1, 1e-1, -1e-1,
          1e-1, 0, 1e0, 1e0, -1e0, 1e0)
> n.start = c(0,rep(25,30))
> fit.svas2sim.01110000 = SNP(as.matrix(sv.as2.sim),
+     model = SNP.model(ar=0, arch=1, garch=1),
+     control = SNP.control(xTransform = "logistic",
+                          n.start=n.start, fOld=fOld,
+                          fNew=fNew),
+     n.drop=4, trace=T)
```

The quasi-maximum-likelihood estimation of the SNP model utilizes random restarts of the optimizer to avoid getting stuck at potential local minima, as recommended by Gallant and Tauchen (2001).

The EMM estimation with $N = 40,000$ simulations and 100 burn-in values is performed using[7]

```
> n.sim = 40000
> n.burn = 100
> nz = n.sim + n.burn
> set.seed(123)
> sv.as2.aux = list(z=rnorm(nz), u=rnorm(nz))
> emm.svas2sim.fit = EMM(fit.svas2sim.01110000, coef=rho.as2,
+                    EMM.control(n.burn=n.burn, n.sim=n.sim),
+                    gensim.fn="sv.as.gensim",
+                    gensim.language="SPLUS",
+                    gensim.aux=sv.as2.aux)
Iteration No. 1, objective = 85170
...
Iteration No. 30, objective = 0.168816

> summary(emm.svas2sim.fit)
```

[7]These are values similar to those used by Andersen and Sorensen (1996). They utilized $N = 20,000$ simulations with $20,000$ antithetic pairs. The `S-PLUS` function `EMM` currently does not support antithetic variates.

```
Call:
EMM(score = fit.svas2sim.01110000, coef = rho.as2,
appcode = EMM.control(n.burn = n.burn, n.sim = n.sim),
 gensim.fn = "sv.as.gensim",
 gensim.language = "SPLUS", gensim.aux = sv.as2.aux)

Coefficients:
          Value Std. Error     95% Conf. Int.
  alpha -0.2631    0.13018 -0.3839    -0.1330
 beta.t  3.3228    0.50730  2.8523     3.8301
sigma.u  0.1950    0.04636  0.1486     0.2378

Final optimization:
  Convergence: relative function convergence
  Iterations: 30
  EMM objective at final iteration: 0.1688
  P-Value:  0.6812  on  1  degrees of freedom

Score Information:
        Mean Score Std. Error  t-ratio Adj. Std. Error t-ratio
      mu    0.8230      2.004 0.410711          2.0029  0.4109
      s0    2.4909     16.076 0.154946          6.0634  0.4108
 arch(1)    0.3551      8.196 0.043326          0.8647  0.4107
garch(1)    0.1139     11.928 0.009551          0.2777  0.4102
```

The EMM estimation converges with a high objective function p-value indicating that the data do not reject the single overidentifying restriction implied by the GARCH(1,1) score generator. The estimates of ω and σ_u are reasonably close to their true values, and the sizes of their estimated standard errors are similar to the Monte Carlo root mean square errors reported by Anderson, Chung, and Sorensen. The estimate of β^* is 3.3228, which implies an estimate for β equal to 0.9652028. The standard error for β, obtained using the delta method approximation, is

```
> bt.hat = emm.svas2sim.fit$coef[2]
> b.hat = exp(bt.hat)/(1 + exp(bt.hat))
> dg = b.hat*(1 - b.hat)
> se.b.hat = sqrt(dg%*%emm.svas2sim.fit$var[2,2]%*%dg)
> se.b.hat
           [,1]
[1,] 0.01703867
```

Notice that all of the adjusted score t-ratios are the same. This occurs because the asymptotic variance of the normalized moments has rank 1 (degree of overidentification).

The GMM estimation of (23.25) based on 24 arbitrary moment conditions is described in Chapter 21. The GMM estimates based on the simulated data in `sv.as2.sim` are computed using

```
> sv.pow = cbind(abs(sv.as2.sim), sv.as2.sim^2,
+               abs(sv.as2.sim)^3, sv.as2.sim^4)
> sv2.pow = sv.pow[-(1:10),]
> sv2.cm = tslag(sv.as2.sim, 1:10, trim=T) *
+         as.vector(sv.as2.sim[-(1:10)])
> sv2.data = cbind(sv2.pow, abs(sv2.cm), sv2.cm^2)
> start.vals = c(0, 0.5, 0.5)
> names(start.vals) = c("omega", "beta", "sigu")
> sv2.fit.gmm = GMM(start.vals, sv.moments, method="iterative",
+                  ts=T, data=sv2.data)

> summary(sv2.fit.gmm,print.moments=F)

Call:
GMM(start = start.vals, moments = sv.moments, method =
"iterative", ts = T, data = sv2.data)

Coefficients:
         Value Std.Error  t value Pr(>|t|)
omega  -0.2650   0.1348   -1.9655   0.0494
 beta   0.9649   0.0179   54.0224   0.0000
 sigu   0.1926   0.0532    3.6214   0.0003

Test of Overidentification:
 J-stat Df P.value
 15.297 21 0.8078
```

The GMM and EMM estimates are remarkably similar. However, based on extensive Monte Carlo exercises, Chumacero (1997) and Anderson, Chung, and Sorensen (1999) recommend EMM over GMM for the following reasons: (1) EMM estimates are numerically more stable; (2) EMM estimates have smaller root mean squared errors; (3) the problems associated with the choice of weighting matrices in the case of GMM are absent in EMM; (4) the EMM test for overidentifying restrictions is more reliable; (5) inference regarding the parameters based on EMM t-statistics is more reliable.

EMM Estimation on S&P 500 Returns

Consider EMM estimation of (23.25) for the S&P 500 daily returns using the best fitting score generator summarized by the "`SNP`" object `fit.sp500.11118000`. The EMM estimation with $N = 40,000$ simulated values and 100 burn-in values is computed using

```
> n.sim = 40000
> n.burn = 100
> nz = n.sim + n.burn
> set.seed(456)
> sv.as.aux = list(z=rnorm(nz), u=rnorm(nz))
> rho.as = c(-0.147, 3.89182, 0.166)
> names(rho.as) = c("alpha", "beta.t", "sigma.u")
> emm.svas.sp500 = EMM(fit.sp500.11118000,coef=rho.as,
+                 EMM.control(n.burn=n.burn, n.sim=n.sim),
+                 gensim.fn = "sv.as.gensim",
+                 gensim.language = "SPLUS",
+                 gensim.aux = sv.as2.aux)
Iteration No. 1, objective = 255798
...
Iteration No. 24, objective = 71.8841

> emm.svas.sp500
Call:
EMM(score = fit.sp500.11118000, coef = rho.as, appcode =
EMM.control(n.burn = n.burn, n.sim = n.sim), gensim.fn
= "sv.as.gensim", gensim.language = "SPLUS", gensim.aux
= sv.as2.aux)

Coefficients:
          Value Std. Error     95% Conf. Int.
  alpha -0.2945    0.13010 -0.4059    -0.1644
 beta.t  3.4229    0.45495  3.0318     3.8778
sigma.u  0.1759    0.03137  0.1445     0.1921

Final optimization:
  Convergence: relative function convergence
  Iterations: 24
  EMM objective at final iteration: 71.88
  P-Value:  1.916e-011  on  10  degrees of freedom
```

The small p-value on the final EMM objective value indicates that the SV model (23.25) is rejected by the S&P 500 returns as summarized through the SNP 11118000 model. A bar plot of the adjusted t-ratios associated with the EMM fit, produced using

```
> plot(emm.svas.sp500)
```

is shown in Figure 23.6.

The large t-ratios on the `z^4`, `z^5`, and `z^7` moments indicate the features of the returns that the SV model (23.25) cannot adequately describe.

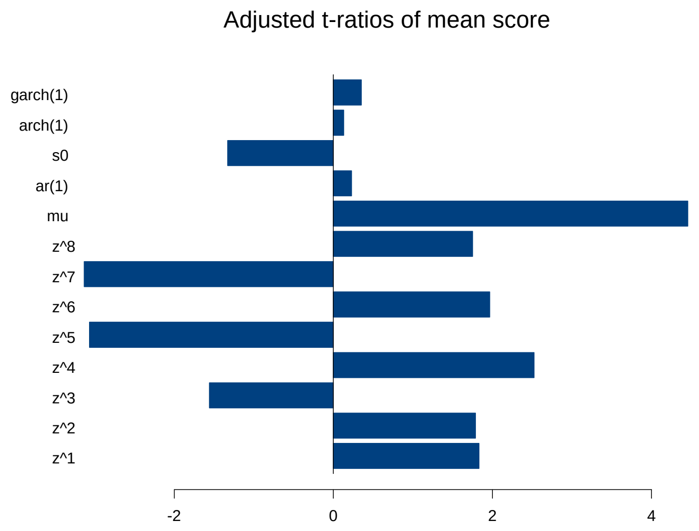


FIGURE 23.6. Adjusted t-ratios from the SNP 11118000 mean score associated with the EMM fit to the SV model (23.25) for the S&P 500 returns.

Gallant, Hsieh, and Tauchen (1997)

Gallant, Hsieh, and Tauchen (1997) considered the general univariate discrete-time SV model

$$\begin{aligned} y_t &= \mu + \sum_{j=1}^{p} \phi_j y_{t-j} + \exp(w_t/2)\sigma_z z_t \\ w_t &= \sum_{j=1}^{q} \beta_j w_{t-j} + \sigma_u u_t \end{aligned}$$

where $\{z_t\}$ and $\{u_t\}$ are iid Gaussian random variables with mean zero, unit variance, and correlation coefficient ρ. The model allows for autoregressive effects in the mean and log-volatility. A negative correlation between the innovations to the level and log-volatility allow for the so-called leverage effect.

23.4.3 Interest Rate Diffusion Models

In this subsection, the one-factor and two-factor interest diffusion models for short-term interest rates introduced in Section 23.2 are fit by EMM. The first example is based on Chan, Karolyi, Longstaff, and Sanders (1992) who fitted a one-factor model to monthly observations on the 3-month U.S.

T-bill. The second and third examples follow Andersen and Lund (1997) who fittd one-factor and two-factor models to weekly observations on the 3-month U.S. T-bill.

CKLS (1992) One-Factor Model

Chan, Karolyi, Longstaff, and Sanders (1992), hereafter CKLS, estimated the generalized Cox-Ingersoll-Ross (GCIR) one-factor continuous-time interest rate diffusion model

$$dr_t = (\alpha_0 + \beta_0 r_t)dt + \sigma_0 r_t^{\gamma_0} dW_t \tag{23.26}$$

using GMM with four moment conditions derived from an Euler discretization. For their estimation, they used $n = 307$ monthly observations on the 3-month U.S. T-bill rate over the period June 1964 through December 1989.[8]

The model (23.26) is of the form (23.22) where $X_t = r_t$ and

$$a(X_t, t) = (\alpha_0 + \beta_0 r_t),\ b(X_t, t) = \sigma_0 r_t^{\gamma_0} \tag{23.27}$$

are the drift and diffusion functions, respectively. In (23.26), the drift function $a(X_t, t) = (\alpha_0 + \beta_0 r_t)dt$ may be reparameterized as $a(X_t, t) = \kappa_0(\mu_0 - r_t)dt$ where $\alpha_0 = \kappa_0\mu_0$ and $\beta_0 = -\kappa_0$. The parameter μ_0 is the long-run mean and the parameter κ_0 determines the speed of mean reversion.

Simulator Function

Simulations of r_t from the GCIR model (23.26) based on Euler's method using the S-PLUS function `euler1d.pcode.gensim`[9] was discussed in Chapter 20. The auxiliary information to set the drift and diffusion functions (23.27) and to calibrate the simulation to monthly data on 3-month T-bills is

```
> rho.names = c("alpha", "beta", "sigma", "gamma")
> ckls.aux = euler.pcode.aux(
+           drift.expr = expression(alpha + beta*X),
+           diffuse.expr = expression(sigma*X^gamma),
+           rho.names = rho.names,
+           t.per.sim = 1/12, ndt = 25,
+           lbound=0, ubound=10)
```

Since the yields are annualized and the data are observed monthly, the argument `t.per.sim` is set to 1/12. Setting `ndt=25` indicates that each

[8]Chapter 21 also gives an example of estimating the GCIR model with monthly data using GMM.

[9]The function `euler1d.pcode.gensim` is more efficient for simulating the solutions to single- factor SDEs than the more general function `euler.pcode.gensim`.

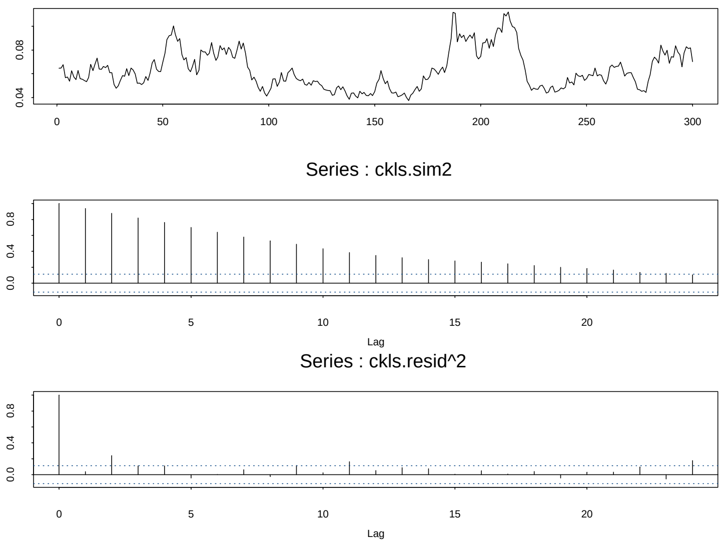


FIGURE 23.7. Simulated annualized short-term interest rates, sampled monthly, from GCIR model.

monthly simulated value is based on 25 discretization points. The arguments `lbound=0` and `ubound=10` truncate the simulation to lie in the interval [0,10]. This prevents nonstationary parameterizations from generating explosive simulations.

CKLS fitted the GCIR model (23.26) to $n = 307$ monthly observations on the 3-month U.S. T-bill rate using GMM, and reported the estimates

$$\hat{\alpha}_0 = 0.0408,\ \hat{\beta}_0 = -0.5921,\ \hat{\sigma}_0^2 = 1.6704,\ \hat{\gamma} = 1.4999 \tag{23.28}$$

The estimates imply a long-run mean interest rate of $0.0408/0.5921 = 0.069$. Figure 23.7 shows a simulation of $n = 300$ annualized interest rates sampled monthly, based on the above GMM estimates, computed using

```
> rho.ckls = c(0.0408, -0.5921, sqrt(1.6704), 1.4999)
> ckls.aux$seed = 123
> ckls.aux$X0 = 0.0408/0.5921
> ckls.sim = euler1d.pcode.gensim(rho.ckls,n.sim=300,
+                                 n.burn=100, aux=ckls.aux)
```

The simulated rates have a long-run mean of about 0.069 and exhibit increased volatility as the level of rates rises. The SACF of the rates shows high persistence in the mean. The SACF of the squared residuals from an AR(1) model fit to the rates shows some evidence of stochastic volatility.

EMM Estimation on Simulated Data

The SNP auxiliary model for the simulated data is determined using the function `SNP.auto`:

```
> ckls.sim = as.matrix(ckls.sim)
> fit.snp.ckls = SNP.auto(ckls.sim,n.drop=8,
+               control = SNP.control(xTransform="spline",
+                                     n.start = n.start,
+                                     seed = 011667,
+                                     fOld = fOld,
+                                     fNew = fNew),
+               xPolyMax=1,lagPMax=1)
```

where the parameters `n.start`, `fOld` and `fNew` are the same as those used in fitting the SNP model for the simulations from the discrete-time SV model discussed earlier. The SNP model that minimizes the BIC is a semi-parameteric AR(1) - GARCH(1,1), 11111000, model:

```
> coef(fit.snp.ckls)

Hermite Polynomial Coefficients:
 z^0     z^1
   1 -0.6622

Conditional Mean Coefficients:
      mu  ar(1)
 -0.3814 0.8125

Conditional Variance Coefficients:
      s0 arch(1) garch(1)
 -0.0798 -0.2568   0.5421
```

A similar model was used by Jensen (2001) in his analysis of monthly interest rate data using EMM. The above SNP model passes the standard residual diagnostics. Figure 23.8 shows simulated values from the fitted SNP model, along with the SACF of the simulated values and the SACF of squared residuals from the SNP fit. The SNP model appears to capture the relevant features of the simulated GCIR data.

Now, consider EMM estimation of the GCIR parameters from the simulated data based on the SNP 11111000 auxiliar model. EMM estimation with $N = 40,000$ simulated values and 100 burn-in values is computed using

```
> n.burn = 100
> n.sim = 40000
> ndt = 25
> set.seed(456)
```

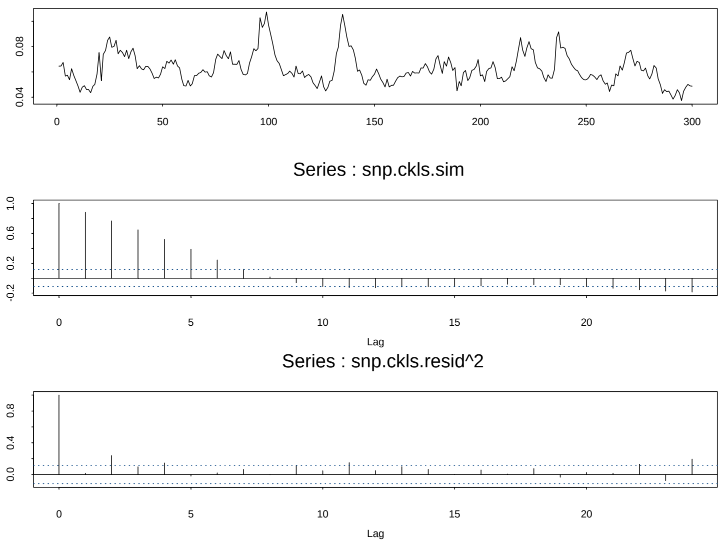


FIGURE 23.8. Simulated data from SNP model fit to simulated values from GCIR model.

```
> z.ckls = rnorm(ndt*(n.burn + n.sim))
> rho.ckls.names = c("alpha", "beta", "sigma", "gamma")
> ckls.aux = euler.pcode.aux(
+           drift.expr = expression(alpha + beta*X),
+           diffuse.expr = expression(sigma*X^gamma),
+           rho.names = rho.ckls.names,
+           t.per.sim = 1/12, ndt = 25,
+           z = z.ckls, X0 = 0.06,
+           lbound=0, ubound=10)

> rho.ckls = c(0.04, -0.6, 1.3 , 1.5)
> emm.ckls.fit = EMM(fit.snp.ckls, coef=rho.ckls,
+                   control = EMM.control(n.sim=n.sim,
+                                         n.burn=n.burn),
+                   gensim.fn = euler1d.pcode.gensim,
+                   gensim.language = "SPLUS",
+                   gensim.aux = ckls.aux, save.sim=T)
Iteration No. 1, objective = 248417
...
Iteration No. 37, objective = 19.9188
> emm.ckls.fit
Call:
```

```
EMM(score = fit.snp.ckls, coef = rho.ckls, control =
EMM.control(n.sim = n.sim, n.burn = n.burn), save.sim
= T, gensim.fn = euler1d.pcode.gensim,
gensim.language = "SPLUS", gensim.aux = ckls.aux)

Coefficients:
         Value Std. Error      95% Conf. Int.
alpha  0.03545   0.004832  0.03416    0.03699
 beta -0.57791   0.083459 -0.61014   -0.55178
sigma  1.25784   0.151522  1.23507    1.29620
gamma  1.49369   0.036179  1.48007    1.50649

Final optimization:
  Convergence: relative function convergence
  Iterations: 37
  EMM objective at final iteration: 19.92
  P-Value:  4.728e-005  on  2  degrees of freedom
```

Interestingly, even though the starting values are set close to the true values, EMM converges with a large objective value. It is possible that the EMM estimates converged to a local minimum. To check this, the GCIR model is re-estimated utilizing the random restart feature of EMM:

```
> rho.ckls = c(0.04, -0.6, 1.3 , 1)
> emm.ckls.fit.restart = EMM(fit.snp.ckls,coef=rho.ckls,
+          control = EMM.control(n.sim = n.sim,
+                                n.burn = n.burn,
+                                init.itmax = 10,
+                                final.itmax = 300,
+                                n.start = rep(5,3),
+                                tweak = c(0.25, 0.5, 1.0)),
+          gensim.fn = euler1d.pcode.gensim,
+          gensim.language = "SPLUS",
+          gensim.aux = ckls.aux, save.sim=T)
Run 1 , 5 starts, tweak = 0.25
Iteration No. 1, objective = 248417
...
Iteration No. 30, objective = 3.38473
```

The random restarts are controled by the arguments `n.start=rep(5,3)` and `tweak=c(0.25, 0.05, 1.0)` to `EMM.control`. The starting values for each coefficient ρ_i are perturbed according to

$$\rho_i \rightarrow \ \rho_i \times (1 + u \times \text{tweak}) \tag{23.29}$$

where u is a uniformly distributed random variable in $(-1, 1)$. The optimization is restarted five times using starting values set by (23.29), with each of the three tweak constants specified in `tweak` and iterated 10 times. The restart that generates the lowest objective function value after 10 iterations is then iterated to convergence. After allowing for random restarts, EMM converges with a much smaller objective function value. The final estimates are

```
> emm.ckls.fit.restart
Call:
EMM(score = fit.snp.ckls, coef = rho.ckls, control =
EMM.control(n.sim = n.sim, n.burn = n.burn, n.start =
rep(5, 3), tweak = c(0.25, 0.5, 1.)), save.sim = T,
gensim.fn = euler1d.pcode.gensim, gensim.language =
"SPLUS", gensim.aux = ckls.aux)

Minimum objective (out of 3 runs) occurred for control
parameters
 tweak n.start
     1       5

Coefficients:
         Value Std. Error      95% Conf. Int.
alpha  0.03246   0.006698  0.02425    0.05684
 beta -0.51219   0.108073 -0.89322   -0.37401
sigma  0.63430   0.170606  0.45965    0.82233
gamma  1.30524   0.099467  1.20165    1.42463

Final optimization:
  Convergence: relative function convergence
  Iterations: 30
  EMM objective at final iteration: 3.385
  P-Value:  0.1841  on  2  degrees of freedom
```

With random restarts of the optimizer, the EMM estimates converge with a much smaller value of the objective function that does not reject the overidentifying restrictions. Interestingly, the EMM estimate of σ is considerably smaller than the true value of 1.292 and it has the largest estimated standard error. Also, the EMM estimate of γ is smaller than its true value of 1.4999. As noted by Tauchen (1997), it is difficult to estimate both σ and γ precisely when their values are large. The small sample sizes afforded by monthly data are likely to exercerbate this problem.

The GMM estimation based on moments derived from the Euler discretization used by CKLS was discussed in Chapter 21. Using the `S+FinMetrics` function `GMM`, the GMM estimates of the GCIR model parameters from the simulated data are

```
> start.vals = c(0.06, -0.5, 1, 1)
> names(start.vals) = c("alpha","beta","sigma","gamma")
> gmm.ckls.sim = GMM(start.vals, ckls.moments, ts=T,
+                    data=data.ckls.sim, dt=1/12)
> summary(gmm.ckls.sim)

Call:
GMM(start = start.vals, moments = ckls.moments, ts = T, data
= data.ckls.sim, dt = 1/12)

Coefficients:
        Value Std.Error t value Pr(>|t|)
alpha  0.0464  0.0157    2.9589  0.0033
 beta -0.7309  0.2701   -2.7064  0.0072
sigma  0.3981  0.2499    1.5932  0.1122
gamma  1.1082  0.2246    4.9336  0.0000

Test of Overidentification:
model is just-identified

Optimization Info:
 Number of Iterations: 5
 Convergence: absolute function convergence
```

Notice that the GMM estimates based on four moments derived from the Euler discretization are more biased and less precise than the EMM estimates based on the six moments derived from the SNP 11111000 auxiliary model.

EMM Estimation on Monthly T-bill Data

Monthly observations on the U.S. T-bill rate over the period June 1964 through November 1989, similiar to the data analyzed by CKLS, are in the `S+FinMetrics` "`timeSeries`" object `ckls.ts`. The SNP auxiliary model for the monthly T-bill data is determined using the function `SNP.auto`:

```
> fit.snp.ckls.ts = SNP.auto(ckls.ts, n.drop=8,
+         control = SNP.control(xTransform = "spline",
+                               n.start = n.start,
+                               fOld = fOld, fNew = fNew),
+         xPolyMax=1,lagPMax=1)
```

The BIC minimizing model is a semiparametric AR(1)-GARCH(1,1) model (11111000):

```
> coef(fit.snp.ckls.ts)

Hermite Polynomial Coefficients:
 z^0     z^1
```

```
   1 -0.5984

Conditional Mean Coefficients:
      mu  ar(1)
 -0.2254 0.8532

Conditional Variance Coefficients:
      s0 arch(1) garch(1)
 -0.0083 -0.1131     0.86
```

The above SNP model passes the standard residual diagnostics, and simulations from the model are dynamically stable.

The EMM estimation with $N = 40,000$ simulated values and 100 burn-in values is computed using

```
> emm.ckls.ts.fit = EMM(fit.snp.ckls.ts, coef=rho.ckls,
+                       control = EMM.control(n.sim=n.sim,
+                                            n.burn=n.burn),
+                       gensim.fn = euler1d.pcode.gensim,
+                       gensim.language = "SPLUS",
+                       gensim.aux=ckls.aux,save.sim=T)
Iteration No. 1, objective = 15897
...
Iteration No. 42, objective = 3.70258

> emm.ckls.ts.fit
Call:
EMM(score = fit.snp.ckls.ts, coef = rho.ckls, control =
EMM.control(n.sim = n.sim, n.burn = n.burn), save.sim
= T, gensim.fn = euler1d.pcode.gensim,
gensim.language = "SPLUS", gensim.aux = ckls.aux)

Coefficients:
         Value Std. Error      95% Conf. Int.
alpha  0.05821    0.05369  0.01217     0.1286
 beta -0.85281    0.82692 -1.88260    -0.7343
sigma  1.11085    0.97791  0.16814     2.1487
gamma  1.44809    0.29755  1.16062     1.7611

Final optimization:
  Convergence: relative function convergence
  Iterations: 42
  EMM objective at final iteration: 3.703
  P-Value:  0.157  on  2  degrees of freedom
```

The EMM estimates converge with a low value of the objective function such that the two overidentifying restrictions implied by the SNP 11111000 model are not rejected. The EMM estimates are similar to the GMM estimates (23.28) for the GCIR model obtained by CKLS, and the GMM estimates using `ckls.ts` reported in Chapter 21. The estimated standard error for the EMM estimate of σ, however, is very large and indicates considerable uncertainty about the value of σ. Although the GCIR model is not rejected using a short span of monthly data, the analysis in the next subsection shows that it may be rejected using a much longer span of weekly data.

Andersen and Lund (1997) One Factor Model

Andersen and Lund (1997) considered fitting the one-factor GCIR model (23.26) to $n = 2155$ weekly observations on the annualized U.S. 3-month T-bill rate (multiplied by 100) over the period January 1954 through April 1995. SNP models were fit to similar weekly interest rate data in the previous chapter. They fit the following reparameterized model

$$dr_t = \kappa(\mu - r_t)dt + \sigma r_t^{\gamma} dW_t,\ \gamma > 0 \qquad (23.30)$$

where μ represents the long-run mean and κ is a speed-of-adjustment parameter. They estimated (23.30) by EMM using a semiparametric AR(5)-EGARCH(1,1) auxiliary model, and reported the following parameter estimates[10]:

$$\hat{\kappa} = 0.082,\ \hat{\mu} = 6.279,\ \hat{\sigma} = 0.670,\ \hat{\gamma} = 0.676 \qquad (23.31)$$

Notice that the estimates of both σ and γ are much smaller than the corresponding estimates (23.28) based on monthly data obtained by CKLS. Based on a large value of the EMM objective function, they rejected the one-factor model. Contrary to the results from EMM estimation of the one-factor model on a short span of monthly data, the one-factor model is strongly rejected using a long span of weekly data.

Simulator Function

The auxiliary information to set the drift and diffusion functions of the one-factor model (23.30) for Euler's method and to calibrate the simulation to weekly data on the 3-month T-bill is

```
> rho.gcir.names = c("k","mu","sigma","gamma")
> gcir.aux = euler.pcode.aux(
+            drift.expr = expression(k*(mu-X)),
```

[10]These estimates are reported in Andersen and Lund (1997, Table 7, p. 370). Andersen and Lund did not use Gallant and Tauchen's FORTRAN code to estimate SNP and EMM models. Their EGARCH specification for the SNP model cannot be implemented using Gallant and Tauchen's SNP code.

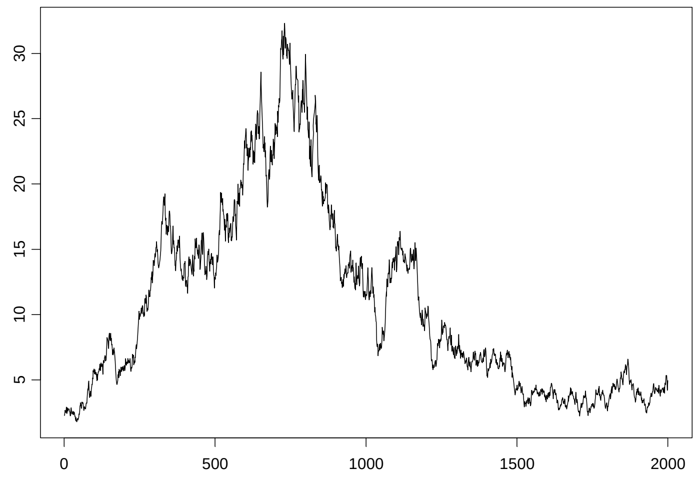


FIGURE 23.9. Simulated values from GCIR process calibrated to weekly observations on the annualized U.S. 3-month T-bill rate.

```
+            diffuse.expr = expression(sigma*X^gamma),
+            rho.names = rho.gcir.names,
+            t.per.sim = 1/52, ndt = 25,
+            lbound=0, ubound=100)
```

For annualized data observed weekly, `t.per.sim` is set to 1/52. To compute a solution path from Euler's method with $n = 2000$ values and 1000 burn-in values, based on the estimates (23.31), use

```
> rho.gcir = c(0.082, 6.279, 0.670, 0.676)
> gcir.aux$seed = 123
> gcir.aux$X0 = 6
> gcir.sim = euler1d.pcode.gensim(rho.gcir, n.sim=2000,
+                                 n.var=1, n.burn=1000,
+                                 aux=gcir.aux)
```

These values are shown in Figure 23.9. Notice that the simulated weekly rates look more like actual rates than the simulated monthly rates shown in Figure 23.7.

EMM Estimation on Simulated Data

The SNP auxiliary model for the simulated weekly data is determined using the function `SNP.auto`:

```
> gcir.sim = as.matrix(gcir.sim)
> fit.snp.gcir = SNP.auto(gcir.sim,n.drop=8,
+         control = SNP.control(xTransform="spline",
+                               n.start = n.start,
+                               fOld = fOld, fNew = fNew),
+         xPolyMax=1,lagPMax=1)
> coef(fit.snp.ckls.ts)

Hermite Polynomial Coefficients:
 z^0     z^1
   1 -0.5984

Conditional Mean Coefficients:
      mu  ar(1)
 -0.2254 0.8532

Conditional Variance Coefficients:
      s0 arch(1) garch(1)
 -0.0083 -0.1131     0.86
> coef(fit.snp.gcir)

Hermite Polynomial Coefficients:
        x^0     x^1
z^0  1.0000  0.2037
z^1 -0.3793 -0.3124

Conditional Mean Coefficients:
      mu  ar(1)
 -0.0487 0.9537

Conditional Variance Coefficients:
      s0 arch(1) garch(1)
 -0.0017 -0.0815   0.9057
```

The BIC minimizing SNP model is a nonlinear nonparametric AR(1)-GARCH(1,1) with $K_z = 1$ and $K_x = 1$, 11111010. The model passes the usual residual diagnostics, and simulations from the model appear dynamically stable.

The EMM estimation with $N = 50,000$ simulated values and 1000 burn-in values may be computed using

```
> n.burn = 1000
> n.sim = 50000
> ndt = 25
> set.seed(456)
> z.gcir = rnorm(ndt*(n.burn+n.sim))
> gcir.aux$z = z.gcir
> emm.gcir.fit = EMM(fit.snp.gcir,coef=rho.gcir,
+                control=EMM.control(n.sim=n.sim,
+                                    n.burn=n.burn),
+                gensim.fn = euler.pcode.gensim,
+                gensim.language = "SPLUS",
+                gensim.aux = gcir.aux, save.sim=T)
Iteration No. 1, objective = 5099.04
...
Iteration No. 31, objective = 1.73639

> emm.gcir.fit
Call:
EMM(score = fit.snp.gcir, coef = rho.gcir, control =
EMM.control(n.sim = n.sim, n.burn = n.burn), save.sim
= T, gensim.fn = euler.pcode.gensim, gensim.language
= "SPLUS", gensim.aux = gcir.aux)

Coefficients:
       Value Std. Error    95% Conf. Int.
    k 0.2768    0.04432 0.2047         NA
   mu 6.8552    0.51202 6.1242     7.6481
sigma 0.8490    0.11229 0.7006     1.0450
gamma 0.5554    0.05951 0.4608     0.6368

Final optimization:
  Convergence: relative function convergence
  Iterations: 31
  EMM objective at final iteration: 1.736
  P-Value:  0.7841  on  4  degrees of freedom
```

The EMM estimates converge with a low value of the objective function that does not reject the four overidentifying restrictions implied by the SNP 11111010 model. All of the estimates, except $\hat{\kappa}$, are reasonably precise and close to their true values. The asymmetric 95% confidence interval for κ indicates that the EMM objective function is nonsmooth in κ near $\hat{\kappa}$. As Gallant and Tauchen (2002) pointed out, estimation can often be improved by utilizing a very long simulation length (e.g., $N = 100,000$) together with random restarts of the optimizer.

EMM Estimation on Monthly T-bill Data

Now, consider EMM estimation of the one-factor model using weekly observations on the U.S. 3-month T-bill contained in the first column of the S+FinMetrics "timeSeries" object tbill.dat. The specification and estimation of SNP models for these data were discussed in the previous chapter.

SNP	$\widehat{\kappa}$	$\widehat{\mu}$	$\widehat{\sigma}$	$\widehat{\gamma}$	J_{EMM}
11118000	1.467	5.559	0.685	1.105	35.7
	(1.095)	(1.419)	(1.073)	(0.593)	(0.000)
10818000	0.227	4.080	0.1373	2.275	47.3
	(0.315)	(3.023)	(0.093)	(0.158)	(0.000)
51118000	0.342	8.219	0.074	1.723	41.8
	(0.446)	(1.731)	(0.068)	(0.327)	(0.000)
10414010	0.445	7.137	2.981	−0.4336	54.4
	(0.078)	(0.401)	(0.506)	(0.1172)	(0.000)

TABLE 23.5. EMM estimates of one-factor GCIR model for weekly 3-month T-bill rates. Starndard errors are in parentheses for estimates; p-vaule in parenthesis for J statistic.

The following four models were suggested: 11118000, 10818000, 51118000 and 10414010. Table 23.5 shows the EMM estimates of the one-factor GCIR model (23.30) based on the above four SNP models. Each model was estimated using $N = 50,000$ simulations and 1000 burn-in values. For example, the EMM estimates based on the SNP 11118000 model are computed using

```
> n.burn = 1000
> n.sim = 50000
> ndt = 25
> set.seed(456)
> z.gcir= rnorm(ndt*(n.burn + n.sim))
> gcir.aux$z = z.gcir
> rho.gcir = c(0.082,6.279,0.670,0.676)
> emm.gcir.11118000.fit = EMM(fit.tb3mo.11118000,
+              coef=rho.gcir,
+              control = EMM.control(n.sim=n.sim,
+                                    n.burn=n.burn),
+              gensim.fn = euler.pcode.gensim,
+              gensim.language = "SPLUS",
+              gensim.aux = gcir.aux, save.sim=T)
```

The one-factor model is rejected using all of the SNP models.[11] Figure 23.10 shows the adjusted score t-ratios (23.21) for the four models. Generally, the one-factor model does not fit the Hermite polynomial terms of the SNP models.

Andersen and Lund (1997) Two-Factor Model

Given the rejection of the one-factor model using weekly data, Andersen and Lund (1997) considered the following two-factor extension of the GCIR

[11]Similar results were obtained using $N = 75,000$ simulations and random restarts.

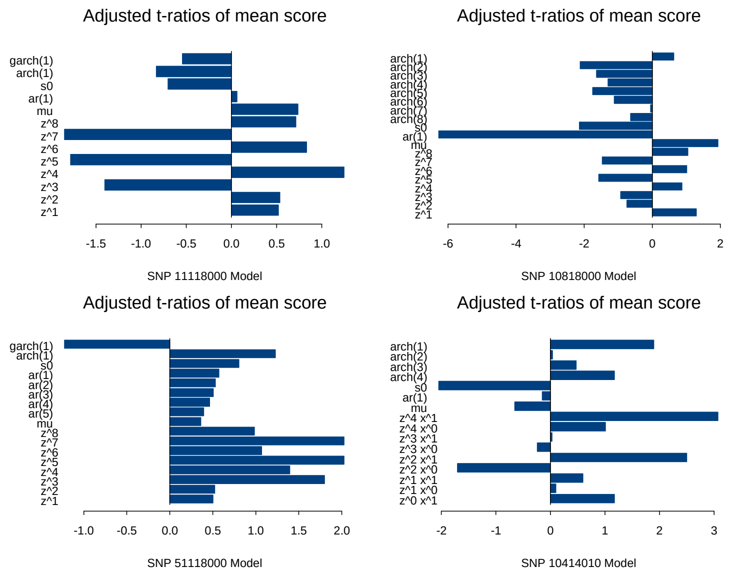


FIGURE 23.10. Adjusted t-ratios for SNP score elements from the EMM estimation of one-factor GCIR model (23.30) for weekly U.S. T-bill rates.

model:

$$
\begin{aligned}
dr_t &= \kappa_1(\mu - r_t)dt + \sigma_t r_t^{\gamma} dW_{1t},\ \gamma > 0 \\
d\ln\sigma_t^2 &= \kappa_2(\alpha - \ln\sigma_t^2)dt + \xi dW_{2t}
\end{aligned} \tag{23.32}
$$

where W_{1t} and W_{2t} are independent Brownian motion processes. The specification implies mean reversion of the interest rate level as well as the (log-) volatility. The parameter α is the long-run mean of log-volatility and μ is the long-run mean for interest rates. Define $v_t = \ln\sigma_t^2 = 2\ln\sigma_t$. Then, $\exp(v_t/2) = \sigma_t$ and (23.32) may be re-expressed as

$$
\begin{aligned}
dr_t &= \kappa_1(\mu - r_t)dt + \exp(v_t/2) r_t^{\gamma} dW_{2t},\ \gamma > 0 \\
dv_t &= \kappa_2(\alpha - v_t)dt + \xi dW_{2t}
\end{aligned}
$$

Let $\mathbf{X}_t = (v_t, r_t)'$ and $\mathbf{W}_t = (W_{1t}, W_{2t})'$. Then (23.32) may be expressed in matrix form

$$
d\mathbf{X}_t = \mathbf{a}(\mathbf{X}_t, t)dt + \mathbf{b}(\mathbf{X}_t, t)d\mathbf{W}_t
$$

where

$$
\mathbf{a}(\mathbf{X}_t, t) = \begin{pmatrix} \kappa_1(\mu - r_t) \\ \kappa_2(\alpha - v_t) \end{pmatrix},\ \mathbf{b}(\mathbf{X}, t) = \begin{pmatrix} \exp(v_t/2) r_t^{\gamma} & 0 \\ 0 & \xi \end{pmatrix} \tag{23.33}
$$

are the drift and diffusion functions, respectively.

Andersen and Lund estimated (23.32) by EMM using a semiparametric AR(5)-EGARCH(1,1) model and reported the following estimates[12]

$$\hat{\kappa}_1 = 0.163,\ \hat{\mu} = 5.95,\ \hat{\gamma} = 0.544,\ \hat{\kappa}_2 = 1.04,\ \hat{\alpha} = -0.282,\ \hat{\xi} = 1.27 \tag{23.34}$$

They reportedd an EMM objective function value of 24.25 with a p-value of 0.019, and concluded that there is mild evidence against the two-factor model for weekly U.S. 3-month T-bill rates.

Simulator Function

Simulated solutions for $\mathbf{X}_t$ from (23.32) based on Euler's method may be computed using the `S-PLUS` function `euler.pcode.gensim`. The auxiliary information to set the drift and diffusion function (23.33) and to calibrate the simulation to the weekly data on 3-month T-bills is

```
> rho.tf.names = c("k1", "mu", "gamma", "k2", "alpha", "xi")
> tf.aux=euler.pcode.aux(
+         rho.names = rho.tf.names,
+         drift.expr = expression(
+                     k1*(mu - X[1]),
+                     k2*(alpha-X[2])),
+         diffuse.expr = expression(
+                       (exp((X[2]-5)*0.5))*X[1]^gamma,
+                        0.0,
+                        0.0,
+                        xi),
+         N = 2, M = 2, t.per.sim = 1/52, ndt = 25,
+         lbound = 0, ubound = 100,
+         returnc = c(T,T))
```

Since there are two factors to simulate and the diffusion matrix is square, `N=2` and `M=2`. Setting `returnc=c(T,T)` specifies that simulated values of both r_t and v_t are returned by the simulator. To ensure that r_t stays positive, `lbound=0`. However, this lower bound applies to both r_t and v_t. Since $v_t = \ln \sigma_t$, v_t can be both positive and negative. To allow v_t to vary freely, the diffusion is reparameterized so that the level of v_t is shifted up by 5 units. As a result, the expression for the drift function for r_t is $\exp((v_t - 5)/2)r_t^{\gamma}$ instead of $\exp(v_t/2)r_t^{\gamma}$.

A solution path of length $n = 2000$, with 1000 burn-in values, using Euler's method calibrated to the estimates (23.34), may be computed using

```
> tf.aux$seed = 678
> tf.aux$X0 = c(6,5-0.28)
```

[12]These estimates are reported in Andersen and Lund (1997, Table 6, p. 368). They are based on a simulation length of $N = 75,000$ with $ndt = 25$.

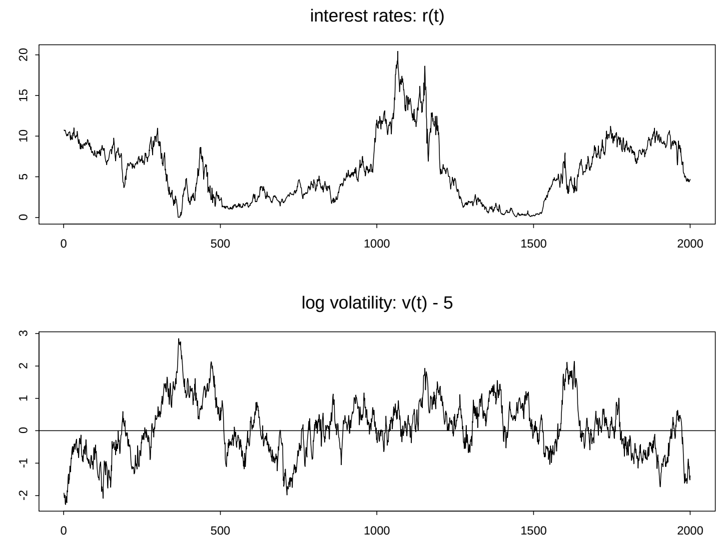


FIGURE 23.11. Simulated values from two-factor GCIR process calibrated to weekly observations on annualized U.S. 3-month T-bill rate.

```
> tf.sim = euler.pcode.gensim(rho=rho.tf,
+                             n.sim=2000, n.var=2,
+                             n.burn=1000, aux=tf.aux)
> tmp = matrix(tf.sim, 2000, 2, byrow=T)
> tfird.sim = as.matrix(tmp[,1, drop=F])
> logvol.sim = tmp[,2] - 5
```

To create the simulation, the initial value for v_t is shifted up by 5 units. Figure 23.11 shows the simulated values for r_t and $v_t - 5$.

EMM Estimation on Simulated Data

The SNP auxiliary model for the simulated data from the two-factor model is determined using the function `SNP.auto`:

```
> fit.snp.tfird = SNP.auto(tfird.sim,n.drop=8,
+                 control = SNP.control(xTransform="spline",
+                                       n.start = n.start,
+                                       fOld = fOld,
+                                       fNew = fNew),
                  lagPMax = 1)
```

The BIC minimizing SNP model is a nonlinear non-parametric AR(1)-GARCH(1,1) with $K_z = 4$ and $K_x = 1$, 11114010:

```
> coef(fit.snp.tfird)

Hermite Polynomial Coefficients:
        x^0     x^1
z^0  1.0000  0.6246
z^1 -0.0765 -0.0517
z^2 -0.0515 -0.0085
z^3 -0.0045 -0.0022
z^4  0.0170  0.0099

Conditional Mean Coefficients:
      mu  ar(1)
 -0.0151 0.9885

Conditional Variance Coefficients:
      s0 arch(1) garch(1)
 -0.0032 -0.1338   0.8569
```

This model is similar to Gallant and Tauchen's preferred SNP model for weekly interest rates. The model passes the usual residual diagnostics, and simulations from the model appear dynamically stable.

The EMM estimation with $N = 50,000$ simulated values and 1000 burn-in values may be computed using

```
> n.burn = 1000
> n.sim = 50000
> ndt = 25
> set.seed(456)
> z.tfird = rnorm(2*ndt*(n.burn + n.sim))
> tfird.aux$z = z.tfird
> tfird.aux$X0 = c(6,5-0.28)
> tfird.aux$returnc = c(T,F))
> rho.tfird = c(0.163, 5.95, 0.544, 1.04, 5-0.282, 1.27)
> emm.tfird.fit = EMM(fit.snp.tfird, coef=rho.tfird,
+                     control = EMM.control(n.sim=n.sim,
+                                           n.burn=n.burn),
+                     gensim.fn = euler.pcode.gensim,
+                     gensim.language = "SPLUS",
+                     gensim.aux = tfird.aux, save.sim=T)
```

Notice that for the two-factor model `2*ndt*(n.burn + n.sim)`, random normals are required for Euler's method. Also, since EMM only requires simulations from r_t, `returnc=c(T,F)` in `tfird.aux`. The results from the estimation are

```
> emm.tfird.fit
Call:
```

```
EMM(score = fit.snp.tfird, coef = rho.tfird, control =
EMM.control(n.sim = n.sim, n.burn = n.burn), save.sim
= T, gensim.fn = euler.pcode.gensim, gensim.language
= "SPLUS", gensim.aux = tfird.aux)

Coefficients:
       Value Std. Error    95% Conf. Int.
   k1 0.2631    0.06132 0.2019     0.4521
   mu 5.4689    0.73000 4.8023     6.4598
gamma 0.4876    0.04728 0.4368     0.5648
   k2 2.0939    0.70286 1.2181     2.9536
alpha 5.2284    0.14385 4.9863     5.3744
   xi 1.6774    0.29380 1.3268     2.1570

Final optimization:
  Convergence: relative function convergence
  Iterations: 29
  EMM objective at final iteration: 5.546
  P-Value:  0.698  on  8  degrees of freedom
```

The high p-value on the EMM objective function indicates that the two-factor model is not rejected by the SNP 11114010 model. All of the estimates are reasonable precise and close to their true values. The estimate of α is 5.2284 - 5 = 0.2284.

EMM estimation on monthly T-bill data

Now consider EMM estimation of the two-factor model using weekly observations on the U.S. 3-month T-bill rate. Table 23.6 shows the EMM estimates of the two-factor GCIR model (23.32) based on the same four SNP models considered for the one-factor model. Each model was estimated using $N = 50,000$ simulations and $1,000$ burn-in values[13]. For example, the EMM estimates based on the SNP 11118000 model are computed using

```
> n.burn = 1000
> n.sim = 50000
> ndt = 25
> set.seed(456)
> z.tfird = rnorm(2*ndt*(n.burn + n.sim))
> tfird.aux$z = z.tfird
> rho.tfird = c(0.163, 5.95, 0.544, 1.04, 5-0.282, 1.27)
> emm.11118000.fit = EMM(fit.tb3mo.11118000, coef=rho.tfird,
+                        control = EMM.control(n.sim=n.sim,
+                                              n.burn=n.burn),
```

[13]Similar results were obtained using $N = 75,000$ simulations.

	11118000	**10818000**	**51118000**	**10414010**
$\hat{\kappa}_1$	1.292	1.215	0.990	1.119
	(1.623)	(3.825)	(0.823)	(0.171)
$\hat{\mu}$	6.848	7.031	10.532	5.977
	(3.371)	(0.893)	(2.869)	(0.314)
$\hat{\gamma}$	0.705	0.014	0.813	0.133
	(0.725)	(1.535)	(0.507)	(0.063)
$\hat{\kappa}_2$	0.879	2.623	1.457	0.775
	(1.650)	(3.079)	(0.983)	(0.151)
$\hat{\alpha}+5$	4.988	3.501	3.576	4.656
	(4.365)	(6.227)	(2.923)	(0.275)
$\hat{\xi}$	1.785	3.251	1.945	1.566
	(0.427)	(0.484)	(0.305)	(0.144)
J_{EMM}	22.61	32.02	29.03	40.77
	(0.002)	(0.002)	(0.002)	(0.000)
df	7	13	11	10

TABLE 23.6. EMM estimates of two factor GCIR model for weekly 3-month T-Bill rates. Starndard errors are in parentheses for estimates; p-vaule in parenthesis for J statistic

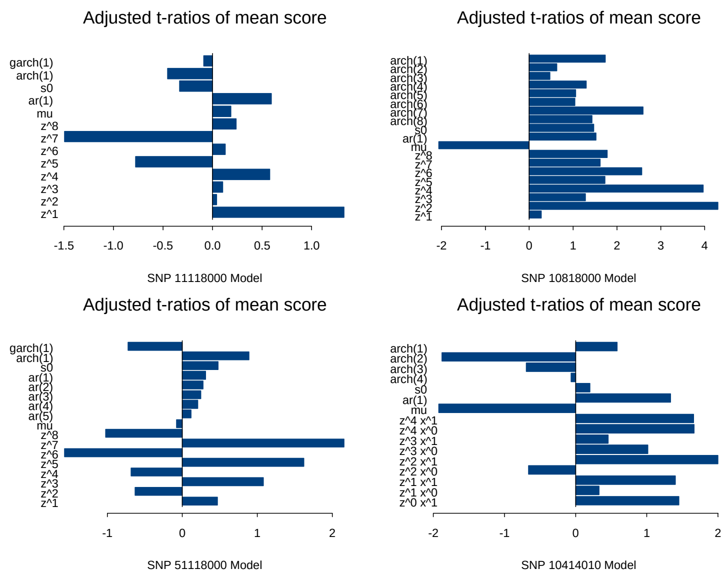


FIGURE 23.12. Adjusted t-ratios for SNP score elements from EMM estimation of two-factor GCIR model (23.32) for weekly U.S. T-bill rates.

```
+                              gensim.fn = euler.pcode.gensim,
+                              gensim.language = "SPLUS",
+                              gensim.aux = tfird.aux, save.sim=T)
```

The one-factor model is rejected using all of the SNP models. Figure 23.12 shows the adjusted score t-ratios (23.21) for the four models. As with the one-factor model, the two-factor model generally does not fit the Hermite polynomial terms of the SNP models.

23.5 References

Ahn, D-H., R.F. Dittmar and A.R. Gallant (2002). "Quadratic Term Structure Models: Theory and Evidence," *Review of Financial Studies*, 15(1), 243-288.

Andersen, T.G., L. Benzoni and J. Lund (2002). "An Empirical Foundation of Continuous-Time Equity Return Models," *Journal of Finance*, 57, 1239-1284.

Andersen, T.G., H.-J. Chung and B.E. Sorensen (1999). "Efficient Method of Moments Estimation of a Stochastic Volatility Model: A Monte Carlo Study," *Journal of Econometrics*, 91, 61-87.

Andersen, T.G. and J. Lund (1997). "Estimating Continuous-Time Stochastic Volatility Models of the Short-Term Interest Rate," *Journal of Econometrics*, 77, 343-377.

Andersen, T.G. and B.E. Sorensen (1996). "GMM Estimation of a Stochastic Volatility Model: A Monte Carlo Study," *Journal of Business and Economic Statistics*, 14, 328-352.

Brandt, M. and P. Santa-Clara (2002). "Simulated Likelihood Estimation of Diffusions with an Application to Exchange Rate Dynamics in Incomplete Markets," *Journal of Financial Economics*, 63, 161-210.

Chan, K.C., G.A. Karolyi, F.A. Longstaff and A.B. Sanders (1992). "An Empirical Comparison of Alternative Models of the Term Structure of Interest Rates," *Journal of Finance*, 47, 1209-1227.

Chernov, M., A.R. Gallant, E. Ghysels and G. Tauchen (2003). "Alternative Models for Stock Price Dynamics," *Journal of Econometrics*, 116, 225-257.

Chumacero, R. A. (1997). "Finite Sample Properties of the Efficient Method of Moments," *Studies in Nonlinear Dynamics & Econometrics*, 2, 35-51.

Chung, C-C. and G. Tauchen (2001). "Testing Target Zone Models Using Efficient Method of Moments," *Journal of Business and Economic Statistics*, 19(3), 255-277.

Coppejans, M. and A.R. Gallant (2002). "Cross Validated SNP Density Estimates," *Journal of Econometrics*, 110(1), 27-65.

Dai, Q. and K.J. Singleton (2000). "Specification Analysis of Affine Term Structure Models," *Journal of Finance*, 55(5), 1943-1978.

Duffie, D. and K.J. Singleton (1993). "Simulated Moments Estimation of Markov Models of Asset Prices," *Econometrica*, 61, 929-952.

Durham, G.B. and A.R. Gallant (2002). "Numerical Techniques for Maximum Likelihood Estimation of Continuous Time Diffusion Precesses," *Journal of Business and Economic Statistics*, 20(3), 297-338.

Elerian, O., S. Chib and N. Shephard (2001). "Likelihood Inference for Discretely Observed Nonlinear Diffusions," *Econometrica*, 69, 959-994.

Fisher, R.A. (1912). "On an Absolute Criterion for Fitting Frequency Curves," *Messages of Mathematics*, 41, 155-157.

Gallant, A.R. (1980). "Explicit Estimators of Parametric Functions in Nonlinear Regression," *Journal of the American Statistical Association* 75, 182-193.

Gallant, A.R., D.A. Hsieh and G.E. Tauchen (1991). "On Fitting a Recalcitrant Series: The Pound/Dollar Exchange Rate, 1974-83," in W. A. Barnett, J. Powell, G. E. Tauchen (eds.), *Nonparametric and Semiparametric Methods in Econometrics and Statistics, Proceedings of the Fifth International Symposium in Economic Theory and Econometrics*. Cambridge University Press, Cambridge, pp. 199-240.

Gallant, A.R., D.A. Hsieh and G.E. Tauchen (1997). "Estimation of Stochastic Volatility Models with Diagnostics," *Journal of Econometrics*, 81(1), 159-192.

Gallant, A.R. and J. Long (1997). "Estimating Stochastic Differential Equations Efficiently by Minimum Chi-squared," *Biometrika*, 84, 125-141.

Gallant, A.R. and D.W. Nychka (1987). "Seminonparametric Maximum Likelihood Estimation," *Econometrica*, 55, 363-390.

Gallant, A.R. and G. Tauchen (1989). "Seminonparametric Estimation of Conditionally Constrained Heterogeneous Processes: Asset Pricing Applications," *Econometrica*, 57, 1091-1120.

Gallant, A.R. and G. Tauchen (1996). "Which Moments to Match," *Econometric Theory*, 12, 657-681.

Gallant, A.R. and G. Tauchen (1999). "The Relative Efficiency of Method of Moments Estimators," *Journal of Econometrics*, 92, 149-172.

Gallant, A.R. and G. Tauchen (2001). "Efficient Method of Moments," unpublished manuscript, Department of Economics, University of North Carolina.

Gallant, A.R. and G. Tauchen (2002). "EMM: A Program for Efficient Method of Moments, Version 1.6, User's Guide," available at `ftp.econ.econ.duke.edu`.

Gallant, A.R. and G. Tauchen (2005). "Simulated Score Methods and Indirect Inference for Continuous-time Models," in Y. Ait-Sahalia and L.P. Hansen (eds.), *Handbook of Financial Econometrics.* Elsevier Science B.V., Amsterdam.

Gauss, C.F. (1816). "Bestimmung der Genauigkeit der Beobachtungen," *Zeitschrift für Astronomie und verwandte Wissenschaften*, 1, 185-196.

Gennotte, G. and T.A. Marsh (1993). "Variations in Economic Uncertainty and Risk Premiums on Capital Assets," *European Economic Review*, 37, 1021-1041.

Ghysels, E., A.C. Harvey and E. Renault (1996). "Stochastic Volatility," in G.S. Maddala and C.R. Rao (eds.), *Handbook of Statistics*, Vol. 14. Elsevier Science B.V., Amsterdam.

Gourieroux, C., A. Monfort and E. Renault (1993). "Indirect Inference," *Journal of Applied Econometrics,* 8, S85-S118.

Hansen, L.P. (1982). "Large Sample Properties of Generalized Method of Moments Estimators," *Econometrica*, 50, 1029-1054.

Heyde, C.C. and R. Morton (1998). "Multiple Roots in General Estimating Equations," *Biometrika,* 85, 949–953.

Ingram, B.F. and B.S. Lee (1991). "Simulation Estimation of Time Series Models," *Journal of Econometrics* 47, 197-250.

Jensen, M. B. (2001). "Efficient Method of Moments Estimation of the Longstaff and Schwartz Interest Rate Model," unpublished manuscript, Department of Business Studies, Aalborg University, Denmark.

Jiang, G.J. and P.J. van der Sluis (2000). "Option Pricing with the Efficient Method of Moments," in Y. S. Abu-Mostafa, B. LeBaron, A. W. Lo and A. S. Weigend (eds.), *Computational Finance*. MIT Press, Cambridge, MA.

Karatzas, I. and S.E. Shreve (1991). *Brownian Motion and Stochastic Calculus, 2nd ed.* Springer-Verlag, Berlin.

Kloeden, P.E. and E. Platen (1992). *Numerical Solution of Stochastic Differential Equations*. Springer-Verlag, Berlin.

Liu, M. (2000). "Modeling Long Memory in Stock Market Volatility," *Journal of Econometrics,* 99, 139-171.

Lo, A.W. (1988). "Maximum Likelihood Estimation of Generalized Ito Process with Discretely Sampled Data," *Econometric Theory,* 4, 231-247.

Mallet, A., F. Mentré, J-L. Steimer and F. Lokiec (1988). "Nonparametric Maximum Likelihood Estimation for Population Pharmacokinetics, with Application to Cyclosporine," *Journal of Pharmacokinetics and Biopharmaceutics,* 16, 311-327.

Nagypal, E. (2001). "Learning-by-Doing versus Selection: Can We Tell Them Apart?" unpublished manuscript, Department of Economics, Stanford University.

Neyman, J. and E.S. Pearson (1928). "On the Use and Interpretation of Certain Test Criteria for Purposes of Statistical Inference," *Biometrika,* 20A, 175-240.

Ng, S. and A. Michaelides (2000). "Estimating the Rational Expectations model of Speculative Storage: A Monte Carlo Comparison of Three Simulation Estimators, *Journal of Econometrics,* 96, 231-266.

Olsen, L.F. and W.M. Schaffer (1990). "Chaos Versus Noisy Periodicity: Alternative Hypotheses for Childhood Epidemics," *Science*, 249, 499-504.

Pearson, K. (1894). "Contributions to the Mathematical Theory of Evolution," *Philosophical Transactions of the Royal Society of London, Series A*, 185, 71-78.

Shephard, N. (1996). "Statistical Aspects of ARCH and Stochastic Volatility", in D.R. Cox, D.V. Hinkley and O.E. Barndorff-Nielsen (eds.), *Time Series Models: In Econometrics, Finance and Other Fields*. Chapman & Hall, London.

Smith, A.A. (1993). “Estimating Nonlinear Time Series Models Using Simulated Vector Autoregressions,” *Journal of Applied Econometrics*, 8, S63-S84.

Tauchen, G. (1997). “New Minimum Chi-square Methods in Empirical Finance,” in D. Kreps and K. Wallis (eds.), *Advances in Econometrics, Seventh World Congress.* Cambridge University Press, Cambridge, pp. 279-317.

Valderrama, D. (2001). “Can a Standard Real Business Cycle Model Explain the Nonlinearities in U.S. National Accounts Data?” Ph.D. thesis, Department of Economics, Duke University.

van der Sluis, P.J. (1999). “Estimation and Inference with the Efficient Method of Moments: With Application to Stochastic Volatility Models and Option Pricing,” Research Series Paper No. 204, Tinbergen Institute, Amsterdam.

Index