

RATS Handbook for ARCH/GARCH and Volatility Models

Thomas A. Doan
Estima

2nd Edition
January 18, 2026

Copyright © 2026 by Thomas A. Doan

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled “GNU Free Documentation License”.

Contents

Preface	vi
1 Introduction	1
1.1 The Example.	4
1.2 Tips and Tricks	7
1.1 Simple Volatility Estimates	8
2 Preliminary Analysis of Returns Data	9
2.1 Robust Sample Statistics	9
2.2 Extending to Multiple Series	13
2.3 Creating a Table	17
2.4 Tips and Tricks	19
2.1 Summary Statistics Allowing for Serial Dependence	22
2.2 Summary Statistics for Multiple Countries	23
2.3 Summary Statistics with Table Generation	24
3 Univariate ARCH and GARCH Models	28
3.1 Introduction	28
3.2 The Example.	31
3.3 Choosing the Mean Model.	32
3.4 Testing for ARCH Effects	35
3.5 Maximum Likelihood Estimation with Gaussian errors	38
3.6 Diagnostics for univariate ARCH models	41
3.7 Maximum Likelihood Estimation with Non-Gaussian Errors	44
3.8 QMLE Estimation	46
3.9 GARCH Models	46
3.1 ARCH Model: Preliminaries	51
3.2 ARCH Model: Estimation	52
3.3 GARCH Model: Estimation	53

4 More on Univariate GARCH Models	55
4.1 Forecasting	55
4.1.1 Evaluating Variance Forecasts	57
4.2 Stationarity	58
4.3 Stability Tests	60
4.4 Variance Equation Dummies	63
4.5 GARCH-M.	66
4.6 Alternative Variance Models.	68
4.7 Asymmetry	74
4.8 Rolling Samples	76
4.1 Univariate Model Forecasting	78
4.2 Stability Tests	78
4.3 GARCH-M, EGARCH and Asymmetry	80
5 Multivariate GARCH: Basics	82
5.1 Preliminaries	82
5.2 GARCH Instruction	85
5.3 Diagnostics	87
5.4 VEC, DVEC and BEKK Models.	90
5.4.1 Diagonal VEC (Standard) Model	91
5.4.2 BEKK Model	96
5.5 Spillover.	100
5.6 CC Models: Constant Correlation	103
5.7 DCC Models: Dynamic Conditional Correlation	106
5.8 RATS Tips and Tricks	113
5.8.1 Graphics with Multiple Series	113
5.8.2 Fancy Table of Diagnostics	114
5.1 Multivariate GARCH: Preliminaries/Diagnostics	116
5.2 Multivariate GARCH: DVEC Estimates	117
5.3 Multivariate GARCH: BEKK Estimates	118
5.4 Multivariate GARCH: Spillover Tests	120
5.5 Multivariate GARCH: CC Estimates	121
5.6 Multivariate GARCH: DCC Estimates	122

6 More on Multivariate GARCH	124
6.1 Forecasting	125
6.2 Volatility Impulse Response Functions	127
6.2.1 Extensions of VIRF	131
6.3 Asymmetry	132
6.4 GARCH-X	138
6.5 Cointegrated Variables	139
6.6 Hedge Ratios and Related Calculations.	146
6.1 Multivariate GARCH: Forecasting	152
6.2 Volatility Impulse Responses	152
6.3 Multivariate GARCH with Asymmetry or Variance Dummies	154
6.4 VECM-GARCH with X Regressor	156
6.5 Hedge Ratios and Portfolio Weights	159
7 Extensions Using MAXIMIZE-Univariate Models	162
7.1 A Simple Example	163
7.2 GMM Estimation.	167
7.3 GARCH Model with Multiplicative Factor	174
7.1 Maximum Likelihood Estimates of Restricted GARCH Model	177
7.2 GMM Estimates of Heteroscedastic Model	178
7.3 GARCH Model with Multiplicative Level Effect	179
8 Extensions Using MAXIMIZE-Multivariate Models	181
8.1 Preparing the Data and Model.	181
8.2 Panel GARCH.	183
8.3 Non-standard Density	190
8.4 Structural VAR with GARCH	195
8.1 Panel GARCH	206
8.2 GARCH Model with Multivariate Skew-t Density	209
8.3 Structural VAR-GARCH	210

9 Simulations and Bootstrapping	215
9.1 Tools	215
9.2 Value-at-risk (VaR) Calculations-Univariate	217
9.3 Value-at-risk (VaR) Calculations-Multivariate.	223
9.4 Variance Impulse Responses by Simulation	227
9.5 RATS Tips and Tricks	232
9.1 VaR Calculations for a Univariate Model	233
9.2 VaR Calculations for a Multivariate Model	235
9.3 VIRF Calculations for with Asymmetry	238
10 Simulation Methods for Model Inference	241
10.1 Bootstrapping	242
10.2 Monte Carlo Methods	245
10.2.1 Importance Sampling	245
10.2.2 Markov Chain Monte Carlo	251
10.2.3 MCMC Estimation of DCC Correlations	256
10.3 RATS Tips and Tricks	261
10.1 Bootstrapping	262
10.2 Univariate Markov Chain Monte Carlo Estimation	263
10.3 Markov Chain Monte Carlo Estimation: Multivariate Model	267
11 GARCH Models with Macroeconomic Data	270
11.1 An SVAR-GARCH-M model	271
11.2 Impulse Responses in an SVAR-GARCH-M Model	277
11.2.1 Error Bands	284
11.1 SVAR-GARCH-M, Estimation	287
11.2 SVAR-GARCH, Impulse Responses	290
12 Stochastic volatility models	297
12.1 State-Space Approximation	297
12.2 Gibbs Sampling	302
12.1 Stochastic Volatility: State-Space Approximation	311
12.2 Stochastic Volatility: Gibbs Sampling	312

A	Probability Distributions	316
A.1	Univariate Normal	316
A.2	Univariate Student (t)	317
A.3	Gamma Distribution	318
A.4	Inverse Gamma Distribution	319
A.5	(Scaled) Inverse Chi-Squared Distribution	320
A.6	Multivariate Normal	321
A.7	Multivariate Student (t).	322
A.8	GED distribution	323
B	Quasi-Maximum Likelihood Estimations (QMLE)	324
C	Diagnostic Tests on Large Data Sets	327
D	Delta method	330
E	Central Limit Theorems with Dependent Data	331
F	Properties of Multivariate Normals	334
G	Pseudo-Code	337
H	Gibbs Sampling and Markov Chain Monte Carlo	339
I	Rejection Method	343
J	GNU Free Documentation License	346
1.	APPLICABILITY AND DEFINITIONS	346
2.	VERBATIM COPYING	348
3.	COPYING IN QUANTITY	348
4.	MODIFICATIONS	349
5.	COMBINING DOCUMENTS	351
6.	COLLECTIONS OF DOCUMENTS	352
7.	AGGREGATION WITH INDEPENDENT WORKS	352
8.	TRANSLATION	352
9.	TERMINATION	353
10.	FUTURE REVISIONS OF THIS LICENSE	353
11.	RELICENSING	354

Contents	vi
Bibliography	355
Index	357

Preface

This workbook is based upon the content of the RATS e-course on ARCH/GARCH and Volatility Models, offered in Fall 2012. Over the years, GARCH models have probably been the second most common application of the RATS software to appear in published articles (after Vector Autoregressions). Roughly half the course concerns the use of the existing **GARCH** instruction—determining the best specification, handling the estimation and doing tests of the adequacy of the model. The second half examines various extensions to the GARCH framework that require either more general likelihood maximization (Chapters 7 and 8), simulation methods (Chapters 9 and 10), or both, with detailed discussions of replications of papers which have been popular downloads by RATS users. The final chapter covers Stochastic Volatility Models, which are similar to GARCH but more complicated technically.

The second edition adds over 50 pages. Some changes reflect improvements to the **GARCH** instruction over the last few years, such as the new **STDRESIDS** and **FACTORBY** options for doing multivariate standardized residuals, and the new **DENSITY** and **PARMSET** options for using non-standard densities.

In addition, there is new or expanded coverage of

- ARMA models for the mean
- evaluating GARCH variance forecasts
- variance shift dummies derived from ICSS pretests
- use/abuse of “rolling window” estimates
- generation of VECM representations for BEKK models
- tests for “spillover” in the mean and in the variance
- DCC models
- Variance Impulse Response Functions both closed form (for models that allow them) or through simulations (for models that don’t)
- methods for handling extreme outliers
- GARCH-X models, particularly as they apply to BEKK
- VECM-GARCH models
- computing and displaying time-varying hedge ratios and portfolio weights

- Cermeño-Grier-style “panel GARCH” models

Style of this Workbook

It assumes that the user is comfortable with such basic instructions as **COMPUTE**, **DISPLAY**, **GRAPH** and **SCATTER** and can use simple programming techniques such as **DO** loops. In many of the chapters, there is a *Tips and Tricks* section which covers in greater detail any functions or instructions that might be unfamiliar.

We use bold-faced Courier (for instance, **GARCH**) for any use of RATS instruction names within the main text, and non-bolded Courier (%RANMVT) for any other pieces of code, such as function and variable names.

For easy reference, the full text of each example is included. The running examples as separate files are available separately. We would strongly suggest that you pull code out of the separate files—PDF often substitutes characters to improve appearance (even in code blocks) so a copy/paste out of the workbook may not work correctly.

Introduction

Variances play a key role in modern finance. Whether it's to price options, compute an optimal hedge ratio, or judge the risk in a portfolio, the variances and covariances of returns on financial instruments are of critical importance. Unfortunately, they are also not observable, and they are, in most cases, extremely large in comparison with what is seen in most types of time series analysis. This course describes methods to estimate these important statistics using RATS.

A common “textbook” assumption in finance, and the assumption on the price process underlying the Black-Scholes pricing model, is that the log of an asset price $\log P$ follows Brownian motion. If $t > s$, this implies that

$$\log P(t) - \log P(s) \sim N(0, \sigma^2(t - s))$$

and this increment is independent of the P process on $(-\infty, s)$. If this is correct, then if we observe equally spaced values of P , the process of log returns $\log P(t) - \log P(t - 1)$ satisfies all of the following:

1. Mean zero
2. Serially uncorrelated
3. Serially independent
4. Constant variance
5. Identically distributed
6. Jointly Normal

Of these, only the first two seem to ever hold empirically and even those will depend upon the asset in question and the sampling frequency. For the moment, however, we'll assume that the underlying assumption is correct. How do we estimate the process volatility σ^2 ? The maximum likelihood estimate (and unbiased estimator) is

$$\frac{1}{T-1} \sum_{t=2}^T (\log P(t) - \log P(t-1))^2$$

Note that this is *not* the standard sample variance estimator. Because the zero mean is assumed, the sample mean of the log returns isn't subtracted off. Also, there is no degrees of freedom correction as there would be to compute an unbiased estimator with an estimated mean—the $T - 1$ divisor is due to the

loss of the data point in computing returns from prices. The RATS **STATISTICS** instruction only does the standard statistics, so it can't be used. Instead, the best choice is **SSTATS**. This could be applied directly to the P series with:

```
sstats(mean) 2 * (log(p)-log(p{1}))^2>>samplevol
```

which computes the mean of the expression into the variable `SAMPLEVOL`. **SSTATS** will prove very useful in the course—see this chapter's Tips and Tricks (Section 1.2) for more on the instruction.

Since all the calculations are done using the log returns, we'll define

$$r(t) = \log P(t) - \log P(t-1) \quad (1.1)$$

and use that in the remainder of the discussion. However, we'll continue to assume that r is only defined from $t = 2$ on, since that's the data you typically would have.

In practice, a full-sample estimate of the volatility is rarely used—the assumption of the constant variance over the complete sample isn't workable. One way to estimate a “slowly-changing” variance while sticking with relatively simple calculations is to use a moving window. With the chosen window width of W , this is

$$\hat{\sigma}_t^2 = \frac{1}{W} \sum_{s=t-W+1}^t r(s)^2$$

which does repeated calculations using the most recent W data points. As defined here, the sequence of estimates starts at $t = W + 2$, which is the first subscript at which we have the full set of required data. The simplest way to compute this with RATS is with **FILTER** applied to the squared log returns. **FILTER** requires a single series (not expression) as its input, so we have to generate the (squared) log returns first:

```
set rsq = (log(p)-log(p{1}))^2
filter(width=w,type=lagging,extend=rescale) rsq / sigsq
```

`TYPE=LAGGING` computes an equally-weighted average of current and lagged values of the input with the number of terms given by the `WIDTH` option. The option `EXTEND=RESCALE` changes the calculation near the start of the data set to average in fewer terms, so this *does* provide estimates for all the data points (beginning with 2), though the usefulness of those early values is low.

While this type of fixed-sized rolling window is very commonly used in financial applications to handle models that are seen as slowly evolving, there is no statistical model which justifies the sharp-cutoff: full weight for entry $t - W + 1$, zero for $t - W$. For many types of models, there isn't a good alternative, but that isn't the case here. An exponentially-weighted average of the squared returns

$$\hat{\sigma}_t^2 = \alpha \sum_{j=0}^{\infty} (1 - \alpha)^j r(t-j)^2$$

smoothly downweights the more distant data. The infinite sum can be eliminated by substitution to give

$$\hat{\sigma}_t^2 = (1 - \alpha) \hat{\sigma}_{t-1}^2 + \alpha r(t)^2$$

which is a weighted average of last period's volatility estimate and this period's squared return. The sum to infinity is replaced with the need for just one pre-sample value. A statistical model which generates an exponentially-weighted average is

$$\begin{aligned} x_t &= x_{t-1} + w_t \\ y_t &= x_t + v_t \end{aligned}$$

which is the local level unobserved components model. x_t is the unobserved local level (which would be the σ_t^2), y_t is the observable ($r(t)^2$), and w_t and v_t are the noise processes. The level of smoothing depends upon the ratio of the two variances—the noisier the observables are relative of the evolution of the level, the more heavily the data are smoothed. The translation of the local level model to volatility isn't perfect: since σ_t^2 needs to be positive, the additive evolution is less than ideal. However, replacing it with a more defensible additive model for $\log \sigma_t^2$ runs into the problem that the observable in the second equation becomes $\log r(t)^2$ which doesn't exist for $r(t) = 0$. The log-additive version of this is the *stochastic volatility model*, which we will cover in Chapter 12. It's a reasonable model, but it requires fairly elaborate statistical methods to deal the behavior of the observation equation.

The simplest way to compute the exponentially-weighted average is with **ESMOOTH** applied to the squared log returns. This requires the **ALPHA** option to choose the smoothing level¹ and a **SMOOTHED** option to choose the target series for the estimates. The default choice on **ESMOOTH** for the one required pre-sample value for the smoothed series is the full-sample average of the input series. If you want to avoid any “forward-looking” behavior in the estimates, you can use the option **INITIAL=START**, which will use just the first observable. As with the case with the fixed window estimates, the early values are of limited value since they are based upon very little data. An example is:

```
esmooth(alpha=.06,initial=start,smoothed=sigsqtex) rsq
```

which will produce the series **SIGSQTEX** as the output. We don't recommend that you “estimate” the smoothing parameter in this type of application. Choose the amount of smoothing you think is proper and input the control for that.

¹Formulas for exponentially-weighted averages often parameterize the smoothing level as the weight on the past value (which would be $1 - \alpha$ in our notation). **ESMOOTH** uses the parameterization that is standard in the exponential smoothing literature, where it's usually the weight on the “correction” term.

1.1 The Example

We'll provide a short example (Example 1.1) to illustrate these calculations. The data set is from West & Cho (1995) which we will use for other examples in this course. This is an XLS file called `WESTCHO-XRATE.XLS`. The raw data are weekly log exchange rates from March 1973 to September 1989 between the U.S. dollar and six currencies: those of Canada, France, Germany, Italy, Japan and the U.K. The weekly value is (if available) the noon bid rate on Wednesday, as Wednesday is rarely a holiday in the U.S. If Wednesday was a non-trading day, the Thursday value was taken, and if Wednesday and Thursday were both non-trading days, Tuesday was used. This is about as close as you can get to “equally-spaced” data in financial markets.

Note the relatively careful choice of the data to use. Even though these were major currencies which were actively traded during the time, the authors used weekly data rather than daily, even though daily would have been available. Daily data can have problems with non-trading days (weekends and holidays) and there is no perfect solution for those—you can either squeeze those out of the data set (so if Friday is a holiday, the entry after Thursday is the following Monday) or you can use treat the missing day as having a zero return from the day prior. The first of these changes the dynamics of the process (you might have three or even four days of “news” separating some entries while most are consecutive days) while the latter has those zero return days which aren't really explained by the model. With daily data in a more thinly traded market, you are much more likely to have serious problems getting a model to fit than with weekly—while volatility clustering declines with increasing frequency, it's still generally apparent with weekly data, even if it's largely gone if you drop all the way to quarterly.

A few other things to note: while they don't directly address this, their analysis starts in 1973, which is when the fixed exchange rates regime ended (and exchange-traded currencies began). A common mistake is to try to work with data which span clearly different regimes (regulated prices, capital controls, etc.) Note also that this paper admits that some of their analysis failed when applied to Italy—it's much more common for a paper to include only the countries/firms where their model worked, giving a misleading impression of how well the model behaves. Just because a model worked with country yyy doesn't mean that it can be applied to country zzz.

We can read the data with:

```
open data westcho-xrate.xls
calendar(w) 1973:3:7
data(format=xls,org=col) 1973:03:07 1989:09:20 $
scan sfra sger sita sjap sukg
```

We'll focus here just on the exchange rate with the Italian lira, in part because the authors chose to leave that out of the rest of the analysis due to some diffi-

culties fitting models to it. Because the data are already in logs, we just need to difference to convert to returns. Note that we multiply by 100. West and Cho also did this, and we recommend that you *always* scale up the data this way when converting to returns. The magnitude of variances of high frequency returns (weekly or finer) is often quite small, and parameters of processes which generate variances are often even smaller than that. With unscaled data, parameters on the order of 10^{-5} to 10^{-8} are common. Aside from being hard to report, these are also difficult to handle numerically, since they are naturally of the order used for “small” values such as convergence checks and perturbations for numerical derivatives. Multiplying the original data by 100 multiplies anything associated with variances by 10^4 , which converts those problematic tiny numbers to something within a few orders of magnitude of 1.

This computes the return and its square and graphs the returns:

```
set xita = 100.0*(sita-sita{1})
set xitasq = xita^2
graph(footer="Lira-USD Exchange Rate Returns")
# xita
```

As we can see in Figure 1.1, there are several huge outliers (in both directions) in a relatively short span in 1976. We will certainly see the effects of those when we take rolling volatility calculations.

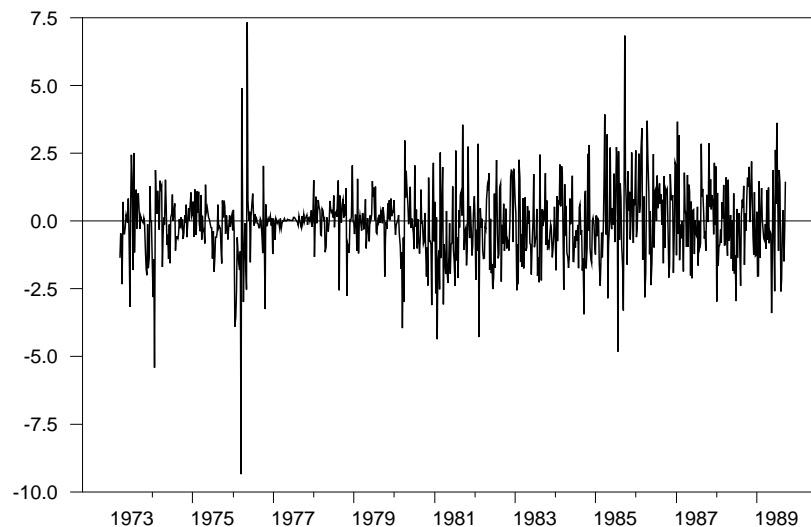


Figure 1.1: Lira-USD Exchange Rate Returns

This computes the full-sample estimate for the variance into the variable ITAVOL:

```
sstats(mean) 2 * xita^2>>itavol
```

In the paper, the authors use a rolling 432 period (roughly eight year) window, which is applied to all models. That’s probably based more on the required sample lengths of the more complicated models. Instead, we’ll do the

rolling window with 52 periods. The “mean lag” of that window is 25.5. An exponentially-weighted window with the same mean lag (and thus somewhat comparable “memory”) is obtained by taking $\alpha = 1/(1 + 25.5)$ (by inverting the mean of a geometric distribution). The calculations for the rolling window and exponentially-weighted estimates are:

```
filter(width=52,extend=rescale,type=lagging) xitasq / itarollvol
esmooth(alpha=1.0/26.5,initial=start,smoothed=itaexpvol) xitasq
```

This graphs the two estimates, with a horizontal line showing the full-sample value,² producing Figure 1.2.

```
graph(footer="Time-Varying Estimates of Historical Volatility",$
      vgrid=itavol,key=below,$
      klabels=||"Rolling Window","Exponentially Weighted"||) 2
# itarollvol
# itaexpvol
```

Clearly neither method seems ideal. The exponentially-weighted average seems to overreact to the outliers, while the rolling window keeps an almost completely flat estimate for a year which suddenly and sharply breaks when the outliers fall off the end of the window. These types of simple calculations will probably prove most useful as comparisons with more complicated models rather than as usable estimates in themselves.

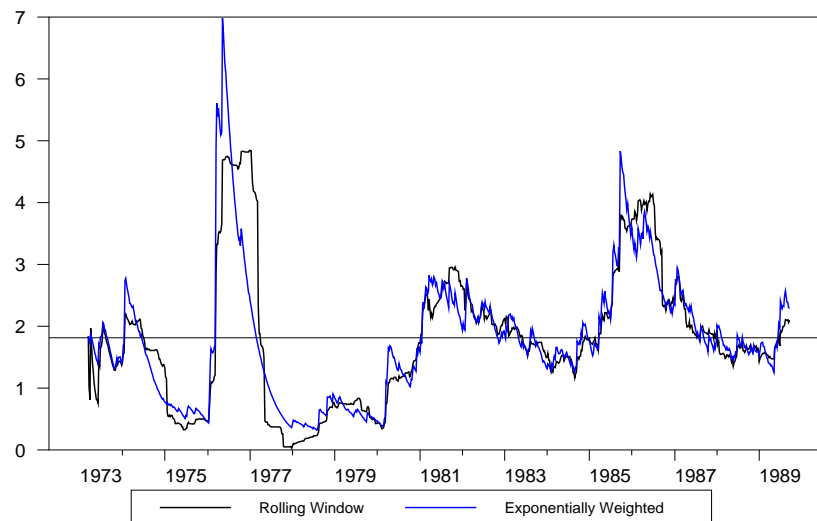


Figure 1.2: Time-Varying Estimates of Historical Volatility

²The horizontal line is done with the `VGRID` option. That can take one value, or a `VECTOR` of values at which you want to draw lines like this.

1.2 Tips and Tricks

The Instruction **SSTATS**

SSTATS is a handy instruction which can be used to compute the sum (or mean or maximum, etc.) of one or more general expressions. Since it accepts a formula, you don't have to take the extra step of generating a separate series with the needed values. By default, it does the sum.

It can be used to answer some (apparently) quite complicated questions. For instance,

```
sstats (min, smpl=peak+trough) start1 endl t>>tp0
```

gives `tp0` as the smallest entry for which either the series `peak` or `trough` (both dummies) is “true”.

```
sstats 1 nob s p1*y>>p1ys p1>>p1s p2*y>>p2ys p2>>p2s
```

computes four parallel sums. Without the **SSTATS**, this would require about eight separate instructions.

```
sstats / date<>date{1}>>daycount
```

computes the number of days in a data set with intra-day data. `date<>date{1}` is 1 when the value of `date(t)` is different from `date(t-1)` and 0 if it's the same. So the **SSTATS** is summing the number of changes in the date series.

Example 1.1 Simple Volatility Estimates

This is the example of simple volatility estimates from this chapter, adapted from West & Cho (1995).

```
open data westcho-xrate.xls
calendar(w) 1973:3:7
data(format=xls,org=col) 1973:03:07 1989:09:20 $
  scan sfra sger sita sjap sukg
*
set xita    = 100.0*(sita-sita{1})
set xitasq = xita^2
*
graph(footer="Lira-USD Exchange Rate Returns")
# xita
*
* Calculation of full-sample volatility estimate
*
sstats(mean) 2 * xita^2>>itavol
*
* Estimation of rolling window estimators using a one-year window,
* and a exponential window with a one-year mean lag.
*
filter(width=52,extend=rescale,type=lagging) xitasq / itarollvol
esmooth(alpha=1.0/26.5,initial=start,smoothed=itaexpvol) xitasq
graph(footer="Time-Varying Estimates of Historical Volatility",$
  vgrid=itavol,key=below,$
  klabels=||"Rolling Window","Exponentially Weighted"||) 2
# itarollvol
# itaexpvol
```

Preliminary Analysis of Returns Data

RATS has quite a few procedures for testing some of the implications of the Brownian motion assumption. If you use the procedure browser on the web site

<https://estima.com/cgi-bin/procbrowser.cgi>

and choose “Dependence Tests” for the subject, you’ll get a list of more than a dozen, many of which are designed specifically for testing for dependence (or lack thereof) in financial returns. At this point, we will stipulate that the overwhelming evidence is that the data *aren’t* sequentially independent—while there may be no linear predictability (that is, the returns may be uncorrelated), there is non-linear information which *is* predictable. Two tests on the list which look at that are the BDS test (@BDS TEST) and the Hinich bi-correlations test (@BICORR TEST). We won’t be using those in this course. A third test that we will use (many times) is @MCLEODLI, from McLeod & Li (1983), which is very specifically aimed at looking for serial dependence in the squares and thus will be more powerful than the more general tests when we are focusing on the properties of variances.

One problem, once we concede that the data aren’t serially independent, is that even computing sample statistics is no longer quite as simple. For instance, one of the implications of the Brownian motion hypothesis is that the returns are Normally distributed. The standard test for Normality in the econometrics literature now is the Jarque-Bera test, which tests (simultaneously) for zero skewness and zero excess kurtosis. The problem is that the calculation of the variances of those higher moments makes assumptions that won’t be holding in financial data. Similarly the standard test for lack of serial correlation (one of the most general implications of the Brownian motion) is the Ljung-Box Q test, which again has asymptotics based upon assumptions which won’t hold if we have higher order dependence.

2.1 Robust Sample Statistics

We will examine these issues using the data from West and Cho, described in Chapter 1. Example 2.1 analyzes the Canadian exchange rate. Their Table 1 gives summary statistics for five of the exchange rate returns (leaving out Italy). In their table footnote 2, it indicates that rows 1-4 (mean, standard deviation, skewness and excess kurtosis of the returns) and 13 and 14 (mean

Table 2.1: Standard Sample Statistics

Statistics on Series XCAN			
Weekly Data From 1973:03:14 To 1989:09:20			
Observations	863		
Sample Mean	-0.020	Variance	0.305
Standard Error	0.552	SE of Sample Mean	0.019
t-Statistic (Mean=0)	-1.047	Signif Level (Mean=0)	0.295
Skewness	-0.424	Signif Level (Sk=0)	0.000
Kurtosis (excess)	5.017	Signif Level (Ku=0)	0.000
Jarque-Bera	930.785	Signif Level (JB=0)	0.000
Minimum	-4.164	Maximum	2.550
01-%ile	-1.355	99-%ile	1.376
05-%ile	-0.872	95-%ile	0.871
10-%ile	-0.644	90-%ile	0.601
25-%ile	-0.312	75-%ile	0.277
Median	-0.030		

and standard deviation of the squared returns) have HAC standard errors. Also, instead of the standard Ljung-Box (LB) test, they have a modified version which is proposed in the paper. Unfortunately, they don't indicate exactly how the HAC standard errors are computed—that it was a Newey-West window seemed rather obvious given that West was a co-author, but the width wasn't. The “rule-of-thumb” from Newey & West (1987) is six for this size data set (as is pointed out later in the paper), but it appears that all these calculations are done with four.

The only number of the four moments and their standard errors that the **STATISTICS** instruction (Table 2.1) will get “right” under these conditions is the sample mean.¹

The formulas that **STATISTICS** uses for the skewness and kurtosis have small sample corrections done under the assumption that the data are i.i.d. Normal, as are the (unreported) standard errors that are used in computing their significance levels. This is handy for testing for Normality when the i.i.d. can be taken as given, but isn't when the data aren't independent.

The HAC standard errors for the mean are easy to calculate, since that can be done by regressing the dependent variable on a `CONSTANT` alone and getting robust errors:

```
linreg(lwindow=newey, lags=4) xcan
# constant
```

The rest are not as simple since they are non-linear functions of the basic moments—the standard deviation is the square root, the skewness is a function of the third moment and the variance, while the kurtosis is a function of the

¹The standard deviation differs in the divisor, which doesn't affect the results at this many digits.

fourth moment and the variance. Standard errors for those can be computed using the *delta method* (Appendix D), but the delta method for the skewness and kurtosis will require not just the HAC estimate of the variances of the second, third and fourth moments, but also their covariance.

If we just needed these one-at-a-time, we could apply the same idea as the mean and do a **LINREG** of the centered powers on **CONSTANT** with HAC standard errors. For instance, for the variance, the following would work:

```
diff(center) xcan / cx
set cxsq = cx^2
linreg(lwindow=newey, lags=4) cxsq
# constant
summarize sqrt(%beta(1))
```

where the **SUMMARIZE** uses the delta method to get the estimated standard error on the estimate of the sample standard deviation. However, since we need a joint covariance matrix, the simplest way to get the full group is with **NLSYSTEM**. We apply method of moments estimation to get the four moments: mean (M1) and the three higher central moments (M2, M3 and M4):

```
nonlin(parmset=meanparms) m1 m2 m3 m4
frml f1 = (xcan-m1)
frml f2 = (xcan-m1)^2-m2
frml f3 = (xcan-m1)^3-m3
frml f4 = (xcan-m1)^4-m4
instruments constant
nlsystem(robust, lags=4, lwindow=newey, parmset=meanparms, inst) 2 * $
    f1 f2 f3 f4
```

Since this is just-identified, the moment estimates are exactly what we would get by just taking averages of the powers of the de-meaned data. What **NLSYSTEM** does is give us a joint HAC covariance matrix. The following uses **SUMMARIZE** to compute the moment statistics and their HAC standard errors:

```
summarize(parmset=meanparms, title="Mean") m1
summarize(parmset=meanparms, title="Standard Deviation") sqrt(m2)
summarize(parmset=meanparms, title="Skewness") m3/m2^1.5
summarize(parmset=meanparms, title="Excess Kurtosis") m4/m2^2.0-3.0
```

The only parts of Panel A left to be done (for Canada) are the modified L-B tests. Because the null here is lack of correlation, the modified tests don't need to allow for the "autocorrelation" part of HAC. However, an assumption underlying the standard L-B test is conditional homoscedasticity:

$$E(\varepsilon_t^2 | \varepsilon_{t-1}, \varepsilon_{t-2}, \dots) = \sigma^2 \quad (2.1)$$

If we look at the calculation of autocovariance j , we have the general result² that

$$\frac{1}{\sqrt{T}} \sum_t \varepsilon_t \varepsilon_{t-j} \xrightarrow{d} N\left(0, E(\varepsilon_t \varepsilon_{t-j})^2\right)$$

With the assumption of conditional homoscedasticity, $E(\varepsilon_t^2 \varepsilon_{t-j}^2) = \sigma^4$ for any $j > 0$ and when converting autocovariances to autocorrelations, the σ factors all cancel. If, however, we have higher-order dependence, (2.1) won't hold and $E(\varepsilon_t^2 \varepsilon_{t-j}^2)$ will be whatever it is, with no simplification possible. The West-Cho test uses the sum of squared autocorrelations just like the Ljung-Box test does, but deflates the individual autocorrelations using the empirical estimates of the variances of the autocovariances. Note that the autocorrelations themselves are the same in each tests—the difference is how these are scaled to create the (asymptotically independent) chi-squareds that form the test statistic. The calculation of these can be done using the `@WESTCHOTEST` procedure. The `NUMBER` option on these sets the number of autocorrelations used. So the three statistics required for the table are done with:

```
@westchotest (number=10) xcan
@westchotest (number=50) xcan
@westchotest (number=90) xcan
```

Panel B

The summary statistics on the squares are most easily done by adapting the `NLSYSTEM` estimates since we already have that written:

```
set esq = xcan^2
nonlin (parmset=meanparms) m1 m2
frml f1 = (esq-m1)
frml f2 = (esq-m1)^2-m2
instruments constant
nlsystem(robust,lags=4,lwindow=newey,parmset=meanparms,inst) 2 * $
    f1 f2
summarize (parmset=meanparms,title="Mean") m1
summarize (parmset=meanparms,title="Standard Deviation") sqrt(m2)
```

The authors do a standard L-B test on the squares, since this is now testing for a null of lack of higher order dependence. What we will use for this is the McLeod-Li test (McLeod & Li (1983)) which is (with very minor changes) exactly the same thing as the L-B test on the squares. This is done with RATS using the `@McLeodLi` procedure. Note that this takes the series itself (not its square) as the input variable—the procedure handles the squaring itself.

```
@mcleodli (number=10) xcan
@mcleodli (number=50) xcan
@mcleodli (number=90) xcan
```

²See Appendix E.

The results of this test (below) show that there *does* seem to be quite significant serial dependence in the second moments. However, it's relatively short-term: the test is *very* strongly significant when 10 lags are used, but moving to 50 or 90 so dilutes the power that it no longer shows as significant.

McLeod-Li Test for Series XCAN		
Using 863 Observations from 1973:03:14 to 1989:09:20		
	Test Stat	Signif
McLeod-Li (10-0)	34.2328241	0.00017
McLeod-Li Test for Series XCAN		
Using 863 Observations from 1973:03:14 to 1989:09:20		
	Test Stat	Signif
McLeod-Li (50-0)	52.4409709	0.37954

2.2 Extending to Multiple Series

The West-Cho paper presents a table with the statistics on five countries. It's quite common for papers analyzing univariate volatility statistics to treat more than one series to provide further evidence of the effect of interest. This section shows a way to convert the above analysis for a single series to produce a table which almost exactly matches the structure in the paper (Table 2.2).

First, note that it is important to get the calculations correct for a single series before expanding to multiple series. A mistake that we see all the time is users starting out trying to do their “final” product, rather than working up to it. You can certainly adjust the set of included statistics after getting a table set up, but don't jump straight into the table without first making sure that you know how to get each statistic when you apply it to just one series. The looping operation to do the set of five series adds another layer of complexity, so if you get answers that appear wrong, you won't know if it's because you are doing the actual calculations wrong, or if you are handling the loop programming wrong. If you start with one series, then you will know what results you *should* be getting for that first column—if your program doesn't produce those, then you're handling the loop structure wrong.

In the typical multi-country or multi-security data set, the looping instruction that will be used is **DOFOR**. This will either be over the list of input series, or (as here) over the abbreviations for the countries. It's a good idea to employ a naming convention for the series from the time it's raw data. With this data set, we have **Sxxx** for the original log exchange rates and **Xxxx** for the log returns, where **xxx** is the country abbreviation—it generally works best to have the individual identifier at the end of the names rather than the beginning, though either is workable.

Our first task is to set up parallel **VECTORS** of **STRINGS** with the abbreviations and (for the table) the full names. The abbreviations are used in referring to the series names; since series names aren't case-sensitive, they can be either upper or lower case. The full names, however, will appear as is in the table, so make them in the desired form.

```
dec vect[string] abbrev(5) full(5)
compute abbrev=||"CAN", "FRA", "GER", "JAP", "UKG"||
compute full = ||"Canada", "France", "Germany", "Japan", "U.K."||
```

Note that the data set also includes data on Italy. The authors decided at some point *not* to include Italy because of problems fitting a later GARCH model. We could put Italy into this table by changing the dimensions to 6, and adding it to the two lists and everything else will go through.³

The next step is to convert the exchange rates to returns. We *could* do this as part of the main calculation loop, but it's a good idea to do any of these first, so you can check that you did them correctly. This could be done almost as easily by copy-paste-edit on

```
set xcan = scan-scan{1}
```

since there are just five countries—if there were twenty, it would be a different story. We're doing the looping version to illustrate the technique of working with series based upon a list of suffixes. Also, the “code cloning” technique will not extend well to the thirty-odd lines of statistical calculations, so we should first make sure we have the structure for walking through the countries correct on the simpler problem.

```
dofor [string] suffix = abbrev
  compute ss=%s("s"+suffix)
  set %s("x"+suffix) = 100.0*(ss{0}-ss{1})
end dofor
```

DOFOR loops over a list of objects. By default, the loop is over integers; if it isn't, we need to make sure the **DOFOR** instruction is told what the loop index should be by declaring it, here, as a **[STRING]**. You could also do this with a separate **DECLARE** instruction for **SUFFIX**:

```
declare string suffix
dofor suffix = abbrev
...
```

but the form we used is more compact, and more understandable.

The **%S** function will be heavily used in any of these types of **DOFOR** loops. It indirectly refers to a series based upon its name using a constructed string. It can even create a new series. In **%s("s"+suffix)**, the current value of **SUFFIX** is appended to the letter **s**, producing, the first time through the loop, **scan**, then **sfra** the second time through, etc.

```
compute ss=%s("s"+suffix)
```

³You could even eliminate the need for the dimensions on the **DECLARE** instruction since the two vectors are dimensioned when set in the **COMPUTE** instructions.

takes that generated name and makes `SS` equal to the (integer) “handle” to the series. This is a way to refer to a series indirectly, but you have to be careful about how you use it. Here, because we need to use the `Sxxx` twice in the transformation, the use of the handle makes the **SET** instruction more readable. The transformation is done with:

```
set %s("x"+suffix) = 100.0*(ss{0}-ss{1})
```

which creates `xcan` the first time through the loop, `xfra` the second time, etc. Note the slight difference in the right-side expression between this and

```
set xcan = 100.0*(scan-scan{1})
```

from the Canada-only example. In the second line, `SCAN` is a series name. A series name by itself in a **SET** (or **FRML**) expression is interpreted to mean the current value of the series, that is, when computing entry `T`, the value `SCAN(T)`. On the other hand, `SS` is merely an **INTEGER** representing a series indirectly. `SS` by itself means the integer value of `SS`, not a series entry reference. Using `SS{0}` forces the desired interpretation—this notation wouldn’t make sense if `SS` weren’t representing a series. This is the one thing about which you must be careful when using the indirect handles. We *could* have avoided using the `SS` variable by writing the expression as the almost unreadable

```
set %s("x"+suffix) = 100.0*(%s("s"+suffix)-%s("s"+suffix){1})
```

but you’re less likely to make mistakes using the simpler coding.

Before you go on, do yourself a big favor and do a **TABLE** instruction to make sure that your data are correct and you got the transformations right. If, for instance, you made a mistake and wrote `SS` rather than `SS{0}` in the **SET** instruction, it would become very obvious by looking at the statistics for the generated `Xxxx` series. If you had an error in one of the abbreviations (say `UK` for `UKG`), the table would show two empty (zero data point) series, one for `SUK` and one for `XUK`. This is because the `%S` function will generate a series if the name doesn’t already exist.

We provide two examples: Example **2.2** just does the calculations for the five countries in a sequence, while Example **2.3** includes both the calculations and the **REPORT** instructions which generate West and Cho’s Table 1. Even if your goal is to have **RATS** produce the table, the first step is to move from Example **2.1** to **2.2**. Since a **REPORT** can be constructed out-of-order (that is, you can put in row 10 before row 2), it’s relatively easy to insert the needed **REPORT** instructions into the loop once you feel confident that you have that correct. This is exactly what we did in this case. Since we will use the looping mechanism again in this course, it’s a good idea to understand how we created Example **2.2**. You can skip over the table generation if you don’t think you are likely to be using **RATS** to create such reports.

To do the computations in a loop, we first copied the **DOFOR** loop from the transformations, deleted the two instructions inside the loop (the **COMPUTE** and **SET**) and pasted into their place the calculations used in Example 2.1, giving us (at first)

```

dofor [string] suffix = abbrev
  *
  * Standard summary statistics
  *
  stats(fractiles) xcan
  .
  .
  .
  @mcleodli(number=90) xcan
end dofor

```

The next step was to indent the body of the loop (everything between the **DOFOR** and **END DOFOR**). See page 20 to see how this is done. The vast majority of programs sent to our technical support department don't indent lines, which can make the program flow almost impossible to follow. Get in the habit of using proper indenting, and you will make fewer errors.

All the calculations done inside this loop refer to **XCAN**. We need to replace those with a reference to the **XXXX** series for the current pass through the loop. This is most easily done using the series handle method:

```

compute x=%s("x"+suffix)

```

then replace **XCAN** with **x** throughout the rest of the loop. There are two places where **XCAN** was used in a formula, where we need to use **x{0}** where **XCAN** itself would be fine:

```

frml f1 = x{0}-m1
frml f2 = (x{0}-m1)^2-m2
frml f3 = (x{0}-m1)^3-m3
frml f4 = (x{0}-m1)^4-m4

```

and

```

set esq = x{0}^2

```

With those corrected, the program to do the calculations for all five series is now done. And, as we said before, if we wanted to add or delete a series from the calculation, all we would have to do is change the definition of the **ABBREV** vector.

2.3 Creating a Table

This section is optional, if you're interested in seeing how to generate high-quality tables using RATS. West and Cho's Table 1 has a rather complicated structure with two "panels" and a varying structure—some of the statistics have standard errors below them, some have significance levels, some have nothing. It's possible to generate this as a single `REPORT`, but it's more flexible to generate separate `REPORTS` for Panel A and Panel B and combine them. That way, if you want to add something to or delete something from Panel A, you won't have to change the row numbers in Panel B.

We'll use three `REPORT` variables: one for the overall table (which we will call `FRAME`), one for Panel A (`PANELA`) and one for Panel B (`PANELB`). `FRAME` will have almost nothing in it until the very end.

```
report (use=frame,action=define,title="Table 1")
```

Because of the different spacing of the statistics, the two left columns of Panel A unfortunately can't be done by any simple method other than inserting strings into the proper slots. This is most easily done using `FILLBY=COLS` and including empty `" "` for cells that are empty by construction:

```
report (use=panela,action=define,title="Panel A")
report (use=panela,atrow=1,atcol=1,fillby=cols,align=right) $
  "1." " " "2." " " "3." " " "4." " " "5." " " "6." " " "7." " " $
  "8." "9." "10." "11." "12."
report (use=panela,atrow=1,atcol=2,fillby=cols,align=left) $
  "Mean" " " "Standard" "deviation" $
  "Skewness" " " "Excess" "kurtosis" $
  "Modified" "L-B(10)" "Modified" "L-B(50)" "Modified" "L-B(90)" $
  "Minimum" "Q1" "Median" "Q3" "Maximum"
```

and similarly for Panel B:

```
report (use=panelb,action=define,title="Panel B")
report (use=panelb,atrow=1,atcol=1,fillby=cols,align=right) $
  "13." " " "14." " " "15." " " "16." " " "17." " "
report (use=panelb,atrow=1,atcol=2,fillby=cols,align=left) $
  "Mean" " " "Standard" "deviation" $
  "L-B(10)" " " "L-B(50)" " " "L-B(90)" " "
```

The **STATS (FRACTILES)** instruction computes the quantiles which will go into rows 15 to 19 of Panel A. Those are inserted using:

```
report (use=panela,atrow=15,col=new,fillby=cols) $
  %minimum %fract25 %median %fract75 %maximum
compute col=%reportcol
```

Since this is the first **REPORT** instruction for the next column, this uses the option `COL=NEW`, which opens a new column. It then saves the number of the new column in the variable `COL` for use in later **REPORT** instructions.

Each of the **@WESTCHOTEST** lines needs to be followed by two **REPORT** instructions: one for the test statistics (saved in `%CDSTAT`), one for its significance level (`%SIGNIF`). The second of these needs the addition of the `SPECIAL=BRACKETS` option.

```
@westchotest(number=10) x
report(use=panela,atrow=9,atcol=col) %cdstat
report(use=panela,atrow=10,atcol=col,special=brackets) %signif
```

You can copy and paste the two **REPORT** lines after each of the next two procedure lines and edit the `ATROW` values.

The moment statistics and their HAC standard errors are inserted by again adding a pair of **REPORT** instructions after each **SUMMARIZE**. The first inserts the variable `%SUMLC` and the second the value of `SQRT(%VARLC)`, both of which are computed by **SUMMARIZE**. For these, we use the `SPECIAL=PARENS` option on the standard error:

```
report(use=panela,atrow=1,atcol=col) %sumlc
report(use=panela,atrow=2,atcol=col,special=parens) sqrt(%varlc)
```

Again, you can copy, paste and edit the row numbers to do the other three moments.

The same types of changes are made to create Panel B. Because we chose to make this a separate report, the row numbers start at 1, which makes it simpler.

It's now time to assemble the final report. That is done with:

```
report(use=frame,atrow=1,atcol=3,align=center) full
report(use=frame,row=new,atcol=1,span) "$\text{Panel A:} e_t$"
report(use=frame,row=new,atcol=1) panela
report(use=frame,row=new,atcol=1,span) "$\text{Panel B:} e_t^2$"
report(use=frame,row=new,atcol=1) panelb
report(use=frame,action=format,align=decimal,picture="*.###")
report(use=frame,action=show)
```

The first line inserts the full names of the countries above columns 3 to 7. The second inserts the descriptor for Panel A. The third inserts the Panel A report at the next line. Note the use of `ROW=NEW` in each insertion after the first. We don't need to know how big `PANELA` was because this simply asks to put the Panel B header at the next line, with the Panel B report after that. We then format to three digits right of the decimal (as is shown in almost all statistics in the published table) with decimal alignment. Decimal alignment is important

when you do the standard errors (in particular), as it gets the numerical values aligned even though the second line has the added parentheses.

To produce the final result, we re-loaded the report (it will be called “Table 1”), did *Select All*, then *Copy to TeX*. The only edits required after pasting were:

1. Adding the `table` environment around the `tabular`
2. Adding rules inside the `tabular` for the top, bottom and mid as shown
3. Adding spacing after the second line in all the multiple line statistics fields. This is done by putting `[.5ex]` after the `\\` line separators on those lines.

The end result is Table 2.2, which, other than some very slight differences in some of the statistics (due to slightly different small-sample corrections) looks almost exactly like the table in the paper.

2.4 Tips and Tricks

How Did the Number of Digits Change in Table 2.1?

When you run the **STATISTICS** instruction on the XCAN return series, you get

Statistics on Series XCAN			
Weekly Data From 1973:03:14 To 1989:09:20			
Observations	863		
Sample Mean	-0.019685	Variance	0.305073
Standard Error	0.552334	SE of Sample Mean	0.018802
t-Statistic (Mean=0)	-1.046990	Signif Level (Mean=0)	0.295398
Skewness	-0.424093	Signif Level (Sk=0)	0.000000
Kurtosis (excess)	5.016541	Signif Level (Ku=0)	0.000000
Jarque-Bera	930.785103	Signif Level (JB=0)	0.000000
Minimum	-4.163577	Maximum	2.550450
01-%ile	-1.355367	99-%ile	1.375713
05-%ile	-0.872448	95-%ile	0.871059
10-%ile	-0.643904	90-%ile	0.600986
25-%ile	-0.312179	75-%ile	0.276547
Median	-0.029924		

which has far too many decimal places for published work. To get the table with the displayed digits cut down to 3, go to the *Window-Report Windows* menu item. This reveals a list of the last twenty “reports” generated by RATS, which will include the output from **STATISTICS** (and **LINREG**, **GARCH**, **MAXIMIZE** and other instructions of that sort). Pick “Statistics on Series XCAN”. In the displayed report, pop up the contextual menu (right-click under Windows) and choose first *Select-All*, then *Reformat*. In the “Report Formatting” dialog that displays, change the *Decimal Width* to 3, and click OK. The report will re-display with everything rounded to three digits. If you do any copy operation or export to any text format, you will paste in text at just the three digits. The actual full precision is still there, and if you copy and paste into a spreadsheet (rather than a text editor), it will use the full precision. The reformatting

is designed for text-to-text copying. In our case, we did *Copy->Tex* (also on the contextual menu) and pasted it into the TeX document. RATS creates the `tabular`, and a `table` needs to be wrapped around it, but relatively little else needs to be done for the finished product.

SUMMARIZE with PARMSET option

RATS version 8.2 added the `PARMSET` option to **SUMMARIZE**, which makes it easier to do the delta method calculations in this chapter—this allows you to analyze the functions in terms of their parameter names rather than just elements of `%BETA`. For instance,

```
nonlin (parmset=meanparms) m1 m2 m3 m4
...
summarize (parmset=meanparms,title="Mean") m1
summarize (parmset=meanparms,title="Standard Deviation") sqrt(m2)
summarize (parmset=meanparms,title="Skewness") m3/m2^1.5
summarize (parmset=meanparms,title="Excess Kurtosis") m4/m2^2.0-3.0
```

Because of the `PARMSET` option, **SUMMARIZE** knows that `M1` is the first estimated parameter, `M2` is the second, etc.

White space: Making Programs More Readable

Well-chosen spaces and line breaks can make it easier to read a program, and will go a long way towards helping you get your calculations correct. At a minimum, you should get into the habit of indenting loops and the like. This makes it much easier to follow the flow of the program, and also makes it easier to skip more easily from one part of the calculation to the next.

Three operations on the *Edit* menu can be helpful with this. *Indent Lines* adds (one level) of indentation at the left; *Unindent Lines* removes one level. The number of spaces per level is set in the preferences in the *Editor* tab. All the programs in this book are done with 3 space indenting, which seems to work well for the way that RATS is structured. In the following, if you select the two lines in the body of the loop and do *Edit-Indent*

```
dofor [string] suffix = abbrev
compute ss=%s("s"+suffix)
set %s("x"+suffix) = 100.0*(ss{0}-ss{1})
end dofor
```

you get the easier to follow

```
dofor [string] suffix = abbrev
  compute ss=%s("s"+suffix)
  set %s("x"+suffix) = 100.0*(ss{0}-ss{1})
end dofor
```

The *Prettify* operation is designed specifically for handling nested **DO** or **DOFOR** loops. If you select a complete loop, and do *Edit-Prettify*, the body of each loop will be indented one (additional) level. For instance, if the original code is the unindented double loop:

```
do i=1,nstep
do j=1,nvar
set %%responses(i,j) draws draws = impulses(j,1)(i)
end do j
end do i
```

if you select the five lines, the *Prettify* operation will produce the much more readable:

```
do i=1,nstep
  do j=1,nvar
    set %%responses(i,j) draws draws = impulses(j,1)(i)
  end do j
end do i
```

Indent Lines and *Unindent Lines* are also on the contextual (right-click under Windows) menu.

Example 2.1 Summary Statistics Allowing for Serial Dependence

This does the summary statistics for the Canadian exchange rate from West & Cho (1995) allowing for serial dependence in the series. The details of the calculations are in Section 2.1.

```
open data westcho-xrate.xls
calendar(w) 1973:3:7
data(format=xls,org=col) 1973:03:07 1989:09:20 $
    scan sfra sger sita sjap sukg
*
set xcan = 100.0*(scan-scan{1})
*
* Standard summary statistics
*
stats(fractiles) xcan
*
* Mean estimate with HAC standard errors
*
linreg(lwindow=newey,lags=4) xcan
# constant
*
* Variance estimate with HAC standard errors
*
diff(center) xcan / cx
set cxsq = cx^2
linreg(lwindow=newey,lags=4) cxsq
# constant
summarize sqrt(%beta(1))
*
* Using delta method for skewness and kurtosis. Get joint estimates
* with HAC covariance matrix.
*
nonlin(parmset=meanparms) m1 m2 m3 m4
*
* Guess values probably don't matter much with this.
*
stats(noprint) xcan
compute m1=%mean,m2=%variance,m3=0.0,m4=3*%variance^2
*
frml f1 = (xcan-m1)
frml f2 = (xcan-m1)^2-m2
frml f3 = (xcan-m1)^3-m3
frml f4 = (xcan-m1)^4-m4
instruments constant
nlsystem(robust,lags=4,lwindow=newey,parmset=meanparms,inst) 2 * $
    f1 f2 f3 f4
*
summarize(parmset=meanparms,title="Mean") m1
summarize(parmset=meanparms,title="Standard Deviation") sqrt(m2)
summarize(parmset=meanparms,title="Skewness") m3/m2^1.5
summarize(parmset=meanparms,title="Excess Kurtosis") m4/m2^2.0-3.0
```

```

*
* West and Cho modified Ljung-Box tests
*
@westchotest(number=10) xcan
@westchotest(number=50) xcan
@westchotest(number=90) xcan
*
* Statistics on squares
*
set esq = xcan^2
nonlin(parmset=meanparms) m1 m2
frml f1 = (esq-m1)
frml f2 = (esq-m1)^2-m2
instruments constant
nlsystem(robust, lags=4, lwindow=newey, parmset=meanparms, inst) 2 * $
    f1 f2
summarize(parmset=meanparms, title="Mean") m1
summarize(parmset=meanparms, title="Standard Deviation") sqrt(m2)
*
* McLeod-Li tests (LB on squares)
*
@mcleodli(number=10) xcan
@mcleodli(number=50) xcan
@mcleodli(number=90) xcan

```

Example 2.2 Summary Statistics for Multiple Countries

This does the summary statistics for the five countries in West & Cho (1995). This is described in Section 2.2.

```

open data westcho-xrate.xls
calendar(w) 1973:3:7
data(format=xls, org=col) 1973:03:07 1989:09:20 $
    scan sfra sger sita sjap sukg
*
dec vect[string] abbrev(5) full(5)
compute abbrev = || "CAN", "FRA", "GER", "JAP", "UKG" ||
compute full = || "Canada", "France", "Germany", "Japan", "U.K." ||
*
do for [string] suffix = abbrev
    compute ss = %s("s"+suffix)
    set %s("x"+suffix) = 100.0*(ss{0}-ss{1})
end do for
table
*
do for [string] suffix = abbrev
    compute x = %s("x"+suffix)
    *
    stats(fractiles) x
    compute m1 = %mean, m2 = %variance, m3 = 0.0, m4 = 3*%variance^2
    *

```



```

* West and Cho modified Ljung-Box tests
*
@westchotest(number=10) x
@westchotest(number=50) x
@westchotest(number=90) x
*
* Using delta method for skewness and kurtosis
*
nonlin(parmset=meanparms) m1 m2 m3 m4
frml f1 = x{0}-m1
frml f2 = (x{0}-m1)^2-m2
frml f3 = (x{0}-m1)^3-m3
frml f4 = (x{0}-m1)^4-m4
*
instruments constant
nlsystem(robust, lags=4, lwindow=newey, parmset=meanparms, inst) 2 * $
    f1 f2 f3 f4
summarize(parmset=meanparms, title="Mean") m1
summarize(parmset=meanparms, title="Standard Deviation") sqrt(m2)
summarize(parmset=meanparms, title="Skewness") m3/m2^1.5
summarize(parmset=meanparms, title="Excess Kurtosis") m4/m2^2.0-3.0
*
set esq = x{0}^2
nonlin(parmset=meanparms) m1 m2
frml f1 = (esq-m1)
frml f2 = (esq-m1)^2-m2
instruments constant
nlsystem(robust, lags=4, lwindow=newey, parmset=meanparms, inst) 2 * $
    f1 f2
summarize(parmset=meanparms, title="Mean") m1
summarize(parmset=meanparms, title="Standard Deviation") sqrt(m2)
*
* McLeod-Li tests (LB on squares)
*
@mcleodli(number=10) x
@mcleodli(number=50) x
@mcleodli(number=90) x
end dofor suffix

```

Example 2.3 Summary Statistics with Table Generation

This does the summary statistics for the five countries in West & Cho (1995), generating (with a small amount of editing) Table 2.2. This does the same statistical calculations as Example 2.2, and is here to show how to create a publication-quality table. The details of how the **REPORT** instruction is used are in Section 2.3.

```

open data westcho-xrate.xls
calendar(w) 1973:3:7
data(format=xls, org=col) 1973:03:07 1989:09:20 $
    scan sfra sger sita sjap sukg

```

```

*
dec vect[string] abbrev(5) full(5)
compute abbrev=||"CAN","FRA","GER","JAP","UKG"||
compute full = ||"Canada","France","Germany","Japan","U.K."||
*
dofor [string] suffix = abbrev
  compute ss=%s("s"+suffix)
  set %s("x"+suffix) = 100.0*(ss{0}-ss{1})
end dofor
table
*
report (use=frame,action=define,title="Table 1")
*
* Define PANELA and PANELB as just the content
*
report (use=panela,action=define,title="Panel A")
report (use=panela,atrow=1,atcol=1,fillby=cols,align=right) $
  "1." "" "2." "" "3." "" "4." "" "5." "" "6." "" "7." "" $
  "8." "9." "10." "11." "12."
report (use=panela,atrow=1,atcol=2,fillby=cols,align=left) $
  "Mean" "" "Standard" "deviation" $
  "Skewness" "" "Excess" "kurtosis" $
  "Modified" "L-B(10)" "Modified" "L-B(50)" "Modified" "L-B(90)" $
  "Minimum" "Q1" "Median" "Q3" "Maximum"
report (use=panelb,action=define,title="Panel B")
report (use=panelb,atrow=1,atcol=1,fillby=cols,align=right) $
  "13." "" "14." "" "15." "" "16." "" "17." ""
report (use=panelb,atrow=1,atcol=2,fillby=cols,align=left) $
  "Mean" "" "Standard" "deviation" $
  "L-B(10)" "" "L-B(50)" "" "L-B(90)" ""
*
dofor [string] suffix = abbrev
  compute x=%s("x"+suffix)
  *
  stats(fractiles) x
  compute m1=%mean,m2=%variance,m3=0.0,m4=3*%variance^2
  report (use=panela,atrow=15,col=new,fillby=cols) $
    %minimum %fract25 %median %fract75 %maximum
  compute col=%reportcol
  *
  * West and Cho modified Ljung-Box tests
  *
  @westchotest (number=10) x
  report (use=panela,atrow=9,atcol=col) %cdstat
  report (use=panela,atrow=10,atcol=col,special=brackets) %signif
  @westchotest (number=50) x
  report (use=panela,atrow=11,atcol=col) %cdstat
  report (use=panela,atrow=12,atcol=col,special=brackets) %signif
  @westchotest (number=90) x
  report (use=panela,atrow=13,atcol=col) %cdstat
  report (use=panela,atrow=14,atcol=col,special=brackets) %signif
  *
  * Using delta method for skewness and kurtosis
  *

```

```

nonlin(parmset=meanparms) m1 m2 m3 m4
frml f1 = x{0}-m1
frml f2 = (x{0}-m1)^2-m2
frml f3 = (x{0}-m1)^3-m3
frml f4 = (x{0}-m1)^4-m4
*
instruments constant
nlssystem(robust, lags=4, lwindow=newey, parmset=meanparms, inst) $
    2 * f1 f2 f3 f4
summarize(parmset=meanparms, title="Mean") m1
report(use=panela, atrow=1, atcol=col) %sumlc
report(use=panela, atrow=2, atcol=col, special=parens) sqrt(%varlc)
summarize(parmset=meanparms, title="Standard Deviation") sqrt(m2)
report(use=panela, atrow=3, atcol=col) %sumlc
report(use=panela, atrow=4, atcol=col, special=parens) sqrt(%varlc)
summarize(parmset=meanparms, title="Skewness") m3/m2^1.5
report(use=panela, atrow=5, atcol=col) %sumlc
report(use=panela, atrow=6, atcol=col, special=parens) sqrt(%varlc)
summarize(parmset=meanparms, title="Excess Kurtosis") m4/m2^2.0-3.0
report(use=panela, atrow=7, atcol=col) %sumlc
report(use=panela, atrow=8, atcol=col, special=parens) sqrt(%varlc)
*
set esq = x{0}^2
nonlin(parmset=meanparms) m1 m2
frml f1 = (esq-m1)
frml f2 = (esq-m1)^2-m2
instruments constant
nlssystem(robust, lags=4, lwindow=newey, parmset=meanparms, inst) 2 * $
    f1 f2
summarize(parmset=meanparms, title="Mean") m1
report(use=panelb, atrow=1, atcol=col) %sumlc
report(use=panelb, atrow=2, atcol=col, special=parens) sqrt(%varlc)
summarize(parmset=meanparms, title="Standard Deviation") sqrt(m2)
report(use=panelb, atrow=3, atcol=col) %sumlc
report(use=panelb, atrow=4, atcol=col, special=parens) sqrt(%varlc)
*
* McLeod-Li tests (LB on squares)
*
@mcleodli(number=10) x
report(use=panelb, atrow=5, atcol=col) %cdstat
report(use=panelb, atrow=6, atcol=col, special=brackets) %signif
@mcleodli(number=50) x
report(use=panelb, atrow=7, atcol=col) %cdstat
report(use=panelb, atrow=8, atcol=col, special=brackets) %signif
@mcleodli(number=90) x
report(use=panelb, atrow=9, atcol=col) %cdstat
report(use=panelb, atrow=10, atcol=col, special=brackets) %signif
end dofor suffix
*
report(use=frame, atrow=1, atcol=3, align=center) full
report(use=frame, row=new, atcol=1, span) "$\text{Panel A:} e_t$"
report(use=frame, row=new, atcol=1) panela
report(use=frame, row=new, atcol=1, span) "$\text{Panel B:} e_t^2$"
report(use=frame, row=new, atcol=1) panelb

```

```
report (use=frame, action=format, align=decimal, picture="*.###")
report (use=frame, action=show)
```

Table 2.2: Reproduction of Table 1 from Paper

	Canada	France	Germany	Japan	U.K.
Panel A: e_t					
1. Mean	-0.020 (0.020)	-0.044 (0.054)	0.042 (0.056)	0.068 (0.054)	-0.052 (0.051)
2. Standard deviation	0.552 (0.029)	1.407 (0.059)	1.465 (0.061)	1.360 (0.059)	1.406 (0.066)
3. Skewness	-0.423 (0.455)	0.103 (0.241)	0.480 (0.207)	0.385 (0.232)	0.261 (0.267)
4. Excess kurtosis	4.981 (2.636)	2.490 (0.837)	2.132 (0.905)	2.587 (0.690)	3.092 (0.830)
5. Modified L-B(10)	7.449 [0.682]	20.881 [0.022]	13.435 [0.200]	20.667 [0.024]	13.572 [0.193]
6. Modified L-B(50)	61.777 [0.123]	65.952 [0.065]	57.118 [0.228]	74.337 [0.014]	55.202 [0.285]
7. Modified L-B(90)	116.218 [0.033]	104.809 [0.136]	96.203 [0.308]	129.329 [0.004]	91.427 [0.438]
8. Minimum	-4.164	-6.825	-4.487	-6.587	-5.691
9. Q1	-0.312	-0.844	-0.837	-0.638	-0.763
10. Median	-0.030	0.000	0.023	-0.027	-0.022
11. Q3	0.277	0.680	0.858	0.609	0.709
12. Maximum	2.550	7.741	8.111	6.546	7.397
Panel B: e_t^2					
13. Mean	0.305 (0.032)	1.983 (0.165)	2.147 (0.181)	1.854 (0.162)	1.978 (0.183)
14. Standard deviation	0.809 (0.213)	4.194 (0.594)	4.392 (0.732)	3.999 (0.483)	4.443 (0.646)
15. L-B(10)	34.233 [0.000]	37.776 [0.000]	56.675 [0.000]	51.860 [0.000]	98.004 [0.000]
16. L-B(50)	52.441 [0.380]	129.434 [0.000]	134.629 [0.000]	101.046 [0.000]	321.813 [0.000]
17. L-B(90)	65.336 [0.977]	178.209 [0.000]	166.104 [0.000]	138.276 [0.001]	336.670 [0.000]

Univariate ARCH and GARCH Models

3.1 Introduction

Heteroscedasticity is a common problem in time series economic data, not just in financial data as shown in Figure 1.2, but also in standard macroeconomic data. While “variance-stabilizing” transformations like converting levels to logs (and thus absolute changes to percentage changes) take care of gross problems with differing variances, there remain inexplicable differences in variance among segments of the data set. However, the standard treatment for heteroscedasticity in the econometrics literature is more appropriate to survey data, with the variance depending upon some function of exogenous variables (like income, population, etc.).

There are several methods to test for variance breaks in time series data based upon time sequencing. The Goldfeld-Quandt test is mainly a test for constancy of variance, rather than an attempt to isolate specific variance regimes, as it mechanically breaks the time series into three segments. Of greater applicability is the test from Inclan & Tiao (1994), which identifies variance breaks using a nested search procedure. This can be done in RATS using the `@ICSS` procedure. If we apply this to the Italian data from Chapter 1 with:

```
@ICSS xita
```

we get Table 3.1.

By construction, these breaks are all significant at the 5% level based upon a test procedure which takes into account that the break points are data-determined and not exogenously input. With seven breaks, this finds eight

Table 3.1: ICSS Analysis of Series XITA

Breakpoints found at entries
1974:04:03
1976:01:14
1976:05:12
1976:09:22
1976:10:27
1978:08:02
1980:10:15

segments with statistically significantly different variances from their neighbors, with some of those segments being as short as five data points. Needless to say, this is not very useful in practice. Even if there appeared to be only a few variance regimes, there remains the problem in forecasting that we would not know which regime would hold into the future. In the textbook use of GLS, the heteroscedasticity is simply a nuisance—we're trying to work around it to get better estimates of the underlying parameters. In finance, the forecast variance is of importance itself—a model where the variance changes based upon an exogenous regime won't be especially helpful.

As noted in Sanso et al. (2004), the original Inlan-Tiao paper assumed the data were Normally distributed, and that that assumption was essential to deriving the statistic.¹ They propose an alternative which allows for non-Gaussian data by estimating the fourth moment instead of taking the implied $3\sigma^4$ that's appropriate for a Normal. That can be done using the @ICSS procedure by adding the ROBUST option. Applying that to the same data set gives the greatly reduced set of breaks:

```
ICSS Analysis of Series XITA
Using significance level 0.05
Breakpoints found at entries
1976:05:12
1980:02:27
```

so the original report was presumably thrown off by poor handling of outliers.

There have been some attempts to combine ICSS analysis with GARCH modeling. This is *not* a good idea, as the two methods are competing to explain the same information. We'll discuss this further in Section 4.4.

To provide a time-series model for heteroscedasticity, Engle (1982) introduced the ARCH model (AutoRegressive Conditional Heteroscedasticity), which makes the variance endogenous to the process. It generates the type of variance clustering evident in the ICSS output, but where the variance is a closed form of the data, so it can be forecast out-of-sample.

ARCH is a description of the error process for a regression. The assumptions are (using our notation):

- (a) $y_t = x_t' \beta + u_t$
- (b) $u_t = \sqrt{h_t} v_t$
- (c) $\{v_t\}$ i.i.d., $Ev_t = 0$, $Ev_t^2 = 1$
- (d) $h_t = c_0 + a_1 u_{t-1}^2 + \dots + a_m u_{t-m}^2$

Engle assumed, specifically, that the v_t process was Normal(0,1). The u_t is conditionally heteroscedastic, that is, the variance of u_t given past u changes

¹Although the ICSS test statistic is based upon the behavior of a Brownian Bridge, which is usually robust to the distribution of the underlying data, the fourth moment is needed to derive the statistic.

with the observed values (with large residuals increasing the variance in the near-future), but is unconditionally homoscedastic, with

$$\sigma^2 = \frac{c_0}{(1 - a_1 - \dots - a_m)}$$

(assuming the denominator is positive). It also has the properties that, while the residuals are conditionally Normal, they are unconditionally fatter-tailed than the Normal. Thus the ARCH process is consistent with two observations regarding financial returns: leptokurtosis (fat-tails) and the clustering of large values. While Engle's initial example (to U.K. inflation data) was not particularly successful, with the improved variant of ARCH in the form of the GARCH model of Bollerslev (1986), and its application to financial data rather than macroeconomic, Engle's idea became the dominant methodology for volatility modeling.

In this chapter, we will start with an example of ARCH and demonstrate preliminary testing for ARCH effects, selection of the mean model, estimation and diagnostics. We will then move on to the much more commonly used GARCH model. Note, however, that most of the steps are the same for either model.

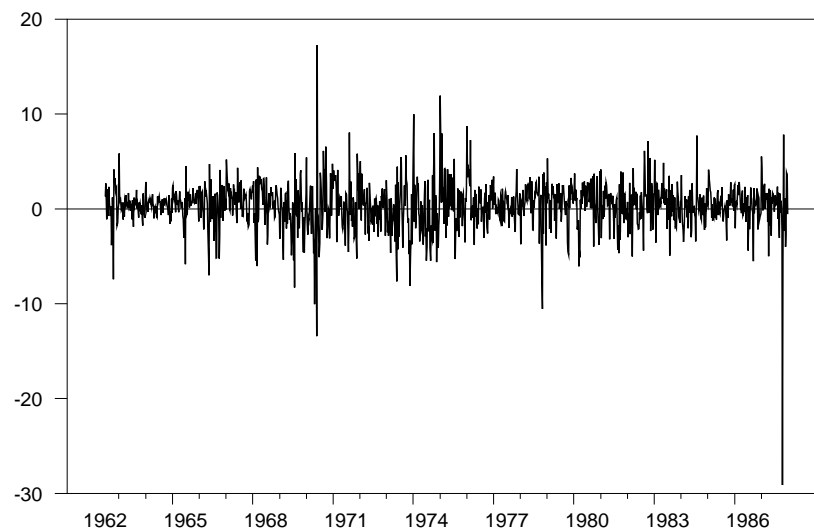


Figure 3.1: Value-weighted NYSE returns

3.2 The Example

The data set is the one used in Hamilton & Susmel (1994), which has weekly returns (Wednesday to Wednesday) on a value-weighted portfolio of the stocks traded on the New York Stock Exchange, roughly 1300 observations from July 1962 to December 1987. This is a sample long enough to include the October 1987 crash. The preliminary analysis of the data is in Example 3.1. We read the data and graph it with

```
open data crspw.txt
calendar(w) 1962:7:3
data(format=prn,nolabels,org=columns) 1962:07:03 1987:12:29 vw
*
* Copy the data over to y to make it easier to change the program.
*
set y = $
*
graph(footer="Figure 3.1 Value-Weighted NYSE Returns")
# y
```

The data in Figure 3.1 show quite a bit of randomness, with several extremely large outliers (of both signs) mixed in. There are long periods of relative quiet (1960's, most of the 1980's) with a period of much higher variance from 1970 to 1975.

Table 3.2: Bayesian IC Lag Analysis of AR Models

Schwarz/Bayesian IC Lag Analysis for Y	
Lags	IC
0	1.831
1	1.793*
2	1.798
3	1.797
4	1.803
5	1.808
6	1.808
7	1.812
8	1.814
9	1.819
10	1.824

3.3 Choosing the Mean Model

It's not obvious to the eye that the data in Figure 3.1 are serially correlated. However, with a data set this size, even very modest levels of autocorrelation can be statistically significant—the rule of thumb for 1300 data points is that an autocorrelation as small as .06 is significant at the .05 level.² The ARCH and GARCH models assume that the residual process being modeled is white noise, so we need to choose a model for the process mean which delivers uncorrelated residuals.

One problem with choosing a model for the mean is that most relatively simple methods for choosing a model for the serial correlation of a series assume that the residuals are homoscedastic. However, if we're expecting ARCH or GARCH residuals, that won't be the case. Despite that, the mean model is generally chosen using standard methods—we then test the residuals to see if any adjustments need to be made once we've estimated a complete model.

The West-Cho test shows that we *do* have significant serial correlation, even when we allow for heteroscedasticity:

```
@westchotest (number=10) y
```

West-Cho Modified Q Test, Series Y	
Q(10)	36.55
Signif.	0.0001

We will use the `@ARAutoLags` procedure to help choose the mean model. The chosen number of lags using BIC is 1 (Table 3.2). Different criteria give different results here, as Hannan-Quinn (CRIT=HQ) chooses 3 and Akaike (CRIT=AIC) chooses 8. For various reasons, it makes sense to start with a smaller model and expand it if that seems to be needed. Remember, again,

²The asymptotic standard deviation of any autocorrelation under the null of no serial correlation is $1/\sqrt{T}$ or .028 for $T = 1300$.

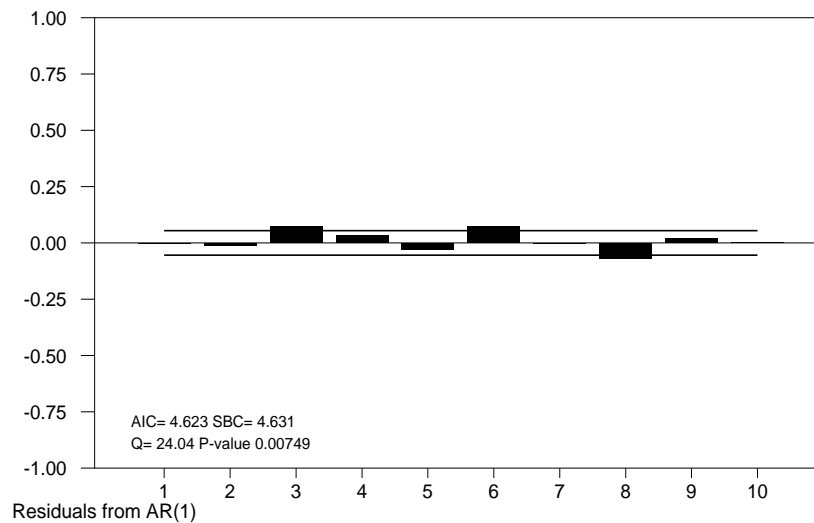


Figure 3.2: Residual Analysis from AR(1) mean model

that all of these are derived under the assumption of homoscedasticity. When there may be (and, in this case are) some overly large residuals, it's much easier to get (apparently) large autocorrelations by chance.

```
@arautolags(maxlags=10,crit=bic,table) y
```

Based upon this, we will use an AR(1) model for the mean throughout. The first step is to estimate the base mean model:

```
linreg y
# constant y{1}
```

If we do a standard Ljung-Box Q test on the residuals from this, we would (strongly) reject that they are white noise:

```
@regcorrs(number=10,qstat,footer="Residuals from AR(1)")
```

produces Figure 3.2. However, despite the strong rejection of white noise, it doesn't suggest any obvious changes to the model to fix this. The “large” autocorrelations at 3 and 6 would not easily be corrected by any simple adjustment.

By contrast, the West-Cho test with the same number of correlations yields quite a different result as

```
@westchotest(number=10) %resids
```

produces

West-Cho Modified Q Test, Series %RESIDS	
Q(10)	16.37
Signif.	0.0895

which isn't significant at the conventional .05 level.

ARMA Models

In almost all applications of ARCH and GARCH models, the mean model is largely treated as a nuisance. Financial returns data is typically, at best, weakly autocorrelated in the mean, so the job of the mean model is to clean up what little serial correlation there is so we can concentrate on the variance model. We'll see examples in Chapter 11 where the main emphasis is more on the mean model with the variance model being of secondary importance, but those are multivariate models with macroeconomic data, not univariate models with financial data.

A common error is to take the mean model used in paper xxx, and try to apply it to a different data set. Where the specific mean model is of little or no direct interest, you want to choose the simplest model that you can that delivers the white noise residuals that you need for a particular series; *not* the model someone else used on a different data set.

A particular example of (incorrectly) basing a model on a published paper is choosing an ARMA model rather than the simpler AR. For data with fairly strong serial correlation, ARMA models are important because they can fit a wide range of serial correlation patterns often with (many) fewer parameters than an AR alone. However, for data with *weak* serial correlation, there is no real advantage to an ARMA model and a major computational disadvantage—ARMA models with both AR and MA terms are subject to “cancellation”, where many different ARMA models produce the same residuals. For instance, if y is white noise (so an ARMA(0,0) model is the correct one), then

$$(1 - \varphi L) y_t = (1 + \theta L) \varepsilon_t \Rightarrow y_t = \frac{(1 + \theta L)}{(1 - \varphi L)} \varepsilon_t$$

where the numerator and denominator “cancel” to produce the correct $y = \varepsilon$ for *any* $\theta = -\varphi$. In practice, you don't get *exact* cancellation, but instead will generally see large AR and MA coefficients which are almost equal of opposite signs and both with very high standard errors. If you try to do an ARMA model and see that, that's a sign that you don't need the MA. (If you read a paper with an GARCH model with an ARMA mean, you'll probably see that.) Note that this has almost no effect on the variance model—the reason you get the cancellation effect is that the two parts cancel out of the calculation of the residuals, leaving the same residuals to generate the same GARCH properties. The numerical problem is that in a joint estimation, the poorly-estimated (nuisance) mean model parameters make it harder to fit the overall model.

3.4 Testing for ARCH Effects

There are two principal methods to test for ARCH effects in the residuals for a regression (or ARMA model). The first is McLeod-Li as already discussed on page 12. The other is the LM test from Engle (1982), which takes the squared residuals and regresses on a constant and lagged squared residuals. The test is similar to a Breusch-Godfrey test for serial correlation applied to the squares instead of the residuals themselves, but with one critical difference: in the BG test, you include the original regressors as well as the lagged residuals. In the ARCH test, you can ignore the fact that the residuals came from a regression because there is a standard result that, in the case of a regression with heteroscedastic errors, the parameter estimates for the regression itself and for the scedastic function are asymptotically uncorrelated (assuming no direct connection like a shared parameter).

This test can be implemented fairly easily using a **LINREG** instruction, though there is also an **@ARCHTEST** procedure, which we will use as well. You need to square the residuals from the AR(1) model (after we copy them to the separate series **U** for use later), and regress the square on the **CONSTANT** and a chosen number of lags (here 10) of the square. We're also including here a test of all but the first two lags (judged by looking at the results of the regression):

```
set u = %resids
set u2 = u^2
linreg u2
# constant u2{1 to 10}
exclude(title="ARCH Test: F Variant")
# u2{1 to 10}
exclude
# u2{3 to 10}
cdf(title="ARCH Test: Chi-Squared Variant") chisqr %trsquared 10
```

There are several ways to convert the information from this auxiliary regression into a test statistic. This shows the “F-test” variant, which just tests the lagged squares using a standard F -test. Note that we don't even have to do the separate **EXCLUDE** instruction, since it is just reproducing the Regression F in the standard **LINREG** output. There's also a chi-squared variant which uses TR^2 —this has asymptotically a χ^2 with degrees of freedom equal to the number of tested lags.³ The two should give *very* similar significance levels, particularly with the high number of degrees in this case (1309).

The coefficients on the regression are typically not of independent interest—we just need the overall significance of the statistic. However, here the test on the final eight lags suggests that for an ARCH model, two lags will be adequate:

³In RATS variables, TR^2 is %TRSQUARED.

```

ARCH Test: F Variant

Null Hypothesis : The Following Coefficients Are Zero
U2              Lag(s) 1 to 10
F(10,1309)=      2.17980 with Significance Level 0.01679564

Null Hypothesis : The Following Coefficients Are Zero
U2              Lag(s) 3 to 10
F(8,1309)=       0.35015 with Significance Level 0.94601519

ARCH Test: Chi-Squared Variant
Chi-Squared(10)= 21.621117 with Significance Level 0.01715551

```

The `@ARCHTEST` procedure can also be used to do the LM test. This does the test for up to 10 lags, showing the results for every number of lags from 1 to 10.⁴ You pass the residuals themselves (or whatever series you want to test) to the procedure, *not* the square: `@ARCHTEST` handles the squaring of the series itself.

```
@ARCHtest(lags=10,span=1) u
```

produces

```

Test for ARCH in U
Using data from 1962:07:10 to 1987:12:29

Lags Statistic Signif. Level
 1    12.712      0.00038
 2     9.620      0.00007
 3     6.423      0.00026
 4     4.860      0.00068
 5     4.170      0.00092
 6     3.594      0.00154
 7     3.110      0.00294
 8     2.725      0.00557
 9     2.424      0.00991
10     2.180      0.01680

```

The `lags=10` value matches what we had above. Given the results on the separate exclusion on lags 3 through 10, it's not a surprise that the test statistic (while still very significant) is not as large for 10 lags as for 2. As with any test like this, the power is reduced if you put in more lags than actually needed. The advantage of doing `@ARCHTEST` with the `SPAN` option is that you can see whether the effect is there at the short lags even if it has been diluted in the longer lags.

The McLeod-Li test with the same number of lags is done with

```
@McLeodLi(number=10) u
```

```

McLeod-Li Test for Series U
Using 1330 Observations from 1962:07:10 to 1987:12:29
McLeod-Li(10-0) 26.3042403 0.00335

```

This strongly agrees with the conclusion from above—the squared residuals are clearly not serially uncorrelated.

⁴The `SPAN` indicates the distance between lag lengths used. So `LAGS=24, SPAN=4` will do 4, 8, 12, ..., 24.

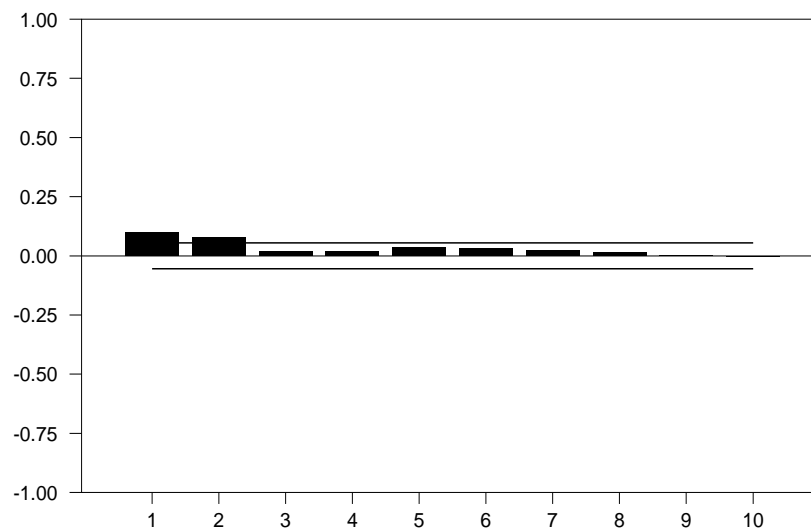


Figure 3.3: Correlations of Squared Residuals

Though not typically necessary, you can also use `@REGCORRS` applied to the squared residuals to get a graphical look at the correlations of the squares. The following creates Figure 3.3:

```
@regcorrs (number=10,nocrits,$
           title="Correlations of Squared Residuals") u2
```

It's important to note that these two tests don't prove that an ARCH or GARCH model is appropriate, just that there is some clustering of large residuals. One of the observations about these data was that there appears to be quieter and noisier segments. Suppose the data had been generated using exogenously-determined variance regimes, more specifically:⁵

```
seed 59343
set utest = %if(t>=1970:1.and.t<1976:1,%ran(3.0),%ran(2.0))
```

Even though the regimes aren't *that* much different (variance 9 in the middle, variance 4 on the two ends), the two tests for ARCH both come in very strongly significant:

```
@archtest(lags=10) utest
@mcLeodli(number=10) utest
```

⁵With weekly or daily data, the notation `year:1` is the first data point in the **CALENDAR** scheme that is in `year`.

```

Test for ARCH in UTEST
Using data from 1962:07:03 to 1987:12:29

Lags Statistic Signif. Level
10      6.080      0.00000

McLeod-Li Test for Series UTEST
Using 1331 Observations from 1962:07:03 to 1987:12:29
McLeod-Li(10-0) 83.8174562    0.00000

```

Both these procedures are testing for persistence in the squares. From the literature on unit root tests under (unmodeled) structural breaks (Perron (1989) and later papers), we know that structural breaks can cause false acceptances of persistence—what we’re seeing here is the same type of behavior in squares rather than levels.

3.5 Maximum Likelihood Estimation with Gaussian errors

The estimation of the ARCH model is in Example 3.2. From the **LINREG** on the squared residuals on its lags, it appears that two ARCH lags will be adequate. The model we will be estimating can be written:

$$\begin{aligned}
 y_t &= \alpha + \rho y_{t-1} + u_t \\
 h_t &= c_0 + a_1 u_{t-1}^2 + a_2 u_{t-2}^2 \\
 u_t | \text{past } y &\sim N(0, h_t)
 \end{aligned}$$

For an ARCH model, the log likelihood can be computed exactly given the observed data if we give up one data point for the lagged y in the mean equation and two more for the lags of u_t required to compute h_t . If $t = 1$ represents the first data point at which u_t can be computed (which would be entry 2 given the data set), then the full sample log likelihood would be

$$\sum_{t=3}^T \left\{ -\frac{1}{2} \log(2\pi) - \frac{1}{2} \log h_t - \frac{u_t^2}{2h_t} \right\} \quad (3.1)$$

However, RATS (by default) does not condition on the extra two data points for the lagged u_t .⁶ Instead, RATS uses

$$h_t = c_0 + a_1 u_{t-1}^2 + a_2 u_{t-2}^2 \quad (3.2)$$

where (using the more generic $x_t' \beta$ for the mean model)

$$u_s^2 = \begin{cases} (y_s - \mathbf{x}_s' \beta)^2 & \text{if } s \geq 1 \\ \hat{\sigma}^2 & \text{if } s \leq 0 \end{cases} \quad (3.3)$$

$\hat{\sigma}^2$ is a sample estimate of the variance of the residuals. Thus pre-sample squared residuals are replaced by a sample estimate of their mean. The difference between the two estimates is likely to be slight. Unlike the case of an

⁶There is a **CONDITION** option which allows you to compute the likelihood for an ARCH conditioning on the early observations. Here you would use **CONDITION=2** to get the likelihood in (3.1).

ARMA model, there is no unconditional density for the pre-sample of an ARCH process. Different software will handle the pre-sample differently and so will give somewhat different results.

The RATS instruction for estimating the model is

```
garch(p=0,q=2,regressors) / y
# constant y{1}
```

The `Q=2` selects an ARCH(2) model. The `REGRESSORS` option is used to indicate that there is a regression model for the mean—the default is to just use a simple process mean, the equivalent of using just the `CONSTANT`. As with a `LINREG`, the explanatory variables follow on a supplementary card using regression format. You'll note that the sample range (`/` to mean the default) comes *before* the name of the dependent variable. This is because the `GARCH` instruction can be used for both univariate and multivariate models, so the list of dependent variables needs to be open-ended. The output is in Table 3.4. By contrast, the OLS results for the mean model are in Table 3.3.

One thing you will note right away is that there are *many* fewer summary statistics in the output for the ARCH model. There are a number of reasons for that, the most important of which is that almost all the summary statistics for least squares are interesting only if you are minimizing the sum of squared residuals, which ARCH most definitely is not.⁷ The one summary statistic that they have in common is the log likelihood, which is comparable here since the two are estimated over the same sample range.⁸ And the log likelihood for the ARCH estimates is *much* higher (-2960 vs -3072).

In the `GARCH` output, the mean model parameters are listed first, followed by the variance parameters. The `C` in the output is the constant in the variance model (c_0) and the `A(1)` and `A(2)` are the coefficients on the lagged variances (a_1 and a_2).

⁷ R^2 is based upon a ratio of sums of squares, the standard errors are estimates of a single variance using sums of squares, etc.

⁸This is possible because of how RATS handles the pre-sample squared residuals. If the likelihood were conditional on the lagged residuals, the ARCH estimates would have two fewer observations.

Table 3.3: OLS Estimates of Mean Model

Linear Regression - Estimation by Least Squares				
Dependent Variable Y				
Weekly Data From 1962:07:10 To 1987:12:29				
Usable Observations		1330		
Degrees of Freedom		1328		
Centered R ²		0.0425		
R-Bar ²		0.0418		
Uncentered R ²		0.0592		
Mean of Dependent Variable		0.3319		
Std Error of Dependent Variable		2.4925		
Standard Error of Estimate		2.4398		
Sum of Squared Residuals	7905.0314			
Regression F(1,1328)	59.0119			
Significance Level of F	0.0000			
Log Likelihood	-3072.4314			
Durbin-Watson Statistic	2.0018			
Variable	Coeff	Std Error	T-Stat	Signif
1. Constant	0.2631	0.0675	3.8984	0.0001
2. Y{1}	0.2063	0.0269	7.6819	0.0000

Table 3.4: ARCH(2) with Gaussian Errors

GARCH Model - Estimation by BFGS					
Convergence in 22 Iterations. Final criterion was 0.0000032 <= 0.0000100					
Dependent Variable Y					
Weekly Data From 1962:07:10 To 1987:12:29					
Usable Observations	1330				
Log Likelihood	-2959.5402				
Variable	Coeff	Std Error	T-Stat	Signif	
1. Constant	0.3959	0.0556	7.1263	0.0000	
2. Y{1}	0.2903	0.0353	8.2267	0.0000	
3. C	2.7312	0.2156	12.6663	0.0000	
4. A{1}	0.5384	0.0683	7.8880	0.0000	
5. A{2}	0.1564	0.0410	3.8101	0.0001	

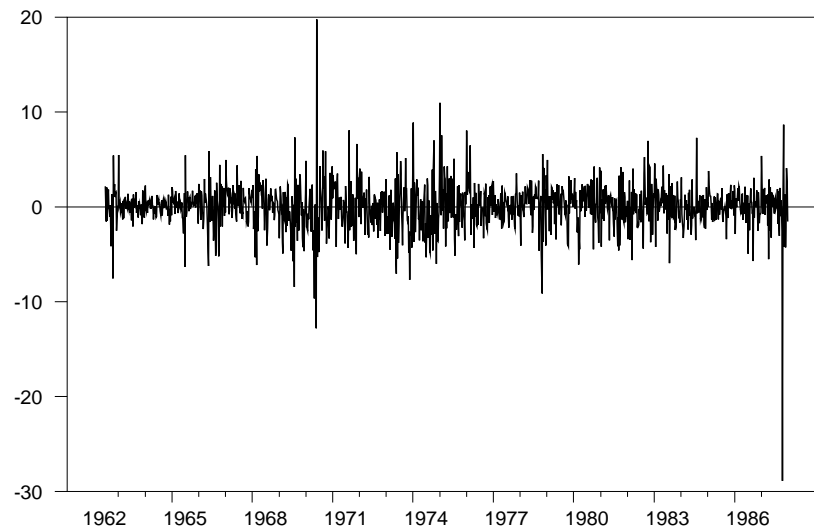


Figure 3.4: Residuals from ARCH(2)

3.6 Diagnostics for univariate ARCH models

The **GARCH** instruction defines the series `%RESIDS`, which are the $y_t - \mathbf{x}_t'\beta$. We can graph those with

```
graph(footer="ARCH(2) Residuals")
# %resids
```

to produce Figure 3.4. These are only very slightly different from the least squares residuals, and aren't directly very useful for diagnostics for the same reasons the least squares residuals weren't—the heteroscedasticity invalidates most diagnostic procedures.

Instead, diagnostics for ARCH and GARCH models are generally based upon *standardized residuals*. The ARCH model provides an estimate of the time-varying variance (h_t) of the residual process. Dividing the model residuals by $\sqrt{h_t}$ gives a set of residuals which should be:

1. Serially uncorrelated if our mean model is correct
2. Homoscedastic (actually variance 1.0) if our variance model is correct

You can save the h_t series on the **GARCH** instruction using the `HSERIES` option. In practice, you would just include that right off, but for illustration we left it off the original estimation. However, now we need it, so we change the instruction to:

```
garch(p=0,q=2,regressors,hseries=h) / y
# constant y{1}
```

We can compute and graph the standardized residuals with

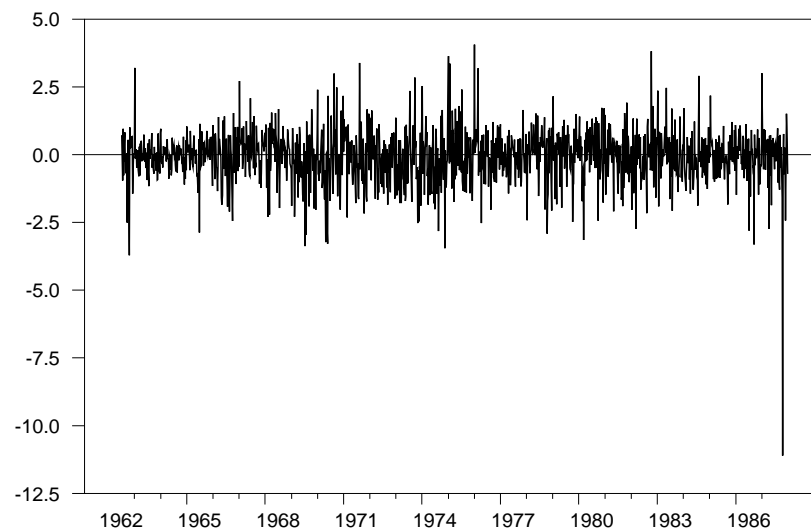


Figure 3.5: Standardized Residuals from ARCH(2)

```
set ustd = %resids/sqrt(h)
graph(footer="ARCH(2) Standardized Residuals")
# ustd
```

which produces Figure 3.5. The first indication of a problem with these is the value of nearly -11 in early 1987. If the residual process were Gaussian (which is what is assumed with the options used on **GARCH**), the probability of a value that large is effectively zero. There are several others which are around 4 in absolute value, which have probabilities of less than .0001. So the data are much fatter-tailed than the assumption of Gaussianity would suggest.

Three diagnostics to apply to the standardized residuals are:

1. Test for Gaussianity (normality)
2. Test for serial correlation
3. Test for remaining ARCH effects

Passing a test for normality isn't strictly required—the estimates are still consistent even under broader assumptions (see Section 3.8). The most commonly used test for normality is Jarque-Bera, which is included in the output from the standard **STATISTICS** instruction. While Gaussianity isn't really required, rejections in the other two will show some inadequacy in the model. We can do both the tests for serial correlation and for remaining ARCH using the **@REGCORRS** procedure, one applied to the standardized residuals, one to the squared standardized residuals:

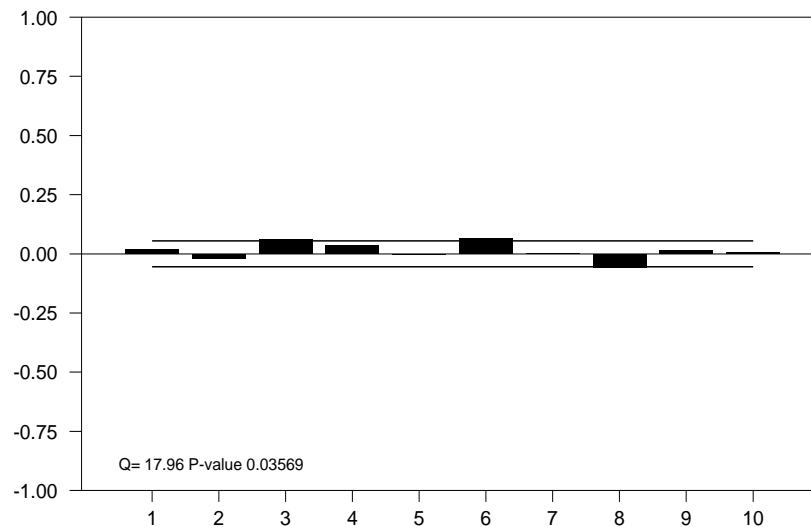


Figure 3.6: Residual Correlations of Standardized Residuals from ARCH(2)

```
stats ustd
@regcorrs (number=10,dfc=1,nocrits,qstat,$
  title="Standardized Residuals") ustd
disp "Q for Residual Serial Correlation" %qstat $
  "significance level" %qsignif
set ustd2 = ustd^2
@regcorrs (number=10,dfc=2,nocrits,qstat,$
  title="Standardized Squared Residuals") ustd2
disp "McLeod-Li for Residual ARCH=" %qstat $
  "significance level" %qsignif
```

The degrees of freedom corrections are 1 for the test for serial correlation (because of the 1 AR lag) and 2 for the test for residual ARCH (because of the 2 ARCH lags).

Statistics on Series USTD			
Weekly Data From 1962:07:10 To 1987:12:29			
Observations 1330			
Sample Mean	-0.079624	Variance	0.994408
Standard Error	0.997200	SE of Sample Mean	0.027344
t-Statistic (Mean=0)	-2.911958	Signif Level (Mean=0)	0.003652
Skewness	-1.060548	Signif Level (Sk=0)	0.000000
Kurtosis (excess)	12.157233	Signif Level (Ku=0)	0.000000
Jarque-Bera	8439.812844	Signif Level (JB=0)	0.000000
Q for Residual Serial Correlation	17.95609	significance level	0.03569
McLeod-Li for Residual ARCH=	3.43534	significance level	0.90415

The normality test is (not surprisingly) overwhelmingly rejected. The Q is barely significant at conventional levels. However, Figure 3.6 shows that none of the remaining correlation is at a position where it would be correctable by any simple change in the mean model. The point in choosing a mean model is to find a specification which eliminates *correctable* serial correlation—it's very

Table 3.5: ARCH(2) with Student- t Errors

GARCH Model - Estimation by BFGS

Convergence in 21 Iterations. Final criterion was 0.0000004 <= 0.0000100

Dependent Variable Y

Weekly Data From 1962:07:10 To 1987:12:29

Usable Observations1330

Log Likelihood-2859.8611

	Variable	Coeff	Std Error	T-Stat	Signif
1.	Constant	0.3683	0.0529	6.9668	0.0000
2.	Y{1}	0.2642	0.0286	9.2336	0.0000
3.	C	2.7200	0.2693	10.1007	0.0000
4.	A{1}	0.3526	0.0686	5.1366	0.0000
5.	A{2}	0.2551	0.0647	3.9421	0.0001
6.	Shape	4.6611	0.5877	7.9306	0.0000

difficult, in practice, to get “white” residuals at the .05 level with data sets this size. See Appendix C for more on that.

So the model seems adequate other than the clearly non-Gaussian residual process. We thus try a different conditional density.

3.7 Maximum Likelihood Estimation with Non-Gaussian Errors

The **GARCH** instruction offers two alternatives to the Normal for the conditional density of the residuals: the Student- t and the generalized error distribution (GED). The t has strictly fatter tails than the Normal, which is the limit distribution as the degrees of freedom $\nu \rightarrow \infty$. The GED family includes both fatter- and thinner-tailed densities. Of the two, the t is much more commonly used.

However, you have to be a bit careful in using these, as the formula (3.2) generates the variance of the residuals. The variance of a standard t with ν degrees of freedom is $\nu/(\nu - 2)$, so the t density has to have its scale reparameterized to give the required variance. Because the variance of a t doesn’t exist when $\nu \leq 2$, 2 is the lower limit on the degrees of freedom—the rescaled likelihood is undefined below that.

The model is estimated by adding the **DISTRIB=T** option to the **GARCH** instruction. We will also save the variances into a different series than before—this one called **HT**. The results are in Table 3.5.

```
garch(p=0,q=2,regressors,distrib=t,hseries=ht) / y
# constant y{1}
```

The new model gives a dramatic improvement to the likelihood over the ARCH(2) with Gaussian errors, now -2860 versus -2960 before. This is mainly due to the handful of very extreme residuals—the log likelihood for those with Gaussian errors is very negative, and not nearly so negative for the t . For the

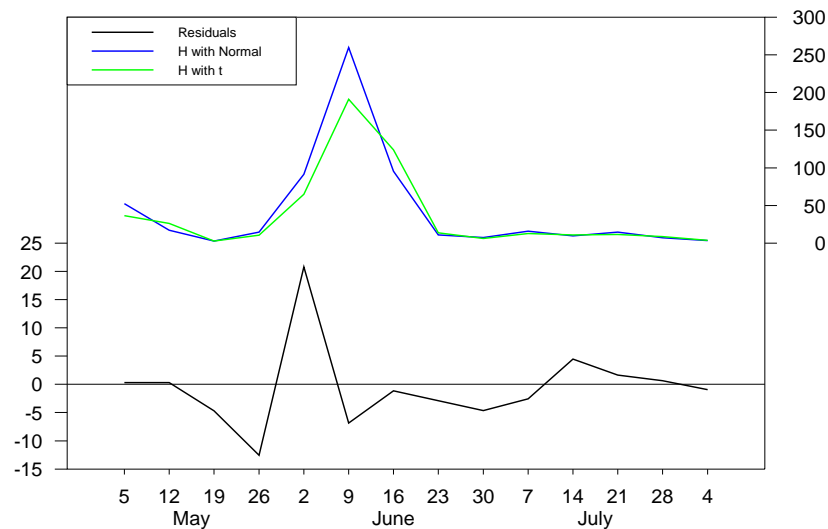


Figure 3.7: Comparison of H Estimates

vast majority of the data points, the models are almost identical. If we graph the two sets of estimated variances over the full data range, they will lie on top of each other at almost every point. The following looks just at a reduced range near one of the outliers (2 June 1970), producing Figure 3.7:

```
graph(overlay=line,ovcount=2,ovrange=.5,key=upleft,$
      klabels=||"Residuals","H with Normal","H with t"||) 3
# %resids 1970:5:1 1970:7:31
# h 1970:5:1 1970:7:31
# ht 1970:5:1 1970:7:31
```

As you can see, other than a few points near the outlier itself, the estimated variances are almost indistinguishable, at least on this scale. Note that the variance estimate for the Normal is a bit higher than for the t on the observation for the outlier. This is an adjustment made to improve the likelihood for these few very influential data points—if the big outliers are preceded by relatively large values (which is happening here), increasing the ARCH lag coefficients will give a higher value to the h estimate at the outlier, which will make the outlier look less severe. The projected variance then shoots quite a bit higher for the data point after the outlier (compared to the t), but the cost in log likelihood to the overly high variance on a relatively modest-sized residual is much less than the cost of a low variance at the outlier. Since the t likelihood isn't as sensitive to outliers, the maximizing estimates don't try as hard to explain a small number of data points.

Table 3.6: ARCH(2) with QMLE Standard Errors

GARCH Model - Estimation by BFGS

Convergence in 22 Iterations. Final criterion was 0.0000032 <= 0.0000100

With Heteroscedasticity/Misspecification Adjusted Standard Errors

Dependent Variable Y

Weekly Data From 1962:07:10 To 1987:12:29

Usable Observations1330

Log Likelihood-2959.5402

	Variable	Coeff	Std Error	T-Stat	Signif
1.	Constant	0.3959	0.0949	4.1696	0.0000
2.	Y{1}	0.2903	0.0384	7.5671	0.0000
3.	C	2.7313	0.3232	8.4494	0.0000
4.	A{1}	0.5384	0.2654	2.0290	0.0425
5.	A{2}	0.1564	0.0598	2.6160	0.0089

3.8 QMLE Estimation

The basic theory behind *Quasi-Maximum Likelihood Estimation* QMLE is described in Appendix B. This is a case where the estimation using Gaussian errors is consistent and asymptotically Normal, even if the assumption of Gaussianity is incorrect. However, the standard errors generated by the estimation routines aren't appropriate as those are done assuming the log likelihood is correct. The corrected standard errors are implemented by adding the `ROBUSTERRORS` option to the `GARCH` instruction, producing Table 3.6 (note the third line in the table):

```
garch(p=0,q=2,regressors,robusterrors) / y
# constant y{1}
```

By construction, the coefficients are the same as in Table 3.4. Note, however, that the standard errors on the ARCH coefficients, particularly the first, are *much* different. The a_1 coefficient was the one making most of the adjustment to better “predict” the variance at the outliers when estimating with the Gaussian likelihood, and once we allow for that in computing the standard errors, we find that it's much less sharply estimated than it appeared. Apparently, the larger value is good for a few extreme data points, but not most of the range.

3.9 GARCH Models

The $\text{ARCH}(m)$ process is different in behavior from the (apparently) similar $\text{MA}(q)$ process. In the $\text{MA}(q)$, correlation is zero for data separated by more than q periods. One might think that the volatility relationship in the $\text{ARCH}(m)$ would similarly cut off after m periods. However, that's not the case. The difference is that the building block of the ARCH is u_t^2 , which has a non-zero expected value, unlike the zero mean u_t used in the MA. Instead, we can rewrite the

ARCH process in terms of the zero-mean building blocks $u_t^2 - h_t$. If we look at the ARCH(2) process, we can rearrange this to

$$h_t = Eu_t^2 = c_0 + a_1(u_{t-1}^2 - h_{t-1}) + a_2(u_{t-2}^2 - h_{t-2}) + a_1h_{t-1} + a_2h_{t-2} \quad (3.4)$$

Assuming the a coefficients are positive (the model makes little sense if they aren't), then shocks which increase the variance going forward are ones where $u_s^2 - h_s > 0$, that is, the residual is bigger than one standard deviation according to the ARCH recursion.

The problem, in practice, with the low order ARCH model is that the variance processes often seem to be fairly persistent. Now, (3.4) can show a high degree of persistence if a_1 and a_2 are non-negative and sum to a number near (but less than) one, due to the behavior of the second-order difference equation in h . The problem is that those same fairly large a coefficients also show up in the transient terms $u_s^2 - h_s$. As a result, the lower order ARCH seems to require overstating the immediate impact of a shock in order to get the persistence correct. Bringing the impact down requires a longer ARCH process, but that runs into the problem faced by Engle in his original paper—even an ARCH(4) process is likely to have negative lag coefficients if run unconstrained, since they are squeezed by the need to sum to less than one to keep the process stable.⁹

The GARCH (Generalized ARCH) process of Bollerslev (1986) corrects the problem by directly including a persistence term for the variance. This very quickly supplanted the ARCH model to the extent that ARCH models themselves are rarely used except in specialized situations (like switching models) where the GARCH recursion is hard to handle.

While it's possible to have higher orders, the vast majority of empirical work with GARCH uses a 1,1 model, which means one ARCH term on the lagged squared residual and one term on the lagged variance:

$$h_t = c_0 + a_1u_{t-1}^2 + b_1h_{t-1} \quad (3.5)$$

If we use the same technique of replacing u_{t-1}^2 with itself minus h_{t-1} , we get

$$h_t = c_0 + a_1(u_{t-1}^2 - h_{t-1}) + (a_1 + b_1)h_{t-1} \quad (3.6)$$

Now, for stability of the variance we need $(a_1 + b_1) < 1$, but with the GARCH the value of that is largely decoupled from the coefficient on $(u_{t-1}^2 - h_{t-1})$ —we can have a persistent variance process with a large b_1 and small a_1 .

The one technical issue with fitting GARCH models is that, while the ARCH variance formula (3.2) *can* be computed exactly given the data and β , the variance in (3.5) depends upon an unobservable presample value for h . Different

⁹Again, a decided difference between ARCH and ARMA processes: there is nothing wrong with negative coefficients in ARMA models because they apply to data which can take either sign, while in the ARCH the recursions apply to positive data and have to produce positive values.

Table 3.7: GARCH(1,1) with Gaussian Errors

GARCH Model - Estimation by BFGS

Convergence in 23 Iterations. Final criterion was 0.0000053 <= 0.0000100

Dependent Variable Y

Weekly Data From 1962:07:10 To 1987:12:29

Usable Observations1330

Log Likelihood-2950.7953

	Variable	Coeff	Std Error	T-Stat	Signif
1.	Constant	0.4259	0.0569	7.4916	0.0000
2.	Y{1}	0.2812	0.0339	8.2849	0.0000
3.	C	0.5905	0.2761	2.1386	0.0325
4.	A	0.3674	0.0741	4.9578	0.0000
5.	B	0.6166	0.0927	6.6526	0.0000

ways of handling this give different likelihoods and thus different estimators. The RATS **GARCH** instruction uses for pre-sample h the same $\hat{\sigma}^2$ value used for presample u_s^2 in (3.3). Thus $h_1 = c_0 + (a_1 + b_1)\hat{\sigma}^2$, with all later values being computed using (3.5). If b_1 is close to one, the results *can* be sensitive to the handling of the presample,¹⁰ so if you have some concern, you can experiment with other values using the option `PRESAMPLE`, which feeds in a value to use for $\hat{\sigma}^2$.

Estimation of GARCH models is in Example 3.3. You estimate a GARCH(1,1) model with the **GARCH** instruction with options `P=1` (number of lagged variance parameters) and `Q=1` (number of lagged squared residuals), along with other options as needed. In our case:

```
garch(p=1,q=1,regressors,hseries=h) / y
# constant y{1}
```

which produces Table 3.7. The log likelihood is somewhat better than the corresponding ARCH(2) model (Table 3.4). Note that there were no real signs that the ARCH model was “wrong”—we only know that the GARCH fits a bit better because we tried both of them. That will be true with quite a few parallel forms of GARCH models.

The standardized residuals (Figure 3.8) still have some quite extreme values, so we re-estimate with t errors, with results in Table 3.8.

The standard diagnostics for the GARCH with t distributed errors are

¹⁰If we do two parallel recursions on h_t and \tilde{h}_t which differ only in the pre-sample value, then $h_t = c_0 + a_1 u_{t-1}^2 + b_1 h_{t-1}$ and $\tilde{h}_t = c_0 + a_1 \tilde{u}_{t-1}^2 + b_1 \tilde{h}_{t-1}$. Since the first two terms are common to the expressions, $h_t - \tilde{h}_t = b_1(h_{t-1} - \tilde{h}_{t-1})$ so the difference declines geometrically at the rate b_1 .

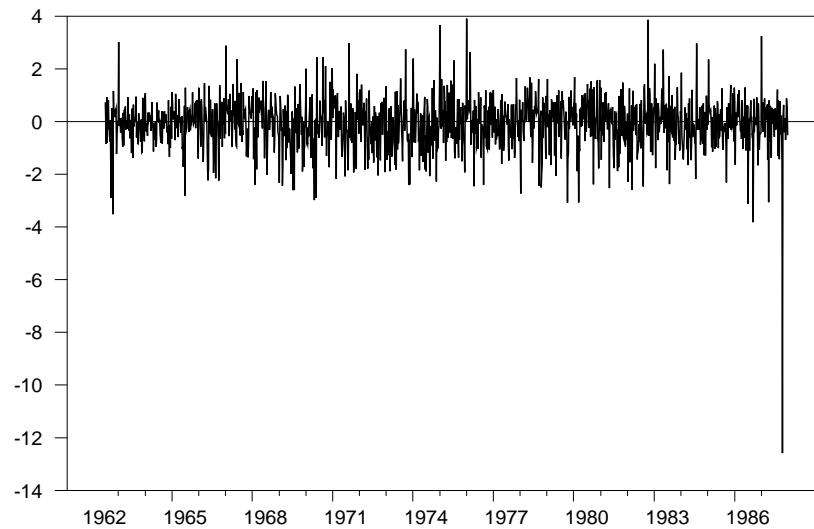


Figure 3.8: Standardized Residuals from GARCH(1,1)

Table 3.8: GARCH(1,1) with t errors

GARCH Model - Estimation by BFGS					
Convergence in 22 Iterations. Final criterion was 0.0000058 <= 0.0000100					
Dependent Variable Y					
Weekly Data From 1962:07:10 To 1987:12:29					
Usable Observations		1330			
Log Likelihood		-2841.3105			
	Variable	Coeff	Std Error	T-Stat	Signif
1.	Constant	0.3707	0.0522	7.0979	0.0000
2.	Y{1}	0.2436	0.0304	8.0259	0.0000
3.	C	0.2849	0.1170	2.4352	0.0149
4.	A	0.2030	0.0462	4.3907	0.0000
5.	B	0.7689	0.0517	14.8859	0.0000
6.	Shape	5.1053	0.6672	7.6516	0.0000

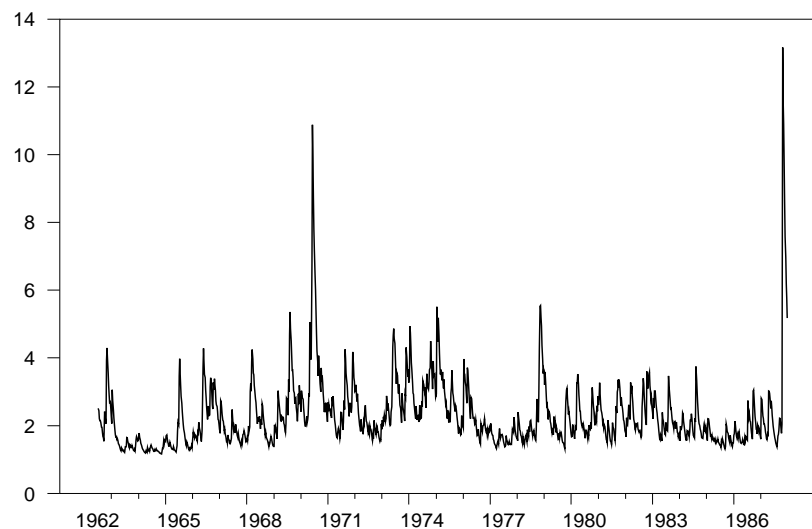


Figure 3.9: Standard Deviations from GARCH(1,1)

Statistics on Series USTD			
Weekly Data From 1962:07:10 To 1987:12:29			
Observations	1330		
Sample Mean	-0.073543	Variance	1.083294
Standard Error	1.040814	SE of Sample Mean	0.028540
t-Statistic (Mean=0)	-2.576882	Signif Level (Mean=0)	0.010077
Skewness	-2.034674	Signif Level (Sk=0)	0.000000
Kurtosis (excess)	25.653967	Signif Level (Ku=0)	0.000000
Jarque-Bera	37388.826659	Signif Level (JB=0)	0.000000
Q for Residual Serial Correlation	12.96698	significance level	0.16411
McLeod-Li for Residual ARCH=	0.46624	significance level	0.99990

which pass our standard criteria for an adequate model. You might wonder why the Jarque-Bera is higher when you use the t likelihood. That's actually to be expected. The Gaussian likelihood is so strongly dominated by the outliers that (as we saw above) it tries to adjust the coefficients to make sure it gets as high a variance as it can on those observations. As a result, when those are standardized they aren't as extreme, thus not pushing the kurtosis as high.

We are now done with the basic models. It would appear that, at least among the four that we've tried, the one which best describes the process is the GARCH(1,1) with t errors. The following graphs the estimated standard deviations from this (Figure 3.9). It's generally better to graph standard deviations rather than variances because the small number of outlier data points will dominate the scale, and you won't see the more subtle behavior in most of the range:

```
set stddev = sqrt(ht)
graph(footer="GARCH(1,1) Standard Deviation Estimate")
# stddev
```

Example 3.1 ARCH Model: Preliminaries

This selects the mean model and tests for ARCH effects in the residuals. The discussion on this starts in Section 3.2 and runs through Section 3.4.

```
open data crspw.txt
calendar(w) 1962:7:3
data(format=prn,nolabels,org=columns) 1962:07:03 1987:12:29 vw
*
* Copy the data over to y to make it easier to change the program.
*
set y = vw
*
graph(footer="Figure 3.1 Value-Weighted NYSE Returns")
# y
*
* Test for serial correlation
*
@westchotest(number=10) y
*
* Choose lag length
*
@arautolags(maxlags=10,crit=bic,table) y
*
* OLS autoregression
*
linreg y
# constant y{1}
*
* Test for residual serial correlation with a standard Q and a Q
* adjusted for possible heteroscedasticity.
*
@regcorrs(number=10,qstat,footer="Residuals from AR(1)")
@westchotest(number=10) %resids
*
* Check for ARCH effects in the residuals by regressing the squared
* residuals on their lags. The exclusion tests would seem to
* indicate that ARCH(2) is appropriate.
*
set u = %resids
set u2 = u^2
linreg u2
# constant u2{1 to 10}
exclude(title="ARCH Test: F Variant")
# u2{1 to 10}
exclude
# u2{3 to 10}
cdf(title="ARCH Test: Chi-Squared Variant") chisqr %trsquared 10
*
* Using the ARCH test procedure to do tests for all lags from 1 to 10.
*
@ARCHtest(lags=10,span=1) u
*
```

```

* McLeod-Li test
*
@McLeodLi(number=10) u
@regcorrs(number=10,nocrits,$
    title="Correlations of Squared Residuals") u2
*
* Test with generated data
*
seed 593430
set utest = %if(t>=1970:1.and.t<1976:1,%ran(3.0),%ran(2.0))
@archtest(lags=10) utest
@mcleodli(number=10) utest

```

Example 3.2 ARCH Model: Estimation

This estimates an ARCH model (Section 3.5, does diagnostics on the results (Section 3.6), re-estimates with t distributed errors (Section 3.7) and finally does estimation with Gaussian errors with a QMLE correction to the covariance matrix (Section 3.8).

```

open data crspw.txt
calendar(w) 1962:7:3
data(format=prn,nolabels,org=columns) 1962:07:03 1987:12:29 vw
*
* Copy the data over to y to make it easier to change the program.
*
set y = vw
*
* ARCH(2) with Gaussian errors
*
garch(p=0,q=2,regressors,hseries=h) / y
# constant y{1}
graph(footer="ARCH(2) Residuals")
# %resids
set ustd = %resids/sqrt(h)
graph(footer="ARCH(2) Standardized Residuals")
# ustd
*
* Diagnostics
*
stats ustd
@regcorrs(number=10,dfc=1,nocrits,qstat,$
    title="Standardized Residuals") ustd
disp "Q for Residual Serial Correlation" %qstat $
    "significance level" %qsignif
set ustd2 = ustd^2
@regcorrs(number=10,dfc=2,nocrits,qstat,$
    title="Standardized Squared Residuals") ustd2
disp "McLeod-Li for Residual ARCH=" %qstat $
    "significance level" %qsignif
*

```

```

* ARCH(2) with t-distributed errors
*
garch(p=0,q=2,regressors,distrib=t,hseries=ht) / y
# constant y{1}
*
* Comparison of H series for t and Gaussian error estimation.
*
graph(overlay=line,ovcount=2,ovrange=.5,key=upleft,$
      klabels=|"Residuals","H with Normal","H with t"|) 3
# %resids 1970:5:1 1970:7:31
# h 1970:5:1 1970:7:31
# ht 1970:5:1 1970:7:31
*
* ARCH(2) using QMLE methods. This uses the Gaussian errors but
* corrects the covariance matrix for misspecification.
*
garch(p=0,q=2,regressors,robusterrors) / y
# constant y{1}

```

Example 3.3 GARCH Model: Estimation

This estimates an GARCH model and does diagnostics on the results (Section 3.9).

```

open data crspw.txt
calendar(w) 1962:7:3
data(format=prn,nolabels,org=columns) 1962:07:03 1987:12:29 vw
*
* Copy the data over to y to make it easier to change the program.
*
set y = vw
*
* GARCH(1,1) with Gaussian errors
*
garch(p=1,q=1,regressors,hseries=h) / y
# constant y{1}
set ustd = %resids/sqrt(h)
graph(footer="GARCH(1,1) Standardized Residuals")
# ustd
*
* Diagnostics
*
stats ustd
@regcorrs(number=10,dfc=1,nocrits,qstat,$
          title="Standardized Residuals") ustd
disp "Q for Residual Serial Correlation" %qstat $
     "significance level" %qsignif
set ustd2 = ustd^2
@regcorrs(number=10,dfc=2,nocrits,qstat,$
          title="Standardized Squared Residuals") ustd2
disp "McLeod-Li for Residual ARCH=" %qstat $

```

```

    "significance level" %qsignif
*
* GARCH(1,1) with t-distributed errors
*
garch(p=1,q=1,regressors,distrib=t,hseries=ht) / y
# constant y{1}
set ustd = %resids/sqrt(ht)
stats ustd
@regcorrs(number=10,dfc=1,nocrits,qstat,$
    title="Standardized Residuals") ustd
disp "Q for Residual Serial Correlation" %qstat $
    "significance level" %qsignif
set ustd2 = ustd^2
@regcorrs(number=10,dfc=2,nocrits,qstat,$
    title="Standardized Squared Residuals") ustd2
disp "McLeod-Li for Residual ARCH=" %qstat $
    "significance level" %qsignif
*
set stddev = sqrt(ht)
graph(footer="GARCH(1,1) Standard Deviation Estimate")
# stddev

```

More on Univariate GARCH Models

4.1 Forecasting

Forecasting the relatively simple ARCH and GARCH models of Chapter 3 is straightforward. First, the mean model can be forecast exactly as it can for a model estimated by OLS—the GARCH error process has no effect on the *expected value* of the process going forward, just its variance. With the default “mean model” of the constant alone, there is nothing to forecast, but in Chapter 3, we had an AR(1) model which has non-trivial forecasts.

Each of the ARCH and GARCH models estimated in Chapter 3 had slightly different estimates for the mean model, from each other, and (even more) from the OLS estimates. There are two reasons for this, one fairly obvious, the other a bit more subtle. The obvious one is that these are models of heteroscedasticity, so the residuals in noisier periods are downweighted relative to those in quieter times. However, if you take the estimated h series and do weighted least squares with **LINREG** with the **SPREAD** option, you will *not* get the same results for the mean model as you do with the **GARCH** instruction that generated the h —while the two models seem to be separate, the presence of the lagged squared residuals in the equation for h means the derivative of the log likelihood with respect to the parameters in the mean model is much more complicated than it is for a standard GLS model. You can’t, for instance, reproduce the likelihood maximizers by estimating the mean parameters given h , then the GARCH parameters given u and repeating, the way Cochrane-Orcutt and iterated SUR work in other types of GLS.

To forecast the mean model, we need to define an equation to pass through to the forecasting routines. The easiest way to do that is to define its form first, then use that with the **EQUATION** option (rather than **REGRESSORS**) on the **GARCH** instruction. For our model (with the chosen form for GARCH), this is done with:

```
equation ar1 y
# constant y{1}
*
garch(p=1,q=1,equation=ar1,hseries=h,dist=t)
```

This is part of Example 4.1. The dependent variable and explanatory variables for the mean are all pulled out of the AR1 equation. **GARCH** then also defines

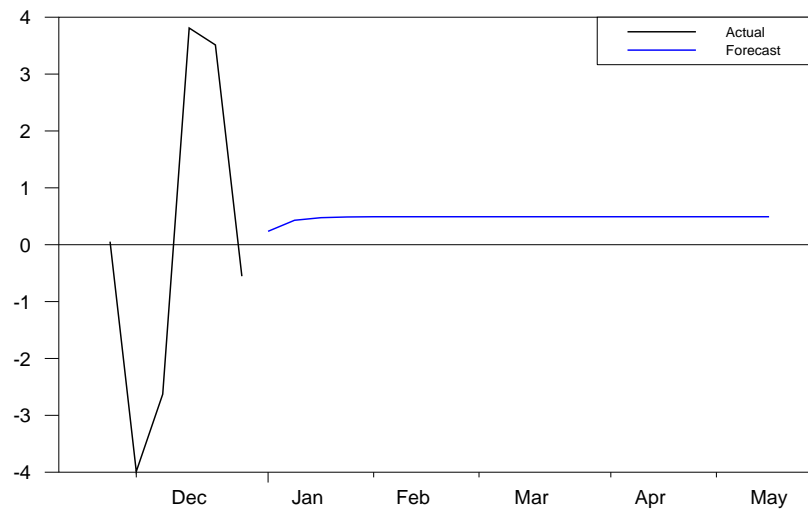


Figure 4.1: Forecasts of Mean

the coefficients for AR1 based upon its estimates. We can forecast that (here for 20 steps) and graph the forecast (Figure 4.1) with the final 6 periods of data using:

```
uforecast(equation=ar1,steps=20) yfore
graph(key=upright,klables=||"Actual","Forecast"||) 2
# y %regend()-5 %regend()
# yfore
```

`%REGEND()` is the final time period in the immediately preceding estimation, so here would be the end of the **GARCH** range. By using that, you can avoid hard-coding ranges when dealing with forecasts.

The *variance* forecasts for a basic GARCH model are computed quite simply. If we use the form:

$$h_t = c_0 + (a_1 + b_1)h_{t-1} + a_1(u_{t-1}^2 - h_{t-1})$$

the forecast for the first period out-of-sample can use values for h_{t-1} and u_{t-1}^2 from the last observed data point. After that, the expected value of u_{t-1}^2 is (by definition) h_{t-1} , so the final term drops out, leaving just the first order difference equation

$$h_t = c_0 + (a_1 + b_1)h_{t-1}$$

for periods $t = T + 2$ on. The procedure `@GARCHFORE` can be used to compute this—it takes as inputs the h and u series from the GARCH estimation to provide the sample values needed for the first forecast, and extends the input h series with the forecasts. In our example, we can forecast the variance out 20 periods (and again graph, producing Figure 4.2) with:

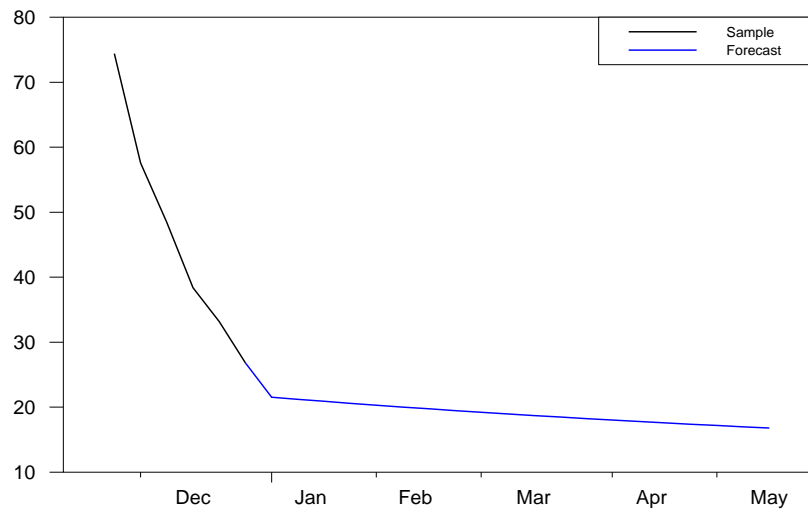


Figure 4.2: GARCH Forecasts

```
@garchfore(steps=20) h %resids
graph(key=upright,klabels=||"Sample","Forecast"||) 2
# h %regend()-5 %regend()
# h %regend() *
```

We can add upper and lower bounds for the mean model forecasts using the GARCH variances with:

```
set upper %regend()+1 %regend()+20 = yfore+2.0*sqrt(h)
set lower %regend()+1 %regend()+20 = yfore-2.0*sqrt(h)
graph(footer="Forecasts with Upper/Lower 2 s.d. bounds") 4
# y %regend()-10 %regend()
# yfore
# upper / 3
# lower / 3
```

However, this (Figure 4.3) isn't completely accurate. The calculations assume the model parameters are known. It also doesn't take into account the AR dynamics in the mean model itself, since the standard error calculations that **UFORECAST** can compute assume constant variance shocks, rather than GARCH shocks. Fully accounting for all sources of uncertainty will require simulation methods, which we will cover in Chapters 9 and 10.

4.1.1 Evaluating Variance Forecasts

As we just saw, the GARCH model can provide out-of-sample forecasts for the variance (volatility) using the estimated time series properties of the variance. The question arises: how good are these forecasts. The difficulty is that, unlike the data itself, the volatility is unobservable. An obvious choice for comparison is the squared residuals. However, as shown in Andersen & Bollerslev (1998),

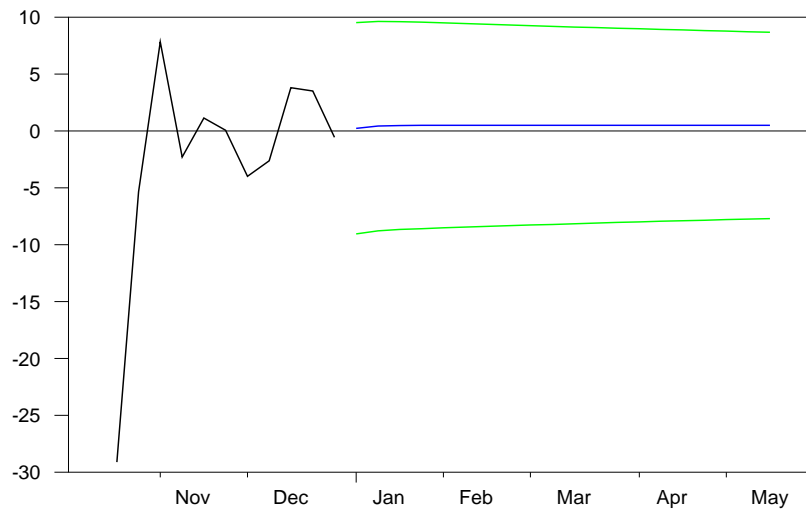


Figure 4.3: GARCH Mean Forecasts, with Bounds

the squared residuals are a (very) weak proxy for the actual volatility. Now, in that paper, the authors use high-frequency data to provide an alternative measure of volatility to compare with the GARCH forecasts, and that type of data may not always be available.

If all you have is the squared residuals proxy, Patton (2011) looks at how various error statistics fare when used to evaluate volatility forecasts. He shows that the RMSE has the desired property of being smallest (theoretically, though noisily) for the correct model, but that the MAE does not—because the squared residual is non-negative and heavily skewed, an (incorrect) model which systematically underestimates the volatility will typically produce a lower MAE. Thus, despite the fact that these have been used in the literature, they aren't a useful tool for choosing a GARCH model.

4.2 Stationarity

In the GARCH(1,1) model, it would appear (based upon intuition from standard autoregressions) that the process is stationary if and only if $b_1 + a_1 < 1$. However, the situation is quite a bit more subtle than that. Take the case of the ARCH model

$$h_t = 1 + 2u_{t-1}^2 = 1 + 2h_{t-1} + 2(u_{t-1}^2 - h_{t-1}) \quad (4.1)$$

If this described a stationary process with a finite variance, then we could solve

$$\bar{h} = 1 + 2\bar{h} + 0$$

for \bar{h} , which gives the nonsense result $\bar{h} = -.5$. So this doesn't describe a process which has a finite unconditional variance. If we forecast the process, the forecasted variances will explode, rather dramatically. However, despite

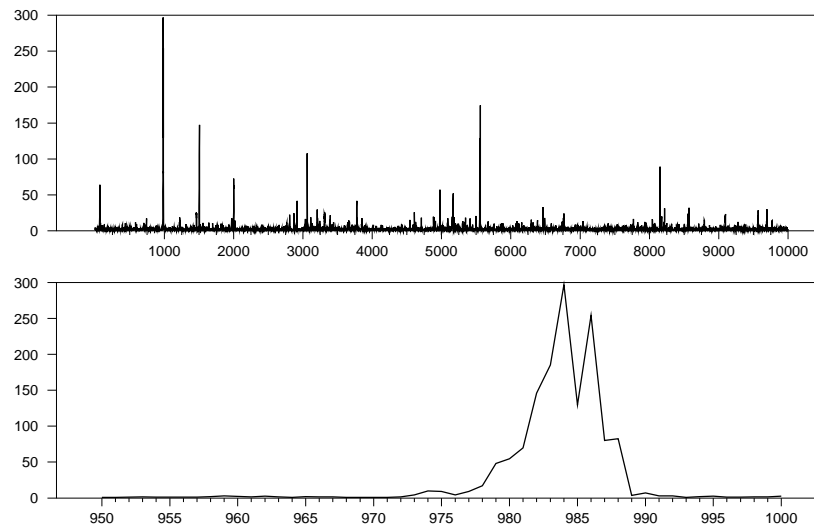


Figure 4.4: Standard Deviations from Explosive ARCH

this, it turns out that this *is* a stationary process, albeit a very strange one. The difference between this and the highly non-stationary and explosive process

$$x_t = 1 + 2x_{t-1} + \varepsilon_t$$

is that the modal (not mean) value of h_t in (4.1) is 1, regardless of the value of h_{t-1} . Thus, it's possible for the process to “reset” itself with a few near-zero residuals in a row. The following generates a sample from a more modestly explosive variance process, and graphs it (Figure 4.4) over a long range (10000 data points) and focuses in on one of the “spikes”:

```
seed 43434
set(first=1.0) h 1 10000 = 1+1.1*%ran(sqrt(h{1}))^2
set stdh = sqrt(h)
spgraph(vfields=2)
graph
# stdh
graph
# stdh 950 1000
spgraph(done)
```

On a broad scale, this appears to be near-zero for most of the range, with the occasional spike. When you zoom in, it is still mainly near-zero (on a relative basis), but the spikes are actually broader. Note, by the way, that the graph is of the standard deviations, not variances—on a variance scale, almost the entire range would appear to be zero.

This rather subtle stationarity behavior was analyzed in Nelson (1990). An alternative way of writing the GARCH(1,1) model which helps understand what

is happening is

$$h_t = c_0 + \left(b_1 + a_1 \frac{u_{t-1}^2}{h_{t-1}} \right) h_{t-1}$$

The expected value of the multiplier on h_{t-1} is (not unexpectedly) $b_1 + a_1$. However (assuming Normal residuals) roughly 2/3 of the time, the multiplier on a_1 will be less than 1. If we expand this in this form, and track the behavior of h_t along a particular path of data, we find that what we need for stability is only that:

$$E \log \left(b_1 + a_1 \frac{u_{t-1}^2}{h_{t-1}} \right) < 0$$

By Jensen's inequality, $E \log Y < \log(EY)$ as long as Y isn't a.s. constant, so, in particular, if $a_1 > 0$, the GARCH process is stationary (regardless of the distribution of u) if $a_1 + b_1 = 1$. It also is stationary even for some values where the sum *exceeds* one, particularly if b_1 is small. If b_1 is zero (as it is in (4.1)), a_1 can be as large as 3.56. Given how odd the process looks even with $a_1 = 1.1$ in Figure 4.4, the process with such a large explosive a_1 will be unrealistic.

In practice, b_1 tends to be fairly close to one, which rather significantly restricts the range of possible processes that are stationary with infinite variance. With $b_1 = .7$, even $a_1 = .4$ gives a non-stationary process.

The most interesting special case is when $a_1 + b_1 = 1$. This gives an integrated GARCH or IGARCH model. If c_0 is non-zero, this is a drifting I-GARCH model and if c_0 is zero, it's a driftless I-GARCH. The driftless I-GARCH has the interesting feature that (almost) every path converges to zero. However, if we start a driftless I-GARCH with a non-zero variance at any point in time, the variance forecasts will be flat out to infinity. While each *path* converges to zero, some take a very long time to do so, and wander a long way from zero along the way. So the distribution of the future process increasingly has a huge mass near zero, but with very long tails, maintaining a constant variance.

To estimate an IGARCH model, add the `I=DRIFT` or `I=NODRIFT` option to the rest of your options. For instance, for the drifting model:

```
garch(p=1,q=1,i=drift,regressors,dist=t) / y
# constant y{1}
```

which gives us Table 4.1. This forces the GARCH lag coefficients to sum to one. Since that isn't too different from the sum shown in Table 3.8, the fit with the constraint is almost identical.

4.3 Stability Tests

The assumption underlying the estimation of a GARCH model is that the same process generates the data throughout the range. In many cases, it's not clear

Table 4.1: IGARCH Estimates

GARCH Model - Estimation by BFGS					
Convergence in 27 Iterations. Final criterion was 0.0000064 <= 0.0000100					
Dependent Variable Y					
Weekly Data From 1962:07:10 To 1987:12:29					
Usable Observations		1330			
Log Likelihood		-2841.8787			
	Variable	Coeff	Std Error	T-Stat	Signif
1.	Constant	0.3718	0.0488	7.6186	0.0000
2.	Y{1}	0.2427	0.0280	8.6640	0.0000
3.	C	0.2257	0.0933	2.4190	0.0156
4.	A	0.2192	0.0496	4.4196	0.0000
5.	B	0.7808	0.0496	15.7419	0.0000
6.	Shape	4.8126	0.5771	8.3387	0.0000

Table 4.2: Stability Test for GARCH Model

Test	Statistic	P-Value
Joint	4.22507610	0.00
1	0.10262527	0.55
2	0.24642283	0.19
3	1.53605532	0.00
4	0.27863094	0.15
5	0.57937732	0.03
6	0.55199997	0.03

that that is the case. Certainly, series which are generated by GARCH processes which are stationary but with infinite variance don't appear to be governed by a single process, and distinguishing those from series which truly have different data generating processes in different ranges can be difficult.

Because the GARCH model is non-linear and because its log likelihood is a recursive function of the data, it's not easy to do a standard Chow-type structural break test. In the Chow test in a linear model (or a non-recursive non-linear model), the estimates from the two subsamples can be done independently of each other. That's not true with the GARCH model—the lagged h values at the start of the second subsample should come from the data in the first subsample if the split is supposed to match the behavior of the full-sample estimates.

A test that can be applied more easily is the fluctuations test from Nyblom (1989). If you have a converged set of maximum likelihood estimates, each component of the gradient of the log likelihood sums to (machine-) zero across the sample. If one of the parameters is subject to a structural break part way through the sample, one would expect to find that its gradient tended to have one sign before the break and the other sign after it, rather than having roughly equally mixed signs throughout. Nyblom's test looks at the fluctuations of those partial sums of the gradient, which form a Brownian Bridge (or

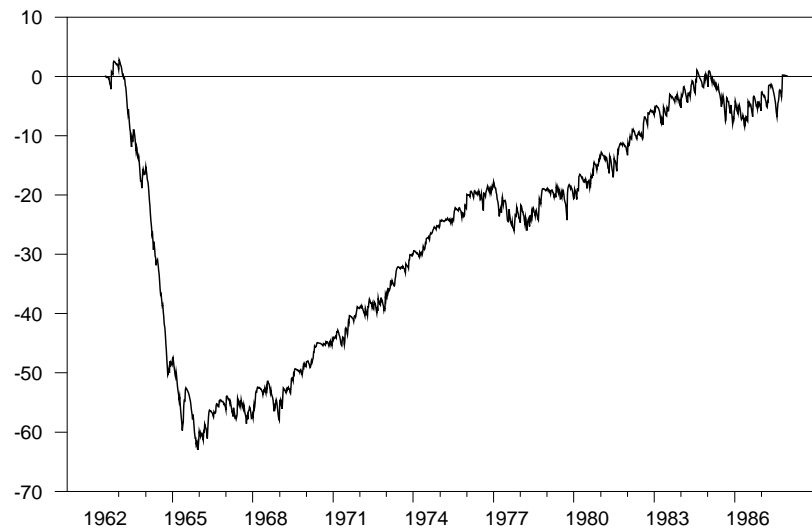


Figure 4.5: Cumulated Gradient for Variance Intercept

“tied-down” Brownian motion) under the null of a stable model, and judges whether the observed fluctuations are significantly different from that. Note that this tests for a once-and-for-all structural break based upon the time sequence. It will have *some* ability to pick up gradual changes over the course of the sample, but not outliers or systematic but scattered effects (like Monday or post-holiday effects).

The Nyblom test can be done using the procedure `@FLUX`. To apply it, we need to save the time sequences of the derivatives of the log likelihood. This is done with the `DERIVES` option on `GARCH`. This produces a `VECTOR[SERIES]`, in this case, called `DD`. `DD(1)` will be the derivatives with respect to the `CONSTANT`, `DD(2)` will be for the lagged `Y`, `DD(3)` for the variance intercept, etc. The test (results in Table 4.2) is done with

```
garch(p=1,q=1,reg,hseries=h,dist=t,derives=dd) / y
# constant y{1}
@flux(title="Stability Test for GARCH Model")
# dd
```

This does tests on each individual coefficient, and also a joint test. In this case, the results point to a major problem with the third coefficient, which is the variance intercept. In Example 4.2, we take a closer look at that with

```
acc dd(3) / fluxstat
graph(footer="Cumulated Gradient for Variance Intercept")
# fluxstat
```

which produces Figure 4.5. This is the behavior that the fluctuations test is designed to discover: the first four years of the sample really want that parameter to be lower, as the derivatives there are almost all negative. If we look back

Table 4.3: Fluctuation Test with Shorter Sample

Test	Statistic	P-Value
Joint	2.30352749	0.00
1	0.15450340	0.36
2	0.44362707	0.06
3	0.51973133	0.04
4	0.89527092	0.00
5	0.83123948	0.01
6	1.02902350	0.00

at the standard deviations generated by this model in Figure 3.9, it's not clear that there is that much of a difference between the 1962-1965 period versus other similar quiet periods like 1977 and 1985 (other than the length). That's, however, a bit misleading, since that graph is generated using the full-sample estimates, and the fluctuation statistic is saying that the variance intercept should be lower to fit the 1962-1965 period.

We can estimate the model starting in 1966 and redo the test (producing Table 4.3) with:

```
clear dd fluxstat
garch(p=1,q=1,reg,hseries=h,dist=t,derives=dd) 1966:1 * y
# constant y{1}
@flux(title="Stability Test for Adjusted GARCH Model") 1966:1 *
# dd
```

While all three GARCH coefficients have significant statistics at conventional levels, none of these are anywhere near as extreme as coefficient 3 was in Table 4.2. A check of the individual results show that much of the apparent instability is due to the huge outlier with the October 1987 crash. Since that's near the end of the sample, the fluctuation statistic can find a “break” at that point, even though it's really most likely a single outlier. Also, again, because of the size of the data set, we need to be a bit more lax with significance levels. Overall, this doesn't seem too bad, particularly if we feel we want to include the crash in the sample.

4.4 Variance Equation Dummies

In Section 4.3, we reacted to an apparent structural break in one of the parameters by shortening the sample. Given that we were losing just a few years at the start, that doesn't seem unreasonable. However, if we isolated a problem in the middle, we might be reluctant to take off half the data set to “fix” it. An alternative is to put shift dummies into the variance equation. This is done by adding the `XREGRESSORS` (or `XREG` for short) option and adding a supplementary card with the list of any variance shift variables. For instance, we can add a dummy for the block of entries before 1966 with:

Table 4.4: GARCH(1,1) with Variance Shift Dummy

GARCH Model - Estimation by BFGS

Convergence in 24 Iterations. Final criterion was 0.0000009 <= 0.0000100

Dependent Variable Y

Weekly Data From 1962:07:10 To 1987:12:29

Usable Observations1330

Log Likelihood-2823.9618

	Variable	Coeff	Std Error	T-Stat	Signif
1.	Constant	0.3794	0.0465	8.1665	0.0000
2.	Y{1}	0.2469	0.0308	8.0244	0.0000
3.	C	0.9875	0.2953	3.3438	0.0008
4.	A	0.2272	0.0496	4.5790	0.0000
5.	B	0.6265	0.0747	8.3807	0.0000
6.	DTO1966	-0.7739	0.2427	-3.1882	0.0014
7.	Shape	5.3481	0.7297	7.3288	0.0000

```

set dto1966 = t<1966:1
garch(p=1,q=1,reg,xreg,hseries=h,dist=t,derives=dd) / y
# constant y{1}
# dto1966

```

This gives us Table 4.4. When we compare this with the estimates *without* the dummy in Table 3.8, we can see that the persistence has dropped a bit with the dummied model, from .97 to .85. This is what we would expect, since an unmodeled break will tend to make the variance appear more persistent.

This data set has one *extremely* large outlier at 1987:10:20. In other types of models, we might be tempted to simply “dummy” this out. In a GARCH model, there are two places where we could put a dummy: in the mean model, or in the variance model. If you put it in the mean model, it will have the expected behavior of pushing the residual at that point towards zero. The problem is that, if the GARCH model is correct, the outlier should provide a “natural experiment” for the period afterwards—the variance should be high after that, and should be high for quite a while. But with the mean shift dummy, the variance never gets the pop from a large residual. Instead, you can put the dummy in the variance equation. If it weren’t for the recursive definition of h , the effect of this would be to make h_t at the dummied point equal (exactly) to the squared residual (a one-data-point estimator for the variance). It won’t be *quite* that because of the effect on the h function for time periods after, but should be at least similar.

We can add the crash period dummy to the variance equation with:

```

set crash = t==1987:10:20
garch(p=1,q=1,reg,xreg,hseries=h,dist=t,derives=dd) / y
# constant y{1}
# dto1966 crash

```

Table 4.5: GARCH(1,1) with Variance Shift and Outlier Dummies

GARCH Model - Estimation by BFGS

Convergence in 34 Iterations. Final criterion was 0.0000025 <= 0.0000100

Dependent Variable Y

Weekly Data From 1962:07:10 To 1987:12:29

Usable Observations1330

Log Likelihood-2816.1871

	Variable	Coeff	Std Error	T-Stat	Signif
1.	Constant	0.3749	0.0499	7.5126	0.0000
2.	Y{1}	0.2495	0.0310	8.0594	0.0000
3.	C	1.0305	0.2797	3.6844	0.0002
4.	A	0.2327	0.0485	4.7947	0.0000
5.	B	0.5993	0.0760	7.8805	0.0000
6.	DTO1966	-0.8038	0.2266	-3.5472	0.0004
7.	CRASH	268.4294	214.7249	1.2501	0.2113
8.	Shape	6.1848	0.9706	6.3721	0.0000

which produces Table 4.5. The t -statistic on the CRASH dummy is somewhat misleading—the likelihood ratio statistic on the addition is about 14, which is highly significant. Note that, despite ridding the sample of the need to explain the outlier, the shape parameter for the t error process is still quite low, so even without the largest outlier there is still strong evidence of fat-tailed conditional residuals.

Dummies from Pre-Testing

In Section 3.1, we looked at the ICSS variance break analysis as an alternative to the GARCH model. There’s a thread in the literature that has been using ICSS as a “pre-test” for variance breaks and uses the results to generate variance shift dummies for a GARCH. For a number of reasons, this is not a good idea—the two methods are really substitutes not complements.

1. Series do not have breaks; models have breaks. You can’t simply transfer information about breaks found in one model to another. (The “model” underlying the ICSS test has independent variables with a fixed variance, which is quite different from a GARCH model.)
2. ICSS is looking for sharp breaks in the variance, that is, the variance through T_i takes one value, and from $T_i + 1$ takes another. On the other hand, a variance shift dummy in a GARCH creates an “innovational” break in the variance process—if the GARCH process is fairly persistent, it may take anywhere from 10 to 30 entries for the variance to converge to the new long-run level. Thus, even if a GARCH with a variance shift is the true DGP, the ICSS procedure is likely to mistime it.
3. Again, assuming that a GARCH process is correct, the ICSS procedure will generally find spurious variance “breaks” because a GARCH, by its nature, generates periods of high variance following outliers.

4. If you generate dummies using the pre-test procedure and then add them as shifts in the GARCH, they will always appear much more significant statistically than they are. First, they were chosen using a search, which itself biases the results. Second, unlike a GARCH model, which has to “predict” variances using past information only, the ICSS dummies get to “look ahead”—for instance, if there’s an outlier (which produces the expected period of high variance following), GARCH will not be able to explain the initial outlier, but, with the benefit of looking at the whole data set, ICSS will “explain” it by saying it’s the start of this new high variance segment that it has found. Obviously, this isn’t very useful for using the model out-of-sample since we would never know what regime will be in effect next period.

The fluctuations test from Section 4.3 offers a superior alternative for looking for breaks that can be modeled as variance shift dummies, since that would show up as a break in the variance constant.

4.5 GARCH-M

If our dependent variable is excess returns from stocks or bonds, the CAPM in particular, or investment theory in general, would predict that the mean return would be higher in volatile periods to compensate holders for the risk. Engle et al. (1987) (ELR for short) proposed the ARCH-M model as a way to incorporate a function of the variance into the mean model.¹ They speculated that the apparent ability of some macroeconomic factors to help “predict” stock or bond returns might be due to their correlations with the (unobservable) volatility. Since GARCH models provide us a computable value for the volatility, we can add that directly into the mean and see whether the factors were simply proxies for volatility.

ELR showed that the functional form in the variance that is needed depends upon assumptions about the utility function and distribution of the residuals, and on various specifics of the markets for the risky and riskless assets. With what is likely to be a non-linear function of unknown form, the best approach is to linearize it. However, if the exact function is $f(h_t)$,² then it can also be written as a function of any other deterministic function of h , such as \sqrt{h} or $\log h$. Which form is best for linearization isn’t clear without knowing more about the structure of f .

¹The paper was originally written in 1982, before Bollerslev introduced the GARCH model, so it is entirely done in terms of ARCH.

²ELR use h as the standard deviation, rather than variance. We’ll use the more common modern notation where h is the variance.

Table 4.6: GARCH-M Model with Variance

GARCH Model - Estimation by BFGS

Convergence in 22 Iterations. Final criterion was 0.0000053 <= 0.0000100

Dependent Variable Y

Weekly Data From 1966:01:04 To 1987:12:29

Usable Observations1148

Log Likelihood-2541.9920

	Variable	Coeff	Std Error	T-Stat	Signif
1.	Constant	0.2874	0.1086	2.6455	0.0082
2.	Y{1}	0.2557	0.0304	8.4064	0.0000
3.	GARCH-V	0.0134	0.0182	0.7337	0.4631
4.	C	0.7930	0.3262	2.4309	0.0151
5.	A	0.1961	0.0537	3.6488	0.0003
6.	B	0.6798	0.0932	7.2917	0.0000
7.	Shape	5.7034	0.8595	6.6354	0.0000

The simplest form of GARCH-M model is

$$h_t = c_0 + a_1 u_{t-1}^2 + b_1 h_{t-1}$$

$$y_t = x_t' \beta + \gamma g(h_t) + u_t$$

where $g(h_t)$ is the chosen form of h about which the variance effect is linearized. Because of linearization effects, an intercept should be included in x , even if Ey_t is theoretically zero if h is zero. The equations are written in the somewhat unnatural order for a reason—when this is implemented h must be computed first. To compute h_1 , RATS again uses the pre-sample value from (3.3) for both u_{t-1}^2 and h_{t-1} .

The RATS **GARCH** instruction requires a model linear in (already constructed) variables for the mean (and also for any variance shifts done using the **XREGRESSORS** option). However, **GARCH** defines the (internal) series **%GARCHV** which has the recursively generated values for h as computed during a function evaluation. Adding **%GARCHV** to the list of regressors gives a GARCH-M with $g(h_t) = h_t$.

In Example 4.3, we restrict the estimates to the subsample starting in 1966 that was chosen using the fluctuations test in Section 4.3. The following estimates a GARCH-M with the variance as regressor:

```
garch(p=1,q=1,reg,hseries=h,dist=t) 1966:1 * y
# constant y{1} %garchv
```

producing Table 4.6. The variance term in the mean is labeled **GARCH-V**. It has the expected sign, but isn't statistically significant—which is a common result for such models.

If you want some form other than the variance as the value around which to linearize, you can use the option **HADJUST**. **HADJUST** and its companion **UADJUST**

give you access to the internal calculations during the recursive evaluation of the likelihood. With `HADJUST`, you provide a formula (a function of the entry number `T`) that you want calculated immediately after the variance for `T` has been computed—`HADJUST` means that you want to adjust something based upon `H`. `UADJUST` is similar, but is an adjustment made after the *residual* `U` has been computed. To allow for an “M” effect, the variance is always computed first.

To estimate a GARCH-M model with the conditional standard deviation (rather than variance) as the mean shifter, you can do the following:

```
set gh = sqrt(h)
garch(p=1,q=1,reg,hseries=h,hadjust=(gh=sqrt(h)),dist=t) 1966:1 * y
# constant y{1} gh
```

This uses `GH` as the “M” variable in the mean equation. The **SET** instruction initializes `GH` to the h series generated by a previous **GARCH** instruction. This series has to be given *some* values or the initial scan of the variables by the **GARCH** instruction will come up with an empty estimation range, and giving them values that at least approximate what they will be when the full model is estimated helps in computing the pre-sample values.

On the **GARCH** instruction itself, the two key options are the `HSERIES` and the `HADJUST`. We’ve been using `HSERIES` all along to get the final time series of estimated variances; however, the target series for the options is also filled in at each function evaluation. Each time the model is evaluated, once the variance for entry `T` has been computed it is saved into `H` series given by the `HSERIES` option. At that point, the `HADJUST` option is applied— $GH(T)$ is set equal to $\sqrt{H(T)}$ (the T ’s are implied in a formula). After the H -based adjustment is made, the residuals for entry `T` are computed using the value of `GH` just generated.

The results of the adjusted “M” model are in Table 4.7. The results are *slightly* better.

Note well that, if the GARCH properties of the data are weak, you can’t estimate an “M” effect. Weak GARCH properties means that the variance is nearly constant, so you can’t really distinguish between the intercept and the desired function of the variance that you put in the mean model.

4.6 Alternative Variance Models

It was recognized rather quickly that the functional form for the GARCH model is somewhat arbitrary. We can rewrite it as

$$h_t = c_0 + (a_1 + b_1)h_{t-1} + a_1(u_{t-1}^2 - h_{t-1})$$

which has a persistence term and a correction term, where the latter is mean zero with positive contributions for more extreme values for u_{t-1} . An alterna-

Table 4.7: GARCH-M model with standard deviation

GARCH Model - Estimation by BFGS					
Convergence in 24 Iterations. Final criterion was 0.0000003 <= 0.0000100					
Dependent Variable Y					
Weekly Data From 1966:01:04 To 1987:12:29					
Usable Observations		1148			
Log Likelihood		-2541.6384			
	Variable	Coeff	Std Error	T-Stat	Signif
1.	Constant	0.0492	0.2767	0.1779	0.8588
2.	Y{1}	0.2545	0.0303	8.4028	0.0000
3.	GH	0.1407	0.1251	1.1241	0.2610
4.	C	0.7425	0.2992	2.4821	0.0131
5.	A	0.1897	0.0508	3.7357	0.0002
6.	B	0.6942	0.0860	8.0736	0.0000
7.	Shape	5.7134	0.8710	6.5597	0.0000

tive would be an additive model in the standard deviation, using the absolute value as the correcting term:

$$\sigma_t = c_0 + b_1\sigma_{t-1} + a_1|u_{t-1}|$$

However, unlike the recursion in the variance, $E|u_{t-1}|$ isn't equal to σ_{t-1} —it's $\sigma\sqrt{2/\pi}$ for the Normal, and takes a different value for a t or any other conditional distribution. Thus, the adjustment to the persistence-correction form isn't as simple as in the variance form—with the same method of rewriting as used before:

$$\sigma_t = c_0 + \left\{ b_1 + a_1 E \frac{|u_{t-1}|}{\sigma_{t-1}} \right\} \sigma_{t-1} + a_1 \sigma_{t-1} \left\{ \frac{|u_{t-1}|}{\sigma_{t-1}} - E \frac{|u_{t-1}|}{\sigma_{t-1}} \right\}$$

A more useful alternative is the exponential GARCH or EGARCH of Nelson (1991). We'll describe Nelson's complete model later, but for now we'll use a simpler form for comparison:

$$\log h_t = c_0 + b_1 \log h_{t-1} + a_1 |u_{t-1}| / \sqrt{h_{t-1}} \quad (4.2)$$

This way of writing this is a bit misleading, since the b_1 and a_1 have different roles than they do in (3.5). Note that the distribution of the correction term is now independent of h_{t-1} —if we center that term, it only shifts the variance intercept, not the persistence term:

$$\log h_t = \left\{ c_0 + a_1 E |u_{t-1}| / \sqrt{h_{t-1}} \right\} + b_1 \log h_{t-1} + a_1 \left\{ |u_{t-1}| / \sqrt{h_{t-1}} - E |u_{t-1}| / \sqrt{h_{t-1}} \right\}$$

Thus in the EGARCH, persistence is determined by the b_1 coefficient itself. And it does not have the peculiar stationarity behavior of the standard GARCH—if $b_1 \geq 1$, it's non-stationary. It has two major disadvantages though:

1. There is no closed-form forecasting calculation for the variance because of the way the variance enters the correction term. Forecasting requires some type of simulation.

2. Manipulations of it (to, for instance, compute the unconditional variance) are specific to a given conditional error distribution. This is due to the presence of the $E|u_{t-1}|$.

To allow for fatter tails than the Normal, Nelson proposed use of the Generalized Error Distribution (GED). This is described in greater detail in Appendix A.8. This is a family which includes the Normal as a special case (when the shape parameter is 1)—the t do that except in the limit. It has members which are both thinner-tailed members than the Normal (when the shape is less than 1) and thicker-tailed (when the shape is bigger than 1). Note that the mapping between shape and kurtosis is the opposite of that in the t , where smaller shapes (degrees of freedom) mean fatter tails. The advantage of the GED is that it is much easier to handle analytically than the t , as it can be transformed easily into a gamma distribution. It *does* have the disadvantage compared to the t that even the fatter-tailed members have finite moments for all powers, so they can't match the fairly thick tails of the t with low degrees of freedom. It also can have numerical problems because the density isn't differentiable at 0 when the shape is bigger than 1. Note well that the use of the GED is not an essential part of the E-GARCH model—they are unrelated ideas which were introduced in the same paper.

To estimate an EGARCH model of the form (4.2), add the `EXP` option to the **GARCH** instruction. To use a conditional GED distribution (rather than the Normal) also add the `DIST=GED` option:

```
garch(exp,p=1,q=1,reg,hseries=hx,dist=ged) 1966:1 * y
# constant y{1}
```

The output from this is in Table 4.8. The standard GARCH over the same sample is in Table 4.9. The latter seems to fit better when we look at the log likelihoods—this is primarily because of the ability of the t to deal with the extreme outliers. The fit of the two is very similar if we restrict the sample to eliminate the crash in 1987.

For most data points, the estimated variances from the two models are almost identical, and graphed over a long range of time, the two lines would sit almost on top of each other. The following does a comparison over a much shorter range (just 1970, which has the second largest outlier), producing Figure 4.6:

```
set sdx = sqrt(hx)
set sd = sqrt(h)
graph(footer="Standard Deviations from EGARCH and GARCH", $
  key=below, klabels=| "EGARCH", "GARCH" |) 2
# sdx 1970:1 1970:52
# sd 1970:1 1970:52
```

Note that the variance in the GARCH actually reacts more than the EGARCH to the outlier, even though the EGARCH has an exponential effect in the tails

Table 4.8: EGARCH without Asymmetry

GARCH Model - Estimation by BFGS

Convergence in 27 Iterations. Final criterion was 0.0000054 <= 0.0000100

Dependent Variable Y

Weekly Data From 1966:01:04 To 1987:12:29

Usable Observations1148

Log Likelihood-2563.1260

	Variable	Coeff	Std Error	T-Stat	Signif
1.	Constant	0.3830	0.0637	6.0172	0.0000
2.	Y{1}	0.2738	0.0308	8.8909	0.0000
3.	C	-0.0114	0.0781	-0.1457	0.8841
4.	A	0.3693	0.0808	4.5685	0.0000
5.	B	0.8444	0.0660	12.7867	0.0000
6.	Shape	1.6120	0.0741	21.7402	0.0000

Table 4.9: GARCH Model with Shortened Sample

GARCH Model - Estimation by BFGS

Convergence in 22 Iterations. Final criterion was 0.0000014 <= 0.0000100

Dependent Variable Y

Weekly Data From 1966:01:04 To 1987:12:29

Usable Observations1148

Log Likelihood-2542.2526

	Variable	Coeff	Std Error	T-Stat	Signif
1.	Constant	0.3527	0.0644	5.4786	0.0000
2.	Y{1}	0.2565	0.0315	8.1509	0.0000
3.	C	0.8280	0.3455	2.3965	0.0166
4.	A	0.2004	0.0550	3.6457	0.0003
5.	B	0.6702	0.0956	7.0089	0.0000
6.	Shape	5.6681	0.8533	6.6428	0.0000

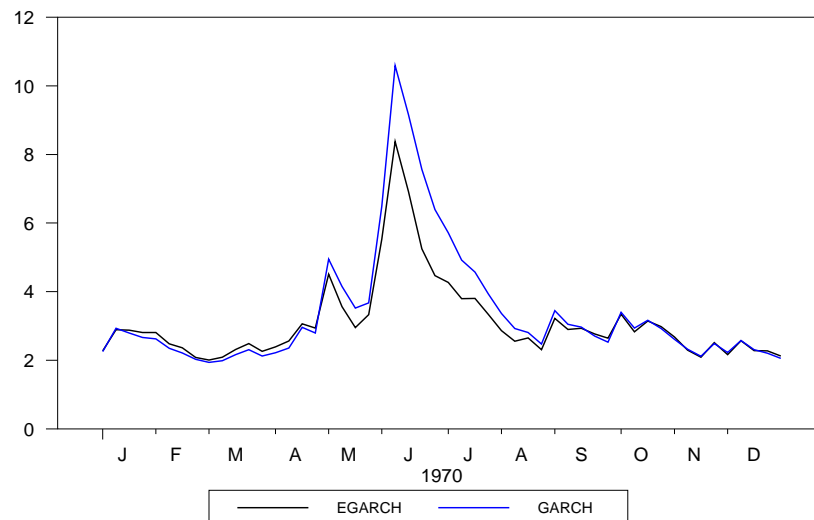


Figure 4.6: Standard Deviations from EGARCH and GARCH

rather than quadratic. In practice, though, the point at which the exponential dominates the quadratic depends upon the estimated parameters, especially the values of b_1 . One way to compare the behavior of the models is by means of the *news impact curve* of Engle & Ng (1993). This measures how the variance reacts to different sized shocks. To allow comparison of models with very different functional forms, this is computed at the unconditional variance. Since the two models will estimate somewhat different unconditional variances, we will use the one from the standard GARCH as the benchmark.

First, we need to set up a grid series for standardized residuals. This creates a 201 point series named `ETA` running from -10 to 10. This is an extremely wide range, but, as we've seen, the standardized residuals actually get quite large in this data set.

```
@gridseries(from=-10.0,to=10.0,n=201) eta
```

The following (re-) estimates the GARCH models and pulls the required values out of the vector `%BETA` of estimated coefficients. `%NREGMEAN` is very handy when working with the **GARCH** instruction when you have a mean model: because the mean parameters come first in the parameter vector, it allows you to skip over them easily to get to the GARCH parameters no matter how large the mean model is. `%PARMSPOKE` takes information from a vector (here the standard `VECTOR` of parameters `%BETA`) and puts them into a parameter set organized to match the estimated model—here the standard `GARCH(1,1)` with a shape parameter has the variance constant, the “A”, the “B” and the shape (degree of freedom for the t).

```
dec vect bmean(%nregmean)
nonlin(parmset=garchparms) bmean c0 a1 b1 shape
compute %parmspoke(garchparms,%beta)
```

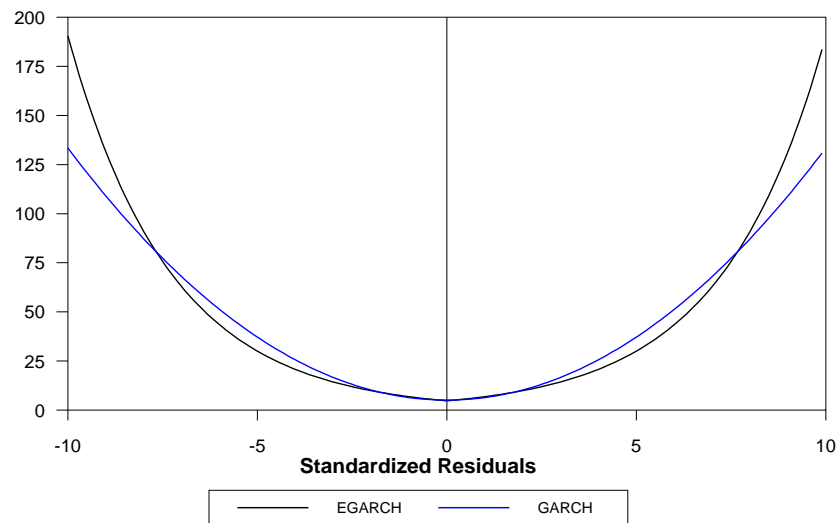


Figure 4.7: News Impact Curves

This computes the unconditional variance (HBAR) and the news impact curve, which substitutes HBAR in the GARCH variance equation wherever h_{t-1} occurs. (This is done in terms of standardized residuals, which is why we have the final HBAR multiplier).

```
compute hbar=c0/(1-a1-b1)
set gimpact 1 201 = c0+b1*hbar+a1*eta^2*hbar
```

The analogous calculation for the EGARCH is done with

```
garch(exp,p=1,q=1,reg,dist=ged) 1966:1 * y
# constant y{1}
dec vect bmean(%nregmean)
nonlin(parmset=garchparms) bmean c0 a1 b1 shape
compute %parmspoke(garchparms,%beta)
set eimpact 1 201 = exp(c0+b1*log(hbar)+a1*abs(eta))
```

You might notice that this is simpler than the formula given in Engle & Ng (1993). That's because **GARCH** doesn't subtract off the expected value—the formula in the article requires adding that back in.

The graphical comparison can be done with

```
scatter(style=lines,key=below,klabels=||"EGARCH","GARCH"||,$
  footer="News Impact Curves",hlabel="Standardized Residuals") 2
# eta eimpact
# eta gimpact
```

which generates Figure 4.7.

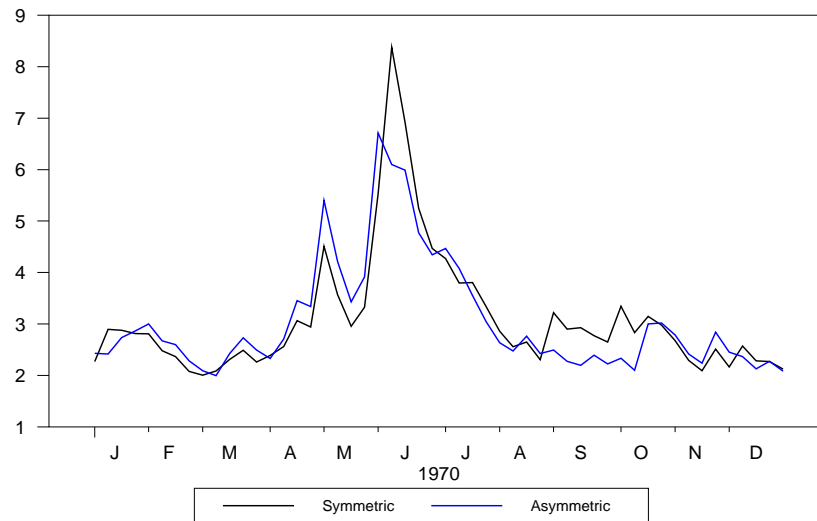


Figure 4.8: Standard Deviations from EGARCH With and Without Asymmetry

4.7 Asymmetry

The GARCH model so far can explain two important observations from analysis of financial data: that the unconditional distribution is (much) fatter-tailed than the Normal, and that large residuals tend to cluster. However, it can't deal with a third, which is that (particularly for stock market data) negative shocks tend to produce greater subsequent volatility than positive shocks.

Nelson's EGARCH model actually incorporated asymmetry from the start, so the model described in (4.2) does not match his. However, we have separated the exponential recursion from the asymmetry as there may be situations where the asymmetry is not needed. Allowing for asymmetric effects, Nelson's model can be written:

$$\log h_t = c_0^* + b_1 \log h_{t-1} + a_1 \left\{ |u_{t-1}| / \sqrt{h_{t-1}} - E |u_{t-1}| / \sqrt{h_{t-1}} \right\} + d_1 u_{t-1} / \sqrt{h_{t-1}} \quad (4.3)$$

There are now two (mean zero) correction terms, one based upon the absolute value, so it is symmetrical, the other being signed. If d_1 is negative (which would be expected), negative shocks have a greater effect on the variance than positive ones. The coefficient on the normalized absolute value is $a_1 + d_1$ for a positive shock and $a_1 - d_1$ for a negative one. The **GARCH** instruction estimates the asymmetric EGARCH model in the slightly simpler form of

$$\log h_t = c_0 + b_1 \log h_{t-1} + a_1 |u_{t-1}| / \sqrt{h_{t-1}} + d_1 u_{t-1} / \sqrt{h_{t-1}}$$

which differs from (4.3) only in the variance intercept. To do this, add the **ASYMMETRIC** option to the **GARCH** instruction:

```
garch(exp, asymmetric, p=1, q=1, reg, hseries=hxa, dist=ged) 1966:1 * y
# constant y{1}
```

Table 4.10: EGARCH with Asymmetry

GARCH Model - Estimation by BFGS

Convergence in 29 Iterations. Final criterion was 0.0000083 <= 0.0000100

Dependent Variable Y

Weekly Data From 1966:01:04 To 1987:12:29

Usable Observations1148

Log Likelihood-2549.3872

	Variable	Coeff	Std Error	T-Stat	Signif
1.	Constant	0.3198	0.0639	5.0049	0.0000
2.	Y{1}	0.2838	0.0314	9.0365	0.0000
3.	C	0.0381	0.0736	0.5175	0.6048
4.	A	0.2925	0.0691	4.2297	0.0000
5.	B	0.8396	0.0611	13.7507	0.0000
6.	D	-0.1692	0.0402	-4.2091	0.0000
7.	Shape	1.5327	0.0705	21.7463	0.0000

producing Table 4.10. The asymmetry coefficient is labeled D , and is clearly significant and has the expected sign.

We can revisit the estimated standard deviations from 1970 and compare those computed with a symmetric EGARCH and those with asymmetry producing Figure 4.8:

```

set sdx = sqrt(hx)
set sdx_a = sqrt(hxa)
graph(footer="Std Deviations from EGARCH With and Without Asymmetry", $
    key=below, klabels=|| "Symmetric", "Asymmetric" ||) 2
# sdx 1970:1 1970:52
# sdx_a 1970:1 1970:52

```

The middle part of 1970 had two fairly large negative shocks in April and May with a much larger positive shock at the beginning of June. Because of the negative shocks, the asymmetric variance moves up above the symmetric estimate until the period after the big positive shock, when it reacts less sharply than the symmetric model.

Asymmetry can also be added to the standard GARCH model. This was proposed in Glosten et al. (1993) and is known as the GJR model. Unfortunately, Nelson and GJR chose different sign conventions—in GJR, a positive coefficient means that *negative* residuals increase the variance. Since the two specifications are sufficiently different, and the sign conventions are well-established, the `ASYMMETRIC` option follows practice, so the D coefficient on a GJR model would be expected to be positive, rather than negative.

With asymmetry, the GJR model is:

$$h_t = c_0 + b_1 h_{t-1} + a_1 u_{t-1}^2 + d_1 u_{t-1}^2 I(u_{t-1} < 0) \quad (4.4)$$

where $I(x)$ is a 0-1 indicator for condition x . Thus, the multiplier on u_{t-1}^2 is a_1 for positive u_{t-1} and $a_1 + d_1$ for negative. If we center the two correction terms,

Table 4.11: GJR GARCH Model

GARCH Model - Estimation by BFGS

Convergence in 23 Iterations. Final criterion was 0.0000082 <= 0.0000100

Dependent Variable Y

Weekly Data From 1966:01:04 To 1987:12:29

Usable Observations1148

Log Likelihood-2531.0500

	Variable	Coeff	Std Error	T-Stat	Signif
1.	Constant	0.2942	0.0636	4.6284	0.0000
2.	Y{1}	0.2809	0.0315	8.9295	0.0000
3.	C	0.8482	0.3102	2.7340	0.0063
4.	A	0.0625	0.0316	1.9763	0.0481
5.	B	0.6613	0.0879	7.5228	0.0000
6.	D	0.2704	0.0834	3.2403	0.0012
7.	Shape	6.2585	1.0234	6.1155	0.0000

we get

$$h_t = c_0 + (b_1 + a_1 + d_1/2) h_{t-1} + a_1(u_{t-1}^2 - h_{t-1}) + d_1(u_{t-1}^2 I(u_{t-1} < 0) - h_{t-1}/2)$$

so the persistence coefficient is $(b_1 + a_1 + d_1/2)$.

To estimate the GJR model with RATS, just add the `ASYMMETRIC` option to the **GARCH** instruction:

```
garch(asyymmetric,p=1,q=1,reg,hseries=ha,dist=t) 1966:1 * y
# constant y{1}
```

This produces Table 4.11. This is better than the corresponding EGARCH, again, due to the fatter tails permitted by the t . In comparing each of the asymmetric models with their symmetric cousin, note that in each case, the shape parameter isn't as extreme, so the asymmetry does appear to be helping to predict at least some of the variance in the neighborhood of the outliers.

4.8 Rolling Samples

The use of “rolling” sample estimates has become common in the literature. Unfortunately, in many cases, the use of these is not well-motivated, and the results presented without any allowance for the effects of the sampling distribution. (Different samples will produce different results, even if the data-generating process is unchanging).

In volatility modeling, one use of these that *does* make sense is in simulated out-of-sample testing. If a model is being used to make predictions or decisions such as portfolio allocations, it isn't a good idea to “back test” it using model estimates through the full data set, since those will have access to the very data you're trying to use to validate the (simulated) out-of-sample performance. Instead, a better procedure is to pick a hold-back period that will be used for

evaluation, estimate the model using only the data prior to that, and look at the performance after the end of the estimation range. You can then add the next entry to the observation window, and look at the performance of the periods beyond that, etc. The `GARCHBACKTEST.RPF` example in the *User's Guide* is an example of this.

However, the standard “rolling window” analysis uses a fixed “window” of data and re-estimates the model for different windows across the data range. So if the window width is 200, the first set of estimates will be using entries 1 to 200, the second entries 2 to 201, etc. Example 1.1 did a very simple example of a rolling window estimator for the volatility using a 52 week flat average of the squares of the data. As we saw there, this can lead to some very wild swings in the volatility estimates as extra data points slide into and out of the window.

The application of rolling windows to GARCH models is a good example of a use of these that isn't well-motivated. What exactly is the point? If the goal is to estimate the “end-of-window” variance (or covariance if it's a multivariate model), what does the rolling window accomplish? Given the parameters of a typical (persistent) GARCH model, the end-of-sample estimator for the variance really only depends upon about the previous 50 data points—as it's an exponentially declining moving average. (If it *isn't* persistent, it will be even less than that). Thus, the only thing the rolling window is doing changing the variance estimates by re-estimating the model with a (usually quite) reduced set of data. GARCH estimates with small data sets have all kinds of problems. In particular, a short window may have only one or sometimes even no real episodes which allow the persistence of volatility to be estimated. (This would be like trying to estimate a linear regression with one explanatory variable that is nearly constant). Thus, the rolling window analysis is injecting a substantial amount of sampling error. And, the GARCH model itself naturally creates periods of higher and lower variance, so the “instability” that the rolling windows are designed to handle is something that doesn't require “handling”.

Example 4.1 Univariate Model Forecasting

This does forecasts of the mean and of the variance for a GARCH model. The details are in Section 4.1.

```
open data crspw.txt
calendar(w) 1962:7:3
data(format=prn,nolabels,org=columns) 1962:07:03 1987:12:29 vw
*
* Copy the data over to y to make it easier to change the program.
*
set y = vw
*
* Define the equation for the mean model
*
equation ar1 y
# constant y{1}
*
garch(p=1,q=1,equation=ar1,hseries=h,dist=t)
*
* Forecast the mean
*
uforecast(equation=ar1,steps=20) yfore
graph(key=upright,klabels=||"Actual","Forecast"||) 2
# y %regend()-5 %regend()
# yfore
*
@garchfore(steps=20) h %resids
graph(key=upright,klabels=||"Sample","Forecast"||) 2
# h %regend()-5 %regend()
# h %regend() *
*
set upper %regend()+1 %regend()+20 = yfore+2.0*sqrt(h)
set lower %regend()+1 %regend()+20 = yfore-2.0*sqrt(h)
graph(footer="Forecasts with Upper/Lower 2 s.d. bounds") 4
# y %regend()-10 %regend()
# yfore
# upper / 3
# lower / 3
```

Example 4.2 Stability Tests

This does a stability test (Section 4.3), and, based upon the results, re-estimates the model adding a variance dummy (Section 4.4).

```
open data crspw.txt
calendar(w) 1962:7:3
data(format=prn,nolabels,org=columns) 1962:07:03 1987:12:29 vw
*
* Copy the data over to y to make it easier to change the program.
```

```

*
set y = vw
*
garch(p=1,q=1,reg,hseries=h,dist=t,derives=dd) / y
# constant y{1}
@flux(title="Stability Test for GARCH Model")
# dd
*
* Check information about the highly significant coefficient
*
acc dd(3) / fluxstat
graph(footer="Cumulated Gradient for Variance Intercept")
# fluxstat
*
* Adjust sample to eliminate the earlier quiet segment
*
clear dd fluxstat
garch(p=1,q=1,reg,hseries=h,dist=t,derives=dd) 1966:1 * y
# constant y{1}
@flux(title="Stability Test for Adjusted GARCH Model") 1966:1 *
# dd
*
* Re-estimate with a variance shift dummy for the early part of the
* sample
*
set dto1966 = t<1966:1
garch(p=1,q=1,reg,xreg,hseries=h,dist=t,derives=dd) / y
# constant y{1}
# dto1966
*
* With crash variance shift as well
*
set crash = t==1987:10:20
garch(p=1,q=1,reg,xreg,hseries=h,dist=t,derives=dd) / y
# constant y{1}
# dto1966 crash

```


Example 4.3 GARCH-M, EGARCH and Asymmetry

This does the GARCH-M models from Section 4.5, the EGARCH models from Section 4.6 and the asymmetric models from Section 4.7.

```
open data crspw.txt
calendar(w) 1962:7:3
data(format=prn,nolabels,org=columns) 1962:07:03 1987:12:29 vw
*
* Copy the data over to y to make it easier to change the program.
*
set y = vw
*
* GARCH-M models
*
garch(p=1,q=1,reg,hseries=h,dist=t) 1966:1 * y
# constant y{1} %garchv
set gh = sqrt(h)
garch(p=1,q=1,reg,hseries=h,hadjust=(gh=sqrt(h)),dist=t) 1966:1 * y
# constant y{1} gh
*
* EGARCH model (without asymmetry)
*
garch(exp,p=1,q=1,reg,hseries=hx,dist=ged) 1966:1 * y
# constant y{1}
garch(p=1,q=1,reg,hseries=h,dist=t) 1966:1 * y
# constant y{1}
set sdx = sqrt(hx)
set sd = sqrt(h)
graph(footer="Standard Deviations from EGARCH and GARCH", $
    key=below,klabels=||"EGARCH","GARCH"||) 2
# sdx 1970:1 1970:52
# sd 1970:1 1970:52
*
* News impact curve
*
@gridseries(from=-10.0,to=10.0,n=201) eta
*
* Use the unconditional variance from the standard GARCH for both models.
*
garch(p=1,q=1,reg,dist=t) 1966:1 * y
# constant y{1}
*
* Pull the parameters out of the estimates
*
dec vect bmean(%nregmean)
nonlin(parmset=garchparms) bmean c0 a1 b1 shape
compute %parmspoke(garchparms,%beta)
*
* Compute around the long-run variance from the standard GARCH
* model.
*
compute hbar=c0/(1-a1-b1)
```

```

set gimpact 1 201 = c0+b1*hbar+a1*eta^2*hbar
*
garch(exp,p=1,q=1,reg,dist=ged) 1966:1 * y
# constant y{1}
*
* Everything is in the same position as above, but we rebuild the
* PARMSET anyway.
*
dec vect bmean(%nregmean)
nonlin(parmset=garchparms) bmean c0 a1 b1 shape
compute %parmspoke(garchparms,%beta)
*
* Again, to permit comparison, we compute everything around the
* long-run variance from the first model.
*
*
set eimpact 1 201 = exp(c0+b1*log(hbar)+a1*abs(eta))
*
scatter(style=lines,key=below,klabels=||"EGARCH","GARCH"||,$
  footer="News Impact Curves",hlabel="Standardized Residuals") 2
# eta eimpact
# eta gimpact
*
* Asymmetry
*
garch(exp,asymmetric,p=1,q=1,reg,hseries=hxa,dist=ged) 1966:1 * y
# constant y{1}
set sdx = sqrt(hx)
set sdx_a = sqrt(hxa)
graph(footer="Standard Deviations from EGARCH With and Without Asymmetry",
  key=below,klabels=||"Symmetric","Asymmetric"||) 2
# sdx 1970:1 1970:52
# sdx_a 1970:1 1970:52
*
garch(asymmetric,p=1,q=1,reg,hseries=ha,dist=t) 1966:1 * y
# constant y{1}

```

Multivariate GARCH: Basics

It didn't take long for GARCH models to make the jump from univariate to multivariate settings. In financial econometrics, it's rare to have only one asset of interest—if you're analyzing bonds, there are different maturities, if exchange rates, multiple currencies, and, of course, there are thousands of equities. Not only is the volatility for each likely to be described fairly well by a GARCH process, but within a class of assets, the movements are likely to be highly correlated. As a result, there would be expected to be substantial gains in (statistical) efficiency in modeling them jointly. In addition, having a time-varying estimate of the covariances would permit calculation of rolling optimal portfolio allocations.

There are *many* more variants of multivariate GARCH models than univariate. In some cases, this is driven by the need to answer particular questions. Many of the differences, though, are due to the difficulties in estimating the more general forms of multivariate GARCH models. You might notice that we really didn't say much about the technical options, such as initial guess values and algorithms, for estimating univariate models in Chapters 3 and 4. This is because the univariate log likelihood is quite well-behaved—even if you estimate a model with a data set which isn't well-described by a GARCH, you will generally get converged estimates without much fuss. That is far from true for multivariate GARCH models, so we will have to be much more careful, both in choice of a specific form for the model, and also in choosing the algorithm to use.

5.1 Preliminaries

Before you do anything else, do yourself a favor and graph the data. Make sure that it at least *looks* like it could be generated by a GARCH process. The data set we will work with here is the daily exchange rate data from section 3.11 of Enders (2010). This has nine years of daily observations on five exchange rates vs the US dollar, though we will work only with three of them: the Euro, pound and Swiss franc.¹ We read the data and convert the three of interest to returns with:

¹The two we are omitting are the Australian and Canadian dollars.

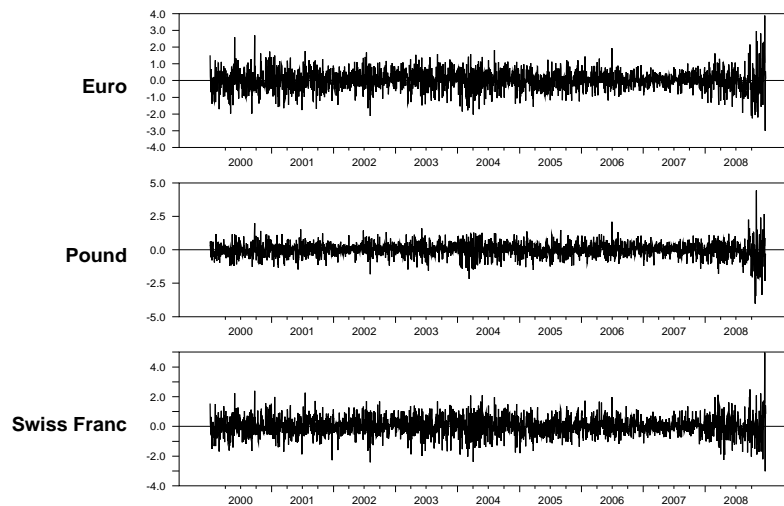


Figure 5.1: Exchange Rate Returns

```
open data "exrates(daily).xls"
calendar(d) 2000:1:3
data(format=xls,org=columns) 2000:01:03 2008:12:23 $
  aust euro pound sw ca
*
set reuro = 100.0*log(euro/euro{1})
set rpound = 100.0*log(pound/pound{1})
set rsw = 100.0*log(sw/sw{1})
```

This scales by 100 as described on page 5, which, in practice, is even more important for multivariate models because of the greater numerical difficulties they create. The following graphs the three return series, producing Figure 5.1.²

```
spgraph(vfields=3,ylabels=||"Euro","Pound","Swiss Franc"||)
  dofor r = reuro rpound rsw
    graph(picture="*.#")
    # r
  end dofor
spgraph(done)
```

What could you see at this point which might raise a flag that the GARCH model might be a mistake (at least across the sample in question)? A common error is including a part of the sample where a price was regulated, so the returns are zero for long stretches with occasional one-period spikes. This may be interesting from an economic standpoint, but you won't be able to include that (at least as an endogenous variable) in any sensible GARCH model. Data

²In this chapter's *Tips and Tricks* (Section 5.8.1), we'll see how to convert this into a better-looking graph.

Table 5.1: Lag Length Selection for Exchange Rates

VAR Lag Selection	
Lags	SBC/BIC
0	7243.00043*
1	7272.86486
2	7319.51443
3	7373.53640
4	7422.21572
5	7465.46548

which have visible variation only at one or two episodes will only fit a non-stationary GARCH model, which may be quite hard to interpret.

As with the univariate case, we need to decide upon a model for the mean. If Ω_{t-1} denotes the information available at time $t - 1$, then a general structure for (the variance model in) a multivariate GARCH model is:

$$E(\mathbf{u}_t | \Omega_{t-1}) = 0$$

$$E(\mathbf{u}_t \mathbf{u}_t' | \Omega_{t-1}) \equiv \mathbf{H}_t = f(\mathbf{H}_{t-1}, \mathbf{H}_{t-2}, \dots, \mathbf{u}_{t-1}, \mathbf{u}_{t-2}, \dots)$$

Before we even concern ourselves with the form of f , note that the first condition means that the series is (at a minimum) a vector white noise—the residuals not only have to be serially uncorrelated individually, but they need to have zero correlation with the lags of the *other* components. An obvious model to entertain for eliminating joint autocorrelation is a low-order VAR. The vector analogue of the `@ARAutoLags` procedure from Section 3.3 is `@VARLagSelect`:

```
@varlagselect(crit=bic, lags=5)
# reuro rpound rsw
```

The minimum BIC lag length (Table 5.1) is 0, which is what is used in the Enders book. Hannan-Quinn (option `CRIT=HQ`) slightly favors 1 over 0, and Akaike (option `CRIT=AIC`) favors five and prefers any positive number of lags over 0.

As with `@ARAutoLags`, `@VARLagSelect` assumes homoscedastic residuals and so can't offer more than a rough guide to choice of lag length. And as before, we're better off starting small and testing the results for residual correlation. Thus, we will use the intercepts only as the mean model, which is the default for the `GARCH` instruction. However, for demonstration purposes, we will use the `MODEL` option on `GARCH`, which is what you would use for any more general type of mean model. The simplest way to define the `MODEL` when we have the same right-side variables in each equation is to use the `SYSTEM` definition instructions:

```

system(model=mvmean)
variables reuro rpound rsw
lags
det constant
end(system)

```

We left an empty `LAGS` instruction to show where you would put in VAR lags if you wanted or needed them.

We can do a preliminary test for multivariate ARCH effects using the procedure `@MVARCHTest`. Since this is mainly designed for diagnostics, it assumes that the input series are already (roughly) mean zero, that is, it doesn't subtract means itself. We can do the least squares estimates and test for ARCH with

```

estimate(resids=resids)
@mvarchtest
# resids

```

which produces the (unsurprising) result that lack of ARCH is overwhelming rejected:

Test for Multivariate ARCH		
Statistic	Degrees	Signif
370.11	36	0.00000

`@MVARCHTest` (from Hacker (2005)) works by running a multivariate regression of all unique combinations of $u_{it}u_{jt}$ on constant and all unique combinations of $u_{i,t-k}u_{j,t-k}$ for each k up to the number of lags tested (which is 1 by default, and controlled by the `LAGS` option) and testing the significance of the lag coefficients. Thus, the null is that the u series has a fixed covariance matrix against the alternative of *some* 2nd order dependence. $u_t u_t'$ is symmetric, so there are $n(n+1)/2$ unique elements in it, thus 6 for the case in question with $n = 3$. The degrees of freedom is the square of that times the number of lags, which can get quite large quite quickly. For diagnostic purposes, we will need to be a bit careful about relying too much on it alone, because some of the tested coefficients will be more likely to be informative than others—only 6 out of 36 will be “own” effects, testing second order correlation of $u_{it}u_{jt}$ with its own lag rather than a different i, j combination.

Again, note that (as with the univariate test) rejecting the null does *not* mean that a GARCH model is correct, only that a fixed covariance *isn't*. Even a change in the correlation structure without any overall changes in the variances can trigger a significant test.

5.2 GARCH Instruction

You use the same `GARCH` instruction for multivariate models as you do for univariate; however, there are a different set of options to deal with choice

of model, and the output information for variances and residuals is different. For the univariate model, we used the `%RESIDS` series for the residuals and the `HSERIES` option to get the estimated variances. Neither of those will be sufficient (and neither even works) with a multivariate model—the residuals will be an n vector at each point in time, and the variances will be a complete symmetric $n \times n$ matrix in order to capture the correlations as well as the variances. The options for getting the multivariate information are now `RVECTORS` for the residuals and `HMATRICES` for the covariance matrices. `RVECTORS` returns a `SERIES OF VECTORS` and `HMATRICES` a `SERIES OF SYMMETRIC matrices`. For instance, if we use the option `HMATRICES=HH`, then in a **SET** instruction or other formula, `HH(T)` is an $n \times n$ symmetric matrix: if we need the 1,1 element of that at time T , we use the double-subscripted expression `HH(T)(1,1)`.

As we will see, there are many ways to set up a multivariate GARCH model. The main option for choosing the type is `MV=model type`. For illustration, we'll use `MV=DIAG`, which isn't really a useful model in practice. This ignores the covariances and models the variances separately using univariate methods. The point estimates should be almost identical to what you would get by estimating separate univariate models—the parameters are estimated jointly so none are considered converged until all of them are, which leads to very slight differences from one-at-a-time estimation. Assuming Normal residuals, the diagonal model can be written:

$$\begin{aligned} h_{ii,t} &= c_i + a_i u_{i,t}^2 + b_i h_{ii,t-1} \\ h_{ij,t} &= 0 \text{ if } i \neq j \\ \log L_t &= \text{const.} - \frac{1}{2} \log |\mathbf{H}_t| - \frac{1}{2} \mathbf{u}'_t \mathbf{H}_t^{-1} \mathbf{u}_t \end{aligned}$$

Because of the diagonality of \mathbf{H} , the log likelihood is just the sum of the log likelihoods across i . The instruction for estimating this (saving the residuals and \mathbf{H} matrices) is:

```
garch(model=mvmean,mv=diag,p=1,q=1,rvectors=rd,hmatrices=hh)
```

You use the `MODEL` option to input the mean model that we defined earlier. Note that the process of estimating the model also sets the coefficients of this model to the GARCH estimates of the mean coefficients, so if we followed this by a **FORECAST**, it would use the **GARCH** estimates rather than the OLS estimates that we calculated first. Note that, because means-only is the default, we could also have estimated this with:

```
garch(mv=diag,p=1,q=1,rvectors=rd,hmatrices=hh) / reuro rpound rsw
```

This is the reason the range parameters come first on **GARCH**: to allow for the open-ended list of dependent variables in this form. The results are in Table 5.2.

Table 5.2: MV-GARCH Diagonal Model

MV-GARCH, Diagonal - Estimation by BFGS					
Convergence in 46 Iterations. Final criterion was 0.0000071 <= 0.0000100					
Daily(5) Data From 2000:01:04 To 2008:12:23					
Usable Observations		2341			
Log Likelihood		-6015.5736			
	Variable	Coeff	Std Error	T-Stat	Signif
1.	Constant	0.0240	0.0114	2.0974	0.0360
2.	Constant	0.0114	0.0107	1.0680	0.2855
3.	Constant	0.0195	0.0130	1.5033	0.1327
4.	C(1)	0.0006	0.0006	1.0818	0.2793
5.	C(2)	0.0035	0.0014	2.5913	0.0096
6.	C(3)	0.0010	0.0008	1.2795	0.2007
7.	A(1)	0.0325	0.0049	6.6572	0.0000
8.	A(2)	0.0483	0.0077	6.2504	0.0000
9.	A(3)	0.0276	0.0045	6.0641	0.0000
10.	B(1)	0.9672	0.0052	187.0732	0.0000
11.	B(2)	0.9406	0.0106	88.8573	0.0000
12.	B(3)	0.9713	0.0050	195.3141	0.0000

First, note that the model being estimated is in the first line of the output—here “MV-GARCH Diagonal”. Rather than just C, A and B, the GARCH coefficients are $C(1)$, $C(2)$, $C(3)$, and similarly for A and B. The C, A and B prefixes will be used for the constant, lagged squared residual and lagged variance terms in all multivariate GARCH models, but the subscripting will change depending upon the form of the multivariate model. The mean model coefficients will always be at the top, grouped by equation.

5.3 Diagnostics

The diagnostics for the univariate GARCH models were based upon the standardized residuals, which should be (if the model is correct) serially uncorrelated and homoscedastic. The residuals from the multivariate model should also be serially uncorrelated, and show no remaining ARCH effects. However, there are two possible approaches to testing these: using univariate tests or multivariate tests. The latter is what is actually implied by the assumptions of the model, but the multivariate tests are both more complicated and not as precisely defined.

The univariate tests would just apply the same pair of tests (for serial correlation on standardized residuals and their squares) to each of the variables. A quick, but not very pretty, set of tests can be done with:


```

do i=1,%nvar
  set  ustd  = rd(t) (i) /sqrt (hh (t) (i,i) )
  set  ustdsq = ustd^2
  @regcorrs (number=10,nocrits,nograph,qstat)  ustd
  compute q1=%cdstat,q1signif=%signif
  @regcorrs (number=10,nocrits,nograph,qstat,dfc=2)  ustdsq
  compute q2=%cdstat,q2signif=%signif
  disp q1 q1signif q2 q2signif
end do i

```

`%NVAR` is defined by **GARCH** as the number of dependent variables, so this will work for any size GARCH model as long as you use `RD` and `HH` for the `RVECTORS` and `HMATRICES` options. The first **SET** instruction shows how to divide component `I` of the residuals by its corresponding (time-varying) standard deviation.

In *Tips and Tricks* (Section 5.8.2), we'll show how to convert this to a nicer table. What this simple version produces is:

8.73373	0.55755	16.30334	0.03824
13.55638	0.19420	9.68835	0.28758
4.44765	0.92490	9.14915	0.32987

The only one of the six numbers that is significant at conventional levels is the one on the squares for the first variable (the Euro, row 1, third number is the statistic and fourth is the significance level) but with over 2000 data points, that's unlikely to be anything fixable with any simple change to the model (Appendix C).

These, however, are just univariate diagnostics, and we have a model which is supposed to produce *vector* white noise. The problem in moving to multivariate diagnostics is dealing with the time-varying covariance matrices. The univariate standardized residuals give three series which are (approximately) each variance one, but they ignore the fact that the correlations vary from time period to time period. Instead, we need a matrix-standardized set of residuals, which transform to (approximately) the identity matrix. However, that standardization isn't unique, as if we take *any* sequence of matrices G_t such that $G_t' G_t = H_t^{-1}$, then $E(G_t u_t u_t' G_t') = I$. Once we have two or more variables, there are an infinite number of such “square root” matrices (at each t) and each sequence of $G_t u_t$ will produce a different test statistic for correlation—whether these are likely to be qualitatively different is unclear.

The simplest such G to compute is the inverse of the Cholesky factor of H . However, a better choice under the circumstances is to use the one derived from an eigen decomposition of H , which will be independent of the order in which you list the dependent variables. Unlike the univariate tests, where we could generate the standardized residuals one-at-a-time, the multivariate tests require that we have the full set of generated residuals simultaneously. They also need to be organized differently: as a `VECTOR` of `SERIES` rather than a `SERIES` like `RD`. You generate these with the `STDRESIDS` option on **GARCH**. The choice of

G is controlled by the **FACTORBY** option, which is either **FACTORBY=CHOLESKY** (the default) or **FACTORBY=EIGEN**.

Adding the options to do the eigen standardization to the **GARCH** instruction gives us:

```
garch(model=mvmean,mv=diag,p=1,q=1,$
      rvector=rd,hmatrices=hh,stdresids=rstd,factorby=eigen)
```

The procedures designed for the diagnostics test on this are **@MVQSTAT** and **@MVARCHTest** that we've already used. We compute a 10-lag Q -statistic with:

```
@mvqstat(lags=10)
# rstd
```

which gives us

Multivariate Q(10)=	195.71117
Significance Level as Chi-Squared(90)=	8.24580e-010

Note that if we were using a VAR with lags, we would need to include the **DFC** (Degrees for Freedom Correction) option on **@MVQSTAT** with correction $n^2 \times L$ where L is the number of VAR lags.

The degrees of freedom on the test will be n^2 times the number of tested lags (here 10). The Q test looks for correlation between $u_{i,t}$ and $u_{j,t-k}$ for all i, j combinations and for each tested lag k .

Despite the fact that the individual Q statistics were insignificant, this is significant beyond any doubt. So what happened? The problem isn't that the residuals are actually serially correlated, it's that the variance model is so completely wrong that the standardization process produces nonsense. **MV=DIAG** assumes the off-diagonal elements of H_t are zero, and that shows up in the **HH** matrices. The sample residuals, however, are quite strongly contemporaneously correlated, so the standardization is simply wrong.

The test for ARCH (here with two lags) confirms the fact that the model estimated is inadequate:

```
@mvarchtest(lags=2)
# rstd
```

gives us

Test for Multivariate ARCH		
Statistic	Degrees	Signif
465.60	72	0.00000

It's important to recognize the difference in reading the diagnostics: in the univariate case, an incorrect GARCH model will rarely produce a highly significant finding of serial correlation in the mean when the mean model is, in fact, adequate, so if we get the result that the Q is significant, the fix will generally be to

change the mean model. The multivariate statistics are much more dependent upon a good variance model, even for testing the adequacy of the mean, because the standardization depends not just on scale (which the diagonal model gets right) but on “shape”.

5.4 VECH, DVECH and BEKK Models

The first important paper to use a multivariate GARCH model was Bollerslev et al. (1988). This actually did a multivariate GARCH-M, but for now we’ll just discuss the variance model in isolation. The first form that they introduce for that is what is now known as the VECH or Full-VECH, after the “vech” operator which takes an $n \times n$ symmetric matrix and converts it to an $n(n+1)/2$ vector by eliminating the duplicated entries. A VECH 1,1 model is written:

$$vech(\mathbf{H}_t) = \mathbf{C} + \mathbf{A} vech(\mathbf{u}_{t-1} \mathbf{u}_{t-1}') + \mathbf{B} vech(\mathbf{H}_{t-1}) \quad (5.1)$$

In this form, \mathbf{C} is a vector which should be the *vech* of a positive semi-definite matrix. \mathbf{A} and \mathbf{B} are full $n(n+1)/2 \times n(n+1)/2$ matrices. Bollerslev, Engle and Wooldridge didn’t estimate this model, and it is rarely used. One reason is fairly obvious—the number of free parameters goes up very fast with n . Even for our example with $n = 3$, we have 78 in the variance model (6 in \mathbf{C} , $36 = 6^2$ for each of \mathbf{A} and \mathbf{B}). That’s a very large number of free parameters for non-linear estimation. Also, this has an unrestricted coefficient on each of the elements of \mathbf{H}_{t-1} for explaining each element of \mathbf{H}_t . If the shocks are highly correlated (which is what we are expecting), then the variances and covariances will tend to move together. Thus (with our $n = 3$ data set), each element of \mathbf{H}_t has free coefficients on six different, and relatively similar, pieces of information. So not only is it a high-dimension set of non-linear parameters, but many of them are poorly determined from the data.

The VECH model is estimated with RATS by using the `MV=VECH` option. However, you have very little chance of getting a model with anything more than two variables to estimate properly without a great deal of experimentation with guess values.

However, even if the unrestricted (5.1) isn’t very useful for estimation, it is (with restrictions) useful for forecasting and similar calculations. If we can cast a GARCH recursion in this form, then, just as with the univariate model, the forecasts can be generated as

$$\begin{aligned} vech(\hat{\mathbf{H}}_{t+1}) &= \mathbf{C} + \mathbf{A} vech(\mathbf{u}_t \mathbf{u}_t') + \mathbf{B} vech(\mathbf{H}_t) \\ vech(\hat{\mathbf{H}}_{t+k}) &= \mathbf{C} + (\mathbf{A} + \mathbf{B}) vech(\hat{\mathbf{H}}_{t+k-1}) \end{aligned} \quad (5.2)$$

that is, the first period out-of-sample forecast uses the computed residuals and variance, while all steps after that are a matrix difference equation in the forecast variance/covariance matrix.

Two important variance models are restrictions on the VEC: the diagonal or DVECH model and the BEKK. Several others (CC and DCC) are not. In this section, we will deal with the restricted VEC forms.

5.4.1 Diagonal VEC (Standard) Model

The model estimated by Bollerslev, Engle and Wooldridge was the diagonal VEC or DVECH model, which, if you don't state otherwise, is the standard for a multivariate GARCH, and is the default for the RATS **GARCH** instruction. This assumes that A and B in (5.1) are diagonal. Another (more compact) way to write this is

$$H_t = C + A \circ (u_{t-1} u'_{t-1}) + B \circ H_{t-1} \quad (5.3)$$

where \circ represents the *Hadamard* or elementwise product,³ and C , A and B are now $n \times n$ symmetric matrices. Written out, this means

$$h_{ij,t} = c_{ij} + a_{ij}u_{i,t-1}u_{j,t-1} + b_{ij}h_{ij,t-1} \quad (5.4)$$

This greatly decreases the number of free parameters: we still have six in C , but only six each in A and B .

Despite being a much smaller model, there are still some numerical problems that must be overcome in estimating this. (5.4) describes “independent” recursions on each of the components of the covariance matrix. For this to produce a computable log likelihood, each H_t matrix must be positive-definite, but there's nothing in the structure of (5.4) that will enforce that. If an off-diagonal recursion has a higher persistence than its corresponding diagonal elements, it's possible for the recursion to stray into non-positive-definite territory if the correlations among the residuals are high. The parameters will never actually go into an area where the log likelihood isn't computable, but if the estimation process ever gets close to a singular H at *any* data point, then it may be hard to move away from that test set of parameters, because the predictions for the log likelihood from the gradient can be quite inaccurate due to the function being nearly non-differentiable.

This is a problem that's specific to the two derivative-based optimization algorithms: BFGS and BHHH. BHHH is almost *never* a good choice as a basic estimation algorithm for GARCH models—its prediction of the curvature of the log likelihood is likely to be good only when you're fairly close to the optimum, and getting *to* the optimum is the challenge. BFGS is generally more successful, but it does have the problem that it estimates the curvature (and thus the covariance matrix of the parameter estimates) using an update method which will give a different answer for different initial guess values. In the GARCH literature, it's not uncommon for standard errors to be reported as generated by either BHHH, or corrected for possible misspecification (in RATS done with the

³ $C = A \circ B \Rightarrow c_{ij} = a_{ij}b_{ij}$

If you need a Hadamard product in RATS as part of a calculation, you can do it with $A . * B$

ROBUSTERRORS option). In practice, the latter is more conservative and should be preferred.

The recommendation that we make for estimating multivariate GARCH models is to use a combination of preliminary “simplex” iterations, followed by BFGS for doing the final estimates. The problem with starting off with BFGS from the initial guesses is that BFGS also has some difficulty with the function curvature at the guess values since it builds up its estimate of the inverse Hessian indirectly by seeing how the gradient changes from iteration to iteration. A poor estimate of the curvature can lead to some wildly inaccurate moves in the early iterations. Because the log likelihood can have a very odd shape, it’s possible to make a big move in the wrong direction and then get stuck on the wrong “hill”—one that doesn’t include the global optimum.

We generally recommend somewhere between 5 and 20 simplex “iterations”.⁴ We frequently see people who think that if 20 simplex iterations is good, then 200 must be better. That’s rarely the case. The simplex method accomplishes quite a bit less in an “iteration” worth of work than one of the climbing algorithms, which is intentional—it relies upon much weaker assumptions about the behavior of the function being optimized and so doesn’t (and *can’t*) make very quick moves up the “hill”. Simplex should *eventually* get to the same optimum, but it can take thousands of iterations to do so with a parameter set of this size—20 vs 200 probably will make little difference. What the first 20 iterations do is to move the parameter set (cautiously) off the guess values in what is likely to be the correct direction. From there, the more standard algorithm can take over.

The estimation code is in Example 5.2. This uses the same data and mean model as Example 5.1. The instruction for estimating the DVECH model using the recommended options is

```
garch(model=mvmean,rvectors=rd,hmatrices=hh,robusterrors,$  
      pmethod=simplex,piters=20,method=bfgs,itors=500)
```

which produces Table 5.3. The MV option isn’t necessary, since this model is the default, but it is called MV=STANDARD. This model nests the MV=DIAG model from Table 5.2 (the diagonal model has all the off-diagonal coefficients zeroed), so the log likelihood must be higher here, but being higher by over 3000 is quite a difference—the diagonal model is obviously thoroughly misspecified.

The A, B and C coefficients are labeled as shown in (5.4) and they are in the rowwise order of the lower triangle that is used for SYMMETRIC matrices in

⁴Iterations is in quotes because, in RATS, a simplex iteration is counted as a set of calculations with roughly the same number of function evaluations as an iteration in one of the derivative-based algorithms. Simplex is very different from “climbing” algorithms, as, rather than explicitly looking for directions of increase, it spends most of its effort eliminating directions of decrease.

Table 5.3: Multivariate GARCH-Diagonal VECH

MV-GARCH - Estimation by BFGS					
Convergence in 194 Iterations. Final criterion was 0.0000000 <= 0.0000100					
With Heteroscedasticity/Misspecification Adjusted Standard Errors					
Daily(5) Data From 2000:01:04 To 2008:12:23					
Usable Observations		2341			
Log Likelihood		-2758.8148			
	Variable	Coeff	Std Error	T-Stat	Signif
1.	Constant	0.0247	0.0103	2.4017	0.0163
2.	Constant	0.0149	0.0100	1.4936	0.1353
3.	Constant	0.0199	0.0123	1.6154	0.1062
4.	C(1,1)	0.0033	0.0017	1.9282	0.0538
5.	C(2,1)	0.0026	0.0017	1.5065	0.1319
6.	C(2,2)	0.0033	0.0022	1.4992	0.1338
7.	C(3,1)	0.0035	0.0018	1.9824	0.0474
8.	C(3,2)	0.0022	0.0012	1.8255	0.0679
9.	C(3,3)	0.0047	0.0023	2.0432	0.0410
10.	A(1,1)	0.0413	0.0104	3.9534	0.0001
11.	A(2,1)	0.0332	0.0092	3.5961	0.0003
12.	A(2,2)	0.0389	0.0091	4.2776	0.0000
13.	A(3,1)	0.0387	0.0099	3.9140	0.0001
14.	A(3,2)	0.0308	0.0064	4.8414	0.0000
15.	A(3,3)	0.0390	0.0102	3.8068	0.0001
16.	B(1,1)	0.9508	0.0147	64.7672	0.0000
17.	B(2,1)	0.9565	0.0165	57.9902	0.0000
18.	B(2,2)	0.9513	0.0156	60.9383	0.0000
19.	B(3,1)	0.9530	0.0141	67.6920	0.0000
20.	B(3,2)	0.9609	0.0108	88.7553	0.0000
21.	B(3,3)	0.9514	0.0145	65.6489	0.0000

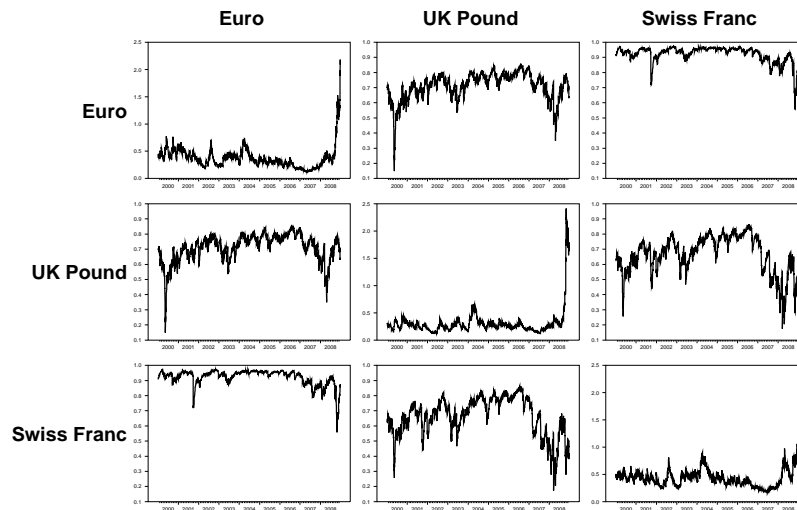


Figure 5.2: Variances (Diagonal) and Correlations (Off-Diagonal)

RATS. If we want to extract the A, B and C matrices from (5.3), we can do it with

```
dec vect meanparms(%nregmean)
dec symm cdvech(%nvar,%nvar) advech(%nvar,%nvar) bdvech(%nvar,%nvar)
nonlin(parmset=dvech) meanparms cdvech advech bdvech
compute %parmspoke(dvech,%beta)
```

which is an extension to a (multivariate) DVECH model of the use of the PARMSET and %PARMSPOKE function. If we extract these and add them, we'll get the persistence on an element by element basis:

0.99210		
0.98966	0.99014	
0.99169	0.99172	0.99043

Note that these are all very close to 1, and the 3,2 element is, in fact, larger than the 2,2 and 3,3 elements, which is the type of behavior that *could* cause problems with the positive-definite boundary.

We can extract the time-varying correlations and the time-varying variances using the following, which organizes them into a SYMMETRIC of SERIES, with variances on the diagonals and correlations on the off-diagonals. The simplest way to get the correlations is to use %CVTOCORR(C), which returns the correlation matrix formed from the covariance matrix C, as we can just pull elements out of that. Note that this works for *any* type of multivariate GARCH model.

```

dec symm[series] hhx(%nvar,%nvar)
do i=1,%nvar
  set hhx(i,i) = hh(t)(i,i)
  do j=1,i-1
    set hhx(i,j) = %cvtocorr(hh(t))(i,j)
  end do j
end do i

```

We can use some of the ideas from Section 5.8.1 to present this information in an interesting way (Figure 5.2) with:

```

table / hhx(2,1) hhx(3,1) hhx(3,2)
compute corrmin=%minimum
table / hhx(1,1) hhx(2,2) hhx(3,3)
compute varmax=%maximum
*

spgraph(vfields=%nvar,hfields=%nvar,$
  xlabel=longlabel,ylabel=longlabel)
do i=1,%nvar
  do j=1,%nvar
    if i==j {
      graph(row=i,col=i,maximum=varmax)
      # hhx(i,i)
    }
    else {
      graph(row=i,col=j,maximum=1.0,min=corrmin)
      # hhx(i,j)
    }
  end do j
end do i
spgraph(done)

```

This puts the variances in the diagonal fields and the correlations in the off-diagonals. To avoid a misleading impression, this forces all the correlations to use the same range. For instance, without that, it wouldn't be clear without a careful look at the scale that the Swiss Franc-Euro correlation is so extremely high across almost the entire range if its values (roughly .6 to 1) are spread across the full box, and the UK correlations (roughly .1 to .9) are graphed in the same sized box.

5.4.2 BEKK Model

There are a few drawbacks to the DVECH model. First, as we described above, it can have some numerical issues because the parameterization doesn't enforce positive-definiteness.⁵ Second, it doesn't allow for certain types of more complicated interactions among the variables; for instance, one interesting possibility is that shocks in one market could have a “spillover” effect on another (Section 5.5). That's precluded by the structure of the DVECH where the only thing that determines the variance of one series is its own shocks.

To allow a greater range of interactions (though not as general as the VECHE), while also enforcing positive-definiteness by construction, Engle & Kroner (1995) proposed what is now known as the BEKK model, after the four authors of an earlier working paper. Note that this does *not* nest the DVECH, so even though it has a few additional parameters, it doesn't necessarily fit better.

For a 1,1 model, the BEKK recursion is

$$\mathbf{H}_t = \mathbf{C}\mathbf{C}' + \mathbf{A}'\mathbf{u}_{t-1}\mathbf{u}_{t-1}'\mathbf{A} + \mathbf{B}'\mathbf{H}_{t-1}\mathbf{B} \quad (5.5)$$

where \mathbf{C} is now a lower triangular matrix and \mathbf{A} and \mathbf{B} are general $n \times n$ matrices. The last two terms could also have the transpose on the post-multiplying matrix rather than the pre-multiplying one—we're following the original Engle and Kroner formula, which is the one most commonly used. By construction, this is positive semi-definite regardless of the values of the parameters, and, in practice, will maintain positive definiteness as long as the \mathbf{B} or \mathbf{C} is full rank.⁶ \mathbf{C} has the same number of parameters as it does in the DVECH, just in a different form. \mathbf{A} and \mathbf{B} now have n^2 rather than $n(n+1)/2$ and so will always have more free parameters than the corresponding matrices in the DVECH: with $n = 3$, there are 9 for each rather than 6.

That this is a restricted version of the VECHE form (5.1) is clear if you look at the expansions of the matrix multiplications. In fact, if we write (5.1) in a “vec” form (which includes the full $n \times n$ matrix including duplicates, stacked by columns), you get

$$\text{vec}(\mathbf{H}_t) = \text{vec}(\mathbf{C}\mathbf{C}') + (\mathbf{A}' \otimes \mathbf{A}') \text{vec}(\mathbf{u}_{t-1}\mathbf{u}_{t-1}') + (\mathbf{B}' \otimes \mathbf{B}') \text{vec}(\mathbf{H}_{t-1}) \quad (5.6)$$

which is easier to analyze for most purposes than the equivalent VECHE form. That the BEKK *can't* be a generalization of DVECH is clear from the fact that $\mathbf{A}'\mathbf{u}_{t-1}\mathbf{u}_{t-1}'\mathbf{A}$ is rank one and the analogous term in the DVECH will be full rank for almost any set of parameters.

⁵Again, sample estimates will *always* give positive-definite matrices for every entry because the likelihood isn't computable otherwise. However, not all sets of parameters will do that, and the boundary between parameters that do and parameters that don't can be very complicated.

⁶The lagged \mathbf{u} term is rank (at most) one, since it's the outer product of the n vector $\mathbf{A}'\mathbf{u}_{t-1}$ and so isn't as critical for maintaining the full rank.

Table 5.4: Multivariate GARCH-BEKK Estimates

MV-GARCH, BEKK - Estimation by BFGS					
Convergence in 172 Iterations. Final criterion was 0.0000000 <= 0.0000100					
With Heteroscedasticity/Misspecification Adjusted Standard Errors					
Daily(5) Data From 2000:01:04 To 2008:12:23					
Usable Observations		2341			
Log Likelihood		-2727.5812			
	Variable	Coeff	Std Error	T-Stat	Signif
1.	Constant	0.0286	0.0115	2.5000	0.0124
2.	Constant	0.0174	0.0109	1.5937	0.1110
3.	Constant	0.0217	0.0129	1.6845	0.0921
4.	C(1,1)	0.0391	0.0094	4.1639	0.0000
5.	C(2,1)	0.0505	0.0105	4.8315	0.0000
6.	C(2,2)	-0.0055	0.0089	-0.6103	0.5417
7.	C(3,1)	0.0230	0.0114	2.0098	0.0445
8.	C(3,2)	-0.0277	0.0086	-3.2393	0.0012
9.	C(3,3)	0.0007	0.0064	0.1144	0.9089
10.	A(1,1)	0.2369	0.0621	3.8180	0.0001
11.	A(1,2)	0.1308	0.0660	1.9837	0.0473
12.	A(1,3)	-0.0862	0.0880	-0.9795	0.3273
13.	A(2,1)	0.0491	0.0289	1.7015	0.0888
14.	A(2,2)	0.1920	0.0318	6.0459	0.0000
15.	A(2,3)	0.0285	0.0287	0.9941	0.3202
16.	A(3,1)	-0.1201	0.0555	-2.1641	0.0305
17.	A(3,2)	-0.1323	0.0554	-2.3898	0.0169
18.	A(3,3)	0.1824	0.0890	2.0499	0.0404
19.	B(1,1)	0.9582	0.0161	59.4924	0.0000
20.	B(1,2)	-0.0254	0.0187	-1.3555	0.1753
21.	B(1,3)	0.0137	0.0268	0.5116	0.6089
22.	B(2,1)	-0.0208	0.0051	-4.0500	0.0001
23.	B(2,2)	0.9707	0.0063	154.5004	0.0000
24.	B(2,3)	-0.0091	0.0078	-1.1661	0.2436
25.	B(3,1)	0.0402	0.0161	2.4943	0.0126
26.	B(3,2)	0.0320	0.0191	1.6765	0.0936
27.	B(3,3)	0.9839	0.0288	34.1251	0.0000

To estimate, use the **GARCH** instruction with the option `MV=BEKK`. This is from Example 5.3, producing Table 5.4.

```
garch(model=mvmean,mv=bekk,robusterrors,pmethod=simplex,piters=20,$
      method=bfgs, iters=500, rvector=rd, hmatrices=hh,$
      stdresids=rstd, factorby=eigen)
```

The log likelihood is quite a bit higher than in Table 5.3. The models don't nest so we can't do a formal likelihood ratio test, but the difference is large enough that the BEKK model would be preferred even for a stringent information criterion like BIC.

We often get questions about the number of negative parameters in the typical BEKK output. One thing to note is that the BEKK model isn't global identified—

you get exactly the same fit if you change the sign of the entire A or B matrix, or even any column of C . However, the guess values used by **GARCH** will steer it in the direction of positive “own” contributions, so it would be very rare that you get the parameters with the opposite from the expected set of signs.

If you first look at the C coefficients, you’ll note that $C(2, 2)$ is (slightly) negative and $C(3, 3)$ is barely positive. Because this is a factor of the variance intercept (rather than the variance intercept itself) the coefficients other than the 1,1 don’t have simple interpretations. However, taken as a whole, these mean that the variance intercept matrix is close to being rank two.

It’s easier to interpret the values in the A matrix. $A'u_{t-1}$ is an n vector, call it v_{t-1} . The contribution to the variance at t is $v_{t-1}v'_{t-1}$, which means that the squares of the elements of v will be the contributions to the variances. To have “spillover” effects, so that shocks in u_j directly affect the variance of i , v will have to be a linear combination of the different components of u . Negative coefficients in the off-diagonals of A' mean that the variance is affected more when the shocks move in opposite directions than when they move in the same direction, which probably isn’t unreasonable in many situations. Here, we see that the most statistically significant spillovers are from 3 (Swiss Franc) to the other two ($A(3, 1)$ and $A(3, 2)$ coefficients) and these *do* have negative signs.

Another common question (which doesn’t apply here) is how it’s possible for the off-diagonals in the A and B matrices to be larger than the diagonals, since one would expect that the “own” effect would be dominant. However, the values of the coefficients are sensitive to the scales of the variables, since nothing in the recursion is standardized to a common variance. If you multiply component i by .01 relative to j , its residuals also go down by a factor of .01, so the coefficient a_{ij} which applies u_i to the variance of j has to go *up* by a factor of 100. Rescaling a variable keeps the *diagonals* of A and B the same, but forces a change in scale of the off-diagonals. Even without asymmetrical scalings, the tendency will be for (relatively) higher variance series to have lower off-diagonal coefficients than lower variance series.

You can convert the coded up BEKK coefficients to the equivalent VECB representation with the procedure `@MVGARCHtoVECH`, here with

```
@MVGARCHtoVECH(mv=bekk)
```

This creates the matrices `%%VECH_C`, `%%VECH_A` and `%%VECH_B`. With $n = 3$, the first will be a 6-vector, the others 6×6 matrices. We can take a closer look at the implied A in the VECB representation with:

```
disp ###.### %%vech_a
```

which gives us

0.056	0.023	0.002	-0.057	-0.012	0.014
0.031	0.052	0.009	-0.047	-0.030	0.016
0.017	0.050	0.037	-0.035	-0.051	0.018
-0.020	0.003	0.001	0.054	0.006	-0.022
-0.011	-0.013	0.005	0.035	0.031	-0.024
0.007	-0.005	0.001	-0.031	0.010	0.033

The “vech” operator is by the rows of the lower triangle, so the elements (in each direction) are in the order (1,1),(2,1),(2,2), etc. The variances (and squared residuals) are thus in positions 1, 3 and 6.

The BEKK estimates give rise to a stable recursion if the sum of the VEC A and B matrices has eigenvalues less than one. Since the eigenvalues could be complex, we can compute and display those with:

```
eigen(cvalues=cv) %%vech_a+%%vech_b
disp ###.### cv
```

and, as we can see, these are just barely on the stable side:

(0.998,-0.000)	(0.994, 0.003)	(0.994,-0.003)
(0.980,-0.015)	(0.980, 0.015)	(0.974, 0.000)

The multivariate diagnostics from Section 5.3 give us

Multivariate Q(10)=	109.59742	
Significance Level as Chi-Squared(90)=	0.07849	
Test for Multivariate ARCH		
Statistic	Degrees	Signif
107.91	72	0.00394

We would like the test for residual ARCH to be a *bit* better than this, but, in practice, with a data set this size (2300 observations), this isn’t unreasonable.

Some people recommend reporting BEKK estimates in the *vech* form with standard errors (which can be computed using the delta method from Appendix D). This seems to be a step backwards, particularly with regards to the coefficients for A where the *vech* form is a reduced form to the more structural BEKK. If you are required to do this by a referee, you can use **SUMMARIZE** using instructions like:

```
dec vect means(%nregmean)
dec packed cx(%nvar,%nvar)
dec rect ax(%nvar,%nvar) bx(%nvar,%nvar)
nonlin(parmset=garchparms) means cx ax bx

summarize(parmset=garchparms) bx(1,1)^2
summarize(parmset=garchparms) bx(1,1)*bx(1,2)*2.0
summarize(parmset=garchparms) bx(1,2)^2
```

The first four lines decompose the estimated parameters into the separate matrices used by a BEKK model. The last three compute the standard errors for

the first three elements of the first row of the VECH B matrix. Calculating standard errors for the full matrix requires the following, which has quite a bit of “bookkeeping” to deal with what’s basically four level subscripting, since each direction in the matrix represents a flattening of a symmetric matrix:

```
compute ncomp=%nvar*(%nvar+1)/2
dec rect %%vech_bse(ncomp,ncomp)
do m=1,ncomp
  do n=1,ncomp
    compute i=%symmrow(m),j=%symmcol(m), $
           k=%symmrow(n),l=%symmcol(n)
    if k==1 {
      summarize(noprint,parmset=garchparms) bx(i,k)*bx(j,l)
      compute %%vech_bse(m,n)=sqrt(%varlc)
    }
    else {
      summarize(noprint,parmset=garchparms) $
      bx(i,k)*bx(j,l)+bx(i,l)*bx(j,k)
      compute %%vech_bse(m,n)=sqrt(%varlc)
    }
  end do n
end do m
```

The values and standard errors can be displayed in a matrix form (Table 5.5) with⁷

```
report(action=define,title="VECH-B from BEKK")
do j=1,ncomp
  report(atrow=1,atcol=j+1,align=center) $
  %string(%symmrow(j))+", "+%symmcol(j)
end do j
do i=1,ncomp
  report(atrow=2*i,atcol=1) %string(%symmrow(i))+", "+%symmcol(i)
  do j=1,ncomp
    report(atrow=2*i,atcol=j+1) %%vech_b(i,j)
    report(atrow=2*i+1,atcol=j+1,special=parens) %%vech_bse(i,j)
  end do j
end do i
report(action=format,atrow=2,picture="##.####",align=decimal)
report(action=show)
```

5.5 Spillover

“Spillover” in the context of multivariate GARCH models is a rather vaguely defined term. In general, the term spillover in economics refers to an event producing an effect elsewhere despite there being no obvious connection. However,

⁷It’s not clear what this really adds.

Table 5.5: VECH-B from BEKK

	1,1	2,1	2,2	3,1	3,2	3,3
1,1	0.9180 (0.0248)	-0.0398 (0.0112)	0.0004 (0.0002)	0.0773 (0.0242)	-0.0017 (0.0008)	0.0016 (0.0011)
2,1	-0.0245 (0.0141)	0.9306 (0.0142)	-0.0202 (0.0056)	0.0298 (0.0129)	0.0385 (0.0124)	0.0013 (0.0010)
2,2	0.0007 (0.0008)	-0.0496 (0.0292)	0.9423 (0.0130)	-0.0016 (0.0017)	0.0624 (0.0285)	0.0010 (0.0009)
3,1	0.0131 (0.0212)	-0.0091 (0.0070)	0.0002 (0.0002)	0.9432 (0.0148)	-0.0208 (0.0061)	0.0397 (0.0137)
3,2	-0.0003 (0.0005)	0.0135 (0.0211)	-0.0089 (0.0073)	-0.0247 (0.0154)	0.9548 (0.0218)	0.0316 (0.0150)
3,3	0.0002 (0.0006)	-0.0003 (0.0003)	0.0001 (0.0001)	0.0269 (0.0426)	-0.0180 (0.0152)	0.9681 (0.0458)

it's hard to argue that there is no obvious connection between, for instance, two exchange rates (which presumably have the same numeraire currency, and thus are both directly affected by any event which touches the numeraire). Spillover has come to mean a model-based “Granger-style” causality, either in the mean or (more commonly) in the variance, that is a question of whether a set of coefficients on “other” variables are zero.

As we described earlier, the DVECH model doesn't allow for spillover/causality in the variance because the variance for each series depends only upon its own lagged variance and its own lagged (squared) residuals. However, BEKK does, so we'll look more carefully at the results from it.

One important thing to note is that, as with standard Granger causality tests, when you have three or more variables (we have three in this example), the exclusion of a single variable from a single equation doesn't really tell you much—a shock to variable Z can affect variable X indirectly if a shock to Z affects Y 's variance and Y 's variance affects X 's. Thus, tests should really be of *block* exclusions, either Z affects neither Y nor X , or X is not affected by either Z nor Y . Example 5.4 does tests for causality (spillover), both in the mean and in the variance. These are all done as “Wald tests”, which are most easily set up using the Regression Tests wizard (the “Exclusion Tests” choice within that) after running the GARCH model.

This time the GARCH model will be run on a one-lag VAR:

```
system(model=mvmean)
variables reuro rpound rsw
lags 1
det constant
end(system)
garch(model=mvmean,mv=bekk,robusterrors,pmethod=simplex,piters=20,$
      method=bfgs, iters=500, rvector=rd, hmatrices=hh)
```

The first test is for exogeneity in the mean for all the variables (basically, that the mean model could have been separate autoregressions on each variable):

```
test(zeros,title="Test of Exogeneity in Mean of All Variables")
# 2 3 5 7 9 10
```

which rather strongly rejects the null:

Test of Exogeneity of All Variables Chi-Squared(6)= 29.165801 or F(6,*)= 4.86097 with Significance Level 0.00005660
--

Note, however, that because these are all exchange rates in terms of the US dollar, it's not all that surprising to find “causality” among them, as the use of multiple series will make it simpler to distinguish between news that affects mainly a single currency, and news that affects all of them (through the effect on the relative value of the dollar).

Exogeneity tests in the mean for each of the three currencies require excluding lags of the other two variables:

```
test(zeros,title="Test of Exogeneity in Mean of Euro")
# 2 3
*
test(zeros,title="Test of Exogeneity in Mean of Pound")
# 5 7
*
test(zeros,title="Test of Exogeneity in Mean of Swiss Franc")
# 9 10
```

all of which are strongly significant at conventional levels.

Moving on to the tests of the variance parameters, a test for whether there is any spillover/causality in the variance for any of the series requires excluding all the “non-diagonal” elements of both **A** and **B**. Diagonality of **A** alone is not enough, as that would only be testing for first-period effects—if **B** is not diagonal, there can be effects at multiple steps.

```
test(zeros,title="Wald Test of Diagonal BEKK")
# 20 21 22 24 25 26 29 30 31 33 34 35
```

Wald Test of Diagonal BEKK Chi-Squared(12)= 94.838308 or F(12,*)= 7.90319 with Significance Level 0.0000

The block exclusions for the variance are:

```

test(zeros,title="Block Exclusion Test, Euro Variance")
# 22 25 31 34
*
test(zeros,title="Block Exclusion Test, Pound Variance")
# 20 26 29 35
*
test(zeros,title="Block Exclusion Test, Swiss Franc Variance")
# 21 24 30 33

```

Note that these are excluding both the **A**'s and the **B**'s for both the other variables—if you don't, you could end up missing multiple-step effects—if those four coefficients are zero, shocks to the other variables can't (according to the model) affect the variance of interest. Note also that, because of the way the BEKK matrices are set up (with the transpose on the pre-multiplication term), the coefficients that hit variance i have column (not row) i , so the test for the Euro (variable 1) is a joint test for $A(2, 1)$, $A(3, 1)$, $B(2, 1)$ and $B(3, 1)$. The only one of these that isn't *strongly* significant is for the Swiss Franc (.019 significance level).

5.6 CC Models: Constant Correlation

An alternative approach for creating a model which is easier to fit than the DVECH model is the *Constant Correlation* (or CC) model of Bollerslev (1990). The DVECH model uses a simple GARCH model for the variances—the numerical problems arise from the lack of connection between the variance recursions and the covariance recursions. The CC model assumes that the covariances are generated with a constant (but unknown) correlation. Whether this is *too* restrictive will depend upon the application: from Figure 5.2, a constant correlation between the Swiss Franc and Euro doesn't seem unreasonable, but it appears to be less likely to do well explaining the relationships of those with the Pound.

The number of free parameters is reduced quite a bit. **C**, **A** and **B** are now just n vectors, and the sub-diagonal of the correlation matrix has only $n(n-1)/2$ free parameters—in our model, that's 3 for a total of 12. The estimation typically behaves well enough that it isn't even necessary to choose a parameterization of the correlation matrix which enforces positive-definiteness—in practice, it doesn't stray close enough to the region where the correlation matrix becomes singular to need that extra protection.

This is estimated with **RATS** using the `MV=CC` option. You typically don't need any help from simplex iterations, so:

```
garch(model=mvmean,mv=cc,rvectors=rd,hmatrices=hh,robusterrors)
```

which gives us Table 5.6. The likelihood is quite a bit worse than for the DVECH and BEKK models, so it seems that the graphical evidence that constant correlation is inappropriate is confirmed.

Table 5.6: Multivariate GARCH, CC Estimates

MV-GARCH, CC - Estimation by BFGS					
Convergence in 50 Iterations. Final criterion was 0.0000017 <= 0.0000100					
With Heteroscedasticity/Misspecification Adjusted Standard Errors					
Daily(5) Data From 2000:01:04 To 2008:12:23					
Usable Observations		2341			
Log Likelihood		-3053.8808			
	Variable	Coeff	Std Error	T-Stat	Signif
1.	Constant	0.0313	0.0136	2.2933	0.0218
2.	Constant	0.0168	0.0124	1.3598	0.1739
3.	Constant	0.0289	0.0157	1.8327	0.0668
4.	C(1)	0.0045	0.0012	3.6068	0.0003
5.	C(2)	0.0048	0.0019	2.5703	0.0102
6.	C(3)	0.0059	0.0016	3.5804	0.0003
7.	A(1)	0.0547	0.0083	6.5670	0.0000
8.	A(2)	0.0496	0.0093	5.3639	0.0000
9.	A(3)	0.0460	0.0064	7.1888	0.0000
10.	B(1)	0.9344	0.0093	100.3212	0.0000
11.	B(2)	0.9343	0.0129	72.2737	0.0000
12.	B(3)	0.9416	0.0071	133.3927	0.0000
13.	R(2,1)	0.7082	0.0110	64.5446	0.0000
14.	R(3,1)	0.9180	0.0048	193.1722	0.0000
15.	R(3,2)	0.6503	0.0142	45.9335	0.0000

Of course, in practice, you would start with the simpler CC. The multivariate diagnostics point to the inadequacy of the model, with a highly significant test for lack of residual ARCH:

```

Multivariate Q(10)=      108.10427
Significance Level as Chi-Squared(90)=      0.09389

Test for Multivariate ARCH
Statistic Degrees Signif
353.00      72 0.00000

```

So if we did these in the more standard order of simplest first, it would point us towards the need for a more complicated model. Tse (2000) provides an LM test for CC against an alternative that the correlation that allows greater adaptation to the observed (lagged) outer product of the residuals. This is implemented with the procedure `@TseCCTest`. It requires that you first estimate the CC model, saving the `VECTOR[SERIES]` of derivatives of the log likelihood with respect to the free parameters. This requires the `DERIVES` option, used before in Section 4.3 in doing fluctuations tests. It also needs the sequences of residuals and variances that we're already saving. For this model, we do:

```

garch(model=mvmean,mv=cc,rvectors=rd,hmatrices=hh,derives=dd)
@tsecctest(rvector=rd,hmatrices=hh,derives=dd)

```

which produces the somewhat disappointing (given what we know) result:

```

Tse Test for CC
Chi-Squared(3)=      7.712254 with Significance Level 0.05234836

```

Table 5.7: Multivariate GARCH, CC with Spillover

MV-GARCH, CC with Spillover Variances - Estimation by BFGS					
Convergence in 64 Iterations. Final criterion was 0.0000087 <= 0.0000100					
With Heteroscedasticity/Misspecification Adjusted Standard Errors					
Daily(5) Data From 2000:01:04 To 2008:12:23					
Usable Observations		2341			
Log Likelihood		-3032.9529			
	Variable	Coeff	Std Error	T-Stat	Signif
1.	Constant	0.0272	0.0085	3.1882	0.0014
2.	Constant	0.0144	0.0091	1.5938	0.1110
3.	Constant	0.0240	0.0098	2.4594	0.0139
4.	C(1)	0.0049	0.0011	4.5878	0.0000
5.	C(2)	0.0047	0.0016	2.9046	0.0037
6.	C(3)	0.0064	0.0015	4.3300	0.0000
7.	A(1,1)	0.0898	0.0135	6.6510	0.0000
8.	A(1,2)	-0.0244	0.0096	-2.5541	0.0106
9.	A(1,3)	-0.0265	0.0094	-2.8074	0.0050
10.	A(2,1)	0.0451	0.0114	3.9589	0.0001
11.	A(2,2)	0.0386	0.0092	4.1735	0.0000
12.	A(2,3)	-0.0457	0.0095	-4.8358	0.0000
13.	A(3,1)	0.0334	0.0156	2.1460	0.0319
14.	A(3,2)	-0.0180	0.0110	-1.6299	0.1031
15.	A(3,3)	0.0240	0.0101	2.3768	0.0175
16.	B(1)	0.9399	0.0078	121.0810	0.0000
17.	B(2)	0.9462	0.0095	99.0982	0.0000
18.	B(3)	0.9435	0.0067	139.8309	0.0000
19.	R(2,1)	0.7120	0.0113	63.0283	0.0000
20.	R(3,1)	0.9181	0.0045	202.9850	0.0000
21.	R(3,2)	0.6556	0.0144	45.5062	0.0000

Since we've estimated a more general model and found that it fit *much* better, the fact that the Tse test doesn't pick up on that is unfortunate. It's an LM test against what would appear in this case to be an alternative which doesn't define a sharp enough contrast. The lesson is not to rely solely upon the Tse test as a diagnostic for the CC model.

One nice feature of the CC framework is that the GARCH models for the variances can take almost any form. For instance, a multivariate EGARCH model is created by using EGARCH models for the variances with the CC to handle the covariances. To choose this, add the option `VARIANCES=EXP` to the **GARCH** instruction:

```
garch (model=mvmean,mv=cc,variances=exp,$
rvectors=rd,hmatrices=hh,robusterrors)
```

which, here, fits even worse (log likelihood -3080.4548).

An alternative variance model which allows for a greater interaction is `VARIANCES=SPILLOVER`, which adds "spillover" terms to the variance calcu-

lation:

$$h_{ii,t} = c_{ii} + \sum_j a_{ij} u_{j,t-1}^2 + b_i h_{ii,t-1}$$

```
garch(model=mvmean,mv=cc,variances=spillover,$
      rvector=rd,hmatrices=hh,robusterrors,$
      pmethod=simplex,piters=20,itters=500)
```

This is no longer a “simpler” model than the DVECH—in this case, it has exactly the same number of free parameters (21). It provides greater flexibility in the variances, but less in the covariances. This produces some interesting results (Table 5.7).

First, despite also having 21 parameters, the fit is much worse than the DVECH. And note that most of the off-diagonal parameters in \mathbf{A} are *negative*. Because these are used directly (not squared, as in the BEKK), this would seem to imply that spillover is negative rather than positive. However, this may be due to a misspecification of the spillover effect. If it is the case that volatility goes up more when the shocks have opposite signs, as we seemed to see in the BEKK model, that can’t be captured by this model which only uses the magnitudes. And, in fact, if it’s a linear combination with differing signs whose square matters, then it turns out that the closest (though not very close) fit to that given only the squares has a negative coefficient on one of them.

5.7 DCC Models: Dynamic Conditional Correlation

The CC model has several advantages over the “vech” forms:

1. It doesn’t have the problem of possible non-positive definite covariance matrices of the VEC and DVECH.
2. It allows more flexibility in handling the variances than BEKK, where the positive-definiteness comes at a cost of a very specific variance recursion.
3. It can handle larger sets of series, as the number of parameters doesn’t increase as fast with n .

But that comes at the major drawback of assuming that the conditional correlation is constant. It didn’t take long for models to be proposed which relaxed that assumption. If we have a model which, at time t , produces an n vector of variances h_t , and an $n \times n$ correlation matrix \mathbf{R}_t (positive-definite symmetric matrix with 1’s on the diagonal), then \mathbf{R}_t and h_t can be converted into a covariance matrix \mathbf{H} by

$$H_{t,ij} = \sqrt{h_{t,i}} \sqrt{h_{t,j}} R_{t,ij}$$

If $n = 2$, there are many ways to produce such an \mathbf{R}_t , as there is only one free value in it (since it's symmetric with 1's on the diagonal), and it's positive definite if and only if the (1, 2) element has absolute value less than 1. For instance, if it is expected that the correlation would be monotonic in some exogenous variable Z , then a logistic mapping such as

$$\log \left(\frac{1 + \rho_t}{1 - \rho_t} \right) = \gamma + \delta Z_t$$

can be used, at least as an approximation.⁸ And it's possible to take this idea and make the logistic index a function of the past residuals, so the correlation adapts to the recent past of the data.

However, the logistic mapping doesn't generalize well to more than $n = 2$. Engle (2002) proposed a more general method which he labeled *Dynamic Conditional Correlations* which can be applied to an arbitrary number of series.⁹ This uses a GARCH-like secondary recursion that generates a symmetric matrix which is converted into the correlation matrix:

$$\begin{aligned} \mathbf{Q}_t &= (1 - a - b)\bar{\mathbf{Q}} + b\mathbf{Q}_{t-1} + a\varepsilon_{t-1}\varepsilon'_{t-1} \\ \mathbf{R}_t &= \text{diag}(\mathbf{Q}_t)^{-1/2}\mathbf{Q}_t\text{diag}(\mathbf{Q}_t)^{-1/2} \end{aligned} \quad (5.7)$$

where $\bar{\mathbf{Q}}$ is a fixed matrix towards which the \mathbf{Q} matrices “shrink”. ε_{t-1} is the lagged univariate standardized residual, that is, each component is divided by its own estimated standard deviation. If the GARCH models are correct, the *expected value* of $\varepsilon_{t-1}\varepsilon'_{t-1}$ is a “correlation matrix”:

1. it *has* to be positive semi-definite (it's rank one, so it is only semi-definite)
2. the expected values of the diagonal elements are 1

Since $\bar{\mathbf{Q}}$ and the pre-sample \mathbf{Q} (which is made equal to $\bar{\mathbf{Q}}$) will be positive definite by construction, as long as b is non-zero, each \mathbf{Q}_t will be positive definite as well. A weighted average like this of actual correlation matrices would also be a correlation matrix, but because the final term is only a correlation matrix in expected value, \mathbf{Q}_t itself isn't a correlation matrix, but as a positive definite symmetric matrix can be converted to one as in (5.7).

In Engle's procedure,

$$\bar{\mathbf{Q}} = \frac{1}{T} \sum_t \varepsilon_t \varepsilon'_t \quad (5.8)$$

that is, it's the average outer product of the standardized residuals. One problem with this in practice is that ε_t , as the standardized residual, depends upon the estimated GARCH variances. While each individual ε_{t-1} in (5.7) will be

⁸This takes a logistic and remaps it from (0,1) to (-1,1).

⁹Note that the phrase “Dynamic Conditional Correlations” has also been applied in some papers to *other* methods of generating time-varying correlations.

known when they're needed, the \bar{Q} depends upon the entire data range (and would change with changes to the univariate GARCH parameters). That specific calculation is really only feasible with the “two-step” procedure that Engle describes:

1. fit univariate GARCH processes to all the series
2. take the residuals and variances from those as given, compute \bar{Q} (which will now be a fixed matrix once the residuals and variances are given) and estimate the a and b values and generate the time-varying correlations with them.

Engle shows that the two-step procedure is consistent, though inefficient (since the univariate GARCH estimates aren't taking into account the correlations with the other processes). However, it can feasibly be applied to rather large sets of series because there are only two free parameters in the multivariate maximization—the univariate GARCH models are estimated separately.

One major drawback to the two-step procedure (inefficiency aside) is that it can only be used if the univariate GARCH models are self-contained. The models designed for use with CC or DCC that provide for some type of “spillover” in computing the variances can't be used since they need joint estimation anyway to generate all the information needed.

To allow for feasible use of DCC for a broader set of models, in **GARCH** with the option `MV=DCC`, the \bar{Q} matrix is computed using the sample correlation matrix of (non-standardized) residuals from a preliminary estimate of the mean model. In practice, the difference is likely to be minor.¹⁰

An alternative for generating the correlation matrices which is, in fact, what the **GARCH** instruction does by default, is to do the recursion (5.7) using covariance matrices rather than correlations—instead of the standardized residuals ε_{t-1} , it uses the equation residuals u_t and instead of \bar{Q} being the sample correlation matrix, it's the sample *covariance* matrix. In practice, the difference ends up being minor, though when it isn't, the recursion in the covariances is generally the one that does better. The recursion in correlations gives greater weight to entries which are outliers according to the GARCH model (large *standardized* residuals), while the recursion in covariances puts greater weight on entries which have large non-standardized residuals—in most cases, the two are the same. The choice between the two is controlled by the option `DCC=COVARIANCES` (default) or `DCC=CORRELATIONS`.¹¹

A third option is the “corrected DCC” of Aielli (2013). Aielli shows that the original Engle recursion isn't internally consistent (in the non-statistical sense)—the expectation of $\varepsilon_{t-1}\varepsilon'_{t-1}$ isn't Q_{t-1} as it would be if (5.7) were a GARCH model,

¹⁰The effect of the \bar{Q} matrix is short-lived unless a and b are fairly small, which generally means that a CC model would have been a better choice.

¹¹These were added with Version 10.

and Engle's \bar{Q} isn't the stationary solution of (5.7) (unless the true DGP is CC rather than DCC). The corrected DCC scales up ε_{t-1} by the (square roots of) the diagonal elements of Q_{t-1} . You get this by using the option `DCC=CDCC`.

While DCC is a common choice, particularly with larger numbers of series, there is no GARCH model which works well under all conditions, and this is no exception:

1. If CC is correct, then the parameters of DCC aren't identified: any combination with $a = 0$ and any value of b will produce the same correlation matrix all the way through the data set, since the presample value of Q is the same as \bar{Q} .
2. If the correlations are small (that is, near zero), even if they aren't fixed, it will be hard to estimate the DCC parameters because the log likelihood is very flat near zero.
3. Because DCC is using only one pair of free parameters for an n variable correlation subsystem, if you have many series and they have very different dynamics, the DCC may end up overdoing the "dynamics" in some pairs to fit the others reasonably.

You should always do a CC model as well (first!) as it may give you some information as to whether you are running into problems with 1 and 2. Note that CC and DCC don't really nest—DCC uses just the two parameters to generate the correlation matrix, as the \bar{Q} isn't freely estimated.

Example 5.6 estimates a standard DCC-GARCH model (that is, with standard GARCH models for the univariate variance estimates) to the Enders data set used in the other examples in this chapter.

```
garch(model=mvmean,mv=dcc,robusterrors,$
      rvector=rd,hmatrices=hh,stdresids=rstd,factorby=eigen,$
      iters=500,pmethod=simplex,piters=10)
```

The output is in Table 5.8. $DCC(A)$ and $DCC(B)$ are the a and b parameters from the recursion to generate the correlation matrix. When DCC "works", this is the type of pattern you will generally see— a is fairly small (typically under .1) and b is large, with the two generally summing to somewhere above .9, often, as here, nearly to 1. If we compare the log likelihood to the other models, we see that this is *much* better than CC, is somewhat worse than DVECH and BEKK, but is better than both of those if we apply the SBC as it has many fewer parameters than those other models.

A common task is to graph the conditional correlations generated by the model. Note that all other forms of multivariate GARCH models (other than CC) will also produce time-varying correlations, which can be computed and displayed in exactly the same way—we already did this in Example 5.2.

Table 5.8: DCC GARCH Model

MV-DCC GARCH - Estimation by BFGS				
Convergence in 74 Iterations. Final criterion was 0.0000071 <= 0.0000100				
With Heteroscedasticity/Misspecification Adjusted Standard Errors				
Daily(5) Data From 2000:01:04 To 2008:12:23				
Usable Observations		2341		
Log Likelihood		-2774.2692		
Variable	Coeff	Std Error	T-Stat	Signif
Mean Model(REURO)				
1. Constant	0.0256	0.0087	2.9312	0.0034
Mean Model(RPOUND)				
2. Constant	0.0154	0.0096	1.6041	0.1087
Mean Model(RSW)				
3. Constant	0.0210	0.0098	2.1345	0.0328
4. C(1)	0.0024	0.0010	2.4998	0.0124
5. C(2)	0.0031	0.0014	2.2454	0.0247
6. C(3)	0.0031	0.0012	2.6588	0.0078
7. A(1)	0.0408	0.0067	6.1000	0.0000
8. A(2)	0.0481	0.0059	8.0903	0.0000
9. A(3)	0.0373	0.0049	7.5720	0.0000
10. B(1)	0.9539	0.0083	115.2548	0.0000
11. B(2)	0.9438	0.0087	108.2947	0.0000
12. B(3)	0.9568	0.0065	146.6784	0.0000
13. DCC(A)	0.0226	0.0059	3.8156	0.0001
14. DCC(B)	0.9752	0.0071	136.4902	0.0000

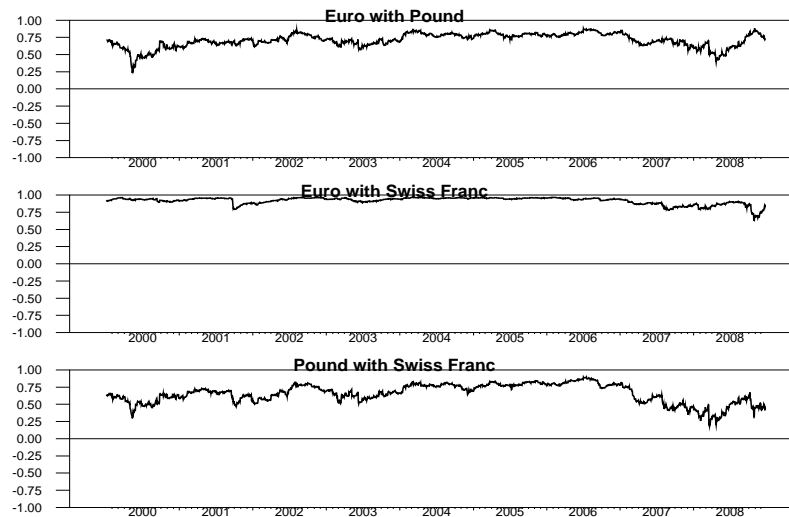


Figure 5.3: DCC Correlations

With two or three series, it's generally easiest to just set up the correlation graphs directly:

```
set r_euro_pound %regstart() %regend() = %cvtocorr(hh(t)) (1,2)
set r_euro_sw    %regstart() %regend() = %cvtocorr(hh(t)) (1,3)
set r_pound_sw   %regstart() %regend() = %cvtocorr(hh(t)) (2,3)
spgraph(vfields=3, footer="Conditional Correlations from DCC")
  graph(header="Euro with Pound", min=-1.0, max=1.0)
  # r_euro_pound
  graph(header="Euro with Swiss Franc", min=-1.0, max=1.0)
  # r_euro_sw
  graph(header="Pound with Swiss Franc", min=-1.0, max=1.0)
  # r_pound_sw
spgraph(done)
```

This (Figure 5.3) uses -1 to 1 ranges on all the series. If you don't include those, you could easily have graphs which overemphasize what may be very small changes to correlations. For instance, the Euro with the Swiss Franc, over almost the entire range (other than the last year) shows almost no change at all, but without pinning the range, the (actually fairly minor) dip in late 2001 would be spread across almost the entire vertical range of the graph. It's important not to overanalyze the movements in these. We'll see later Section 10.2.3) how it is possible to put some type of error band on these, but remember that these are just point estimates.

An alternative to manually putting together a correlation graph is the following, which generates the same graph programmatically using "long labels" input for the series. This is useful if you have more than 3 series (though the graphs get rather crowded above 5), or if you're using 2 or 3 series from a longer list.


```
compute totalfields=%nvar*(%nvar-1)/2
compute hfields=fix(sqrt(totalfields))
compute vfields=(totalfields-1)/hfields+1
spgraph(vfields=vfields,hfields=hfields,$
  footer="Conditional Correlations from DCC")
do i=1,%nvar
  do j=i+1,%nvar
    set ccorr = %cvtocorr(hh(t))(i,j)
    graph(header=longlabel(i)+" with "+longlabel(j),$
      min=-1.0,max=1.0)
    # ccorr
  end do j
end do i
spgraph(done)
```

5.8 RATS Tips and Tricks

5.8.1 Graphics with Multiple Series

It's very common in working with multivariate GARCH models to need something similar to Figure 5.1: a graph with multiple fields of relatively similar types of data. There are a number of decisions you need to make in trying to make these presentable for publication. One is how to label the fields. Here, we did it by using the `YLABELS` option on the enclosing **SPGRAPH**, which labels them in the left margin, as shown in the figure. The most common alternative is to include a label on each **GRAPH** instruction. In many cases, the graphs are simply lettered with (a), (b) and (c), with descriptions in the captions. For three series, it's probably easiest to do three separate **GRAPH** instructions, rather than using the **DOFOR** loop:

```
spgraph(vfields=3)
  graph(picture="*.#",hlabel="(a)")
  # reuro
  graph(picture="*.#",hlabel="(b)")
  # rpound
  graph(picture="*.#",hlabel="(c)")
  # rsw
spgraph(end)
```

(or the same with more descriptive titles). It usually looks best to use an `HLABEL` (which puts the information below each graph) rather than a `HEADER` (which puts it above). However, labeling on the left margin in some form typically gives a better appearance. There are two problems with inserting the labels between graphs

1. It's not clear (to the reader) at a quick glance to which graph the label applies.
2. The individual graphs are already disproportionately wide. Inserting labels between graphs makes them even smaller in the vertical direction, while marginal labels make them narrower horizontally.

Aside from the layout, as you can see in Figure 5.1, the labeling of the values on the vertical axis varies from graph to graph, which can look a bit odd, particularly when the scales aren't actually that different. We used the `PICTURE` option on the **GRAPH** instructions in the example to force them to at least align the same, even if the values were different.¹² However, where the series are as similar in scale as these are, a better solution both visually, and for interpretation, is to force them onto a common scale. The easiest way to do that is to

¹²The pound has labels at 2.5 and -2.5 and thus needs a digit right of the decimal, while the other two can (and by default will) just use integers.

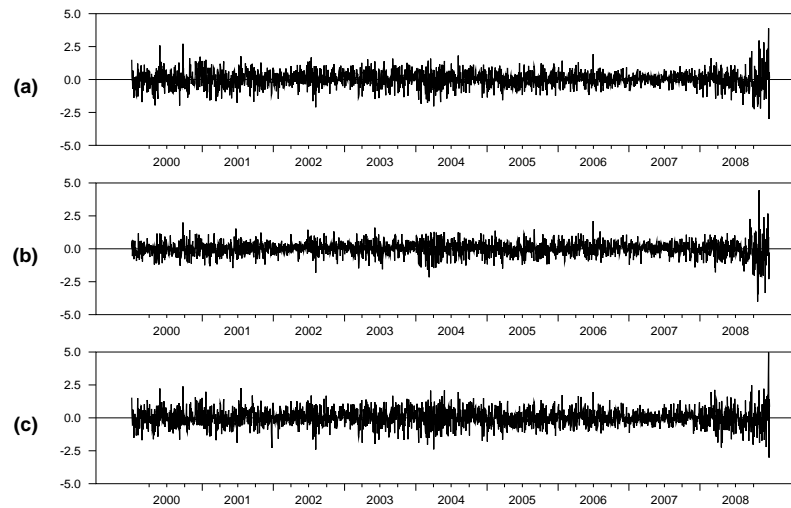


Figure 5.4: Exchange Rate Returns, Prettied Up

use the **TABLE** instruction, which sets the variables `%MAXIMUM` and `%MINIMUM` as the overall maximum and minimum values of all the series covered. If we switch now to marginal labels with (a), (b), and (c):

```
table / reuro rpound rsw
spgraph(vfields=3,ylabels=|| "(a)", "(b)", "(c)" ||)
do for r = reuro rpound rsw
    graph(min=%minimum,max=%maximum)
    # r
end do for
spgraph(done)
```

we get Figure 5.4, which has a much cleaner appearance.

5.8.2 Fancy Table of Diagnostics

To make an interesting report out of this, we will need more descriptive labels of the currencies, which we can get with:

```
dec vect[string] longlabel(3)
compute longlabel=|| "Euro", "UK Pound", "Swiss Franc" ||
```

A first go at this would insert a few **REPORT** instructions (and delete the **DISPLAY**).

Table 5.9: Sample Univariate Diagnostics

Currency	Q	Signif	ARCH	Signif
Euro	8.734	0.558	16.303	0.038
UK Pound	13.556	0.194	9.688	0.288
Swiss Franc	4.448	0.925	9.149	0.330

```

report(action=define)
report(atrow=1,atcol=1) "Currency" "Q" "Signif" "ARCH" "Signif"
do i=1,%nvar
  set ustd  = rd(t)(i)/sqrt(hh(t)(i,i))
  set ustdsq = ustd^2
  @regcorrs(number=10,nocrits,nograph,qstat) ustd
  compute q1=%cdstat,q1signif=%signif
  @regcorrs(number=10,nocrits,nograph,qstat,dfc=2) ustdsq
  compute q2=%cdstat,q2signif=%signif
  report(atrow=i+1,atcol=1) longlabel(i) q1 q1signif q2 q2signif
end do i
report(action=show)

```

Without additional formatting, that produces:

Currency	Q	Signif	ARCH	Signif
Euro	8.733736	0.557546	16.303304	0.038239
UK Pound	13.556423	0.194203	9.688304	0.287588
Swiss Franc	4.447671	0.924901	9.149159	0.329870

We need to show fewer digits (3 is probably good for everything), and it would look better to center the column headers over the numerical columns. However, the “Currency” header should still be left-justified. Two **REPORT** instructions with **ACTION=FORMAT** will take care of this. These are inserted just before the **REPORT (ACTION=SHOW)**:

```

report(action=format,align=center,atrow=1,torow=1,atcol=2)
report(action=format,picture="*.###")

```

On the first of these, we need to make sure the **ALIGN=CENTER** only applies where we want it: **ATROW=1** and **TOROW=1** limit it to row 1, and **ATCOL=2** (without **TOCOL**) limits it to columns from 2 until the end. We don’t have to worry about that with the second, since **PICTURE** only applies to numerical cells. The end result is Table 5.9.

Example 5.1 Multivariate GARCH: Preliminaries/Diagnostics

This includes the programs from Sections 5.1, 5.2 and 5.3.

```

open data "exrates(daily).xls"
calendar(d) 2000:1:3
data(format=xls,org=columns) 2000:01:03 2008:12:23 $
    aust euro pound sw ca
*
set reuro = 100.0*log(euro/euro{1})
set rpound = 100.0*log(pound/pound{1})
set rsw = 100.0*log(sw/sw{1})
*
spgraph(vfields=3,ylabels=||"Euro","Pound","Swiss Franc"||)
    dofor r = reuro rpound rsw
        graph(picture="*.#")
        # r
    end dofor
spgraph(done)
*
* Look at possible lag lengths
*
@varlagselect(crit=bic,lags=5)
# reuro rpound rsw
*
system(model=mvmean)
variables reuro rpound rsw
lags
det constant
end(system)
*
* Test for ARCH effects
*
estimate(resids=resids)
@mvarchtest
# resids
*
garch(model=mvmean,mv=diag,p=1,q=1,$
    rvectors=rd,hmatrices=hh,stdresids=rstd, factorby=eigen)
*
* Univariate diagnostics
*
do i=1,%nvar
    set ustd = rd(t)(i)/sqrt(hh(t)(i,i))
    set ustdsq = ustd^2
    @regcorrs(number=10,nocrits,nograph,qstat) ustd
    compute q1=%cdstat,q1signif=%signif
    @regcorrs(number=10,nocrits,nograph,qstat,dfc=2) ustdsq
    compute q2=%cdstat,q2signif=%signif
    disp q1 q1signif q2 q2signif
end do i
*
* Multivariate diagnostics

```

```

*
@mvqstat(lags=10)
# rstd
@mvarchtest(lags=2)
# rstd

```

Example 5.2 Multivariate GARCH: DVECH Estimates

This is the program from Section 5.4.1.

```

open data "exrates(daily).xls"
calendar(d) 2000:1:3
data(format=xls,org=columns) 2000:01:03 2008:12:23 $
    aust euro pound sw ca
*
set reuro = 100.0*log(euro/euro{1})
set rpound = 100.0*log(pound/pound{1})
set rsw = 100.0*log(sw/sw{1})
*
dec vect[string] longlabel(3)
compute longlabel=||"Euro","UK Pound","Swiss Franc"||
*
system(model=mvmean)
variables reuro rpound rsw
lags
det constant
end(system)
*
garch(model=mvmean,rvectors=rd,hmatrices=hh,robusterrors,$
    pmethod=simplex,piters=20,method=bfgs,itters=500)
*
dec vect meanparms(%nregmean)
dec symm cdvech(%nvar,%nvar) advech(%nvar,%nvar) bdvech(%nvar,%nvar)
nonlin(parmset=dvech) meanparms cdvech advech bdvech
compute %parmspoke(dvech,%beta)
*
* Display the sum of the A and B matrices
*
disp advech+bdvech
*
* Extract the time-varying correlations and the variances
*
dec symm[series] hhx(%nvar,%nvar)
do i=1,%nvar
    set hhx(i,i) = hh(t)(i,i)
    do j=1,i-1
        set hhx(i,j) = %cvtocorr(hh(t))(i,j)
    end do j
end do i
*
* This is specific to a 3 variable system

```

```

*
table / hhx(2,1) hhx(3,1) hhx(3,2)
compute corrmin=%minimum
*
table / hhx(1,1) hhx(2,2) hhx(3,3)
compute varmax=%maximum
*
spgraph(vfields=%nvar,hfields=%nvar,$
        xlabel=longlabel, ylabel=longlabel)
do i=1,%nvar
  do j=1,%nvar
    if i==j {
      graph(row=i,col=i,maximum=varmax)
      # hhx(i,i)
    }
    else {
      graph(row=i,col=j,maximum=1.0,min=corrmin)
      # hhx(i,j)
    }
  end do j
end do i
spgraph(done)

```

Example 5.3 Multivariate GARCH: BEKK Estimates

This is the program from Section 5.4.2.

```

open data "exrates(daily).xls"
calendar(d) 2000:1:3
data(format=xls,org=columns) 2000:01:03 2008:12:23 $
  aust euro pound sw ca
*
set reuro = 100.0*log(euro/euro{1})
set rpound = 100.0*log(pound/pound{1})
set rsw = 100.0*log(sw/sw{1})
*
dec vect[string] longlabel(3)
compute longlabel=||"Euro","UK Pound","Swiss Franc"||
*
system(model=mvmean)
variables reuro rpound rsw
lags
det constant
end(system)
*
garch(model=mvmean,mv=bekk,robusterrors,pmethod=simplex,piters=20,$
      method=bfgs, iters=500, rvector=rd, hmatrices=hh,$
      stdresids=rstd, factorby=eigen)
*
* Convert to equivalent VECH representation
*

```

```

@MVGARCHToVECH(mv=bekk)
disp ###.### %%vech_a
*
* Display eigenvalues of persistence matrix
*
eigen(cvalues=cv) %%vech_a%%vech_b
disp ###.### cv
*
@mvqstat(lags=10)
# rstd
@mvarchtest(lags=2)
# rstd
*****
*
dec vect means(%nregmean)
dec packed cx(%nvar,%nvar)
dec rect ax(%nvar,%nvar) bx(%nvar,%nvar)
nonlin(parmset=garchparms) means cx ax bx
summarize(parmset=garchparms) bx(1,1)^2
summarize(parmset=garchparms) bx(1,1)*bx(1,2)*2.0
summarize(parmset=garchparms) bx(1,2)^2
*
compute ncomp=%nvar*(%nvar+1)/2
dec rect %%vech_bse(ncomp,ncomp)
do m=1,ncomp
  do n=1,ncomp
    compute i=%symmrow(m),j=%symmcol(m),$
             k=%symmrow(n),l=%symmcol(n)
    if k==l {
      summarize(noprint,parmset=garchparms) bx(i,k)*bx(j,l)
      compute %%vech_bse(m,n)=sqrt(%varlc)
    }
    else {
      summarize(noprint,parmset=garchparms) $
        bx(i,k)*bx(j,l)+bx(i,l)*bx(j,k)
      compute %%vech_bse(m,n)=sqrt(%varlc)
    }
  end do n
end do m
*
report(action=define,title="VECH-B from BEKK")
do j=1,ncomp
  report(atrow=1,atcol=j+1,align=center) $
    %string(%symmrow(j))+", "+%symmcol(j)
end do j
do i=1,ncomp
  report(atrow=2*i,atcol=1) %string(%symmrow(i))+", "+%symmcol(i)
  do j=1,ncomp
    report(atrow=2*i,atcol=j+1) %%vech_b(i,j)
    report(atrow=2*i+1,atcol=j+1,special=parens) %%vech_bse(i,j)
  end do j
end do i
report(action=format,atrow=2,picture="###.####",align=decimal)
report(action=show)

```


Example 5.4 Multivariate GARCH: Spillover Tests

This is the example from Section 5.5.

```

open data "exrates(daily).xls"
calendar(d) 2000:1:3
data(format=xls,org=columns) 2000:01:03 2008:12:23 $
    aust euro pound sw ca
*
set reuro = 100.0*log(euro/euro{1})
set rpound = 100.0*log(pound/pound{1})
set rsw = 100.0*log(sw/sw{1})
*
dec vect[string] longlabel(3)
compute longlabel=||"Euro","UK Pound","Swiss Franc"||
*
system(model=mvmean)
variables reuro rpound rsw
lags 1
det constant
end(system)
*
garch(model=mvmean,mv=bekk,robusterrors,pmethod=simplex,piters=20,$
    method=bfgs, iters=500, rvectors=rd, hmatrices=hh)
*
test(zeros,title="Test of Exogeneity in Mean of All Variables")
# 2 3 5 7 9 10
*
test(zeros,title="Test of Exogeneity in Mean of Euro")
# 2 3
*
test(zeros,title="Test of Exogeneity in Mean of Pound")
# 5 7
*
test(zeros,title="Test of Exogeneity in Mean of Swiss Franc")
# 9 10
*
test(zeros,title="Wald Test of Diagonal BEKK")
# 20 21 22 24 25 26 29 30 31 33 34 35
*
test(zeros,title="Block Exclusion Test, Euro Variance")
# 22 25 31 34
*
test(zeros,title="Block Exclusion Test, Pound Variance")
# 20 26 29 35
*
test(zeros,title="Block Exclusion Test, Swiss Franc Variance")
# 21 24 30 33

```

Example 5.5 Multivariate GARCH: CC Estimates

This is the example from Section 5.6.

```
open data "exrates(daily).xls"
calendar(d) 2000:1:3
data(format=xls,org=columns) 2000:01:03 2008:12:23 $
    aust euro pound sw ca
*
set reuro = 100.0*log(euro/euro{1})
set rpound = 100.0*log(pound/pound{1})
set rsw = 100.0*log(sw/sw{1})
*
dec vect[string] longlabel(3)
compute longlabel=||"Euro","UK Pound","Swiss Franc"||
*
system(model=mvmean)
variables reuro rpound rsw
lags
det constant
end(system)
*
garch(model=mvmean,mv=cc,robusterrors,$
    rvectors=rd,hmatrices=hh,stdresids=rstd,factorby=eigen)
*
* Multivariate diagnostics
*
@mvqstat(lags=10)
# rstd
@mvarchtest(lags=2)
# rstd
*
* Tse test for CC
* We need the derivatives of the log likelihood, which we save into
* the VECTOR[SERIES] called DD.
*
garch(model=mvmean,mv=cc,rvectors=rd,hmatrices=hh,derives=dd)
@tsecctest(rvector=rd,hmatrices=hh,derives=dd)
*
* EGARCH with CC
*
garch(model=mvmean,mv=cc,variances=exp,$
    rvectors=rd,hmatrices=hh,robusterrors)
*
* CC GARCH with spillover
*
garch(model=mvmean,mv=cc,variances=spillover,$
    rvectors=rd,hmatrices=hh,robusterrors,$
    pmethod=simplex,piters=20,itors=500)
```

Example 5.6 Multivariate GARCH: DCC Estimates

This is the example from Section 5.7.

```

open data "exrates(daily).xls"
calendar(d) 2000:1:3
data(format=xls,org=columns) 2000:01:03 2008:12:23 $
    aust euro pound sw ca
*
set reuro = 100.0*log(euro/euro{1})
set rpound = 100.0*log(pound/pound{1})
set rsw = 100.0*log(sw/sw{1})
*
dec vect[string] longlabel(3)
compute longlabel=||"Euro","UK Pound","Swiss Franc"||
*
system(model=mvmean)
variables reuro rpound rsw
lags
det constant
end(system)
*
garch(model=mvmean,mv=dcc,robusterrors,$
    rvectors=rd,hmatrices=hh,stdresids=rstd,factorby=eigen,$
    iters=500,pmethod=simplex,piters=10)
*
* Multivariate diagnostics
*
@mvqstat(lags=10)
# rstd
@mvarchtest(lags=2)
# rstd
*
* This does the conditional correlation graphs with hard coding
* (which probably is simplest with 2 or 3 series).
*
set r_euro_pound %regstart() %regend() = %cvtocorr(hh(t))(1,2)
set r_euro_sw %regstart() %regend() = %cvtocorr(hh(t))(1,3)
set r_pound_sw %regstart() %regend() = %cvtocorr(hh(t))(2,3)
spgraph(vfields=3,footer="Conditional Correlations from DCC")
graph(header="Euro with Pound",min=-1.0,max=1.0)
# r_euro_pound
graph(header="Euro with Swiss Franc",min=-1.0,max=1.0)
# r_euro_sw
graph(header="Pound with Swiss Franc",min=-1.0,max=1.0)
# r_pound_sw
spgraph(done)
*
* This does the conditional correlations for an arbitrary number of
* series using the <<longlabel>> VECTOR of STRINGS to manage the
* labels. This probably gets too cluttered when N is bigger than 5.
*
compute totalfields=%nvar*(%nvar-1)/2

```

```
compute hfields=fix(sqrt(totalfields))
compute vfields=(totalfields-1)/hfields+1
spgraph(vfields=vfields,hfields=hfields,$
  footer="Conditional Correlations from DCC")
do i=1,%nvar
  do j=i+1,%nvar
    set ccorr = %cvtocorr(hh(t))(i,j)
    graph(header=longlabel(i)+" with "+longlabel(j),$
      min=-1.0,max=1.0)
    # ccorr
  end do j
end do i
spgraph(done)
```

More on Multivariate GARCH

We'll now discuss adjustments to the basic multivariate GARCH models and other types of analysis which can be done with results from the **GARCH** instruction.

We'll look at three new data sets. For Sections 6.1 and 6.2 we will use (a reconstruction of) the data set from Hafner & Herwartz (2006) (from now on HH). The full data set (called `HHDATA.XLS`) has 3270 daily observations on ten exchange rates vs the dollar, running from 31 December 1979 to 1 April 1994. The data as used in the paper are expressed in local currency/USD, while the data on the file are USD/local currency, so we have to change the sign when computing the returns.¹ There's a separate date column, with the date coded numerically as a six digit number *yymmdd*. Although the data set includes all five days a week and so could be handled as **CALENDAR(D)** with RATS, we'll treat it as irregular, and show how to locate an entry based upon a coded date field like that. The data are read with

```
open data hhdata.xls
data(format=xls,org=columns) 1 3720 usxjpn usxfra usxsui $
  usxnld usxuk usxbel usxger usxswe usxcan usxita date
```

Our focus will be on bivariate models on returns for two of the currencies: the British pound and the Deutsche mark. The authors choose to use separate univariate autoregressions for the mean models—if you estimate a one lag VAR, the “other” lags have *t*-stats less than 1, so leaving them out isn't unreasonable. The mean model can be set up with

```
set demret = -100.0*log(usxger/usxger{1})
set gbpret = -100.0*log(usxuk/usxuk{1})
*
equation demeqn demret
# constant demret{1}
equation gbpeqn gbpret
# constant gbpret{1}
group uniar1 demeqn gbpeqn
```

The **GROUP** instruction combines the separate equations into a single model for input into **GARCH**. This is the only way to create a mean model with different

¹Though this has no effect on any variance calculations.

Table 6.1: BEKK Estimates for Hafner-Herwartz Data

MV-GARCH, BEKK - Estimation by BFGS					
Convergence in 54 Iterations. Final criterion was 0.0000075 <= 0.0000100					
Usable Observations		3718			
Log Likelihood		-5259.4997			
	Variable	Coeff	Std Error	T-Stat	Signif
1.	Constant	0.0086	0.0089	0.9677	0.3332
2.	DEMRET{1}	0.0035	0.0123	0.2853	0.7754
3.	Constant	-0.0030	0.0085	-0.3546	0.7229
4.	GBPRET{1}	0.0184	0.0125	1.4662	0.1426
5.	C(1,1)	0.0963	0.0122	7.9131	0.0000
6.	C(2,1)	0.0708	0.0138	5.1138	0.0000
7.	C(2,2)	-0.0456	0.0039	-11.7169	0.0000
8.	A(1,1)	0.2891	0.0235	12.2816	0.0000
9.	A(1,2)	-0.0072	0.0225	-0.3205	0.7486
10.	A(2,1)	-0.0518	0.0239	-2.1674	0.0302
11.	A(2,2)	0.2459	0.0209	11.7592	0.0000
12.	B(1,1)	0.9565	0.0071	134.6023	0.0000
13.	B(1,2)	0.0046	0.0065	0.7170	0.4734
14.	B(2,1)	0.0100	0.0075	1.3378	0.1810
15.	B(2,2)	0.9636	0.0055	175.5495	0.0000
16.	Shape	4.3861	0.2451	17.8967	0.0000

regressors—a *full* VAR *can* be done using the `REGRESSORS` option, as that puts the same set of regressors into each equation, though we would recommend the more convenient **SYSTEM** definition for the model.

They choose a BEKK model with t errors, which produces Table 6.1.

```
garch(model=uniar1,mv=bekk,rvectors=rd,hmatrices=hh,distrib=t,$
      pmethod=simplex,piters=20,iters=500)
```

If we compare this with the BEKK estimates in Table 5.4, off-diagonal elements for A and B are, in this data set, close to zero. A further restricted version of the BEKK called a diagonal BEKK or DBEKK appears to be appropriate. A DBEKK is also a restricted version of the DVECH where the coefficients on A and B in the covariance recursions are the geometric means of the coefficients on their corresponding variance recursions. This can be done using the option `MV=DBEKK`. Since it won't make much difference, we'll use the unrestricted BEKK in the examples in Sections 6.1 and 6.2.

6.1 Forecasting

For any model that can be cast into VEC form ((5.1)), the out-of-sample forecasts can be done using the recursion (5.2). As with univariate forecasts, this will require input of the estimates for the residuals and covariance matrices. The calculations can be done most easily using the procedure `@MVGARCHFore`.

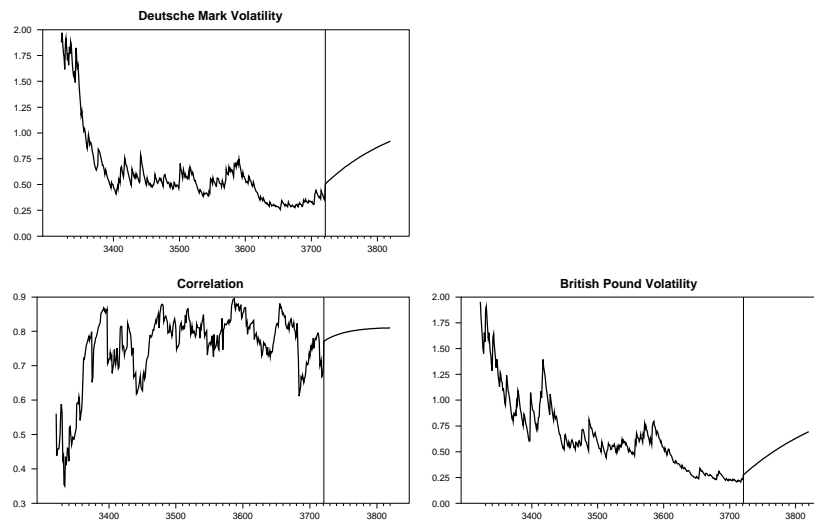


Figure 6.1: Forecasts from a BEKK GARCH Model

In Example 6.1, we'll use the BEKK model estimated above. To forecast out-of-sample for 100 steps, we use

```
@MVGARCHFore (mv=bekk, steps=100) hh rd
```

The `HH` and `RD` are input to the procedure, and `HH` is also used for the output. The following graphs the last 400 observed values and the 100 forecast steps for the two volatilities and the correlations, producing Figure 6.1:

```
spgraph(vfields=2,hfields=2)
compute gstart=%regend()-399,gend=%regend()+100
set h11fore gstart gend = hh(t) (1,1)
set h22fore gstart gend = hh(t) (2,2)
set h12fore gstart gend = hh(t) (1,2)/sqrt(h11fore*h22fore)
graph(row=1,col=1,grid=(t==%regend()+1),min=0.0,$
  header="Deutsche Mark Volatility")
# h11fore
graph(row=2,col=2,grid=(t==%regend()+1),min=0.0,$
  header="British Pound Volatility")
# h22fore
graph(row=2,col=1,grid=(t==%regend()+1),header="Correlation")
# h12fore
spgraph(done)
```

As you can see, there's a quick move on the D-Mark, which results from a fairly large residual in its last data point. If you graphed just the forecasts without the context of the actual data, the forecasts would appear to be explosive. However, the model has eigenvalues just below 1, and is simply (very slowly) converging back from the historically low volatility at the end of the sample towards the average.

6.2 Volatility Impulse Response Functions

Hafner & Herwartz (2006) introduced the concept of the *volatility impulse response function* (VIRF) for multivariate GARCH models. The recursion (5.1) is very similar to the one governing a one-lag VAR in the mean, and we generally describe the dynamics of a VAR in terms of its impulse responses or some information derived from them.

The calculation of the VIRF is not difficult for models which can be put into the Vech form (5.1), once we define what an “impulse” means in this context. For the IRF for a VAR, a shock is any non-zero u_t . Because of linearity, we can do responses to a standardized set of shocks (for instance, unit shocks to each variable in turn) and add or rescale them to get responses to any other set of shocks. IRF’s are usually presented as an $n \times n$ set of graphs, one for each combination of shock and target variable. For a GARCH model, that no longer will work, since the u_t enters as a “square” (actually as the outer product). Now a “unit” shock may be completely out-of-scale, and we can’t simply aggregate them anyway, since the inputs get squared before being used. Instead, VIRF’s need to be calculated as the responses to a complete vector of shocks.

HH describe several ways to create interesting sets of shocks to input to the recursion. To make sense, these must somehow be “typical” of the data. We pick either u_t and transform to $u_t u_t'$ or we pick $u_t u_t'$ directly. The difference in the volatility forecasts (5.2) with the input shocks (compared to $u_t = 0$) is then:

$$\begin{aligned} \text{vech}(\mathbf{v}_{t+1}) &= \mathbf{A} \text{vech}(\mathbf{u}_t \mathbf{u}_t') \\ \text{vech}(\mathbf{v}_{t+k}) &= (\mathbf{A} + \mathbf{B}) \text{vech}(\mathbf{v}_{t+k-1}) \end{aligned}$$

This defines what the authors call the *conditional volatility profile*. This is a function just of the model coefficients and the shock, and not the data. The VIRF itself is the same basic formula with a slightly different input. In the standard IRF, we’re computing the revision in the forecast due to observing the given shock. For the analogous idea in the volatility equation, we need to do the calculations as:

$$\begin{aligned} \text{vech}(\mathbf{V}_{t+1}) &= \mathbf{A} \text{vech}(\mathbf{u}_t \mathbf{u}_t' - \mathbf{H}_t) \\ \text{vech}(\mathbf{V}_{t+k}) &= (\mathbf{A} + \mathbf{B}) \text{vech}(\mathbf{V}_{t+k-1}) \end{aligned} \tag{6.1}$$

where \mathbf{H}_t is the GARCH covariance matrix for time t . The VIRF depends upon the data now through \mathbf{H}_t —the “shock” to the variance is the amount by which the $u_t u_t'$ exceeds its expected value. Of course, it’s possible for that to be negative, even on the diagonals, while the input to the conditional volatility profile has to have positive diagonals.

HH computed VIRF for two historical episodes, and we’ll show how to handle those. Note, however, that the data set is a reproduction, so the results don’t quite match. We are also scaling the data by 100 relative to what they do, which eliminates the need to rescale the responses themselves.

The model in Example 6.2 is the same as the one used in Section 6.1—a BEKK with t errors. To compute the VIRF, we need to convert the BEKK estimates to VECH. This also looks at the eigenvalues of the persistence matrix:

```
@MVGARCHtoVECH(mv=bekk)
eigen(cvalues=cv) %%vech_a+%%vech_b
disp "Eigenvalues from BEKK-t" * .### cv
```

The eigenvalues are just barely inside the stable region:

Eigenvalues from BEKK-t (0.994,-0.000) (0.993,0.004) (0.993,-0.004)

The shocks used in the two experiments are from “Black Wednesday” (September 16, 1992), when the Italian lira and the UK pound dropped out of the European Exchange Rate Mechanism (ERM), and August 2, 1993, when the EC finance ministers changed the variability bands on currencies in the ERM. The simplest way to locate those entries given the `DATE` field on the file is with:

```
sstats(max) / %if(date==920916,t,0)>>xblackwed $
               %if(date==930802,t,0)>>xecpolicy
compute blackwed=fix(xblackwed),ecpolicy=fix(xecpolicy)
```

Now, if we had a daily `CALENDAR` date scheme, we could simply use

```
compute blackwed=92:9:16
compute ecpolicy=93:8:2
```

for the entries, but we didn’t set that up with this data set, so we’re showing how to locate an entry given the coded date field. `XBLACKWED` and `XECPOLICY` are both `REAL` variables (which is what `SSTATS` returns), so the `COMPUTE` instruction is used to convert them to `INTEGER` entry numbers.

Example 6.2 wraps the calculations in an `SPGRAPH` which does two columns, one for each shock, with three fields in each, one for the each variance and one for the covariance. In practice, that would be the *last* thing you did—you would want to get the calculations right first before being concerned with the appearance of the output. So we’ll skip that for later and jump right to the calculation.

The description of the calculation for the VIRF in the paper is more complicated than it needs to be—they transform the observed residuals to a standardized vector using (in our notation) H_t , but the standardization gets reversed in putting it back into the recursion. Instead, you can just compute the final matrix in the first row of (6.1) using

```
compute eps0=rd(blackwed)
compute sigma0=hh(blackwed)
compute shock=%vec(%outerxx(eps0)-sigma0)
```

Because the VEC recursion is in *vech* form, we need to convert the shock from a matrix to a vector. %VEC does the VEC stacking as long as its argument is identifiable as a symmetric array. That's the case here because SIGMA0, as an element of HH, is SYMMETRIC and %OUTERXX creates a SYMMETRIC by construction. If you want to be careful, you can write %VEC([SYMMETRIC]...) to force the desired interpretation.

The calculation will generate 3 (that is, $n(n+1)/2$ —the %SIZE function gives the number of free elements in a matrix) output series over the requested number of steps (400 in this example, which has already been saved into the variable NSTEP). This creates a target set of SERIES for this:

```
dec vect[series] sept92virf(%size(sigma0))
do i=1,%size(sigma0)
    set sept92virf(i) 1 nstep = 0.0
end do i
```

The actual calculation is quite simple. HVEC is an $n(n+1)/2$ vector which, at each point, has the previous period's *vech*'ed response. This gets overwritten by the recalculated value. At each step, this gets split up into the SEPT92VIRF series using %PT.

```
do step=1,nstep
    if step==1
        compute hvec=%vech_a*shock
    else
        compute hvec=(%vech_a+%vech_b)*hvec
        compute %pt(sept92virf,step,hvec)
    end do step
```

The first column of graphs is done with the following, which has a few options added to improve the appearance:

```
do i=1,%size(sigma0)
    graph(column=1,row=i,picture="*.###",vticks=5)
    # sept92virf(i) 1 nstep
end do i
```

Note that because of the way the *vech* operator works, the *second* graph is the covariance between the two currencies.

The calculation of the VIRF for the EC policy shock is similar:

```
compute eps0=rd(ecpolicy)
compute sigma0=hh(ecpolicy)
compute shock=%vec(%outerxx(eps0)-sigma0)
```

Given that, the rest of the calculation is the same as above, with results put into a different VECTOR[SERIES]:

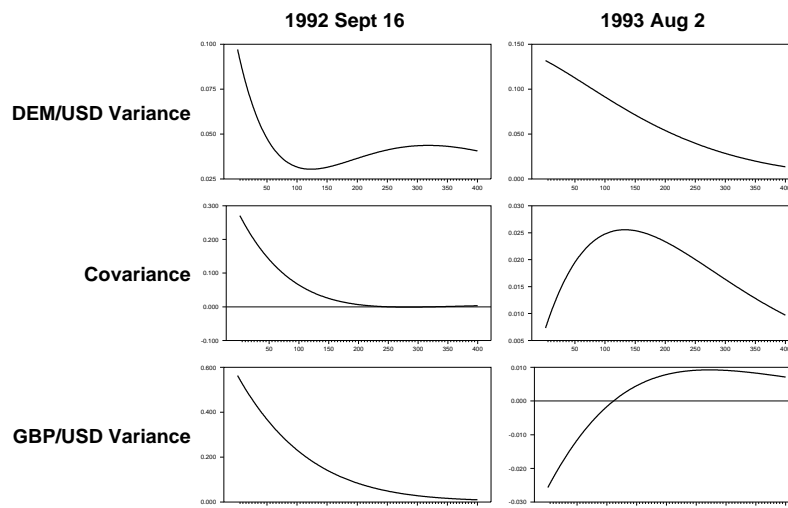


Figure 6.2: Volatility Impulse Response Functions

```

dec vect[series] aug93virf(%size(sigma0))
do i=1,%size(sigma0)
    set aug93virf(i) 1 nstep = 0.0
end do i
*

do step=1,nstep
    if step==1
        compute hvec=%vech_a*shock
    else
        compute hvec=(%vech_a+%vech_b)*hvec
        compute %pt(aug93virf,step,hvec)
    end do step
do i=1,%size(sigma0)
    graph(column=2,row=i,picture="*.###",vticks=5)
    # aug93virf(i) 1 nstep
end do i

```

The **SPGRAPH** that encloses the graphs puts labels on the rows and columns on the margins of the graph array. The result is Figure 6.2.

```

spgraph(vfields=3,hfields=2,footer="Figure 1",$
    xlabel=||"1992 Sept 16","1993 Aug 2"||,$
    ylabel=||"DEM/USD Variance","Covariance","GBP/USD Variance"||)

```

Ordinarily, with standard impulse response functions, we recommend graphing all responses *of* a variable over a common range, as that makes it easier to compare the importance of the different shocks in explaining a variable. HH recommend scaling the responses by the historical variances at the time of the

shock in question.² In this case, it would make almost no difference, since the variances (in our scaling of the data) are very near 1 in both situations.³ And that would not change the fact that the first incident disproportionately hit the pound, while the second had almost no effect on it. It probably still makes sense to use a common scale across a row, but be aware of the fact that, unlike the IRF for a VAR, the size of the responses isn't just a function of the model, but depends upon your choice of historical shocks to apply.

6.2.1 Extensions of VIRF

The VIRF, as described above, can be calculated the same way for any multivariate GARCH that can be put into VECH form, which means VECH, DVECH, BEKK and its triangular (TBEKK) and diagonal (DBEKK) restrictions. It does *not* apply to any of those model forms (other than the DBEKK) if you add asymmetry (Section 6.3)—as we'll see there, asymmetry makes it impossible to calculate a change in the covariance without having the entire covariance matrix. Instead, you would have to use simulation methods (Chapter 9), and the entire calculation has to be conditioned on the entire situation at a specific time period. By contrast, the VIRF is merely using an episode to provide the shock—the *change* due to the shock will be the same regardless of the initial conditions.

There is another whole class of models where the responses of the variances only (not the covariances) can be computed using a similar idea. For instance, many of the models for the individual variances used in CC and DCC GARCH can be extrapolated using the same basic idea. For instance, the VARMA variances (VARIANCES=VARMA option) are:

$$H_{ii}(t) = c_{ii} + \sum_j a_{ij} u_j(t-1)^2 + \sum_j b_{ij} H_{jj}(t-1)$$

This uses no covariances, and no cross terms on the u , so this can be written as

$$H_t = C + A(u_{t-1} \circ u_{t-1}) + B H_{t-1} \quad (6.2)$$

where H is a vector of just the variances, C is a vector of the c coefficients and A and B are $n \times n$ matrices of coefficients. Since

$$E(u_{t-1} \circ u_{t-1}) = H_{t-1}$$

the same idea as before will apply to *this* recursion. The “spillover” variance model (VARIANCES=SPILLOVER) is similar, but with a diagonal B matrix, while the default VARIANCES=SIMPLE has diagonal A and B matrices

²Presumably dividing the covariance by the square root of the product of the variances, though that's not explicit in the paper.

³With their scaling, these would all be smaller by a factor of 10000.

(and thus no interactions among the variables in their variances). The C, A and B matrices can be obtained from a **GARCH** estimation using the procedure **MVGARCHVarMats**.

```
@MVGARCHVarMats(VARIANCES=VARIANCE model from GARCH)
```

In the option, repeat the **VARIANCES** option from the preceding **GARCH** instruction.⁴ The procedure will create the VECTOR `%%VAR_C`, and the matrices `%%VAR_A` and `%%VAR_B` from (6.2). (The C matrix isn't needed for impulse response functions, but is needed for forecasting). See the `GARCHVIRFDIAG.RPF` program for an example of its use, which closely parallels the Hafner-Herwartz example.

6.3 Asymmetry

Asymmetry is more complicated with multivariate models than univariate. With the univariate model, there are many equivalent ways to parameterize asymmetry. For instance,

$$\begin{aligned} au_{t-1}^2 + du_{t-1}^2 I(u_{t-1} < 0) &= (a + d)u_{t-1}^2 - du_{t-1}^2 I(u_{t-1} > 0) \\ &= au_{t-1}^2 I(u_{t-1} > 0) + (a + d)u_{t-1}^2 I(u_{t-1} < 0) \end{aligned}$$

so it doesn't matter whether the asymmetry term uses the positive or negative sign. The first formulation is used most often because the assumption is that negative shocks increase variance more than positive ones, so that would give $d > 0$. But if that assumption is incorrect—if it's positive shocks which increase the variance—there is nothing preventing d from being negative.

In a multivariate setting, it's no longer true that the sign convention is innocuous. The problem comes from the interaction terms between the shocks. The most commonly used adjustment for asymmetry in the standard multivariate GARCH is to define \mathbf{v}_t as

$$\mathbf{v}_t = \mathbf{u}_t \circ I(\mathbf{u}_t < 0)$$

that is $v_{it} = u_{it}$ if $u_{it} < 0$ and $v_{it} = 0$ otherwise, done component by component. The recursion for **H** is

$$\mathbf{H}_t = \mathbf{C} + \mathbf{A} \circ (\mathbf{u}_{t-1} \mathbf{u}_{t-1}') + \mathbf{B} \circ \mathbf{H}_{t-1} + \mathbf{D} \circ (\mathbf{v}_{t-1} \mathbf{v}_{t-1}') \quad (6.3)$$

The diagonal terms are identical to those in a corresponding asymmetric univariate GARCH. However, for off-diagonal element ij , the asymmetry term is non-zero only if *both* $u_{i,t-1}$ and $u_{j,t-1}$ are negative. The recursion differentiates between those data points with both negative (where **D** comes into play) and those where at least one is positive, which only get the first three terms. If

⁴The **VARIANCES=EXPONENTIAL** and **VARIANCES=KOUTMOS** models don't fit into this form—if you use them, the procedure will return with an error message.

we change the sign convention in the definition of v , the data points which get the added term are those where *both* u are positive. Both of these leave out those data points with one positive and one negative. Unless you add an additional term or terms to deal with those, the likelihood function will be different depending upon which sign convention you choose for the asymmetry.

For univariate models with the basic GARCH recursion (not EGARCH), forecasting can be done relatively easily with the asymmetry, because

$$E(u_{t-1}^2 I(u_{t-1} > 0)) = .5 E u_{t-1}^2$$

which is true for any symmetric density for u . And that's true on the *diagonals* for (6.3), but *not* the off-diagonals. The expected value of $v_{it}v_{jt}$ for $i \neq j$ is a complicated function of the covariance matrix. For a Normal distribution, this is:

$$E v_i v_j = h_{ij} F(0, 0, \rho) + \frac{\sqrt{1 - \rho^2}}{2\pi} \sqrt{h_{ii} h_{jj}} \quad (6.4)$$

where $F(x, y, r)$ is the standard bivariate normal CDF with correlation r .⁵ (6.4) is a special case of the general formula in Muthen (1990) with ρ as the correlation between u_i and u_j —(6.4) only gives $.5h_{ij}$ if $\rho = 1$. We've seen several papers that (incorrectly) did calculations with asymmetric models assuming that the .5 factor could apply to the full matrix. Not only is it not a simple function, but it also depends upon the distribution—(6.4) is specific to the Normal. For the same reason, there is no simple extension of the VIRF of Section 6.2 to an asymmetric model—even if you *want* to incorporate the calculation in (6.4), it can't be computed without H itself at every stage, so there is no way to compute it as an impulse response. Instead, you have to do simulations (Chapter 9) with and without added shocks.

Adding Asymmetry to a Model

As with the univariate model, you add asymmetry to the GARCH model by adding the `ASYMMETRIC` option to **GARCH**. What parameters that adds to the model will depend upon the choice for the `MV` option.

The data file used in Example 6.3 is daily return data on the US SP500, the Japanese Nikkei and the Hong Kong Hang Seng, daily (holidays included) from 1 January 1986 to 10 April 2000. The original data are just raw daily returns, so we scale up by 100:

⁵The standard bivariate normal has variance 1 for each component and correlation r . The CDF is the probability of the quadrant southwest of (x, y) . This can't be calculated using the univariate CDF unless $r = 0$; instead, you can use the RATS function `%BICDF`.

```

open data mhcdata.xls
cal(d) 1986:1:1
data(format=xls,org=columns) 1 3724 rnk rhs rsp
*
* We scale up the original data by 100.0
*
set rsp = rsp*100.0
set rnk = rnk*100.0
set rhs = rhs*100.0

```

This is taken from the working paper “Empirical Modelling of Multivariate Asymmetric Financial Volatility” by Chan and McAleer. Asymmetry is a common feature in GARCH models of stock market return data. As with the example above, the authors chose univariate AR1 models for the mean:

```

equation speq rsp
# constant rsp{1}
equation nkeq rnk
# constant rnk{1}
equation hseq rhs
# constant rhs{1}
*
group armeans speq nkeq hseq

```

The simplest form of asymmetry is in the basic CC model—this just adds the one univariate asymmetry parameter to each variance equation. As we will be looking at several non-nested models, we’ll compute a set of information criteria on each model:

```

garch(model=armeans,mv=cc,asymm)
@regcrits(title="CC Model with Asymmetry")

```

For the simple CC model, the asymmetry coefficients are labeled as $D(1)$, $D(2)$, $D(3)$ for the three equations. The output is Table 6.2. As you can see, the asymmetry terms are *highly* significant, and are larger (even when multiplied by .5) than the corresponding A coefficients.

The final model in the working paper was the CC model with asymmetric VARMA GARCH variances. The added asymmetry parameters are one per variable, just like the basic CC. The VARMA GARCH is an extension of the spillover model (page 106) which includes B coefficients on all the lagged variances, not just own variances.⁶ It is estimated with `MV=CC` and `VARIANCES=VARMA`.

```

garch(model=armeans,mv=cc,variances=varma,asymm,$
pmethod=simplex,piters=10,method=bfgs)
@regcrits(title="CC-VARMA Model with Asymmetry")

```

⁶Unfortunately, the phrase VARMA-GARCH is used to mean both a GARCH with the VARMA recursion for the variances, and a standard GARCH model with a VARMA model for the mean.

Table 6.2: CC Model with Asymmetry

MV-GARCH, CC - Estimation by BFGS					
Convergence in 51 Iterations. Final criterion was 0.0000053 <= 0.0000100					
Daily(5) Data From 1986:01:03 To 2000:04:10					
Usable Observations		3722			
Log Likelihood		-16751.3755			
	Variable	Coeff	Std Error	T-Stat	Signif
1.	Constant	0.0565	0.0136	4.1561	0.0000
2.	RSP{1}	0.0301	0.0179	1.6866	0.0917
3.	Constant	0.0502	0.0172	2.9117	0.0036
4.	RNK{1}	-0.0031	0.0178	-0.1761	0.8602
5.	Constant	0.0902	0.0210	4.3053	0.0000
6.	RHS{1}	0.1006	0.0187	5.3759	0.0000
7.	C(1)	0.0198	0.0042	4.7367	0.0000
8.	C(2)	0.0297	0.0051	5.7849	0.0000
9.	C(3)	0.0892	0.0107	8.3576	0.0000
10.	A(1)	0.0276	0.0090	3.0629	0.0022
11.	A(2)	0.0377	0.0079	4.7964	0.0000
12.	A(3)	0.0528	0.0094	5.6350	0.0000
13.	B(1)	0.9087	0.0122	74.7463	0.0000
14.	B(2)	0.8702	0.0095	91.4406	0.0000
15.	B(3)	0.8369	0.0103	80.8687	0.0000
16.	D(1)	0.0847	0.0138	6.1593	0.0000
17.	D(2)	0.1750	0.0165	10.6008	0.0000
18.	D(3)	0.1738	0.0207	8.3871	0.0000
19.	R(2,1)	0.2693	0.0136	19.7330	0.0000
20.	R(3,1)	0.3148	0.0144	21.8621	0.0000
21.	R(3,2)	0.2540	0.0156	16.3017	0.0000

This adds an extra 12 parameters compared to the basic CC model. We'll omit the output—it's does somewhat worse than the asymmetric CC on the BIC and somewhat better on less stringent criteria like the AIC.

With asymmetry, the BEKK model becomes

$$H_t = CC' + A'u_{t-1}u'_{t-1}A + B'H_{t-1}B + D'v_{t-1}v'_{t-1}D$$

which adds the $n \times n$ matrix D . More than in the other models, this is sensitive to the choice of sign for the asymmetric effect since that final term is forcibly positive semi-definite—the variance increment has to be at least as high for the data points covered by the final term as it is for those that aren't. The default is for v to be constructed using the negative branch. For a different convention, you can use the `SIGNS` option. `SIGNS` provides a n vector with the desired signs for the asymmetry for each variable. The default is all -1, but you can change that to all 1's for the positive branch, or can mix-and-match if you have a combination of variables for which that's appropriate. For instance, `SIGNS=||-1,1||` in a bivariate system would have negative asymmetry on the first variable and positive on the second.


```
garch(model=armmeans,mv=bekk,asymm,$
      pmethod=simplex,piters=10,method=bfgs)
@regcrits(title="BEKK Model with Asymmetry")
```

Asymmetry in the standard GARCH uses the formula (6.3). D is symmetric, so it adds $n(n+1)/2$ new parameters. This ends up being the (slightly) preferred specification of the four, at least by the BIC and HQ. BEKK is very slightly favored by AIC. The output from **GARCH** is in Table 6.3.

```
garch(model=armmeans,asymm,rvectors=rd,hmatrices=hh,$
      pmethod=simplex,piters=10,method=bfgs,$
      stdresids=rstd,factorby=eigen)
```

We did standard multivariate diagnostics for this model:

```
@regcrits(title="Standard GARCH with Asymmetry")
*
@mvqstat(lags=10)
# rstd
@mvarchtest
# rstd
```

producing

Multivariate Q(10)=	110.19303
Significance Level as Chi-Squared(90)=	0.07295
Test for Multivariate ARCH	
Statistic	Degrees
850.40	36
Signif	0.00000

The test for residual ARCH is clearly much worse than we would like. One thing to note, however, is that none of the estimates used t errors. We can do a multivariate Jarque-Bera test on the standardized residuals using the **@MVJB** procedure with

```
@mvjb(factor=%identity(3))
# rstd
```

Var	JB	P-Value
1	1861.392	0.000
2	5726.954	0.000
3	1265.721	0.000
All	8854.067	0.000

which shows that each component and the joint test are highly significant. If we wanted to spend more time with this model, we would go back and re-estimate with t errors. It's also the case that the ARCH test fails largely because of the period around the October 1987 crash in the US which creates huge outliers in the standardized residuals.⁷

⁷You can add range parameters like 1988:1 * to the **@mvarchtest** to restrict the sample.

Table 6.3: Standard GARCH with Asymmetry

MV-GARCH - Estimation by BFGS					
Convergence in 111 Iterations. Final criterion was 0.0000059 <= 0.0000100					
Daily(5) Data From 1986:01:03 To 2000:04:10					
Usable Observations		3722			
Log Likelihood		-16686.6651			
	Variable	Coeff	Std Error	T-Stat	Signif
1.	Constant	0.0597	0.0135	4.4183	0.0000
2.	RSP{1}	0.0316	0.0174	1.8130	0.0698
3.	Constant	0.0575	0.0165	3.4867	0.0005
4.	RNK{1}	-0.0022	0.0165	-0.1347	0.8928
5.	Constant	0.1006	0.0187	5.3884	0.0000
6.	RHS{1}	0.0948	0.0159	5.9826	0.0000
7.	C(1,1)	0.0140	0.0031	4.4976	0.0000
8.	C(2,1)	0.0033	0.0017	1.9017	0.0572
9.	C(2,2)	0.0267	0.0046	5.7624	0.0000
10.	C(3,1)	0.0093	0.0022	4.2025	0.0000
11.	C(3,2)	0.0086	0.0030	2.8602	0.0042
12.	C(3,3)	0.0785	0.0085	9.2891	0.0000
13.	A(1,1)	0.0272	0.0066	4.1069	0.0000
14.	A(2,1)	0.0047	0.0070	0.6664	0.5051
15.	A(2,2)	0.0330	0.0075	4.3865	0.0000
16.	A(3,1)	0.0116	0.0063	1.8307	0.0671
17.	A(3,2)	0.0142	0.0078	1.8051	0.0711
18.	A(3,3)	0.0517	0.0075	6.8830	0.0000
19.	B(1,1)	0.9231	0.0087	106.1192	0.0000
20.	B(2,1)	0.9144	0.0091	99.9792	0.0000
21.	B(2,2)	0.8761	0.0092	95.4095	0.0000
22.	B(3,1)	0.9269	0.0095	97.6067	0.0000
23.	B(3,2)	0.9033	0.0098	92.4051	0.0000
24.	B(3,3)	0.8484	0.0088	96.8754	0.0000
25.	D(1,1)	0.0709	0.0109	6.4983	0.0000
26.	D(2,1)	0.0917	0.0117	7.8123	0.0000
27.	D(2,2)	0.1760	0.0152	11.6115	0.0000
28.	D(3,1)	0.0566	0.0122	4.6524	0.0000
29.	D(3,2)	0.0823	0.0145	5.6790	0.0000
30.	D(3,3)	0.1604	0.0168	9.5343	0.0000

6.4 GARCH-X

Just as with univariate models, shifts can be added to a multivariate GARCH with the option `XREGRESSORS`. Typically, these are dummies of some form. `XREGRESSORS` adjusts the `C` term in the GARCH recursion. For any form of `CC` or `DCC`, this adds a coefficient for each variance equation for each of the “X” variables. For a standard GARCH, it adds a coefficient for each regressor to each of the components. In either case, it’s relatively straightforward.

The only model type for which it’s *not* clear how to handle such regressors is the BEKK because of the desire to enforce positive-definiteness. If we restrict our attention to dummy variables, it’s a standard result that it *should* be irrelevant to the fit whether a dummy is 0-1 or 1-0 for an event and its complement. For instance, if we’re working with scalars:

$$\alpha + \beta d_t = (\alpha + \beta) - \beta(1 - d_t) = \alpha(1 - d_t) + (\alpha + \beta) d_t$$

If there are no sign restrictions, all three of these are equivalent. The problem with the BEKK is that there *are* sign restrictions, since each term is required to be positive semi-definite. If we apply the dummy to a factor matrix and try to alter the interpretation of the dummy, we get

$$CC' + EE'd_t = (CC' + EE') + (-EE')(1 - d_t)$$

If the left side is OK, the right side isn’t.

An alternative (which is what the **GARCH** instruction does by default) is to replace `CC'` with

$$(C + Ed_t)(C + Ed_t)'$$

where `E` is (like `C`) a lower triangular matrix. This enforces positive-definiteness, but doesn’t require that the dummy itself add a positive semi-definite increment to the variance. Because the adjustments are made before “squaring”, the model isn’t sensitive to the choice of representation for the dummy. As we said, this is the default method of handling the X regressors in a BEKK model. The alternative (of separately added terms) can be chosen using `XBEKK=SEPARATE`.

As an illustration of `XREGRESSORS`, we will add a “Monday” dummy to the variance model for the three stock market returns in Example 6.3. With a daily **CALENDAR**, it’s easy to create that with

```
set monday = %weekday(t)==1
```

`%WEEKDAY` maps the entry number to 1 to 7 for Monday to Sunday. We’ll add this to the standard and BEKK models, while also adding `DISTRIB=T` to deal with the fat tails:

Table 6.4: Monday Variance Dummies in DVECH Model

	Variable	Coeff	Std Error	T-Stat	Signif
31.	MONDAY(1,1)	-0.0080	0.0460	-0.1733	0.8624
32.	MONDAY(2,1)	-0.0652	0.0441	-1.4797	0.1389
33.	MONDAY(2,2)	-0.1074	0.0740	-1.4502	0.1470
34.	MONDAY(3,1)	-0.1984	0.0536	-3.7042	0.0002
35.	MONDAY(3,2)	-0.0965	0.0683	-1.4131	0.1576
36.	MONDAY(3,3)	-0.1286	0.1061	-1.2122	0.2254
37.	Shape	6.3246	0.3285	19.2548	0.0000

```
garch(model=armean, asymm, xreg, distrib=t, $
  pmethod=simplex, peters=10, method=bfgs)
# monday
garch(model=armean, mv=bekk, asymm, xreg, distrib=t, $
  pmethod=simplex, peters=10, method=bfgs)
# monday
```

In the DVECH model, the Monday dummies are generally insignificant (and have what would generally be seen as the “wrong” sign, since one would generally expect higher variances on Monday after several non-trading days). See (abbreviated) Table 6.4. The one that comes in significant is for the covariance between the SP500 and HS.

6.5 Cointegrated Variables

To this point, the examples of multivariate GARCH have had, at most, fairly weak serial correlation—a low order VAR was generally adequate, and sometimes it was questionable whether any lagged variables were needed at all. However, it is possible to have very strongly correlated data with GARCH error processes. For instance, Lee (1994) fits GARCH models to the pair of spot and forward rates for different currencies. Because the spot and forward rates are seen as being cointegrated, a VAR in differences is misspecified, so a Vector Error Correction Model (VECM) is required for the mean. With daily data, there is substantial second-moment dependence as well, so a VECM-GARCH is used.

Note well that Lee uses the theoretical cointegrating vector of $(+1, -1)$ (that is, $z_t = f_t - s_t$ is the disequilibrium condition, where f and s are in logs). While it's *possible* to estimate the cointegrating vector (using, for instance, @JOHMLE) as the estimates are generally robust to GARCH errors, here a theoretical value exists and (logically) there is no sensible reason for the actual cointegrating vector to be different than the theoretical one other than through sampling error, so it is a mistake to inject that sampling error into the calculations.

Lee chose to use a diagonal BEKK model with the lagged disequilibrium as an “X” regressor in the variance process.⁸ He uses the “separate” terms method

⁸The paper doesn't discuss why the DBEKK rather than any other form of multivariate

for this (Section 6.4), so the variance process evolves according to

$$\mathbf{H}_t = \mathbf{C}\mathbf{C}' + \mathbf{A}'u_{t-1}u_{t-1}'\mathbf{A} + \mathbf{B}'\mathbf{H}_{t-1}\mathbf{B} + \mathbf{D}\mathbf{D}'z_{t-1}^2 \quad (6.5)$$

where \mathbf{A} and \mathbf{B} are restricted to be diagonal (by the choice of DBEKK) and \mathbf{D} is lower triangular. The separate BEKK term for the z_{t-1} is more sensible here than it would be with just a dummy (it's probably not unreasonable to expect that a gap between the forward and spot rates would increase the process variance), it does carry the restriction that the added variance is symmetric in z_{t-1} , which might *not* be so clear.

Example 6.4 does analysis for the exchange rates of seven currencies vs the US dollar. The data set has the seven pairs of series nicely labeled with a two character suffix after an “F” (forward) and “S” (spot) prefix. That makes it easier to create a loop which runs over the list of suffixes:

```
open data bbjof1989.xls
data(format=xls,org=columns) 1 1245 fuk fwg fja fca ffr fit fsw $
                                suk swg sja sca sfr sit ssw

*
dec vect[labels] suffix(7)
input suffix
    uk wg ja ca fr it sw
```

This creates a lookup table of full names for each currency. A `HASH` is a form of array where the “subscript” is a string (known as a key).

```
dec hash[string] longnames
compute longnames("uk")="United Kingdom"
compute longnames("wg")="West Germany"
compute longnames("ja")="Japan"
compute longnames("ca")="Canada"
compute longnames("fr")="France"
compute longnames("it")="Italy"
compute longnames("sw")="Switzerland"
```

The pseudo-code (Appendix G) for the loop is:

```
dec label country
dofor country = list of suffixes
    << do analysis when suffix is country >>
end dofor country
```

The raw data are the (30-day) forward and spot rates, which need to be transformed to logs. We're following the standard recommendation of using $100 * \log(x)$, while the paper itself did just $\log(x)$. This affects the *values* of the

log likelihoods, but not test statistics (all log likelihoods go down by a constant amount, so the differences between them don't change).

This creates series `FORWARD`, `SPOT` and `Z` for the log forward, log spot and gap between them. Each pass through the loop replaces these with the values based upon the country in the loop index. `SFORWARD` and `SSPOT` are the series “handles” to the currently analyzed forward and spot rates—the 0's in the `SET` instructions are required because handles only represent series; they aren't the series themselves.

```
compute sforward=%s("f"+country)
compute sspot    =%s("s"+country)
*
set forward      = 100*log(sforward{0})
set spot         = 100*log(sspot{0})
set z            = forward-spot
```

This does Dickey-Fuller tests on the log spot, log forward and the difference. The expected results are unit roots on the first two and not (i.e. rejection of the unit root) on the difference.

```
@dfunit(lags=10,method=bic) spot
@dfunit(lags=10,method=bic) forward
@dfunit(lags=10,method=bic) z
```

Before we go on with this, let us note that one of the problems with an analysis of this type (large number of parallel calculations) is that you may not be able to give the proper attention to each one. In the paper, Lee was unable to fit one of the models (the GARCH-X) to two of the currencies: the British pound and Japanese yen. If we do what we should always do anyway, and look more carefully at the GBP data (Figure 6.3), we see that there are three spikes in the forward premium (top panel). If we focus in on the first of those (bottom panel), we see that it's a one-day spike in the forward rate. (Since this series is in USD/GBP, a negative spike means a weak dollar or a strong pound). The second is also a one-day spike in the forward rate (in the opposite direction), while the third is a one-day spike in the spot. All are one-day outliers, and all seem to have no persistent effect on the data.

If we were fitting a univariate GARCH model to the z series, the best way to handle this statistically would be to put single period dummies into the mean model. (We should, of course, try to determine what happened on those days, so we can explain our choice of action.) However, we aren't—we're doing a bivariate GARCH model and z is derived from the two other data series, and will enter the model in two locations (variance model and the error correction term in the VECM). The closest thing to dummifying out the strange z values is to interpolate the series whose value is out-of-line (forward in the first two cases, spot in the third). You should only do this type of adjustment if it seems

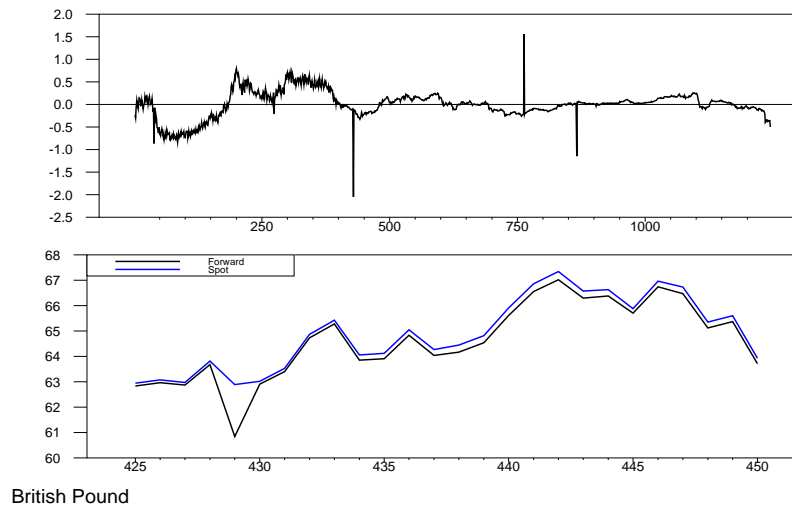


Figure 6.3: British Pound, Forward Premium and Focus Subrange

fairly clear that the odd data points are completely outside what the model would predict—huge outliers should be followed by substantial volatility (in both forward and spot markets), and these aren't.⁹

However, continuing on, for each series pair, the number of lags in the VECM is chosen using BIC (from a maximum of 10 lags). Though `@VARLagSelect` is for a standard VAR, it can be used just as easily to find the lag length in a VECM. The procedure sets the variable `%%AUTOP` to the chosen number of lags, which is what is used in setting up the VECM.

```
@varlagselect(lags=10,crit=bic)
# spot forward
*
equation(coeffs=||1.0,-1.0||) ecteq z
# forward spot
*
system(model=vecm)
variables spot forward
lags 1 to %%autop
det constant
ect ecteq
end(system)
```

This estimates the VECM without GARCH errors:

```
estimate(model=vecm)
compute loglmodel1=%logl
@regcrits
```

⁹These almost look like data errors or a data point being mis-timed—again, part of doing this correctly would require checking out the circumstances.

and this estimates the VECM with DBEKK errors but without the X variable:

```
garch(model=vecm,mv=dbekk,pmethod=simplex,piters=10,itors=500)
compute loglmodel2=%logl
@regcrits
```

Note that **@REGCRITS** for computing the information criteria properly accounts for *all* the free parameters in a model. That isn't really an issue with the GARCH model, but is with the non-GARCH VECM, where the covariance matrix often isn't counted among the parameters,¹⁰ but needs to be included if we're comparing its estimates with the GARCH which explicitly models the covariance matrices.

This computes the DBEKK with the lagged error correction term as an "X" regressor.

```
garch(model=vecm,mv=dbekk,pmethod=simplex,piters=10,itors=500,$
      xbekk=separate,xreg)
# z{1}
compute loglmodel3=%logl
@regcrits
```

This does a simple listing of the log likelihoods of the three models (VECM, VECM-GARCH, VECM-GARCH-X), with likelihood ratio tests comparing the first two (the GARCH adds 2 parameters in each of the diagonal A and B matrices—the C exactly matches the three free parameters in the fixed covariance matrix in the VECM), and another comparing the last two (the GARCH-M adds the 3 parameters in the D matrix).

```
disp longnames(country) loglmodel1 loglmodel2 loglmodel3
cdf(title="GARCH vs No GARCH") chisqr 2*(loglmodel2-loglmodel1) 4
cdf(title="GARCH-X vs GARCH") chisqr 2*(loglmodel3-loglmodel2) 3
```

If we were writing a paper based upon this, we would at some point, add **REPORT** instructions to create tables with the information, however, for illustrating calculations, that isn't necessary. You also wouldn't do it until you were *sure* this is the analysis that you wanted to use.

All of the countries rather clearly reject the regular VECM in favor of the VECM-GARCH. All except the pound reject the GARCH in favor of the GARCH-X.

The results of the final model for the Swiss Franc are in Table 6.5. One thing to note (and this is fairly consistent across currencies) is that the coefficients on the disequilibrium are the same sign. Since $z = f - s$, and the adjustment coefficients are negative, a positive value of z (that is, positive forward premium)

¹⁰If you're comparing, for instance, two VECM's with different lags, both have covariance matrices which are concentrated out, so whether you include the covariance matrix in the parameter count would have no effect on the *relative* parameter counts.

Table 6.5: VECM-GARCH-X for Switzerland

MV-GARCH, Diagonal BEKK - Estimation by BFGS
 Convergence in 277 Iterations. Final criterion was 0.0000018 <= 0.0000100

Usable Observations		1240			
Log Likelihood		450.8412			
	Variable	Coeff	Std Error	T-Stat	Signif
Mean Model(SPOT)					
1.	D_SPOT{1}	-0.1439	0.2112	-0.6813	0.4957
2.	D_SPOT{2}	-0.1281	0.2603	-0.4922	0.6226
3.	D_SPOT{3}	-0.3103	0.2455	-1.2637	0.2063
4.	D_SPOT{4}	-0.0084	0.1900	-0.0441	0.9648
5.	D_FORWARD{1}	0.1248	0.2128	0.5864	0.5576
6.	D_FORWARD{2}	0.1521	0.2627	0.5790	0.5626
7.	D_FORWARD{3}	0.2819	0.2476	1.1387	0.2548
8.	D_FORWARD{4}	0.0471	0.1901	0.2475	0.8046
9.	Constant	0.0355	0.0523	0.6780	0.4978
10.	EC1{1}	-0.1795	0.0926	-1.9396	0.0524
Mean Model(FORWARD)					
11.	D_SPOT{1}	0.0313	0.2097	0.1491	0.8815
12.	D_SPOT{2}	0.0290	0.2593	0.1119	0.9109
13.	D_SPOT{3}	-0.1089	0.2450	-0.4443	0.6568
14.	D_SPOT{4}	0.0420	0.1880	0.2234	0.8232
15.	D_FORWARD{1}	-0.0569	0.2118	-0.2687	0.7881
16.	D_FORWARD{2}	0.0001	0.2621	0.0003	0.9998
17.	D_FORWARD{3}	0.0858	0.2474	0.3468	0.7287
18.	D_FORWARD{4}	-0.0099	0.1886	-0.0525	0.9581
19.	Constant	0.0434	0.0522	0.8323	0.4052
20.	EC1{1}	-0.1958	0.0923	-2.1197	0.0340
21.	C(1,1)	0.0640	0.0252	2.5444	0.0109
22.	C(2,1)	0.0692	0.0244	2.8345	0.0046
23.	C(2,2)	0.0000	0.0230	0.0000	1.0000
24.	A(1)	0.3635	0.0186	19.4954	0.0000
25.	A(2)	0.3548	0.0183	19.3546	0.0000
26.	B(1)	0.9291	0.0057	163.1765	0.0000
27.	B(2)	0.9303	0.0056	165.0863	0.0000
28.	Z{1}(1,1)	-0.1800	0.0288	-6.2479	0.0000
29.	Z{1}(2,1)	-0.1829	0.0282	-6.4788	0.0000
30.	Z{1}(2,2)	0.0118	0.0035	3.4217	0.0006

tends to mean both f and s will go down; s by just slightly less than f . Thus, the adjustment to equilibrium is fairly slow. A common question is why the A and B coefficients add up to more than one—however, remember that those are both squared in a DBEKK, and their *squares* add up to just *below* one.

Finally, as with all other matrices in a BEKK, the signs on the D matrices aren't identified. Here, they're (mainly) negative, but you would get the exact same results from changing the signs of all elements of the matrix. And, as with pretty much all parameters in a BEKK, it's difficult (if not impossible) to interpret the D coefficients directly. The parameters show in the output as $z\{1\}(i, j)$, which means the i, j element of the D matrix for the X-regressor $z\{1\}$. If there were more than one X-regressor, there would be a separate block of 3 parameters for it. If you multiply out the D matrix (recall that it enters in the form DD'), you get

$$\begin{bmatrix} 0.03240 & 0.03292 \\ 0.03292 & 0.03359 \end{bmatrix}$$

Because the values are very similar, this says that the effect of disequilibrium is a common increase in the variance of each series (and their covariance). An alternative way to look at that is that the shock at t is the sum of two independent components, one with covariance determined by the C, A and B parts of the recursion, the other a (random) common shift in each series with standard deviation proportional to the disequilibrium. That fairly neat interpretation would apply to a few of the series—it requires the (1,1) and (1,2) elements of the D matrix to be roughly equal and the 2,2 element to be (relatively) near zero.

Let's now revisit the data for the pound and see what happens if we patch over the three really bad spots. This does linear interpolation on the logged data. While it might be *possible* to do something more sophisticated, Kalman smoothing a random walk over a missing data point is (effectively) the same as linear interpolation and spot and forward are both near-random walks, so the difference between this and a more complex patch procedure will be negligible (particularly when we are fixing just 3 data points out of 1200).

```
set forward = 100.0*log(fuk)
compute forward(429)=.5*(forward(428)+forward(430))
compute forward(763)=.5*(forward(762)+forward(764))
set spot    = 100.0*log(suk)
compute spot(866)=.5*(spot(865)+spot(867))
*
set z       = forward-spot
```

If we run the same analysis as before, the difference is rather striking. The log likelihood goes from -105.5389 without the patches to 996.3297 .¹¹ The DBEKK coefficients, which showed quite low persistence originally (both the A and B

¹¹Note: while the difference is extraordinarily high, there is no “test” for whether the patch

were in the .3-.4 range) are now much more similar to the other countries (A's around .5 and B's around .8—again, remember that these get squared). And the test for GARCH vs GARCH-X which originally was not significant is very, very strongly significant. So just three data points (again, out of 1200!) had a really staggering impact on the estimates. This should serve as a reminder to check your data, and carefully select your sample ranges.

As a final experiment, let's estimate the GARCH-X using the default method for adding "X" variables to a BEKK:

$$\mathbf{H}_t = (\mathbf{C} + \mathbf{D}z_{t-1})(\mathbf{C} + \mathbf{D}z_{t-1})' + \mathbf{A}'u_{t-1}u_{t-1}'\mathbf{A} + \mathbf{B}'\mathbf{H}_{t-1}\mathbf{B}$$

D is still parameterized as a lower-triangular matrix, so this has the same number of free parameters. The two methods don't nest, so there is no formal test. However

```
garch(model=vecm,mv=dbekk,pmethod=simplex,piters=10,itors=500,$
      xreg,xbekk=combined)
# z{1}
```

gives a log likelihood of 1094.1975, which is higher by almost 100 compared with the XBEEK=SEPARATE estimates. Across the seven countries, the two forms produce effectively the same log likelihood for three countries (West Germany, Canada and Switzerland), and better to much better likelihoods for the other four. So it would appear that the restriction that the variance change be proportional to the square of the disequilibrium doesn't seem to be correct.

6.6 Hedge Ratios and Related Calculations

One possible use of multivariate GARCH models is to compute hedge ratios and portfolio weights for securities or exchange rates. For effectively every type of multivariate GARCH models, those will be time-varying functions of the covariance matrices. (Even the CC model, since those calculations depend upon the covariances, and not just the correlations.)

Hedge ratios are computed pairwise. If you hold a unit position in asset i , the short position in asset j that minimizes the variance is H_{ij}/H_{ii} . (If this comes up negative, you would need a long position in j). If you save the HMATRICES when you estimate the **GARCH** instruction, these are easily computed. Example 6.5 uses the 3-variable DVECH model from Example 5.2. There will be uses for both "long" and "short" labels for the three currencies, which are defined here:

```
dec vect[string] longlabel(3) shortlabel(3)
compute longlabel=||"Euro","UK Pound","Swiss Franc"||
compute shortlabel=||"Euro","GBP","SWF"||
```

was "correct", as this isn't a different model for the same data, but different data. The fact that it's easier to fit good data than bad data isn't much of a surprise. What you would need to do in practice is to justify patching over those three data points based upon how they represent some failure in the data, rather than how patching over them makes the model easier to fit.

With the `HMATRICES` saved in `HH`, this computes the hedge ratios. Because there is no hedge ratio of a series with itself, the diagonals aren't defined. This allows for any number of series, though the pairwise values probably won't be very interesting when you get much above 3 series.

```
dec rect[series] hedges(%nvar,%nvar)
do i=1,%nvar
  do j=1,%nvar
    if i==j
      next
    set hedges(i,j) = hh(t)(i,j)/hh(t)(i,i)
  end do j
end do i
```

This graphs the results (Figure 6.4). Again, this allows for any number of series, but probably gets rather crowded when you get to five or more.

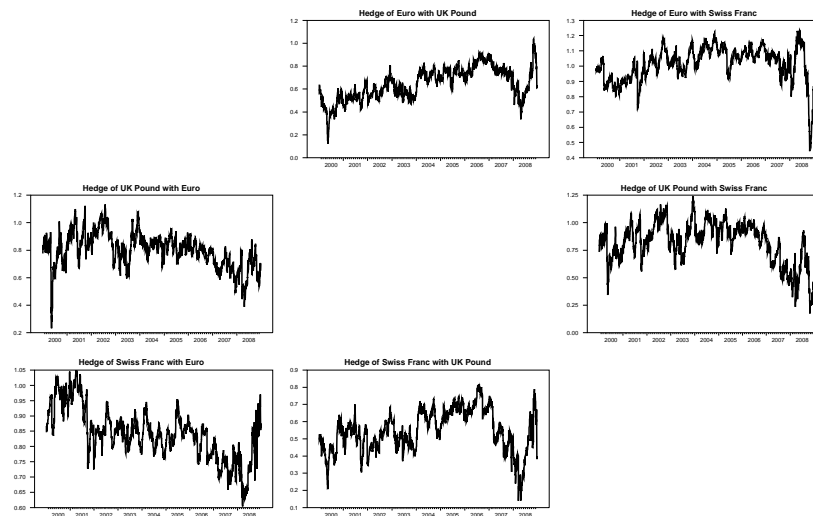


Figure 6.4: Hedge Ratios

```
spgraph(hfields=%nvar,vfields=%nvar,footer="Hedge Ratios")
do i=1,%nvar
  do j=1,%nvar
    if i==j
      next
    graph(row=i,col=j,$
      header="Hedge of "+longlabel(i)+" with "+longlabel(j))
    # hedges(i,j)
  end do j
end do i
spgraph(done)
```

In Sadorsky (2012), the author presented summary statistics on the time-varying hedge ratios with one line per pair. That can be done (using the short labels rather than a long ones) using

```
report (use=hreport, action=define, $
        title="Hedge ratio (long/short) summary statistics")
report (use=hreport, atrow=1, atcol=2, align=center) $
        "Mean" "St Dev" "Min" "Max"
do i=1,%nvar
  do j=1,%nvar
    if i==j
      next
    stats (noprint, fract) hedges (i, j)
    report (use=hreport, row=new, atcol=1) $
            shortlabel(i)+"/"+shortlabel(j) $
            %mean sqrt(%variance) %minimum %maximum
  end do j
end do i
report (use=hreport, atcol=2, atrow=2, action=format, $
        align=decimal, picture="*.###")
report (use=hreport, action=show)
```

producing

	Mean	St Dev	Min	Max
Euro/GBP	0.65	0.14	0.12	1.04
Euro/SWF	1.02	0.11	0.44	1.23
GBP/Euro	0.80	0.13	0.23	1.13
GBP/SWF	0.82	0.19	0.18	1.24
SWF/Euro	0.84	0.08	0.60	1.05
SWF/GBP	0.55	0.12	0.14	0.81

Portfolio weights can be computed either pairwise or across the whole set of variables, and can be computed allowing for short positions, or restricted to long positions only. Note that the calculations here are made assuming the mean returns are zero and so are designed to minimize variance with the portfolio standardized to a net long position of 1. Pairwise calculations can be done with

$$w_{ij} = \frac{(h_{jj} - h_{ij})}{(h_{ii} - 2h_{ij} + h_{jj})} \quad (6.6)$$

where restricting to long positions only requires pinning this between 0 and 1. With more than two assets, it's easiest to use **LQPROG** to solve the problem, whether you are constraining it to long positions only or not.

The pairwise weights can be computed using:

```

dec rect[series] weights(%nvar,%nvar)
do i=1,%nvar
  do j=1,%nvar
    if i==j
      next
    set weights(i,j) = $
      (hh(t)(j,j)-hh(t)(i,j))/(hh(t)(i,i)-2*hh(t)(i,j)+hh(t)(j,j))
    *
    * This constrains the positions to the range[0,1]
    *
    set weights(i,j) = %min(1.0,%max(0.0,weights(i,j)))
    *
  end do j
end do i

```

where you can remove the long position restriction by commenting out the second **SET** instruction. Unlike the hedge ratios, where one direction doesn't necessarily tell you much about the other, the weights have to add up to one in a pair. The following does all the pairs (Figure 6.5), but note that the i,j and j,i pairs are reflections of each other around .5.

```

spgraph(vfields=%nvar,hfields=%nvar,xlabels=longlabel,$
  ylabel=longlabel,footer="Pairwise Portfolio Weights")
do i=1,%nvar
  do j=1,%nvar
    if i==j
      next
    graph(row=i,col=j,style=bar,min=0.0,max=1.0)
    # weights(i,j)
  end do j
end do i
spgraph(done)

```

Again, as in Sadorsky, the summary statistics across time are presented in a report

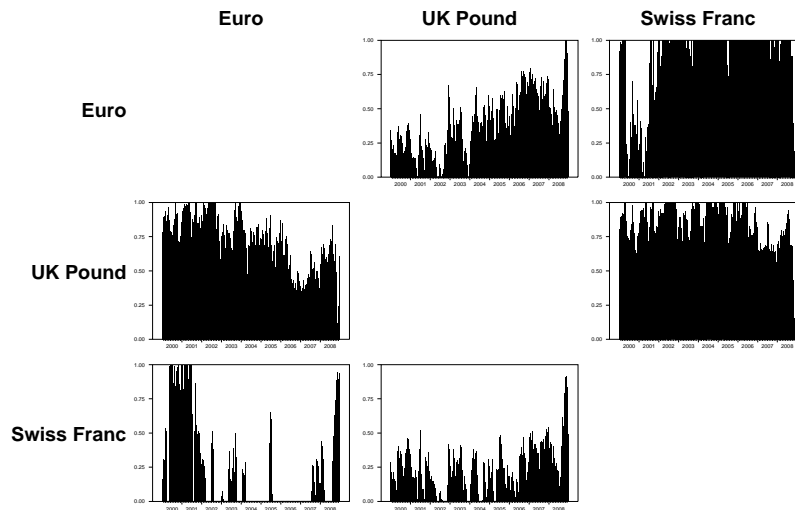


Figure 6.5: Pairwise Portfolio Weights

```

report (use=wreport,action=define,$
        title="Portfolio weights summary statistics")
report (use=wreport,atrow=1,atcol=2,align=center) $
"Mean" "St Dev" "Min" "Max"
do i=1,%nvar
  do j=i+1,%nvar
    stats(noprint,fract) weights(i,j)
    report (use=wreport,row=new,atcol=1) $
      shortlabel(i)+"/"+shortlabel(j) $
      %mean sqrt(%variance) %minimum %maximum
  end do j
end do i
report (use=wreport,action=format,atcol=2,atrow=2,$
        picture="*.##",align=decimal)
report (use=wreport,action=show)

```

This does each pair just once—the reverse has the mean, min and max being 1 minus the number shown, and the standard deviation won't change.

	Mean	St Dev	Min	Max
Euro/GBP	0.34	0.21	0.00	1.00
Euro/SWF	0.81	0.31	0.00	1.00
GBP/SWF	0.80	0.17	0.08	1.00

If you want to show *all* pairs, replace the `DO J` instruction with

```

do j=1,%nvar
  if i==j
    next

```

Finally, portfolio weights across the full set of three variables can be computed with

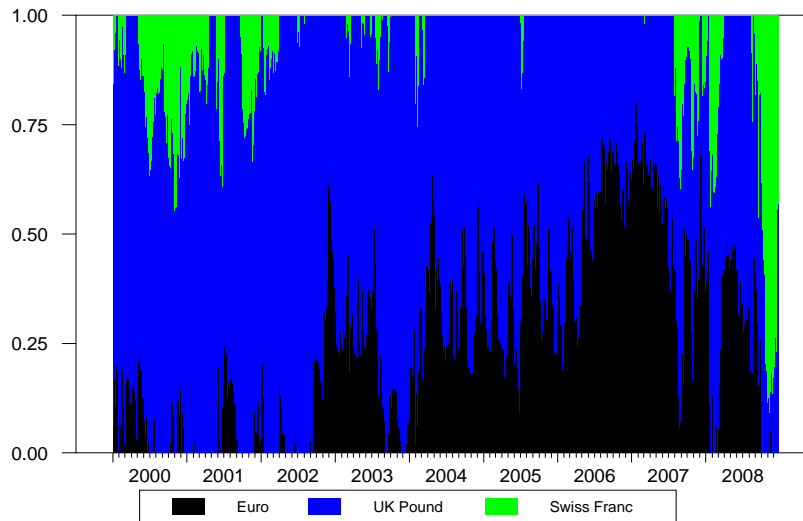


Figure 6.6: Long-position Portfolio Weights

```
dec vect[series] pweights(%nvar)
clear(zeros) pweights
do t=%regstart(),%regend()
    *
    lqprog(noprint,q=hh(t),a=%ones(1,%nvar),b=1.0,$
        equalities=1,nneg=%nvar) x
    compute %pt(pweights,t,x)
end do t
```

This uses a constraint requiring only long positions, as the `NNEG=%NVAR` option means that all the elements need to be non-negative. If you want to allow short positions, you can replace that with `NNEG=0`.

This just has one set of 3 time-varying numbers which must add up to one. This presents them as a stacked bar graph (Figure 6.6):

```
graph(style=stacked,series=pweights,min=0.0,max=1.0,$
    key=below,klabels=longlabel,$
    footer="Long-position Portfolio Weights")
```


Example 6.1 Multivariate GARCH: Forecasting

This is the example from Section 6.1.

```
open data hhdata.xls
data(format=xls,org=columns) 1 3720 usxjpn usxfra usxsui $
  usxnld usxuk usxbel usxger usxswe usxcan usxita date
*
* This is rescaled to 100.0 compared with data used in the paper.
* The paper also uses local currency/USD, while the data set has
* USD/local currency, so we change the sign on the return.
*
set demret = -100.0*log(usxger/usxger{1})
set gbpret = -100.0*log(usxuk/usxuk{1})
*
* The mean model is a univariate AR on each variable separately.
*
equation demeqn demret
# constant demret{1}
equation gbpeqn gbpret
# constant gbpret{1}
group uniar1 demeqn gbpeqn
*
* Estimate BEKK model with student t errors
*
garch(model=uniar1,mv=bekk,rvectors=rd,hmatrices=hh,distrib=t,$
  pmethod=simplex,piters=20,itors=500)
*
@MVGARCHFore(mv=bekk,steps=100) hh rd
*
spgraph(vfields=2,hfields=2)
  compute gstart=%regend()-399,gend=%regend()+100
  set h11fore gstart gend = hh(t)(1,1)
  set h22fore gstart gend = hh(t)(2,2)
  set h12fore gstart gend = hh(t)(1,2)/sqrt(h11fore*h22fore)
  graph(row=1,col=1,grid=(t==%regend()+1),min=0.0,$
    header="Deutsche Mark Volatility")
  # h11fore
  graph(row=2,col=2,grid=(t==%regend()+1),min=0.0,$
    header="British Pound Volatility")
  # h22fore
  graph(row=2,col=1,grid=(t==%regend()+1),header="Correlation")
  # h12fore
spgraph(done)
```

Example 6.2 Volatility Impulse Responses

This is the example from Section 6.2.

```
open data hhdata.xls
data(format=xls,org=columns) 1 3720 usxjpn usxfra usxsui $
```

```

usxnld usxuk usxbel usxger usxswe usxcan usxita date
*
* This is rescaled to 100.0 compared with data used in the paper.
* The paper also uses local currency/USD, while the data set has
* USD/local currency, so we change the sign on the return.
*
set demret = -100.0*log(usxger/usxger{1})
set gbpret = -100.0*log(usxuk/usxuk{1})
*
* The mean model is a univariate AR on each variable separately.
*
equation demeqn demret
# constant demret{1}
equation gbpeqn gbpret
# constant gbpret{1}
group uniar1 demeqn gbpeqn
*
garch(model=uniar1,mv=bekk,rvectors=rd,hmatrices=hh,distrib=t,$
      pmethod=simplex,piters=20,itors=500)
*
* Transform the BEKK model to its equivalent Vech representation
*
@MVGARCHtoVECH(mv=bekk)
eigen(cvalues=cv) %%vech_a+%%vech_b
disp "Eigenvalues from BEKK-t" *.### cv
*
* VIRF with historical incidents
*
sstats(max) / %if(date==920916,t,0)>>xblackwed $
              %if(date==930802,t,0)>>xecpolicy
compute blackwed=fix(xblackwed),ecpolicy=fix(xecpolicy)
*
compute nstep=400
spgraph(vfields=3,hfields=2,footer="Figure 1",$
      xlabel=||"1992 Sept 16","1993 Aug 2"||,$
      ylabel=||"DEM/USD Variance","Covariance","GBP/USD Variance"||)
*
* Black Wednesday shocks. These are computed using a baseline of the
* estimated volatility state, so they are excess over the predicted
* covariance.
*
compute eps0=rd(blackwed)
compute sigma0=hh(blackwed)
compute shock=%vec(%outerxx(eps0)-sigma0)
*
* This generates responses for the 3 elements of the covariance matrix,
* which will be (in order) the (1,1), (1,2) and (2,2).
*
dec vect[series] sept92virf(%size(sigma0))
do i=1,%size(sigma0)
  set sept92virf(i) 1 nstep = 0.0
end do i
*
* Use the Vech representation to compute the VIRF to the original shock.

```

```

*
do step=1,nstep
  if step==1
    compute hvec=%vech_a*shock
  else
    compute hvec=(%vech_a+%vech_b)*hvec
    compute %pt(sept92virf,step,hvec)
end do step
do i=1,%size(sigma0)
  graph(column=1,row=i,picture="*.###",vticks=5)
  # sept92virf(i) 1 nstep
end do i
*
* EC Policy Change shock.
*
compute eps0=rd(ecpolicy)
compute sigma0=hh(ecpolicy)
compute shock=%vec(%outerxx(eps0)-sigma0)
dec vect[series] aug93virf(%size(sigma0))
do i=1,%size(sigma0)
  set aug93virf(i) 1 nstep = 0.0
end do i
*
do step=1,nstep
  if step==1
    compute hvec=%vech_a*shock
  else
    compute hvec=(%vech_a+%vech_b)*hvec
    compute %pt(aug93virf,step,hvec)
end do step
do i=1,%size(sigma0)
  graph(column=2,row=i,picture="*.###",vticks=5)
  # aug93virf(i) 1 nstep
end do i
spgraph(done)

```

Example 6.3 Multivariate GARCH with Asymmetry or Variance Dummies

This is the example from Section 6.3.

```

open data mhcddata.xls
cal(d) 1986:1:1
data(format=xls,org=columns) 1 3724 rnk rhs rsp
*
* We scale up the original data by 100.0
*
set rsp = rsp*100.0
set rnk = rnk*100.0
set rhs = rhs*100.0
*

```

```

* Define univariate AR's for each dependent variable
*
equation speq rsp
# constant rsp{1}
equation nkeq rnk
# constant rnk{1}
equation hseq rhs
# constant rhs{1}
*
group armeans speq nkeq hseq
*
* CC models
*
garch(model=armean, mv=cc, asymm)
@regcrits(title="CC Model with Asymmetry")
garch(model=armean, mv=cc, variances=varma, asymm, $
  pmethod=simplex, peters=10, method=bfgs)
@regcrits(title="CC-VARMA Model with Asymmetry")
*
* BEKK model
*
garch(model=armean, mv=bekk, asymm, $
  pmethod=simplex, peters=10, method=bfgs)
@regcrits(title="BEKK Model with Asymmetry")
*
* Standard GARCH
*
garch(model=armean, asymm, rvector=rd, hmatrix=hh, $
  pmethod=simplex, peters=10, method=bfgs, $
  stdresids=rstd, factorby=eigen)
@regcrits(title="Standard GARCH with Asymmetry")
*
@mvqstat(lags=10)
# rstd
@mvarchtest
# rstd
@mvjb(factor=%identity(3))
# rstd
*
* Standard GARCH with shift dummy
*
set monday = %weekday(t)==1
garch(model=armean, asymm, xreg, distrib=t, $
  pmethod=simplex, peters=10, method=bfgs)
# monday
*
* BEKK GARCH with shift dummy
*
garch(model=armean, mv=bekk, asymm, xreg, distrib=t, $
  pmethod=simplex, peters=10, method=bfgs)
# monday

```

Example 6.4 VECM-GARCH with X Regressor

This is the example from Section 6.5. It estimates a series of VECM models for spot and forward rates with DBEKK covariance matrices with a variance shift based upon the VECM disequilibrium. This is based upon Lee (1994).

```

open data bbjof1989.xls
data(format=xls,org=columns) 1 1245 fuk fwg fja fca ffr fit fsw $
                                suk swg sja sca sfr sit ssw

*
dec vect[labels] suffix(7)
input suffix
    uk wg ja ca fr it sw
*
dec hash[string] longnames
compute longnames("uk")="United Kingdom"
compute longnames("wg")="West Germany"
compute longnames("ja")="Japan"
compute longnames("ca")="Canada"
compute longnames("fr")="France"
compute longnames("it")="Italy"
compute longnames("sw")="Switzerland"
*
* This uses more exact derivatives to improve behavior of estimates.
*
nlpar(derives=fourth,exactline)
*
dec label country
dofor country = suffix
    disp
    disp %STRREP(" ",80)
    disp "Analysis for "+longnames(country)
    *
    * SFORWARD and SSPOT are the "handles" for the series of forward and
    * spot for the country being analyzed.
    *
    compute sforward=%s("f"+country)
    compute sspot    =%s("s"+country)
    *
    * The empirical results in the paper use just log(x) in the
    * transformation. Using 100 log(x) improves the ability to estimate
    * the model by changing the scale on the variance constants. The only
    * other things that will be affected are the CONSTANT in the mean
    * model, and the values of the log likelihoods and regression
    * criteria---the other GARCH coefficients, the lag coefficients in
    * the mean model, the likelihood ratio statistics and the ordering of
    * the models by information criteria (in short, everything that
    * actually matters) will be unaffected. You can (closely) replicate
    * the actual results by changing the 100* to 1*. However, the GARCH
    * models actually fit quite a bit better than shown in the paper.
    *
    * Because sforward and sspot are handles to series (not the series
    * themselves), we need the {0} notation to get the current value of

```

```

* the series.
*
set forward = 100*log(sforward{0})
set spot    = 100*log(sspot{0})
set z       = forward-spot
*
* Do Dickey-Fuller tests on the log spot, log forward and the
* difference. Expected results are unit roots on the first two
* and not on the difference.
*
@dfunit(lags=10,method=bic) spot
@dfunit(lags=10,method=bic) forward
@dfunit(lags=10,method=bic) z
*
@varlagselect(lags=10,crit=bic)
# spot forward
*
equation(coeffs=||1.0,-1.0||) ecteq z
# forward spot
*
system(model=vecm)
variables spot forward
lags 1 to %%autop
det constant
ect ecteq
end(system)
*
* Straight VECM without GARCH errors
*
estimate(model=vecm)
compute loglmodel1=%logl
@regcrits
*
* Diagonal BEKK without X regressor
*
garch(model=vecm,mv=dbekk,pmethod=simplex,piters=10,itors=500)
compute loglmodel2=%logl
@regcrits
*
* Diagonal BEKK with lagged error correction term as an "X"
* regressor
*
garch(model=vecm,mv=dbekk,pmethod=simplex,piters=10,itors=500,$
      xbekk=separate,xreg)
# z{1}
compute loglmodel3=%logl
@regcrits
*
disp longnames(country) loglmodel1 loglmodel2 loglmodel3
cdf(title="GARCH vs No GARCH") chisqr 2*(loglmodel2-loglmodel1) 4
cdf(title="GARCH-X vs GARCH") chisqr 2*(loglmodel3-loglmodel2) 3
end dofor country
*
* Fix the problems with the UK data and re-do the estimates

```

```

*
* There are two entries where forward seems off (429 and 763) and
* one where spot is the problem (866). This does linear
* interpolation (on the logs), which is probably almost exactly
* what a more sophisticated patch procedure would produce. (Kalman
* smoothing across a missing value on a near random walk process is
* the same as linear interpolation).
*
set forward = 100.0*log(fuk)
compute forward(429)=.5*(forward(428)+forward(430))
compute forward(763)=.5*(forward(762)+forward(764))
set spot    = 100.0*log(suk)
compute spot(866)=.5*(spot(865)+spot(867))
*
set z        = forward-spot
*
@varlagselect(lags=10,crit=bic)
# spot forward
*
equation(coeffs=||1.0,-1.0||) ecteq z
# forward spot
*
system(model=vecm)
variables spot forward
lags 1 to %%autop
det constant
ect ecteq
end(system)
*
* Straight VECM without GARCH errors
*
estimate(model=vecm)
compute loglmodel1=%logl
@regcrits
*
* Diagonal BEKK without X regressor
*
garch(model=vecm,mv=dbekk,pmethod=simplex,piters=10,itters=500)
compute loglmodel2=%logl
@regcrits
*
* Diagonal BEKK with lagged error correction term as an "X"
* regressor
*
garch(model=vecm,mv=dbekk,pmethod=simplex,piters=10,itters=500,$
      xreg,xbekk=separate)
# z{1}
compute loglmodel3=%logl
@regcrits
*
cdf(title="GARCH vs No GARCH") chisqr 2*(loglmodel2-loglmodel1) 4
cdf(title="GARCH-X vs GARCH") chisqr 2*(loglmodel3-loglmodel2) 3
*
* Diagonal BEKK with lagged error correction term as an "X"

```

```

* regressor, using the combined rather than separate form.
*
garch(model=vecm,mv=dbekk,pmethod=simplex,piters=10,itters=500,$
      xreg,xbekk=combined)
# z{1}

```

Example 6.5 Hedge Ratios and Portfolio Weights

This is the example from Section 6.6.

```

open data "exrates(daily).xls"
calendar(d) 2000:1:3
data(format=xls,org=columns) 2000:01:03 2008:12:23 $
  aust euro pound sw ca
*
set reuro = 100.0*log(euro/euro{1})
set rpound = 100.0*log(pound/pound{1})
set rsw = 100.0*log(sw/sw{1})
*
* These are used for labeling the graphs and the tables
*
dec vect[string] longlabel(3) shortlabel(3)
compute longlabel=||"Euro","UK Pound","Swiss Franc"||
compute shortlabel=||"Euro","GBP","SWF"||
*
system(model=mvmean)
variables reuro rpound rsw
lags
det constant
end(system)
*
garch(model=mvmean,rvectors=rd,hmatrices=hh,robusterrors,$
      pmethod=simplex,piters=20,method=bfgs,itters=500)
*
* Compute hedge ratios.
*
dec rect[series] hedges(%nvar,%nvar)
do i=1,%nvar
  do j=1,%nvar
    if i==j
      next
    set hedges(i,j) = hh(t)(i,j)/hh(t)(i,i)
  end do j
end do i
*
* Graph them. Position being hedged is in the row.
*
spgraph(hfields=%nvar,vfields=%nvar,footer="Hedge Ratios")
do i=1,%nvar
  do j=1,%nvar
    if i==j

```



```

        next
    graph(row=i,col=j,$
        header="Hedge of "+longlabel(i)+" with "+longlabel(j))
    # hedges(i,j)
end do j
end do i
spgraph(done)
*
* This is how information is presented in Sadorsky(2012), with one
* line per pair.
*
report(use=hreport,action=define,$
    title="Hedge ratio (long/short) summary statistics")
report(use=hreport,atrow=1,atcol=2,align=center) $
    "Mean" "St Dev" "Min" "Max"
do i=1,%nvar
    do j=1,%nvar
        if i==j
            next
        stats(noprint,fract) hedges(i,j)
        report(use=hreport,row=new,atcol=1) $
            shortlabel(i)+"/"+shortlabel(j) $
            %mean sqrt(%variance) %minimum %maximum
    end do j
end do i
report(use=hreport,atcol=2,atrow=2,action=format,$
    align=decimal,picture="*.###")
report(use=hreport,action=show)
*
* Portfolio weights (pairwise, long positions only)
*
dec rect[series] weights(%nvar,%nvar)
do i=1,%nvar
    do j=1,%nvar
        if i==j
            next
        set weights(i,j) = $
            (hh(t)(j,j)-hh(t)(i,j))/(hh(t)(i,i)-2*hh(t)(i,j)+hh(t)(j,j))
        *
        * This constrains the positions to the range[0,1]
        *
        set weights(i,j) = %min(1.0,%max(0.0,weights(i,j)))
        *
    end do j
end do i
*
spgraph(vfields=%nvar,hfields=%nvar,xlabels=longlabel,$
    ylabels=longlabel,footer="Pairwise Portfolio Weights")
do i=1,%nvar
    do j=1,%nvar
        if i==j
            next
        graph(row=i,col=j,style=bar,min=0.0,max=1.0)
        # weights(i,j)
    end do j
end do i

```

```

        end do j
    end do i
    spgraph(done)
    *
    report(use=wreport,action=define,$
        title="Portfolio weights summary statistics")
    report(use=wreport,atrow=1,atcol=2,align=center) $
        "Mean" "St Dev" "Min" "Max"
    do i=1,%nvar
        do j=i+1,%nvar
            stats(noprint,fract) weights(i,j)
            report(use=wreport,row=new,atcol=1) $
                shortlabel(i)+"/"+shortlabel(j) $
                %mean sqrt(%variance) %minimum %maximum
        end do j
    end do i
    report(use=wreport,action=format,atcol=2,atrow=2,$
        picture="*.###",align=decimal)
    report(use=wreport,action=show)
    *
    * Portfolio weights (system-wide, long positions only)
    *
    dec vect[series] pweights(%nvar)
    clear(zeros) pweights
    do t=%regstart(),%regend()
        *
        * The long position constraint is governed by the NNEG=%NVAR
        * option (which is the default anyway). That forces all elements
        * of the solution to be non-negative. NNEG=0 will allow positive
        * or negative values.
        *
        lqprog(noprint,q=hh(t),a=%ones(1,%nvar),b=1.0,equalities=1,nneg=%nvar) x
        compute %pt(pweights,t,x)
    end do t
    *
    * Graph the positions. Whether a stacked bar graph is the best
    * choice for this is debateable---with three or more series, the
    * middle one has floating limits, which makes it harder to judge
    * period-to-period changes to it.
    *
    graph(style=stacked,series=pweights,min=0.0,max=1.0,$
        key=below,klabels=longlabel,$
        footer="Long-position Portfolio Weights")

```

Extensions Using MAXIMIZE-Univariate Models

The **GARCH** instruction was added to RATS with Version 6 in 2004. Before that, there was a substantial body of programs available for estimating GARCH models using the general optimizer **MAXIMIZE**. Given the lags between when empirical work is done and when papers get published, **GARCH** probably didn't start showing up in the literature until about 2008. It's still common to see users actively working with **MAXIMIZE** code to estimate a model that could be done with the original version 6 **GARCH**.

It usually isn't that difficult to do the programming for the (log) likelihood function itself for using **MAXIMIZE**. Much of the work, in fact, is a set of preliminary tasks that are handled automatically by **GARCH**. These are:

1. Examining the input series (dependent variables and any **REGRESSOR** or **XREGRESSORS** variables) to determine the maximum range.
2. Running a preliminary regression for the mean model to get guess values for it, and pre-sample values for the covariance matrix.
3. Setting the guess values for GARCH parameters.

If a model *can* be estimated using **GARCH**, you should use it. Aside from being easier (since you don't have to do the above steps), it's also faster, usually by a factor of about four.

What types of models *can't* be done using **GARCH**?

- Any model with restrictions on the lagged variance or squared residuals terms in the GARCH recursion. For instance, the ARCH model in Engle (1982) enforces a declining pattern on the ARCH lags.
- Any model with lagged variance or squared residuals terms which aren't fixed across the sample—breaks or dummy effects can only be in the variance *intercepts*, not in the “garch” lag coefficients.
- Any model with non-linear interactions among the coefficients.

For a univariate model, the (very) general structure of a GARCH model is

$$\begin{aligned}
 y_t &= f(X_t, h_t, \beta) + u_t \\
 h_t &= g(h_{t-1}, h_{t-2}, \dots, u_{t-1}, u_{t-2}, \dots, Z_t, \theta) \\
 Eu_t &= 0 \\
 Eu_t^2 &= h_t
 \end{aligned}$$

Table 7.1: GARCH(1,2) Estimates

GARCH Model - Estimation by BFGS

Convergence in 23 Iterations. Final criterion was 0.0000024 <= 0.0000100

Dependent Variable SP

Monthly Data From 1926:01 To 1999:12

Usable Observations888

Log Likelihood-2640.3396

	Variable	Coeff	Std Error	T-Stat	Signif
1.	Mean	0.6317	0.1351	4.6745	0.0000
2.	C	0.7151	0.2598	2.7526	0.0059
3.	A{1}	0.0500	0.0314	1.5929	0.1112
4.	A{2}	0.0941	0.0438	2.1481	0.0317
5.	B	0.8393	0.0247	34.0131	0.0000

We can estimate this by maximum likelihood if we choose a distribution for u_t , or by GMM otherwise.

7.1 A Simple Example

In Example 7.1, we'll look at Example 3.5 from Tsay (2010). This uses monthly returns data on the SP500 from 1926 to 1999—a total of 888 observations.¹

```
open data m-ibmspln.dat
calendar(m) 1926
data(format=prn,org=columns) 1926:1 1999:12 ibm sp
```

He first fits a GARCH(1,2) model with a fixed mean. This is done with **GARCH** using

```
garch(p=1,q=2) / sp
```

On this basis (Table 7.1), he decided to run a 1,2 model with the first ARCH lag dropped. This is a questionable idea, as it produces an odd dynamic for the GARCH effect. However, we'll do it for illustration.

The basic setup for estimating such a model by maximum likelihood is:

```
linreg sp
# constant
frml(lastreg,vector=beta) meanf
nonlin(parmset=meanparms) beta

set u = %resids
set uu = %seesq
set h = %seesq
*
```

¹The data set also includes returns on IBM stock, which we won't use here.

```

nonlin(parmset=garchparms) a0 a2 b1
compute a2=.05,b1=.75,a0=%seesq*(1-a2-b1)
frml hf = a0+a2*uu{2}+b1*h{1}
frml logl = u=sp-meanf,uu=u^2,h=hf,%logdensity(h,u)

```

There's an example similar to this in the RATS *User's Guide*. By using a **LINREG** for the mean model (even for a simple model like this with just a fixed mean), you can more easily adapt what you've done to a different data set, or can adjust the model if diagnostics require. The **UU** and **H** are used to hold the squared residuals and generated variance—these are initialized to the sample variance from the mean model, which is exactly what **GARCH** does by default (3.3). While the **SET** instructions fill in the entire range, only the pre-sample values actually matter at this point.

The **FRML HF** defines the formula to compute the variance at time t . In this case, it uses the second lag of **UU** and the first lag of **H**. The **LOGL** formula does most of the work: the four components of this

1. Compute the current residual U (element of a series) using the dependent variable and the mean formula.
2. Compute and place the square of U into the current entry of **UU** for future calculations.
3. Compute the current variance and save it into the **H** series.
4. Compute the log likelihood (assuming Normal residuals). Since this is the final calculation, it's the value that the **LOGL** formula returns.

The **A2** and **B1** coefficients are “scale-free” values—these are fairly typical guess values when a definite GARCH effect is anticipated. The **A0** is not, which is why it is initialized to a scale of the variance (which has the proper scale for **A0**). This adapts to the guess values given to **A2** and **B1** to give a process with the same (unconditional) variance as the linear regression.

A “feature” of some legacy code for using **MAXIMIZE** is writing out the log likelihood, which would here be done with something like

```
frml logl = u=sp-meanf,uu=u^2,h=hf,-.5*log(h)-.5*u^2/h
```

Using **%LOGDENSITY** is more efficient, clearer, and also includes all the integrating constants, so the log likelihood is directly comparable if you compare it with estimates using t or GED residuals.

The actual estimation can be done with

```
maximize(parmset=regparms+garchparms) logl 3 *
```

One thing that you have to do when estimating such a model is to pick the proper start date for the estimation. **GARCH** handles this automatically by scanning the mean model (and any exogenous shifts in the variance) as the lagged

squared residuals and lagged variance are all generated series. When you have to write out the formulas yourself, the `UU` and `H` (at least pre-sample) are now “data”. The start of the data range for which you can compute the log likelihood depends upon:

1. Data points lost in initial transformation of the data (for instance, to returns). Here none, since the data are already returns.
2. Data points lost to lags in the mean model (here also none).
3. Lags in the variance recursion (here two on the `UU`).

Thus, the earliest point in this case is 3. You can also use **INQUIRE** with the `REGLIST` option to figure this out. Here,

```
inquire(reglist) gstart
# sp uu{2} h{1}
```

will give a value of 3 for `STARTG`—the “regressors” that you need to list are the dependent variable (here `SP`), and other regressors in the mean or variance models (none here), and any required lags of `UU`, `U` and `H` needed in the variance recursion. You would then use

```
maximize(parmset=meanparms+garchparms) logl gstart *
```

to estimate the model over the maximum possible range.

If you look at “legacy” programs for doing GARCH with **MAXIMIZE**, you will often see something like:

```
maximize(method=simplex,recursive,itors=15,noprint) logl gstart gend
maximize(method=bfgs,robust,recursive,itors=300) logl gstart gend
```

First, these two can now be combined into a single instruction by use of the `PMETHOD` and `PITERS` options for the initial simplex iterations. The `RECURSIVE` option is no longer needed (and is simply ignored)—until version 5, the default behavior for **MAXIMIZE** was to compute numerical derivatives an observation at a time. This produces incorrect results when (as in the case of a GARCH model), the derivatives “cumulate” because of the recursive calculation of the h function. The `RECURSIVE` option was to force the derivatives to be calculated one parameter at a time through the entire data set. There’s a (very) slight computational advantage to the original way of computing these, but it’s so small that we changed **MAXIMIZE** to always do the parameter by parameter derivatives.

The modern way to do this is

```
maximize(pmethod=simplex,piters=15,method=bfgs,itors=300,robust) $
logl gstart gend
```

Table 7.2: McLeod-Li Tests for GARCH with Skipped Lag

Lag	Corr	Partial	LB Q	Q Signif
1	0.074	0.074	4.928619	
2	-0.031	-0.037	5.778517	
3	-0.008	-0.003	5.838929	0.0157
4	-0.021	-0.022	6.243785	0.0441
5	-0.028	-0.025	6.922088	0.0744
6	-0.004	-0.002	6.939929	0.1391
7	-0.042	-0.044	8.509042	0.1303
8	0.027	0.032	9.141259	0.1658
9	0.021	0.013	9.538460	0.2163
10	-0.002	-0.003	9.540797	0.2987
11	0.041	0.042	11.067047	0.2711
12	0.003	-0.004	11.076703	0.3516
13	-0.028	-0.023	11.789011	0.3797
14	-0.011	-0.008	11.894464	0.4542
15	0.003	0.006	11.901673	0.5357
16	-0.022	-0.021	12.346864	0.5785
17	-0.003	-0.002	12.355386	0.6520
18	-0.021	-0.021	12.764822	0.6899
19	0.016	0.017	13.007010	0.7357
20	-0.016	-0.025	13.249936	0.7765

Because the `FRML` generates the residuals and variances as a side effect of the calculation, doing standard diagnostics is fairly simple. This uses `@REGCORRS` to do the Q tests since it can print out a sequential table of the statistics.

```
set stdu %regstart() %regend() = u/sqrt(h)
@regcorrs(number=20,dfc=1,qstat,report,title="Ljung-Box Tests") stdu
set usq = stdu^2
@regcorrs(number=20,dfc=2,qstat,report,title="McLeod-Li Tests") usq
```

This turns out to be particularly useful here, since the table of McLeod-Li tests is shown in Table 7.2. While the overall test on 20 lags is fine, the tests on the shorter lags aren't as clean.²

For comparison, we can run a standard one-lag GARCH model using the same setup and over the same range.³

```
frml hf = a0+a2*uu{1}+b1*h{1}
maximize(parmset=meanparms+garchparms) logl gstart *
*
set stdu %regstart() %regend() = u/sqrt(h)
@regcorrs(number=20,dfc=1,qstat,report,title="Ljung-Box Tests") stdu
set usq = stdu^2
@regcorrs(number=20,dfc=2,qstat,report,title="McLeod-Li Tests") usq
```

²There are no significance levels until lag 3 since we have to allow for two degrees of freedom for the estimated GARCH parameters.

³For simplicity, we apply the A_2 coefficient to the first lag.

Table 7.3: GARCH(1,1) Model for Comparison

MAXIMIZE - Estimation by BFGS					
Convergence in 15 Iterations. Final criterion was 0.0000005 <= 0.0000100					
Monthly Data From 1926:03 To 1999:12					
Usable Observations		886			
Function Value		-2636.8241			
	Variable	Coeff	Std Error	T-Stat	Signif
1.	BETA(1)	0.6904	0.1409	4.9000	0.0000
2.	A0	0.6465	0.2257	2.8643	0.0042
3.	A2	0.1171	0.0198	5.9012	0.0000
4.	B1	0.8653	0.0187	46.3271	0.0000

Despite the fact that, in the original GARCH(1,2) model, the second lag showed as more significant, the fit with just lag 1 (Table 7.3) is somewhat better than with just lag 2 (Table 7.1) and the low lag McLeod-Li tests (not shown) are clearly better. The ARCH lag coefficient itself is slightly smaller on lag 1, but the higher GARCH lag compensates for that. This is a reminder that, unlike in linear models, the t -statistics on non-linear models only *approximate* the effect of dropping a variable.

7.2 GMM Estimation

The fact that GMM estimates aren't tied to a specific distribution for the residuals would seem to be a strong point in their favor, but, in practice, they often produce much *poorer* estimates.

The two conditions used in estimating GARCH and similar models using GMM are

$$Eu_t = 0$$

$$E(u_t^2 - h_t) = 0$$

where u_t and h_t are functions of their own lags, the data and the free parameters. These are actually both expectations given information through $t - 1$, so any product between either one and a function of the data through $t - 1$ is usable as a moment condition. It's the choice of the particular moment conditions that affects how well the estimator functions.

An example which includes estimation by GMM comes from Gospodinov (2005). This is an analysis of the dynamics of the yields on one-month Treasury notes, monthly data from 1954:1 to 1997:6. The full data set (only R1M is used) is read with:

```
open data fama5497.dat
calendar(m) 1954:1
data(format=prn,nolabels,org=columns) 1954:01 1997:06 $
r1m r3m r1yr r2yr r3yr r4yr r5yr
```

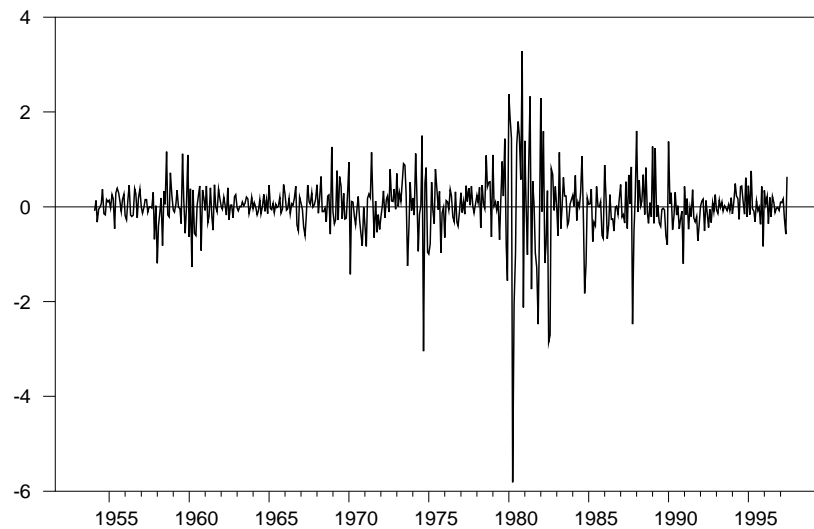



Figure 7.1: Dynamics of Changes in One-Month Rate

Figure 7.1 shows the graph of the first difference from:

```
graph(footer="Figure 1 Dynamics of one-month rate")
# r1m
set d1m = r1m-r1m{1}
graph(footer="Figure 2 Dynamics of changes in one-month rate")
# d1m
```

This has very quiet periods for much of the data range with a very noisy stretch from 1980 to 1982. As a first possible model for the heteroscedasticity (his Table 1), the author chooses a power scedastic function:

$$h_t = \sigma^2 r_{t-1}^{2\rho}$$

where σ^2 and ρ are free parameters. This is analyzed in Example 7.2. The mean model used is

$$\Delta r_t = a_0 + a_1 r_{t-1} + a_2 \Delta r_{t-1} + a_3 \Delta r_{t-2} + u_t$$

The number of lags was chosen using AIC though the results (Table 7.4) of a Dickey-Fuller test would indicate that three lags *on the difference* (thus one more than used) is more appropriate. However, we'll use the model from the paper.

We first of all set up the mean model using

```
linreg r1m
# constant r1m{1} d1m{1 2}
*
frml(lastreg,vector=arc,parmset=meanparms) linear
```

Table 7.4: Dickey-Fuller Test on Interest Rate

Dickey-Fuller Unit Root Test, Series R1M		
Regression Run From 1954:05 to 1997:06		
Observations		519
With intercept		
With 3 lags chosen from 6 by AIC		
Sig Level	Crit Value	
1%(**)		-3.44517
5%(*)		-2.86741
10%		-2.56989
T-Statistic		-2.57537

Table 7.5: Just Identified GMM Estimates

GMM-Continuously Updated Weight Matrix					
Convergence in 16 Iterations. Final criterion was 0.0000084 <= 0.0000100					
Monthly Data From 1954:01 To 1997:06					
Usable Observations		519			
Skipped/Missing (from 522)		3			
Function Value		0.0000			
	Variable	Coeff	Std Error	T-Stat	Signif
1.	ARC(1)=Constant	0.1628	0.0858	1.8978	0.0577
2.	ARC(2)=R1M{1}	0.9708	0.0198	49.0439	0.0000
3.	ARC(3)=D1M{1}	0.0012	0.0908	0.0129	0.9897
4.	ARC(4)=D1M{2}	-0.0589	0.0839	-0.7021	0.4826
5.	SIGSQ	0.0024	0.0022	1.1093	0.2673
6.	RHO	1.3853	0.2337	5.9275	0.0000

This creates `LINEAR` as a `FRML` which evaluates the mean model for any setting of `ARC` for the four free parameters. An obvious set of guess values for the variance model are to make $\rho = 0$ with σ^2 taken from the OLS estimates:

```
compute sigsq=%seesq, rho=0.0
nonlin(parmset=varparms) sigsq rho
```

The following two formulas will have expected value zero (if the model is correct) when multiplied by any function of data through $t - 1$:

```
frml mean      = (r1m-linear)
frml variance = (r1m-linear)^2-(sigsq*r1m{1}^(2*rho))
```

The estimates in the paper are done using the four regressors as instruments for the mean model and just 1 and the lagged interest rate as instruments for the variance model. This type of asymmetrical handling of the instruments can be done using `NLSYSTEM` with the `MASK` option. The mask matrix has dimensions number of instruments \times number of equations with 1's and 0's to indicate which instruments are used in each equation. In this case, we would want

Table 7.6: Overidentified GMM Estimates

GMM-Continuously Updated Weight Matrix

Convergence in 7 Iterations. Final criterion was 0.0000019 <= 0.0000100

Monthly Data From 1954:01 To 1997:06

Usable Observations519

Skipped/Missing (from 522)3

Function Value0.9376

J-Specification(2)0.9376

Significance Level of J0.6257

	Variable	Coeff	Std Error	T-Stat	Signif
1.	ARC(1)=Constant	0.1309	0.0767	1.7058	0.0880
2.	ARC(2)=R1M{1}	0.9779	0.0178	54.9634	0.0000
3.	ARC(3)=D1M{1}	0.0103	0.0902	0.1137	0.9094
4.	ARC(4)=D1M{2}	-0.0303	0.0752	-0.4032	0.6868
5.	SIGSQ	0.0038	0.0025	1.5391	0.1238
6.	RHO	1.2636	0.1698	7.4401	0.0000

```
nlsystem(parmset=meanparms+varparms,inst,zudep,$
mask=|1,1|1,1|1,0|1,0|) / mean variance
```

which produces Table 7.5, which matches the left column in Table 1 in the paper. This doesn't look unreasonable, until you look at the OLS estimates and note that the two sets of estimates of the autoregressive model are identical. This is by construction given the choice for the GMM conditions, as, with a just identified model, all the sample moment conditions will be zero (note the function value), and zeroing the sums of residuals \times regressors gives OLS. This may be by construction, but probably isn't by design, since the model has an explicit description of the heteroscedasticity and then these estimates ignore it in estimating the mean parameters.

If you use the full set of four instruments for both equations, the simpler

```
nlsystem(parmset=meanparms+varparms,inst,zudep) / mean variance
```

gives Table 7.6. Now the regression estimates are no longer matching OLS, but are only slightly different—the rather insignificant J statistic means that the least squares moment conditions have to be *almost* met, even if not exactly. In some ways, this estimation is more “dangerous” than the previous one, since it *doesn't* match OLS and thus doesn't look as obviously wrong.

The problem with both of these is that they use a poor choice of moment conditions given the model. While the estimates generated are consistent, the sums of the sample moments for both the mean and the variance will be dominated by the data points where the variance is high.

An alternative way to write the variance condition which doesn't have this

Table 7.7: GMM with Conditions Corrected for Heteroscedasticity

GMM-Continuously Updated Weight Matrix

Convergence in 9 Iterations. Final criterion was 0.0000027 <= 0.0000100

Monthly Data From 1954:01 To 1997:06

Usable Observations	519
Skipped/Missing (from 522)	3
Function Value	0.8500
J-Specification(2)	0.8500
Significance Level of J	0.6538

	Variable	Coeff	Std Error	T-Stat	Signif
1.	ARC(1)=Constant	0.1294	0.0626	2.0669	0.0387
2.	ARC(2)=R1M{1}	0.9768	0.0156	62.6467	0.0000
3.	ARC(3)=D1M{1}	-0.0139	0.0803	-0.1730	0.8627
4.	ARC(4)=D1M{2}	-0.0498	0.0720	-0.6922	0.4888
5.	SIGSQ	0.0313	0.0101	3.0863	0.0020
6.	RHO	0.7350	0.0920	7.9906	0.0000

problem is

$$\frac{h_t}{\sigma^2 r_{t-1}^{2\rho}} - 1 = 0$$

Combine with a mean equation which corrects for heteroscedasticity, and you get

```
frml mean      = (rlm-linear)/(sigsq*rlm{1}^(2*rho))
frml variance = (rlm-linear)^2/(sigsq*rlm{1}^(2*rho))-1.0
nlsystem(parmset=meanparms+varparms,inst,zudep) / mean variance
```

which produces Table 7.7, which has a substantially smaller value for ρ than GMM with uncorrected moment conditions.

You can do maximum likelihood estimates (assuming Gaussian residuals) with:

```
frml logl = %logdensity(sigsq*rlm{1}^(rho*2),rlm-linear)
maximize(parmset=meanparms+varparms) logl
```

The output from maximum likelihood is Table 7.8, which shows an even smaller value for ρ .

We can compute the estimated entry-by-entry variances with

```
set mlvariance = sigsq*rlm{1}^(2*rho)
```

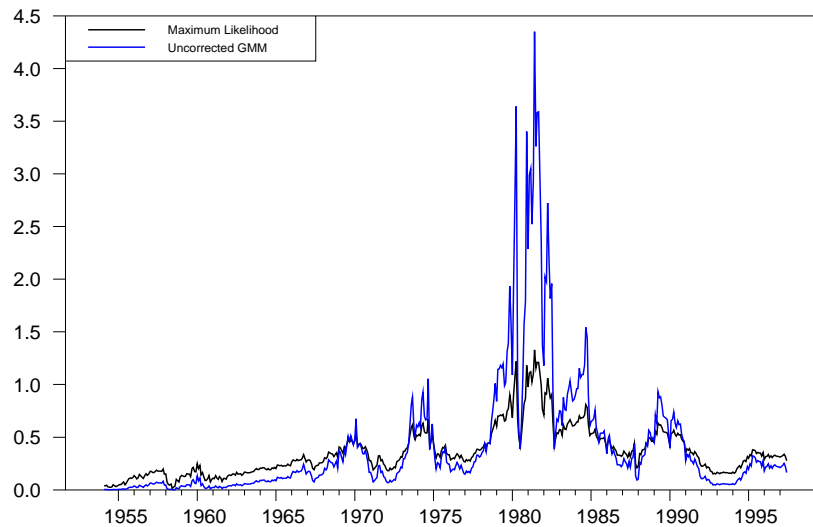
We can compare this with the same calculation done after the original GMM estimates:

```
graph(footer="Comparison of Variance Estimates",$
  key=upleft,klabels=||"Maximum Likelihood","Uncorrected GMM"||) 2
# mlvariance
# gmmvariance
```

Table 7.8: Maximum Likelihood Estimates

MAXIMIZE - Estimation by BFGS					
Convergence in 16 Iterations. Final criterion was 0.0000078 <= 0.0000100					
Monthly Data From 1954:01 To 1997:06					
Usable Observations	519				
Skipped/Missing (from 522)	3				
Function Value	-411.2797				
Variable	Coeff	Std Error	T-Stat	Signif	
1. ARC(1)=Constant	0.1318	0.0325	4.0525	0.0001	
2. ARC(2)=R1M{1}	0.9766	0.0083	117.1640	0.0000	
3. ARC(3)=D1M{1}	-0.0377	0.0396	-0.9514	0.3414	
4. ARC(4)=D1M{2}	-0.0289	0.0315	-0.9188	0.3582	
5. SIGSQ	0.0455	0.0055	8.2871	0.0000	
6. RHO	0.6064	0.0339	17.9056	0.0000	

to get Figure 7.2. Note how the (uncorrected) GMM estimates are much more volatile—lower in the quiet periods, (much) higher in the noisy ones. In order to avoid the high cost of a few very large outliers, the GMM overestimates the variance for a large segment of the data set.

**Figure 7.2:** Comparison of Variance Estimates

For a model such as this with a static variance function, it's possible to interpret maximum likelihood as a (relatively) simple GMM estimator. Ignoring constants, and using θ to represent the free parameters, the log likelihood for entry t is:

$$-\frac{1}{2} \log h_t(\theta) - \frac{u_t^2(\theta)}{2h_t(\theta)}$$

Its derivative with respect to θ is:

$$-\frac{1}{2} \frac{h'_t(\theta)}{h_t(\theta)} + \frac{h'_t(\theta) u_t^2(\theta)}{2[h_t(\theta)]^2} - \frac{u_t(\theta)}{h_t(\theta)} u'_t(\theta) = \frac{1}{2} \frac{h'_t(\theta)}{h_t(\theta)} \left[1 - \frac{u_t^2(\theta)}{h_t(\theta)} \right] - \frac{u_t(\theta)}{h_t(\theta)} u'_t(\theta)$$

For a GARCH model, all elements of θ will enter into the derivatives of h since the residuals at t feed through to compute h_{t+1} . Here, however, parameters just in the mean model will only be in the final term, and the derivatives are (minus) the regressors. When summed across t , this will just be the moment condition for GLS estimates given h . With

$$h_t(\theta) = \sigma^2 r_{t-1}^{2\rho}$$

we can write

$$\log h_t(\theta) = \log \sigma^2 + 2\rho \log r_{t-1}$$

With respect to σ^2 , $h'/h = \sigma^{-2}$ which is just a constant, and with respect to ρ , $h'/h = 2 \log r_{t-1}$.

The six conditions require the same instruments as before (for the mean model), plus $\log r_{t-1}$ for the variance,⁴ with the forms for the mean and variance used in the corrected form of GMM.

With a mask on the conditions to pick out the six interactions that we need, we get an (almost perfect) match with maximum likelihood:

```
set logrlm = log(rlm)
instruments constant rlm{1} dlm{1 2} logrlm{1}
nlssystem(parmset=meanparms+varparms, inst, zudep, $
  mask=|1,1|1,0|1,0|1,0|0,1|) / mean variance
```

⁴Constant scaling factors like the -1 in the mean moments and the σ^{-2} and 2 in the variance moments can be ignored.

7.3 GARCH Model with Multiplicative Factor

In the previous section, we estimated a model with a static form of heteroscedasticity. The next step in Gospodinov (2005) (shown in Example 7.3) was to combine the “level” effect with the dynamics of a GARCH model. Thus the variance is high when the (lagged) interest rate is high, and also can be high due to inertia from an underlying GARCH process. In his model, these are combined by multiplication:

$$\sigma_t^2 = r_{t-1}^{2\rho} \tilde{h}_t$$

$$\tilde{h}_t = c + b\tilde{h}_{t-1} + cu_{t-1}^2/r_{t-2}^{2\rho}$$

The GARCH recursion is thus in “deflated” data. To get both appropriate guess values for the parameters, and to get presample values for \tilde{h} , we first estimate the static model with level effect by maximum likelihood as was done above:

```
linreg rlm
# constant rlm{1} dlm{1 2}
*
frml (lastreg,vector=arc,paramset=meanparms) linear
compute sigsq=%seesq,rho=0.0
nonlin(paramset=varparms) sigsq rho
frml logl = %logdensity(sigsq*rlm{1}^(rho*2),rlm-linear)
maximize (paramset=meanparms+varparms) logl
```

The appropriate pre-sample values for the \tilde{h} and the lagged deflated u_t^2 is the SIGSQ estimate.

We’ll still use H and UU for the series entering the GARCH recursion. At this point, everything will look very similar to a standard GARCH:

```
set uu = sigsq
set h = sigsq
set u = 0.0
*
nonlin(paramset=garchparms) c a b rho
compute a=.05,b=.75,c=sigsq*(1-a-b),rho=0.0
frml varf = (c+a*uu{1}+b*h{1})
```

The difference comes in the LOGL formula, which computes U as the residual with the level effect removed, does the standard recursion based upon that, then puts the level effect back when computing the log likelihood:

```
frml logl = effect=rlm{1}^rho, (u=(rlm-linear)/effect), $
      (uu(t)=u^2), (h(t)=varf(t)), %logdensity(effect^2*h,effect*u)
maximize (paramset=meanparms+garchparms,robusterrors) logl 4 *
```

EFFECT isn’t a series in the formula above, so to get the model’s estimate of the variance, we have to compute it again with each data point:

Table 7.9: GARCH with Level Effect

MAXIMIZE - Estimation by BFGS

Convergence in 40 Iterations. Final criterion was 0.0000009 <= 0.0000100

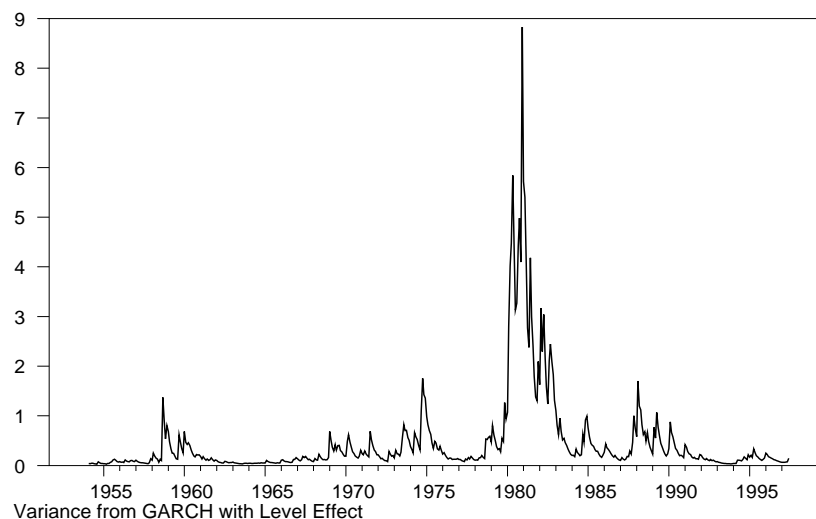
With Heteroscedasticity/Misspecification Adjusted Standard Errors

Monthly Data From 1954:04 To 1997:06

Usable Observations519

Function Value-329.6089

	Variable	Coeff	Std Error	T-Stat	Signif
1.	ARC(1)=Constant	0.1070	0.0360	2.9728	0.0030
2.	ARC(2)=R1M{1}	0.9836	0.0098	100.3450	0.0000
3.	ARC(3)=D1M{1}	-0.1164	0.0502	-2.3202	0.0203
4.	ARC(4)=D1M{2}	-0.0662	0.0504	-1.3138	0.1889
5.	C	0.0018	0.0014	1.2598	0.2078
6.	A	0.2700	0.0962	2.8058	0.0050
7.	B	0.7135	0.0874	8.1610	0.0000
8.	RHO	0.6549	0.2803	2.3367	0.0195

**Figure 7.3:** Level GARCH Variance

```

set combinedvar = effect=r1m{1}^(rho),effect^2*h
graph(footer="Variance from GARCH with Level Effect")
# combinedvar

```

The results of the estimation are seen in Table 7.9 and Figure 7.3.

This fits much better than the model without GARCH (Table 7.8). We can estimate the GARCH without the level effect either by using **GARCH**, or imposing $\text{RHO}=0$ on the model just estimated (full output is omitted):

```

nonlin(parmset=pegrho) rho=0.0
maximize(parmset=meanparms+garchparms+pegrho,robusterrors) logl 4 *

```

The combined model definitely fits better (-329.6 vs -336.8 log likelihood),

Table 7.10: EGARCH-X Model

GARCH Model - Estimation by BFGS					
Convergence in 28 Iterations. Final criterion was 0.0000033 <= 0.0000100					
Dependent Variable R1M					
Monthly Data From 1954:04 To 1997:06					
Usable Observations		519			
Log Likelihood		-328.6893			
	Variable	Coeff	Std Error	T-Stat	Signif
1.	Constant	0.1122	0.0238	4.7255	0.0000
2.	R1M{1}	0.9800	0.0063	156.4781	0.0000
3.	D1M{1}	-0.1536	0.0359	-4.2739	0.0000
4.	D1M{2}	-0.0933	0.0476	-1.9591	0.0501
5.	C	-0.7130	0.1086	-6.5671	0.0000
6.	A	0.5142	0.0769	6.6842	0.0000
7.	B	0.9141	0.0201	45.4717	0.0000
8.	LOGR1M{1}	0.1319	0.0357	3.6958	0.0002

though a standard GARCH effect goes a long way towards explaining the variance.

A model which has some similarities to the model with the multiplicative level effect is an EGARCH model with the log lagged interest rate as an XREGRESSOR. This will also have a variance which is a power of lagged R1M times a GARCH recursion. If we feed into this the same presample variance used above, the model fits slightly better than the level effect model (Table 7.10).

```

set logrlm = log(r1m{1})
garch(exp,reg,xreg,p=1,q=1,presample=sigsq,$
      pmethod=simplex,piters=30,hseries=egarchvar) / r1m
# constant r1m{1} d1m{1 2}
# logrlm{1}

```

Example 7.1 Maximum Likelihood Estimates of Restricted GARCH Model

This is based upon Example 3.5 from Tsay (2010), and is discussed in Section 7.1.

```

open data m-ibmspln.dat
calendar(m) 1926
data(format=prn,org=columns) 1926:1 1999:12 ibm sp
*
* Estimate GARCH(1,2) model
*
garch(p=1,q=2) / sp
*
* Estimate with first ARCH lag constrained to zero.
*
linreg sp
# constant
frml(lastreg,vector=beta) meanf
nonlin(parmset=meanparms) beta
*
set u = %resids
set uu = %seesq
set h = %seesq
*
nonlin(parmset=garchparms) a0 a2 b1
compute a2=.05,b1=.75,a0=%seesq*(1-a2-b1)
frml hf = a0+a2*uu{2}+b1*h{1}
frml logl = u=sp-meanf,uu=u^2,h=hf,%logdensity(h,u)
*
* Restricted GARCH(2,1) model
*
maximize(parmset=meanparms+garchparms) logl 3 *
*
* Using INQUIRE to determine the start of the estimation range
*
inquire(reglist) gstart
# sp uu{2} h{1}
maximize(parmset=meanparms+garchparms) logl gstart *
*
* Do diagnostics
*
set stdu %regstart() %regend() = u/sqrt(h)
@regcorrs(number=20,dfc=1,qstat,report,title="Ljung-Box Tests") stdu
set usq = stdu^2
@regcorrs(number=20,dfc=2,qstat,report,title="McLeod-Li Tests") usq
*
* Re-estimate with an standard GARCH(1,1) using a comparable setup.
*
frml hf = a0+a2*uu{1}+b1*h{1}
maximize(parmset=meanparms+garchparms) logl gstart *
*
set stdu %regstart() %regend() = u/sqrt(h)

```

```
@regcorrs (number=20,dfc=1,qstat,report,title="Ljung-Box Tests") stdu
set usq = stdu^2
@regcorrs (number=20,dfc=2,qstat,report,title="McLeod-Li Tests") usq
```

Example 7.2 GMM Estimates of Heteroscedastic Model

This estimates the model with heteroscedastic level effect from Gospodinov (2005), discussed in Section 7.2.

```
open data fama5497.dat
calendar(m) 1954:1
data(format=prn,nolabels,org=columns) 1954:01 1997:06 $
    r1m r3m r1yr r2yr r3yr r4yr r5yr
*
graph(footer="Figure 1 Dynamics of one-month rate")
# r1m
set d1m = r1m-r1m{1}
graph(footer="Figure 2 Dynamics of changes in one-month rate")
# d1m
*
@dfunit(method=aic,lags=6) r1m
*
linreg r1m
# constant r1m{1} d1m{1 2}
*
frml(lastreg,vector=arc,parmset=meanparms) linear
compute sigsq=%seesq,rho=0.0
nonlin(parmset=varparms) sigsq rho
frml mean      = (r1m-linear)
frml variance = (r1m-linear)^2-(sigsq*r1m{1}^(2*rho))
*
instruments constant r1m{1} d1m{1 2}
*
* Using just the minimal set of conditions
*
nlsystem(parmset=meanparms+varparms,inst,zudep,$
    mask=|1,1|1,1|1,0|1,0|) / mean variance
*
* Using the overidentified model with all instruments used for
* both conditions.
*
nlsystem(parmset=meanparms+varparms,inst,zudep) / mean variance
set gmmvariance = sigsq*r1m{1}^(2*rho)
*
* An alternative for the variance moment condition that's superior
* as it isn't dominated by a few large values is to divide by the
* predicted variance to produce:
*
frml mean      = (r1m-linear)/(sigsq*r1m{1}^(2*rho))
frml variance = (r1m-linear)^2/(sigsq*r1m{1}^(2*rho))-1.0
nlsystem(parmset=meanparms+varparms,inst,zudep) / mean variance
```

```

*
* This estimates the model by maximum likelihood.
*
frml logl = %logdensity(sigsq*rlm{1}^(rho*2),rlm-linear)
maximize(parmset=meanparms+varparms) logl
set mlvariance = sigsq*rlm{1}^(2*rho)
*
* Graph a comparison of the variance estimates
*
graph(footer="Comparison of Variance Estimates",$
      key=upleft,klabels=|"Maximum Likelihood","Uncorrected GMM"|) 2
# mlvariance
# gmmvariance
*
* This does the identical estimates using GMM by figuring out the
* moment conditions implied by maximum likelihood.
*
set logrlm = log(rlm)
instruments constant rlm{1} dlm{1 2} logrlm{1}
nlsystem(parmset=meanparms+varparms,inst,zudep,$
        mask=|1,1|1,0|1,0|1,0|0,1|) / mean variance

```

Example 7.3 GARCH Model with Multiplicative Level Effect

This does the GARCH model with a multiplicative power level effect from Gospodinov (2005), discussed in Section 7.3.

```

open data fama5497.dat
calendar(m) 1954:1
data(format=prn,nolabels,org=columns) 1954:01 1997:06 $
      rlm r3m r1yr r2yr r3yr r4yr r5yr
set dlm = rlm-rlm{1}
*
* Estimate the static level effect model by maximum likelihood
*
linreg rlm
# constant rlm{1} dlm{1 2}
*
frml(lastreg,vector=arc,parmset=meanparms) linear
compute sigsq=%seesq,rho=0.0
nonlin(parmset=varparms) sigsq rho
frml logl = %logdensity(sigsq*rlm{1}^(rho*2),rlm-linear)
maximize(parmset=meanparms+varparms) logl
*
* This does the GARCH estimate with the level effect. To make this
* as similar as possible to other RATS programs for univariate
* GARCH, we'll use U for the author's epsilon.
*
* The UU and H are initialized to the estimates of the variance
* from the system with the level effect.
*

```

```

set uu = sigsq
set h = sigsq
set u = 0.0
*
nonlin(parmset=garchparms) c a b rho
compute a=.05,b=.75,c=sigsq*(1-a-b),rho=0.0
frml varf = (c+a*uu{1}+b*h{1})
*
* The GARCH variance propagates on the process with the level effect
* removed. We have to put the level effect back in when evaluating
* the likelihood.
*
frml logl = effect=r1m{1}^rho, (u=(r1m-linear)/effect), $
      (uu(t)=u^2), (h(t)=varf(t)), %logdensity(effect^2*h, effect*u)
maximize(parmset=meanparms+garchparms, robusterrors) logl 4 *
*
set combinedvar = effect=r1m{1}^(rho), effect^2*h
graph(footer="Variance from GARCH with Level Effect")
# combinedvar
*
* GARCH without level effect
*
nonlin(parmset=pegrho) rho=0.0
maximize(parmset=meanparms+garchparms+pegrho, robusterrors) logl 4 *
*
* Similar model to level effect done with EGARCH
*
set logr1m = log(r1m{1})
garch(exp, reg, xreg, p=1, q=1, presample=sigsq, $
      pmethod=simplex, pitters=30, hseries=egarchvar) / r1m
# constant r1m{1} d1m{1 2}
# logr1m{1}

```

Extensions Using MAXIMIZE-Multivariate Models

The advice from the previous chapter about using **GARCH** (possibly with **UADJUST**, **HADJUST** or **SIGNS** options) wherever possible is even more important with multivariate models. For multivariate models, the preparatory steps are more complicated, the parameter setups more involved and the execution time longer.

Also, note that there have been many extensions and adjustments made to multivariate GARCH models which have been different from more standard methods, but not necessarily better. Be very wary of any paper that proposes a new technique without showing how it, in practice, is better (or at least no worse) than existing models.

When do you need to move towards the use of **MAXIMIZE**?

- **GARCH** only allows mean models which are linear in things that can be observed directly or which can be constructed from the lagged residuals (created with the **UADJUST** option) or current and lagged variances (created with the **HADJUST** option), and which have no cross-equation restrictions. If you have non-linearities or restrictions (such as a common coefficient on $X_{i,t}$ for different i), then you can't use **GARCH**.
- Effectively any recursion in the variances which isn't included in an option for **GARCH** will need the use of **MAXIMIZE**. Note that it's possible to use the **DENSITY** option to get **GARCH** to work with some of these (Section 8.3).
- Some model types are less about GARCH and more about some other modeling strategy. For instance, in a Markov Switching ARCH or GARCH the "Markov Switching" part is the more important in terms of the approach.

8.1 Preparing the Data and Model

While some multivariate GARCH models are designed to be applied to a fixed number of series (typically two), it's more common that they can be applied to two or more, where the only limit is what can be handled numerically. In general, it's a good idea to write everything using various forms of arrays with variable dimensions. That way, you will generally only have to make some modest changes near the top of the program to change the data set and number of variables.

The easiest way to handle the variables flexibly is to use a `VECTOR[EQUATION]` to define the mean models (if they're linear) or `VECTOR[FRML]` if they aren't. For instance, the early setup in Example 8.1 (for a three variable system) is:

```
compute n=3
*
dec vect[series] u(n)
dec series[symm] h uu
*
dec vect[equation] eqn(n)
dec vect[vect] bmean(n)
nonlin(parmset=meanparms) bmean
*
equation eqn(1) xjpn
# constant xjpn{1}
equation eqn(2) xfra
# constant xfra{1}
equation eqn(3) xsui
# constant xsui{1}
```

which is the last time in the program that the identity of the three dependent variables (XJPN, XFRA and XSUI) needs to be used. Later on, those can be estimated (to get guess values) using `LINREG` with the `EQUATION` option, and the values of the residuals of those equations can be computed using the function `%EQNRVALUE`.

With any multivariate GARCH, we will, in the end, need a formula which, at time t , delivers a recursively calculated $n \times n$ covariance matrix and an n vector of residuals. The log likelihood will be computed with `%LOGDENSITY` for normally distributed residuals and `%LOGTDENSITY` for Student t residuals. How we get there is often quite a bit more cumbersome than with a univariate model. In particular, it's quite common to need a **FUNCTION** to evaluate some key part or parts of the calculation. This is due to the need to evaluate variances or residuals separately for each variable in the system. While this sometimes can be done with a clever use of element-wise matrix operators like `.*`, and usually can be done using the `%DO` function, those often lead to coding that's hard to read and hard to fix.

Most of the time, the function is best evaluated just at a single time period. That makes it easy to use as part of a `FRML` which itself a function of an entry number. The following, for instance, computes the covariance matrix at entry `T` for an SVAR-GARCH model (from Example 8.3):

```

function SVARHMatrix t
type symm SVARHMatrix
type integer t
*
local integer i
*
compute vv(t)=%outerxx(binvs*xt(u,t))
do i=1,n
  compute hhd(t)(i)=$
    g(i)(1)+g(i)(2)*vv(t-1)(i,i)+g(i)(3)*hhd(t-1)(i)
end do i
compute SVARHMatrix=bmat*%diag(hhd(t))*tr(bmat)
end

```

The name of the function is **SVARHMatrix** (not case-sensitive) which evaluates to a SYMMETRIC matrix (**TYPE** instruction in the second line). Its one parameter is **T**, an **INTEGER** (**TYPE** instruction in the third line) which will represent the entry for which we are doing the calculation. Lagged values of various time-dependent objects are computed using `series(T-1)`—you can’t use the typical `{1}` notation for a lag because this is a **FUNCTION**, not a **FRML**.

8.2 Panel GARCH

Example 8.1 is based upon Cermeño & Grier (2006). They propose a set of models which apply individual homogeneity to various parameters in a multivariate GARCH model—these are collectively known as “panel GARCH” models. It’s important to note that (effectively) *all* multivariate GARCH models are applied to a “panel data set”, in the sense that the data are a set of parallel time series data for each of a number of individuals. However, in the typical multivariate GARCH model, the linkages between the equations are generally fairly minor. For instance, in a CC model, the mean and variance equations are separately parametrized across individuals and the only connection is through the contemporaneous correlation—in effect, you have a “seemingly unrelated regression” with a GARCH error process. However, if the dynamics for individuals are quite similar, there may be some gain to joint estimation with some parameters that are common to different equations; this is especially true if the T dimension is not large. However, if the dynamics are *not* similar, you should not be using this.

The applications in the paper (and the working paper that preceded it) are to what would generally be seen as standard regressions in panel data with the addition of GARCH error processes. For instance, one has $N = 5$ and $T = 20$, estimating the well-known Grunfeld investment equation:

$$I_{it} = \mu_i + \beta_F F_{it} + \beta_C C_{it} + u_{it}$$

which is commonly used in textbooks to illustrate either panel data techniques, or seemingly unrelated regressions.¹ Note that the mean equation has common coefficients on F and C , which means that the standard GARCH couldn't be applied no matter what we did to model the variance of u_{it} . Their second example is a dynamic panel data model on inflation in seven Latin American countries over 36 quarters:

$$\pi_{it} = \mu_i + \beta\pi_{i,t-1} + u_{it}$$

Again, note that there is a common coefficient in the mean model. There are well-known methods for estimating both of these models if the residuals are assumed to have the typical structure assumed with panel data regressions (generally serially independent over t both across and within i with fixed (possibly zero) covariance across i at t). What Cermeño and Grier address is what to do if the error process is GARCH, rather than serially independent. Note that both of these applications have a small T and, particularly, a small T relative to N . No typical multivariate GARCH model is likely to be very successful in working with a data set of that size. A CC has the three GARCH model coefficients per equation plus the correlation matrix with $N(N-1)/2$ free parameters while a DCC has the problem that it needs that freely estimated $N \times N$ covariance on which to base the recursion, which may be nearly singular with those types of dimensions. The authors instead use common coefficients on the lag terms in the GARCH recursion: (using mostly our notation)

$$\begin{aligned} h_{ii,t} &= c_i + \gamma u_{i,t-1}^2 + \delta h_{ii,t-1} \\ h_{ij,t} &= \eta + \rho u_{i,t-1} u_{j,t-1} + \lambda h_{ij,t-1} \quad \text{if } i \neq j \end{aligned} \tag{8.1}$$

This is a DVECH recursion with common lag coefficients on the variance terms (with an individual-specific intercept) and common coefficients for all terms in the covariance. The common η may be a problem in practice, as, if the variances are allowed to differ across individuals (c_i is individual-specific), it's a bit hard to imagine that a single fixed number can handle all the off-diagonal terms adequately. On the other hand, freely estimating all the off-diagonal variance constants pushes the parameter count up to $4 + N(N+1)/2$ which is the type of size that we would be trying to avoid with N so large relative to T .

Example 8.1 applies these ideas to the three exchange rates used in the `GARCHMV.RPF` example. This has a very large T (6237 observations) so would not be an obvious candidate for panel GARCH from that standpoint. However, there might still be some interest in seeing how similar the variance dynamics are for the series, and there may be data sets of this size where a common coefficient in the mean model might be of interest, so we'll demonstrate how to do that as well.

We already described the initial setup of the model on page 182. This uses univariate AR's for each equation, though there is really no need for the lagged

¹ I is investment, F is the firm's market value and C is the firm's capital stock.

dependent variable—we’re including that to illustrate how to impose the restriction of common coefficients.

This estimates each of the equations (in a loop) to get base residuals and guess values for the coefficients. `BMEAN` is a `VECTOR[VECTOR]` where the outer index is over equations.

```
do i=1,n
  linreg(equation=eqn(i))
  compute bmean(i)=%beta
  set u(i) = %resids
end do i
```

This sets the later estimation range based upon the range of the last regression. If you have data that aren’t defined over the same range, you may need to do a joint estimation first to get an overall range instead.

```
compute gstart=%imax(%regstart(),2),gend=%regend()
```

and this computes the covariance matrix of the residuals, saves it and then initializes the pre-sample variances and lagged outer product of residuals. This is the same method used internally by **GARCH**.

```
vcv
# u
compute rr=%sigma
*
gset h * gend = rr
gset uu * gend = rr
```

This next is a general function for computing the log likelihood at entry `T` assuming conditionally Normal residuals. It has “plug-in” functions for computing the `VECTOR` of residuals (`UFNC`) and the `SYMMETRIC` covariance matrix (`HFNC`). We’ll define the functions that will be used for those later—different applications would require different functions for those.

```
function MVLogLNormal t ufnc hfnc
type real MVLogLNormal
type integer t
type function[vector](integer) ufnc
type function[symm](integer) hfnc
*
local symm hx
local vect ux
```

```

*
compute hx=hfnc(t)
compute h(t)=hx
compute ux=ufnc(t)
compute %pt(u,t,ux)
compute uu(t)=%outerxx(ux)
compute MVLogLNormal=%logdensity(hx,ux)
end

```

MVLogLNormal first computes the covariance matrix, and saves it into the current slot in the SERIES[SYMM] H, which we're using for the generated covariance matrices. The covariance matrix needs to be computed first in case the mean has an "M" effect and so depends upon the current variance. It then computes the residuals, and saves those into the current slots in the VECT[SERIES] U (that's what the %PT function does). UU(T) is made equal to the outer product of the residuals, so it will be available in the next period. Note that U and UU are organized differently—U is a VECT[SERIES] so there are separately accessible series of residuals for each series, while UU is a SERIES[SYMM], so at entry T each element is a SYMMETRIC matrix. This is because the separate elements of UU(T) are really not of direct interest.

This next defines the function of entry T which can be used to return the residuals from the equations set up earlier. This will work for any set of linear equations using BMEAN(I) as the coefficient vector for equation I. We'll see later how to impose homogeneity restrictions on the equations without having to change this function.

```

function ULinearF t
type vector    ULinearF
type integer   t
*
local integer i
*
dim ULinearF(n)
ewise ULinearF(i)=%eqnrvalue(eqn(i),t,bmean(i))
end

```

As mentioned earlier, the GARCH part of panel GARCH is a DVECH with restrictions. This evaluates the DVECH variance recursion given the matrices of coefficients VCS (variance model constants), VBS (coefficients on the lagged covariance matrix) and VAS (coefficients on the lagged outer product of residuals). This assumes that H is a SERIES[SYMM] for the variances and UU is a SERIES[SYMM] for the outer products, as we set up earlier. The panel GARCH restrictions will be imposed by changing how VCS, VBS and VAS are constructed.

```

dec symm vcs(n,n) vbs(n,n) vas(n,n)
*
function HDVECHF t
type symm    HDVECHF
type integer t
*
compute HDVECHF=vcs+vbs.*h(t-1)+vas.*uu(t-1)
end

```

The first variant will have no restrictions on the mean, and will allow for freely estimated variance constants in (8.1). Thus the GARCH variance parameters will be the full matrix of variance constants (the VCS array), plus the four separate values for the lagged terms.

```

dec real delta lambda gamma rho
compute delta=lambda=.8,gamma=rho=.1
compute vcs=rr*(1-gamma-delta)
nonlin(parmset=garchparms) vcs delta lambda gamma rho

```

This function will be called during the START processing for **MAXIMIZE** to fill out the VBS and VAS arrays. With both, the diagonal elements (when $I=J$, which are the variance recursions) take one value and the off-diagonal the other:

```

function PGARCHInit
local integer i j
ewise vbs(i,j)=%if(i==j,delta,lambda)
ewise vas(i,j)=%if(i==j,gamma,rho)
end

```

This now defines a FRML (which is what **MAXIMIZE** needs) that returns the log likelihood at T using the previously defined functions ULinearF to compute the residuals and HDVECHF to evaluate the covariance matrix at T.

```

frml thisLogL = MVLogLNormal(t,ULinearF,HDVECHF)

```

and the final working instruction is:

```

maximize(start=PGARCHInit(),parmset=meanparms+garchparms,$
  pmethod=simplex,piters=10,method=bfgs,iters=400) $
  thisLogL gstart gend

```

The START option calls the **PGARCHInit** function before each function evaluation to do the calculations which depend upon parameters, but not the entry—here, that creates the desired VBS and VAS matrices from the basic parameters.

The output is in Table 8.1. BMEAN(1) has the coefficients from the first equation (for XJPN), where BMEAN(1)(1) is the first variable in that equation (its

Table 8.1: Panel GARCH Estimates with Free Variance Constants

MAXIMIZE - Estimation by BFGS
Convergence in 63 Iterations. Final criterion was 0.0000000 <= 0.0000100

Usable Observations	6235				
Function Value	-11867.3628				

	Variable	Coeff	Std Error	T-Stat	Signif
1.	BMEAN(1)(1)	0.0047	0.0064	0.7434	0.4572
2.	BMEAN(1)(2)	0.0211	0.0107	1.9668	0.0492
3.	BMEAN(2)(1)	-0.0044	0.0060	-0.7348	0.4624
4.	BMEAN(2)(2)	0.0099	0.0096	1.0325	0.3018
5.	BMEAN(3)(1)	-0.0035	0.0075	-0.4597	0.6457
6.	BMEAN(3)(2)	0.0045	0.0093	0.4841	0.6283
7.	VCS(1,1)	0.0096	0.0009	10.5009	0.0000
8.	VCS(2,1)	0.0054	0.0007	7.2801	0.0000
9.	VCS(2,2)	0.0094	0.0010	9.3936	0.0000
10.	VCS(3,1)	0.0064	0.0008	7.6850	0.0000
11.	VCS(3,2)	0.0087	0.0011	8.0552	0.0000
12.	VCS(3,3)	0.0124	0.0014	8.6323	0.0000
13.	DELTA	0.8791	0.0047	185.2194	0.0000
14.	LAMBDA	0.8860	0.0047	187.8156	0.0000
15.	GAMMA	0.1115	0.0052	21.3987	0.0000
16.	RHO	0.1038	0.0051	20.2875	0.0000

CONSTANT) and **BMEAN**(1)(2) is for its AR term, similarly for the other. The **DELTA-GAMMA** and **LAMBDA-RHO** combinations have sensible-looking values for series with strong GARCH properties. In fact, it looks like the further restrictions of $\delta = \lambda$ and $\gamma = \rho$ might not be unreasonable.

We now set up the further restriction of a common η in (8.1). This requires a different parameter set for the variance constants, with a **VECTOR** (called **VCI** here) for the diagonal and the single **ETA** value for the off-diagonals and a new **START** function which fills in **VCS** as well.

```
compute delta=lambda=.8,gamma=rho=.1
dec vect vci(n)
ewise vci(i)=rr(i,i)*(1-gamma-delta)
compute eta=.5*%avg(rr)*(1-gamma-delta)
nonlin(parmset=garchparmsr) vci delta lambda gamma rho eta

function PGARCHInitR
local integer i j
ewise vbs(i,j)=%if(i==j,delta,lambda)
ewise vas(i,j)=%if(i==j,gamma,rho)
ewise vcs(i,j)=%if(i==j,vci(i),eta)
end
```

The only changes needed to the previous **MAXIMIZE** instruction are to the

START option and the PARMSET (which now uses GARCHPARMSR rather than GARCHPARMS):

```
maximize(start=PGARCHInitR(), parmset=meanparms+garchparmsr, $
    pmethod=simplex, pitters=10, method=bfgs, iters=400) $
    thisLogL gstart gend
```

We won't show the full output from this. The log likelihood drops to -11903 vs -11867 without the restriction on η . When there are just two fewer parameters, that's too big a drop even with over 6000 data points. As we said earlier, the restriction down to a single off-diagonal element in the covariance recursions when the variances themselves aren't restricted seems likely to cause problems in practice. Unless you have a very high ratio of N to T (and data with very, very similar scales), you probably want to stick to the less restrictive form.

We now add the still further restriction of common "slope coefficients" in the mean models. As with the previous restriction, this can be done by using a new START function which copies information into the BMEAN VECTOR's. This will be a different function than is used for the GARCH parameters so it can be easily used in any combination with the GARCH recursion restrictions.

The new parameter set will have separate intercepts for each equation (we put those in the VECTOR named MU) and a single VECTOR (which will here have dimension just 1) for the slope coefficients, called BCOMMON. As constructed, this can be used for a homogeneity restriction on any number of slope coefficients in the model.

```
dec vect mu(n)
do i=1,n
    linreg(noprint, equation=eqn(i))
    compute mu(i)=%beta(1)
end do i
compute bcommon=%xsubvec(%beta, 2, %nreg)
nonlin(parmset=meanparmsr) mu bcommon

*
function PGARCHInitMean
local integer i j
do i=1,n
    ewise bmean(i)(j)=%if(j==1, mu(i), bcommon(j-1))
end do i
end
```

This assumes the explanatory variables are the constant followed by the "slope" variables in that order. The %IF replaces the first slot in BMEAN(I) with MU(I) and otherwise uses the elements of the common vector BCOMMON which have to be shifted because element 1 in BCOMMON is coefficient 2 in the equation itself.

The **MAXIMIZE** now “adds” the `PGARCHInitMean()` to the `START` option, replaces the `MEANPARMS` with `MEANPARMSR` and that’s it:

```
maximize(start=PGARCHInitR()+PGARCHInitMean(), $
  parmset=meanparmsr+garchparmsr, $
  pmethod=simplex, pitters=10, method=bfgs, iters=400) $
  thisLogL gstart gend
```

This is barely different from the previous results, which isn’t much of a surprise since all the lagged dependent variables are close to zero.

Finally, we compare to a full DVECH without restrictions (Table 8.2):

```
group meanmodel eqn(1) eqn(2) eqn(3)
garch(model=meanmodel, mv=dvech) gstart gend
```

While the log likelihood gap between this and the first (and least restricted) panel GARCH is around 35, this time, here there are 8 additional parameters, so it’s just *barely* better than panel GARCH using BIC (the penalty is $8\log T = 69.9$ while $2 \times 35.4 = 70.8$).

8.3 Non-standard Density

Multivariate GARCH models are typically estimated using either a multivariate Normal, or multivariate t as the density for the residuals. Both of these are examples of *elliptically symmetrical* (or just elliptical) densities: these have the property that their density depends on x only through $x'\Sigma^{-1}x$, that is, they are constant on ellipses whose shape is determined by Σ . The t can’t have fatter tails in some directions, and neither is permitted to be skewed.

Bauwens & Laurent (2005) propose an alternative density which doesn’t have elliptical symmetry. What they describe is a general method of converting symmetrical densities to skewed ones which can be applied to (effectively) any well-behaved symmetrical univariate density to create a skewed multivariate one. The one on which they focus is the multivariate skew- t , which is based upon the univariate t . Note that it’s hard, when dealing with correlated variables, to get skew that’s easy to interpret. The authors define the skewness at the level of a standardized (to identity covariance) multivariate distribution, then transform to the desired covariance. The skew coefficients thus aren’t really associated with any specific variables since they’re defined for unobserved uncorrelated random variables.

In their formulation, a skew parameter ξ is a positive number where $\xi > 1$ means that the component is skewed positive and $\xi < 1$ means it’s skewed negative. The skewed (univariate) distribution can be constructed from a symmetrical distribution by multiplying positive values by ξ and dividing negative ones by ξ . To keep the density continuous at 0, the probability of a positive

Table 8.2: Unrestricted DVECH Estimates

MV-GARCH - Estimation by BFGS

Convergence in 63 Iterations. Final criterion was 0.0000035 <= 0.0000100

Usable Observations6235

Log Likelihood-11831.9199

Variable	Coeff	Std Error	T-Stat	Signif
Mean Model(XJPN)				
1. Constant	0.0048	0.0064	0.7527	0.4516
2. XJPN{1}	0.0237	0.0118	2.0046	0.0450
Mean Model(XFRA)				
3. Constant	-0.0034	0.0065	-0.5190	0.6037
4. XFRA{1}	0.0085	0.0104	0.8206	0.4119
Mean Model(XSUI)				
5. Constant	-0.0023	0.0077	-0.2955	0.7676
6. XSUI{1}	0.0029	0.0099	0.2895	0.7722
7. C(1,1)	0.0087	0.0012	7.4153	0.0000
8. C(2,1)	0.0055	0.0008	6.9436	0.0000
9. C(2,2)	0.0112	0.0012	9.1345	0.0000
10. C(3,1)	0.0058	0.0008	7.2068	0.0000
11. C(3,2)	0.0097	0.0012	8.4050	0.0000
12. C(3,3)	0.0125	0.0015	8.4013	0.0000
13. A(1,1)	0.1048	0.0086	12.1346	0.0000
14. A(2,1)	0.0929	0.0063	14.6675	0.0000
15. A(2,2)	0.1266	0.0071	17.8934	0.0000
16. A(3,1)	0.0878	0.0058	15.1496	0.0000
17. A(3,2)	0.1123	0.0059	18.9309	0.0000
18. A(3,3)	0.1103	0.0059	18.7046	0.0000
19. B(1,1)	0.8855	0.0086	103.4202	0.0000
20. B(2,1)	0.8926	0.0070	128.3619	0.0000
21. B(2,2)	0.8627	0.0070	123.3260	0.0000
22. B(3,1)	0.8990	0.0063	142.9845	0.0000
23. B(3,2)	0.8764	0.0061	143.3082	0.0000
24. B(3,3)	0.8788	0.0061	143.1096	0.0000

value must be ξ^2 times the probability of a negative value, so the method of generation (given ξ) would be to:

1. Pick the sign using a Bernoulli that has $P(\text{positive}) = \xi^2 / (1 + \xi^2)$
2. Draw the absolute value $|x|$ from the underlying symmetrical density.
3. Compute the draw $y = |x|\xi$ for positive and $y = -|x|/\xi$ for negative.

As this is described, the mean is positive if $\xi > 1$ and negative if $\xi < 1$. Fortunately, the moments of the skewed variable are a computable function of the skew parameter ξ and the absolute moments of the base distribution. If these exist through the second moment, then the skewed variable can be transformed to zero mean and unit variance by a straightforward transformation. The probability that the *transformed* variable is positive will be less than .5 if $\xi > 1$ in order to offset the positive skew.

The density itself in the standardized form can be computed using the RATS (external) function %LOGMVSKEWT. To make this available, you need to use a SOURCE instruction

```
source logmvskewt.src
```

The added parameters for this are an n vector (of skew parameters) modeled in log form (LOGXI) and the degrees of freedom for the t (NU). ξ values of 1 are non-skewed, so we initialize the LOGXI to zeroes.

```
compute n=3
*
dec vector logxi(n)
nonlin(parmset=skewparms) nu logxi
*
compute logxi=%zeros(n,1)
compute nu=10.0
```

%LOGMVSKEWT(z, xi, nu) returns the log density of a standardized (mean zero, identity covariance) distribution at Z , with NU degrees of freedom. XI is the n -vector of skew parameters. In order to use this in practice, you need to incorporate the GARCH covariance matrix into it. The following takes an input SYMMETRIC covariance matrix H and residual VECTOR U and returns the log density. Because the %LOGMVSKEWT takes XI (and not its logs), we have to pass through %EXP(LOGXI).

```
function logmvskewtfnc h u
type real logmvskewtfnc
type symm h
type vect u
*
compute logmvskewtfnc=$
  %logmvskewt(inv(%decomp(h))*u,%exp(logxi),nu) - .5*logdetxx(h)
end
```

This maps the observable U to the standardized Z , and corrects for the Jacobian.²

If a model is otherwise a standard GARCH other than the non-standard density, you can use the `DENSITY` and `PARMSET` options on **GARCH**, which were added with version 9.2. The `PARMSET` provides just the added parameters needed for the density function:

```
garch(model=exrates,mv=dcc,pmethod=simplex,piters=10,$
      parmset=skewparms,density=logmvskewtfnc)
```

This does the authors' exchange rate example. The data are already in returns.

```
open data txch.xls
data(format=xls,org=columns,left=2) 1 3065 date eurUSD yenUSD $
  gbpUSD r_eurUSD r_yenUSD r_gbpUSD
```

The authors use (univariate) 1 lag AR for the euro and yen—it's not clear how they came to that decision on the yen, which seems to have almost no noticeable serial correlation. BIC picks zero for all, but we'll stick with the choices in the paper.

The estimates of the final model are shown in Table 8.3. For comparison, a model with just a standard t is run. This is a restriction on the previous one, so a likelihood ratio test can be done:

```
garch(model=exrates,mv=dcc,distrib=t,pmethod=simplex,piters=10)
compute loglnoskew=%logl
*
cdf(title="LR Test of Skewness") chisqr 2.0*(loglskew-loglnoskew) n
```

The likelihood ratio test on the skewness parameters is highly significant:

Chi-Squared(3)=	41.721746 with Significance Level 0.00000000
-----------------	--

Even more significant in this case, is the move from Gaussian to t —with a ν of roughly six the tails are quite fat.

²The paper doesn't address the fact that, precisely because the density isn't elliptically symmetrical, the likelihood depends upon the method used to factor the covariance matrix—a different ordering of the variables (and thus a different triangular factorization) will lead to a somewhat different log likelihood function.

Table 8.3: GARCH with Multivariate Skew-T Residuals

MV-DCC GARCH - Estimation by BFGS					
Convergence in 44 Iterations. Final criterion was 0.0000054 <= 0.0000100					
Usable Observations		3064			
Log Likelihood		-6680.2496			
	Variable	Coeff	Std Error	T-Stat	Signif
Mean Model(R.EURUSD)					
1.	Constant	-0.0020	0.0098	-0.2041	0.8383
2.	R.EURUSD{1}	-0.0110	0.0117	-0.9384	0.3480
Mean Model(R.YENUSD)					
3.	Constant	-0.0078	0.0095	-0.8189	0.4129
4.	R.YENUSD{1}	-0.0075	0.0149	-0.4998	0.6172
Mean Model(R.GBPUSD)					
5.	Constant	-0.0053	0.0083	-0.6398	0.5223
6.	R.GBPUSD{1}	0.0048	0.0123	0.3876	0.6983
7.	C(1)	0.0091	0.0020	4.4801	0.0000
8.	C(2)	0.0077	0.0021	3.6064	0.0003
9.	C(3)	0.0040	0.0010	3.9008	0.0001
10.	A(1)	0.0537	0.0062	8.7076	0.0000
11.	A(2)	0.0705	0.0087	8.1299	0.0000
12.	A(3)	0.0530	0.0065	8.1722	0.0000
13.	B(1)	0.9318	0.0077	120.4772	0.0000
14.	B(2)	0.9192	0.0097	94.4037	0.0000
15.	B(3)	0.9381	0.0075	124.3029	0.0000
16.	DCC(A)	0.0342	0.0032	10.6168	0.0000
17.	DCC(B)	0.9631	0.0036	266.5062	0.0000
18.	NU	6.1915	0.3637	17.0241	0.0000
19.	LOGXI(1)	-0.0761	0.0236	-3.2318	0.0012
20.	LOGXI(2)	0.1043	0.0222	4.6879	0.0000
21.	LOGXI(3)	-0.0917	0.0225	-4.0794	0.0000

8.4 Structural VAR with GARCH

There's actually nothing about this type of GARCH model that's specific to the mean model being a VAR. However, the idea behind the handling of the GARCH model is borrowed from the Bernanke-Sims structural VAR model.

In the basic SVAR, the one-step forecast errors (or non-orthogonal shocks) u_t are related to a set of orthogonalized shocks by:

$$A(\theta)u_t = B(\theta)v_t$$

where θ is a set of free parameters. The identifying assumption is that the covariance matrix of v_t is diagonal. Most SVAR's have either an A and no B or a B and no A. These are called A models and B models respectively. With both an A and a B, you have an A-B model.

For an SVAR, due to the properties of the moving average representation, the v_t process has components which are uncorrelated with each other, both contemporaneously and across time. If they were jointly Normal, they would be independent random variables. The assumption in the SVAR GARCH is that, while each component of v_t follows a GARCH process (and so has second moment serial dependence), there is no first or second moment dependence *between* the different components. Thus a GARCH model for n variables will have n separate univariate GARCH models, plus a structural model for mapping the orthogonal process to the observable residuals. If that model is just-identified, there will be $n(n-1)/2$ free parameters in θ . For a just-identified SVAR, there will be exactly the same number of free parameters as there is in a CC GARCH model, so it's about as tightly restricted a model as you can create for a multivariate GARCH.³ The structural model can also be over-identified, which restricts the parameterization still further.

One significant difference between the SVAR GARCH and a regular SVAR is that the maximized likelihood in an SVAR is *identical* for any just-identified model. That's not true with GARCH components, since one linear combination being well-represented by a GARCH process doesn't mean another one is—a perfectly reasonable structural model may fail to fit well because the orthogonalized processes don't seem to follow well-behaved GARCH processes.

Example 8.3 is from Martin et al. (2012). This does a six variable SVAR GARCH model on the UK variables in the data set:

```
calendar(d) 2004:1:1
open data "libor_data.xls"
data(format=xls,nolabels,org=columns,top=3,right=15) $
  2004:01:01 2008:05:28 liborus liboreuro liboruk $
  liboreuribor defeulteuro defeultuk defeultus defeulteuribor $
  repo swapeuro swapgbp liquidity volatility lbspxr
```

³The only way to create a more parsimonious model would be the (closely related) factor GARCH model where there are $r < n$ underlying univariate GARCH processes.

The six variables included in the model are:

- VOLATILITY represented by the VIX index constructed from European put and call option prices
- LIQUIDITY measured by the on-off run U.S. 5-year spread
- SWAP the GDP/USD swap rate
- REPO credit spreads as measured by the REPO spread
- DEFAULT measure of the default risk
- LIBOR 3-month U.K. LIBOR rate spread

Three of those are in the data set under the names above, the three others are created with:

```
set libor    = liboruk
set swap     = swapgbp
set default  = 10000.0*defaultuk
```

The DEFAULT variable is rescaled as the raw data were at a very small magnitude.

Similar to the typical process with a regular SVAR, the lag coefficients are estimated in a first step and the GARCH model is applied to the residuals. For a VAR with a fixed (but unknown) covariance matrix, the two-step procedure is fully justified as maximum likelihood since the likelihood-maximizing estimates of the lag coefficients are just OLS equation by equation regardless of the value of the covariance matrix. For GARCH residuals (of any form), this is no longer the case—however, the OLS estimates *are* consistent, and (more importantly) taking them out of the non-linear parameter set greatly reduces the calculation time. With a two-lag VAR, the number of parameters in the VAR part alone is 78, more than double the number in the GARCH process itself.

```
compute nlags=2
compute nstep=20
*
system(model=basicvar)
variables volatility liquidity swap repo default libor
lags 1 to nlags
det constant
end(system)
*
estimate(resids=u)
```

We grab the N (for use in the GARCH coding) and the start and end limits out of the VAR with:

```
compute n=%nvar
compute gstart=%regstart(),gend=%regend()
```

The structural model is a “B” type with

$$\mathbf{B} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ b_{21} & 1 & b_{23} & 0 & 0 & 0 \\ b_{31} & 0 & 1 & 0 & 0 & 0 \\ b_{41} & 0 & 0 & 1 & 0 & 0 \\ b_{51} & b_{52} & b_{53} & b_{54} & 1 & 0 \\ b_{61} & b_{62} & b_{63} & b_{64} & b_{65} & 1 \end{bmatrix}$$

The six shocks are to be interpreted as (in order):

1. Global risk factor
2. Narrow liquidity factor
3. Broad liquidity factor
4. Credit factor
5. Default factor
6. Idiosyncratic factor

The model is overidentified by 2 parameters. It’s basically a recursive model—the “above” diagonal b_{23} is actually due to ordering: if variables 2 and 3 and shocks 2 and 3 were reversed, the model would be lower triangular. Structural models that (in some arrangement) are lower triangular usually are very well-behaved numerically.

The setup for the \mathbf{B} matrix is largely the same as what you would have for estimating using **CVMODEL**:

```
nonlin(parmset=svarparms) b21 b23 b31 b41 b51 b52 b53 b54 $
    b61 b62 b63 b64 b65
*
frml bfrml = ||$
    1.0,0.0,0.0,0.0,0.0,0.0,0.0|$
    b21,1.0,b23,0.0,0.0,0.0|$
    b31,0.0,1.0,0.0,0.0,0.0|$
    b41,0.0,0.0,1.0,0.0,0.0|$
    b51,b52,b53,b54,1.0,0.0|$
    b61,b62,b63,b64,b65,1.0||
```

We’ll just initialize all of those to zero.

```
compute b21=b23=b31=b41=0.0
compute b51=b52=b53=b54=0.0
compute b61=b62=b63=b64=b65=0.0
```

With those guess values, the underlying orthogonal components are just the observable residuals. Each of those will have a standard univariate GARCH process, so we initialize the parameters to those using a fairly persistent GARCH

and the variance constant which would produce the observed variance of the VAR.

```
dec vect[vect] g(n)
do i=1,n
    ewise g(i)=||%sigma(i,i)*(1-.10-.80),.10,.80||
end do i
nonlin(parmset=garchparms) g
```

Because the GARCH processes are on the unobservable orthogonalized processes, the bookkeeping is a bit different than would be typical. We will have to transform the observed residuals (u) to the unobserved (v) using the B matrix from the structural model and the univariate GARCH models are on the v 's. The `SERIES` below are to keep track of the outer product of the v 's (VV), the variances of the v processes (HHD) and the full model covariance matrices (HH).

```
dec series[symm] vv
dec series[vect] hhd
dec series[symm] hh
dec rect binv
*
gset hhd 1 gend = %zeros(n,1)
gset hh 1 gend = %zeros(n,n)
gset vv 1 gend = %zeros(n,n)
```

Next is the `START` option function which does any calculations required that depend upon parameters, but not upon time, such as the B matrix and its inverse. We need to get the transformed pre-sample values for the (squared) structural residuals and their variances. This uses the fact that

$$u_t = Bv_t \Rightarrow v_t = B^{-1}u_t \Rightarrow Ev_t v_t' = B^{-1} (Eu_t u_t') B^{-1'}$$

```
function SVARStart
local symm vsigma
*
compute bmat=bfrml(1)
compute binv=inv(bmat)
compute vsigma=binv*%sigma*tr(binv)
gset vv 1 gstart-1 = vsigma
gset hhd 1 gstart-1 = %xdiag(vsigma)
end
```

After this is the function which generates the model's covariance matrix for the observables at entry `T`.

```

function SVARHMatrix t
type symm SVARHMatrix
type integer t
*
local integer i
*
compute vv(t)=%outerxx(binvs*%xt(u,t))
do i=1,n
  compute hhd(t)(i)=$
    g(i)(1)+g(i)(2)*vv(t-1)(i,i)+g(i)(3)*hhd(t-1)(i)
end do i
compute SVARHMatrix=bmat*%diag(hhd(t))*tr(bmat)
end

```

The first working line computes the outer product of the orthogonalized residuals. The %XT function takes a “time t ” cross section of a VECT[SERIES] like U : %XT(U, T) is the vector that we would generally describe mathematically as u_t . So

```
compute vv(t)=%outerxx(binvs*%xt(u,t))
```

does the transformation from u to v by pre-multiplying by the matrix BINVS, then uses %OUTERXX to compute $v_t v_t'$ and saves it into entry T of the VV SERIES of SYMMETRICS—that will get used in the *next* period.

```

do i=1,n
  compute hhd(t)(i)=$
    g(i)(1)+g(i)(2)*vv(t-1)(i,i)+g(i)(3)*hhd(t-1)(i)
end do i

```

evaluates the recursion for each component of v . $g(i)$ is the VECTOR of parameters for component i , with $g(i)(1)$ the variance intercept, $g(i)(2)$ the “ARCH” parameter and $g(i)(3)$ the “GARCH” parameter. The target for the calculation is element I of entry T for the SERIES[VECT] HHD. Because this is not being used in a SET instruction or a FRML, we have to use explicit T and $T-1$ subscripts on the series references in this.

The final working line

```
compute SVARHMatrix=bmat*%diag(hhd(t))*tr(bmat)
```

transforms from the variances of the (assumed) orthogonal v to the full covariance of the original u that we need for evaluating the likelihood function. Because SVARHMatrix is the name of the function, this sets the return value.

The only line in the function which we haven’t discussed is

```
local integer i
```


This makes the `T` used in the `DO` loop a different variable than the `T` used in the main program. The formal parameter `T` is also local. The other variables are (intentionally) global in scope as they are either defined outside (such as `N`, `BMAT` and `BINV`) or set and used in multiple calls to the `FUNCTION` (such as `VV` and `HHV`). In general, any variable which is not intentionally in the global name space should be declared `LOCAL` so you can't reset something in the main program by accident.

Because of a great deal of preparation, the log likelihood `FRML` for this model is the (apparently) simple:

```
frml SVARLog1 = hh=SVARHMatrix(t),%logdensity(hh,%xt(u,t))
```

As described above, `HH` is a `SERIES[SYMM]` which has the full covariance matrix. In this case, `HH` isn't used in any of the recursions (since the recursions are on the variances of the transformed data), but will be available for diagnostics or graphics. The formula evaluates the `SVARHMatrix` function for entry `T`, puts the returned matrix information into `HH` (at entry `T`—inside a `FRML`, you don't need the `T` subscript when referring to the current entry of a series), and evaluates the log multivariate Normal density at u_t with covariance matrix `HH(T)` (again, the `T` is assumed here).

```
maximize(start=SVARStart(),parmset=garchparms+svarparms) $
SVARLog1 gstart gend
```

The `START` option calls the `SVARStart` function before each function evaluation to do the calculations which depend upon parameters, but not the entry. In many cases, a `START` option is just for efficiency, to avoid doing a calculation for each entry that could be done once. Here, it's more important than that, because we need to compute a new set of pre-sample variances in order to start the recursion correctly.

The output is shown in Table 8.4. There's nothing about this that looks "wrong". The fourth and fifth components are borderline unstable GARCH recursions, but the fifth (in particular) has such a small variance intercept that it will be a very minor part of the explanation of the data. The first, on the other hand, has a very large intercept and a very persistent GARCH process, so we would expect that it would tend to be a major factor in any variable on which it loads.

One thing about the model that might give us some concern is that a `CC` model fits quite a bit better (log likelihood of -11524.7146). The two don't nest, so there's no simple test, but a log likelihood gap of better than 70 in favor of a much simpler type of model should cause some concern.

We can do the same types of diagnostics discussed in Section 5.3—here, it's even simpler since we have an obvious candidate for what should be mutually uncorrelated, homoscedastic transformed residuals, which are the `v` deflated

Table 8.4: SVAR-GARCH Estimates

MAXIMIZE - Estimation by BFGS					
Convergence in 83 Iterations. Final criterion was 0.0000010 <= 0.0000100					
Daily(5) Data From 2004:01:05 To 2008:05:28					
Usable Observations		1148			
Function Value		-11596.9419			
	Variable	Coeff	Std Error	T-Stat	Signif
1.	G(1)(1)	1.2164	0.2468	4.9279	0.0000
2.	G(1)(2)	0.1198	0.0185	6.4811	0.0000
3.	G(1)(3)	0.8447	0.0214	39.3977	0.0000
4.	G(2)(1)	0.0009	0.0005	1.6408	0.1008
5.	G(2)(2)	0.0359	0.0070	5.1593	0.0000
6.	G(2)(3)	0.9633	0.0064	150.1309	0.0000
7.	G(3)(1)	0.2456	0.0406	6.0550	0.0000
8.	G(3)(2)	0.1447	0.0200	7.2516	0.0000
9.	G(3)(3)	0.8458	0.0170	49.7059	0.0000
10.	G(4)(1)	0.1333	0.0578	2.3037	0.0212
11.	G(4)(2)	0.2926	0.0282	10.3631	0.0000
12.	G(4)(3)	0.7856	0.0150	52.3796	0.0000
13.	G(5)(1)	0.0001	0.0000	3.1604	0.0016
14.	G(5)(2)	0.2327	0.0283	8.2141	0.0000
15.	G(5)(3)	0.7936	0.0203	39.1839	0.0000
16.	G(6)(1)	0.4236	0.0524	8.0862	0.0000
17.	G(6)(2)	0.3439	0.0453	7.5827	0.0000
18.	G(6)(3)	0.6325	0.0336	18.8340	0.0000
19.	B21	0.0018	0.0021	0.8620	0.3887
20.	B23	-0.0068	0.0035	-1.9272	0.0540
21.	B31	0.0687	0.0135	5.0858	0.0000
22.	B41	0.0239	0.0155	1.5446	0.1224
23.	B51	0.0020	0.0004	5.4846	0.0000
24.	B52	0.0003	0.0040	0.0705	0.9438
25.	B53	0.0026	0.0010	2.7274	0.0064
26.	B54	0.0012	0.0005	2.4315	0.0150
27.	B61	0.0137	0.0094	1.4492	0.1473
28.	B62	-0.1220	0.0818	-1.4921	0.1357
29.	B63	0.2184	0.0192	11.3841	0.0000
30.	B64	0.0086	0.0086	1.0038	0.3155
31.	B65	0.1755	0.3198	0.5488	0.5831

by their GARCH standard errors. Preparation of the standardized residuals can be done with:

```
dec vect[series] stdv(n)
do time=gstart,gend
    compute %pt(stdv,time,binv*%xt(u,time)./%sqrt(hhd(time)))
end do time
```

The BINV will have the value from the final function evaluation (which will be at the converged estimates), so `BINV*%XT(U,TIME)` computes the vector `V(TIME)`. Similarly `HHD(TIME)` will have the values of the variances of `V` at the final estimates, so we divide by the square root of that (element by element) to get the standardized residuals.

As before, we can use `@MVQSTAT` and `@MVARCHTEST` to test for remaining (joint) serial correlation and ARCH effects:

```
@mvqstat(lags=10)
# stdv
@mvarchtest(lags=2)
# stdv
```

We get the somewhat sobering results:

Multivariate Q Test		
Test Run Over 2004:01:05 to 2008:05:28		
Lags Tested	10	
Degrees of Freedom	360	
Q Statistic	1252.702	
Signif Level	0.000	
Multivariate ARCH Test		
Statistic	Degrees	Signif
2893.78	882	0.00000

both significant beyond any doubt. If we check the individual elements of `STDV`, there don't seem to be problems at *that* level, but the assumption that there's no joint ARCH effect in the orthogonalized process would appear to not hold with this structural model.

Although we've seen that the structural model is inadequate, for illustration we'll demonstrate how to compute forecast error variance decompositions (FEVD) based upon this model. Variance decompositions with a standard SVAR are relatively simple because the weights on the orthogonalized components are fixed to reflect the (assumed) fixed covariance matrix Σ . With a GARCH process governing the covariances, there will be a different set of weights for each data point—as we've computed this, these will be the (square roots of the) `HHD` matrices. With a traditional VAR, a graphical presentation is usually an $n \times n$ array of graphs, with each showing the responses for a certain number of steps of variable i to a shock j . With a GARCH model, we could do the same

thing, but we would have a different such graph for each data point. An alternative (which is basically what we'll do here) is to fix the horizon and graph across time.

Assuming fixed VAR lag coefficients, the moving average representation is still

$$\mathbf{y}_t = \sum_{s=0}^{\infty} \Psi_s \mathbf{u}_{t-s} = \sum_{s=0}^{\infty} (\Psi_s \mathbf{G}^{-1}) \mathbf{v}_{t-s}$$

where \mathbf{G}^{-1} is the mapping from the orthogonalized \mathbf{v} process to the non-orthogonal \mathbf{u} process—for our model $\mathbf{G}^{-1} = \mathbf{B}$. The forecast error for $t + H$ given information through t is

$$\mathbf{e}_{t+H|t} = \sum_{s=0}^{H-1} (\Psi_s \mathbf{G}^{-1}) \mathbf{v}_{t+H-s}$$

Because (by assumption), the \mathbf{v} are uncorrelated both across time and across shocks, the covariance matrix of this can be written:

$$\sum_{s=0}^{H-1} (\Psi_s \mathbf{G}^{-1}) E(\mathbf{v}_{t+H-s} \mathbf{v}_{t+H-s}' | t) (\Psi_s \mathbf{G}^{-1})'$$

where

$$E(\mathbf{v}_{t+H-s} \mathbf{v}_{t+H-s}' | t)$$

is the variance forecast for the orthogonalized process. For a standard VAR, this is constant and diagonal—for the SVAR GARCH, it is still diagonal (which allows us to decompose the variance into the component due to each shock), but is different for each t and each s given t .

The moving average weights $(\Psi_s \mathbf{G}^{-1})$ are, however, independent of t , and can be computed using a standard **IMPULSE** instruction:

```
impulse(model=basicvar, factor=bmat, results=irf, steps=nstep, $
  labels=|| "Global Risk", "Narrow Liquidity", "Broad Liquidity", $
  "Credit", "Default", "Idiosyncratic" ||)
```

We then also need the forecasts for variances. At each t , we need the forecasts for up to H steps for each of the n components. We organize that into a `VECT[VECT[SERIES]]`, with the outer VECTOR for the components, the inner for the horizon. This does all the time periods simultaneously by using a **SET** instruction. Note that the way this is computed, time `T` will have forecasts using data through `T-1` (since `HHD(T)` are the time `T` forecasts using `T-1`). This is the forecast calculation described in Section 4.1, repeated for different variables and start periods.

```

dec vect[vect[series]] hhdfore(n)
do i=1,n
  dim hhdfore(i) (nstep)
  set hhdfore(i) (1) gstart gend = hhd(t) (i)
  do fh=2,nstep
    set hhdfore(i) (fh) gstart gend = g(i) (1)+$
      (g(i) (2)+g(i) (3)) *hhdfore(i) (fh-1)
  end do fh
end do i

```

The structure for the components to the variances is even more complicated. This will be a `VECT[RECT[SERIES]]`, where outer VECTOR is horizon, the inner RECT is variables \times shocks. When we're done, `VARSUMS(h) (i, j) (t)` will be the contribution of shock j to the variance for variable i to the h -step-ahead forecast given data through $t - 1$. This gives the raw sums of the contributions to the variance—to get the decomposition, these will have to be divided into their sums across j and scaled by 100.0. Note how the forecast variances are used in “reverse” order—the short lags on the impulse response function (`PHIK`) are applied to the longest steps in the `HHDFORE`.

```

dec vect[rect[series]] varsums(nstep)
do fh=1,nstep
  dim varsums(fh) (n,n)
  clear(zeros) varsums(fh)
  do k=1,fh
    compute phik=%xt(irf,k)
    do i=1,n
      do j=1,n
        set varsums(fh) (i, j) = varsums(fh) (i, j) + $
          phik(i, j) ^2 *hhdfore(j) (fh-k+1)
      end do j
    end do i
  end do k
end do fh

```

You could create an entire book with the graphs that could be generated from this. The text does only decompositions for the Libor spread (sixth variable in the system), for one step, five step and twenty step and restricts the display to just three of the shocks: global risk (#1), broad liquidity (#3) and idiosyncratic (#6).

The information in `VARSUMS` is converted into decompositions as needed and put into the `VECT[SERIES] GARCHDECOMP`:

```

dec vect[series] garchdecomp(n)

```

where `N` here represents the shocks. The calculation for the 1 step decomposition is done with:

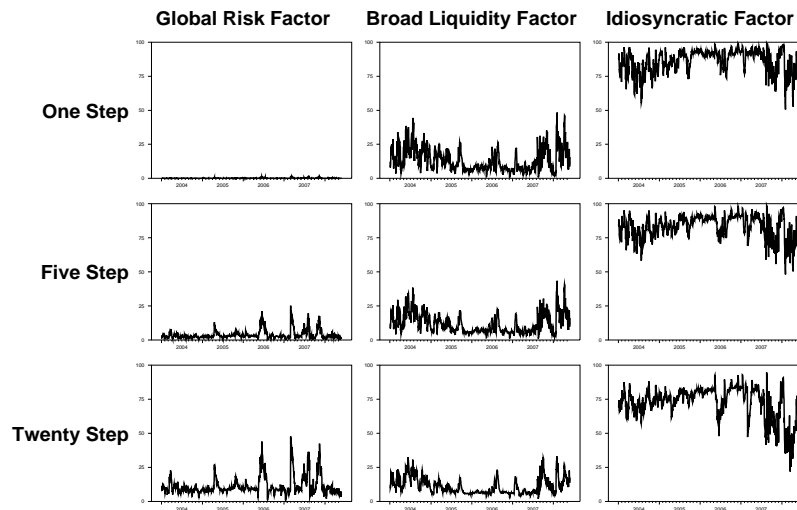


Figure 8.1: Decompositions of Variance for Libor from SVAR-GARCH

```
do j=1,n
  set garchdecomp(j) = varmat=%xt(varsums(1),t), $
    100.0*varmat(6,j)/%sum(%xrow(varmat,6))
end do j
```

The 1 in the `VARSUMS(1)` is the step number. `VARMAT` will be an $n \times n$ matrix with variable in the row and shock in the column so the sum across row i will be the total one step variance for variable i . The decomposition is computed only for $i = 6$ in this case—if you wanted the entire $n \times n$ decomposition, you would need to make change `GARCHDECOMP` to a `RECT[SERIES]` and add a loop over the variable, that is:

```
dec rect[series] garchdecomp(n,n)
do i=1,n
  do j=1,n
    set garchdecomp(i,j) = varmat=%xt(varsums(1),t), $
      100.0*varmat(i,j)/%sum(%xrow(varmat,i))
  end do j
end do i
```

The end result of the graphics instructions is seen in Figure 8.1. Because we only included three of the six shocks, these won't sum to 100% across a row, but generally comes fairly close. The decompositions are a complicated interaction of the IRF coefficients (if the response to a shock is zero, it doesn't matter how large the shock is) and the variance forecasts. For instance, while the Global Risk Factor shock has a high variance relative to the others, it also has a nearly zero impact response on the Libor spread, thus the (effectively) flat line at zero for one step. Its IRF increases fairly steadily through 20 steps and its forecast variance stays high so it steadily increases its percentage explained.

Example 8.1 Panel GARCH

This demonstrates estimation of a multivariate GARCH model with various types of homogeneity restrictions as proposed in Cermeño & Grier (2006) (with one of the standard RATS data sets, rather than the authors’).

```
all 6237
open data g10xrate.xls
data(format=xls,org=columns) / usxjpn usxfra usxsui
*
set xjpn = 100.0*log(usxjpn/usxjpn{1})
set xfra = 100.0*log(usxfra/usxfra{1})
set xsui = 100.0*log(usxsui/usxsui{1})
*
* Estimation using MAXIMIZE
*
compute n=3
*
dec vect[series] u(n)
dec series[symm] h uu
*
dec vect[equation] eqn(n)
dec vect[vect] bmean(n)
nonlin(parmset=meanparms) bmean
*
* Mean equations
* For illustration, this includes a lag of the dependent variable
* in each, though there is no reason for it in this application.
*
equation eqn(1) xjpn
# constant xjpn{1}
equation eqn(2) xfra
# constant xfra{1}
equation eqn(3) xsui
# constant xsui{1}
*
* Estimate by OLS to get guess values
*
do i=1,n
    linreg(equation=eqn(i))
    compute bmean(i)=%beta
    set u(i) = %resids
end do i
*
* Set the range based upon the regression range. (This will adapt
* automatically to lags in the mean model).
*
compute gstart=%imax(%regstart(),2),gend=%regend()
*
* Get the covariance matrix of the OLS residuals to get presample
* values for the H matrices and to use for guess values for the
* GARCH parameters.
*
```

```

vcv
# u
compute rr=%sigma
*
* These are used to initialize pre-sample variances.
*
gset h * gend = rr
gset uu * gend = rr
*
*
function MVLogLNormal t ufnc hfnc
type real MVLogLNormal
type integer t
type function[vector] (integer) ufnc
type function[symm] (integer) hfnc
*
local symm hx
local vect ux
*
compute hx=hfnc(t)
compute h(t)=hx
compute ux=ufnc(t)
compute %pt(u,t,ux)
compute uu(t)=%outerxx(ux)
compute MVLogLNormal=%logdensity(hx,ux)
end
*****
*
* This is a general function for evaluating the residuals. Any
* restrictions on the coefficients will be handled by setting the
* BMEAN vectors based upon a smaller number of free parameters.
*
function ULinearF t
type vector ULinearF
type integer t
*
local integer i
*
dim ULinearF(n)
ewise ULinearF(i)=%eqnrvalue(eqn(i),t,bmean(i))
end
*****
*
* The Panel GARCH is a DVECH with restrictions. This sets up
* functions for evaluating the variance recursion for a DVECH. The
* restrictions will be handled by generating VCS, VBS and VAS
* matrices with the proper forms.
*
dec symm vcs(n,n) vbs(n,n) vas(n,n)
*
function HDVECHF t
type symm HDVECHF
type integer t
*

```



```

compute HDVECHF=vcs+vbs.*h(t-1)+vas.*uu(t-1)
end
*****
*
* Panel GARCH - DVECH with restrictions. This allows for different
* constants in the covariance recursions (so the VCS is fully free).
*
dec real delta lambda gamma rho
compute delta=lambda=.8, gamma=rho=.1
compute vcs=rr*(1-gamma-delta)
nonlin(parmset=garchparms) vcs delta lambda gamma rho
*
* Call once during START option to fill in the VAS and VBS arrays
*
function PGARCHInit
local integer i j
ewise vbs(i,j)=%if(i==j,delta,lambda)
ewise vas(i,j)=%if(i==j,gamma,rho)
end
*
frml thisLogL = MVLogLNormal(t,ULinearF,HDVECHF)
maximize(start=PGARCHInit(),parmset=meanparms+garchparms,$
  pmethod=simplex,piters=10,method=bfgs,itters=400) $
  thisLogL gstart gend
*****
*
* With the further restriction that the covariance recursions have
* a fixed eta value.
*
compute delta=lambda=.8, gamma=rho=.1
dec vect vci(n)
ewise vci(i)=rr(i,i)*(1-gamma-delta)
compute eta=.5*%avg(rr)*(1-gamma-delta)
nonlin(parmset=garchparmsr) vci delta lambda gamma rho eta
*
function PGARCHInitR
local integer i j
ewise vbs(i,j)=%if(i==j,delta,lambda)
ewise vas(i,j)=%if(i==j,gamma,rho)
ewise vcs(i,j)=%if(i==j,vci(i),eta)
end
maximize(start=PGARCHInitR(),parmset=meanparms+garchparmsr,$
  pmethod=simplex,piters=10,method=bfgs,itters=400) $
  thisLogL gstart gend
*****
*
* With the still further restriction of a common AR coefficient
* (with individual-specific intercepts). This is designed to handle
* any number of homogeneous slope coefficients (with
* individual-specific intercepts).
*
dec vect mu(n)
do i=1,n
  linreg(noprint,equation=eqn(i))

```

```

        compute mu(i)=%beta(1)
    end do i
    compute bcommon=%xsubvec(%beta,2,%nreg)
    nonlin(parmset=meanparmsr) mu bcommon
*
function PGARCHInitMean
local integer i j
do i=1,n
    ewise bmean(i)(j)=%if(j==1,mu(i),bcommon(j-1))
end do i
end
*
maximize(start=PGARCHInitR()+PGARCHInitMean(),$
    parmset=meanparmsr+garchparmsr,$
    pmethod=simplex,piters=10,method=bfgs,itters=400) $
    thisLogL gstart gend
*****
*
* Comparison to standard DVECH (no restrictions)
*
group meanmodel eqn(1) eqn(2) eqn(3)
garch(model=meanmodel,mv=dvech) gstart gend

```

Example 8.2 GARCH Model with Multivariate Skew-t Density

This is based upon an example from Bauwens & Laurent (2005), using a DVECH GARCH rather than DCC. This is covered in Section 8.3.

```

open data txch.xls
data(format=xls,org=columns,left=2) 1 3065 date eurUSD yenUSD gbpUSD $
    r_eurUSD r_yenUSD r_gbpUSD
*
* The returns series exist from the start of the data set.
*
* The authors use a 1 lag AR for the euro and yen---it's not clear how
* they came to that decision on the yen, which seems to have almost no
* noticeable serial correlation. BIC picks zero for all, but we'll stick
* with the choices in the paper.
*
@arautolags(maxlags=10,crit=bic,table) r_eurUSD
@arautolags(maxlags=10,crit=bic,table) r_yenUSD
@arautolags(maxlags=10,crit=bic,table) r_gbpUSD
*
source logmvskewt.src
*
compute n=3
*
* The multivariate skew-t has a VECTOR of skewness parameters (one per
* variable), and a single degrees of freedom. This parameterizes the
* "xi" vector in log form so it will have values unbounded above and
* below.

```

```

*
dec vector logxi(n)
nonlin(parmset=skewparms) nu logxi
*
compute logxi=%zeros(n,1)
compute nu=10.0
*
equation eq_eurusd r_eurusd
# constant r_eurusd{1}
equation eq_yenusd r_yenusd
# constant r_yenusd{1}
equation eq_gbpusd r_gbpusd
# constant r_gbpusd{1}
*
group exrates eq_eurusd eq_yenusd eq_gbpusd
*
* Function to take the GARCH U and H and returns the log multivariate
* skew-t density with the current settings for xi and nu.
*
function logmvskewtfnc h u
type real logmvskewtfnc
type symm h
type vect u
*
compute logmvskewtfnc=$
    %logmvskewt(inv(%decomp(h))*u,%exp(logxi),nu)-.5*%logdetxx(h)
end
*
garch(model=exrates,mv=dcc,pmethod=simplex,piters=10,$
    parmset=skewparms,density=logmvskewtfnc)
compute loglskew=%logl
*
* t only (no skew)
*
garch(model=exrates,mv=dcc,distrib=t,pmethod=simplex,piters=10)
compute loglnoskew=%logl
*
cdf(title="LR Test of Skewness") chisqr 2.0*(loglskew-loglnoskew) n

```

Example 8.3 Structural VAR-GARCH

This is Application 20.7.2 from Martin et al. (2012). This is described in detail in Section 8.4.

```

calendar(d) 2004:1:1
open data "libor_data.xls"
data(format=xls,nolabels,org=columns,top=3,right=15) $
    2004:01:01 2008:05:28 liborus liboreuro liboruk $
    liboreuribor defaulteuro defaultuk defaultus defaulteuribor $
    repo swapeuro swapgbp liquidity volatility lbspvx
*

```

```

* Pull out the UK variables, rescaling the default rate.
*
set libor    = liboruk
set swap     = swapgbp
set default  = 10000.0*defaultuk
*
* Number of VAR lags and steps for the impulse responses
*
compute nlags=2
compute nstep=20
*
system(model=basicvar)
variables volatility liquidity swap repo default libor
lags 1 to nlags
det constant
end(system)
*
* The GARCH model is estimated as a two-step procedure, with the VAR
* estimated first and the GARCH model fit to the residuals.
*
estimate(resids=u)
compute n=%nvar
compute gstart=%regstart(),gend=%regend()
*
* The structural model is a "B" type, that is  $u=Bv$  where the  $v$ 's are
* assumed to be orthogonal. (The book uses the opposite convention
* for which is  $u$  and which is  $v$ ---we'll stick with the notation
* used in the RATS documentation).
*
dec frml[rect] bfrml
nonlin(parmset=svarparms) b21 b23 b31 b41 b51 b52 b53 b54 $
    b61 b62 b63 b64 b65
*
frml bfrml = ||$
    1.0,0.0,0.0,0.0,0.0,0.0|$
    b21,1.0,b23,0.0,0.0,0.0|$
    b31,0.0,1.0,0.0,0.0,0.0|$
    b41,0.0,0.0,1.0,0.0,0.0|$
    b51,b52,b53,b54,1.0,0.0|$
    b61,b62,b63,b64,b65,1.0||
*
compute b21=b23=b31=b41=0.0
compute b51=b52=b53=b54=0.0
compute b61=b62=b63=b64=b65=0.0
*
* Parameters for the GARCH processes for the orthogonal V
*
dec vect[vect] g(n)
do i=1,n
    ewise g(i)=||%sigma(i,i)*(1-.10-.80),.10,.80||
end do i
*
nonlin(parmset=garchparms) g
*

```

```

dec series[symm] vv
dec series[vect] hhd
dec series[symm] hh
dec rect binv
*
gset hhd 1 gend = %zeros(n,1)
gset hh 1 gend = %zeros(n,n)
gset vv 1 gend = %zeros(n,n)
*****
*
* This is a "startup" function which needs to be executed at the
* start of a function evaluation. The B matrix and its inverse
* change with the parameters, but not with time. And we need to get
* the transformed pre-sample values for the (squared) structural
* residuals and their variances.
*
function SVARStart
local symm vsigma
*
compute bmat=bfrml(1)
compute binv=inv(bmat)
compute vsigma=binv*%sigma*tr(binv)
gset vv 1 gstart-1 = vsigma
gset hhd 1 gstart-1 = %xdiag(vsigma)
end
*****
*
* This is the function evaluated at each entry to do the recursion
* in the variance of the structural residuals, then maps it back to
* the standard residuals.
*
function SVARHMatrix t
type symm SVARHMatrix
type integer t
*
local integer i
*
compute vv(t)=%outerxx(binv*%xt(u,t))
do i=1,n
  compute hhd(t)(i)=$
    g(i)(1)+g(i)(2)*vv(t-1)(i,i)+g(i)(3)*hhd(t-1)(i)
end do i
compute SVARHMatrix=bmat*%diag(hhd(t))*tr(bmat)
end
*****
frml SVARLogl = hh=SVARHMatrix(t),%logdensity(hh,%xt(u,t))
maximize(start=SVARStart(),parmset=garchparms+svarparms) $
  SVARLogl gstart gend
*
* CC GARCH for comparison
*
garch(p=1,q=1,mv=cc) / u
*
* Do diagnostics on the standardized (orthogonalized) residuals.

```

```

*
dec vect[series] stdv(n)
do time=gstart,gend
    compute %pt(stdv,time,binv*%xt(u,time)./%sqrt(hhd(time)))
end do time
@mvqstat(lags=10)
# stdv
@mvarchtest(lags=2)
# stdv
*
* Compute responses to orthogonalized shocks
*
impulse(model=basicvar,factor=bmat,results=irf,steps=nstep,$
    labels=||"Global Risk","Narrow Liquidity","Broad Liquidity",$
        "Credit","Default","Idiosyncratic"||)
*
* Do up to the requested number of variance forecasts starting at
* each period.
*
dec vect[vect[series]] hhdfore(n)
do i=1,n
    dim hhdfore(i)(nstep)
    set hhdfore(i)(1) gstart gend = hhd(t)(i)
    do fh=2,nstep
        set hhdfore(i)(fh) gstart gend = g(i)(1)+$
            (g(i)(2)+g(i)(3))*hhdfore(i)(fh-1)
    end do fh
end do i
*
* The decomposition of variance will be different at each point
* because of the time-varying variances. varsums(h)(i,j) is the
* contribution of orthogonal shock j to the variance of variable i
* at horizon h. The GARCH variances are used in the opposite order
* of the IRF coefficients.
*
dec vect[rect[series]] varsums(nstep)
do fh=1,nstep
    dim varsums(fh)(n,n)
    clear(zeros) varsums(fh)
    do k=1,fh
        compute phik=%xt(irf,k)
        do i=1,n
            do j=1,n
                set varsums(fh)(i,j) = varsums(fh)(i,j)+$
                    phik(i,j)^2*hhdfore(j)(fh-k+1)
            end do j
        end do i
    end do k
end do fh
*
* The graph is for decompositions of the Libor spread, which is
* variable 6 (hence the 6's in the SET instructions below). The
* varsums are the contributions of the components to the variance,
* so those have to be converted to percentages (by multiplying by

```

```

* 100 and dividing by the total) before graphing.
*
dec vect[series] garchdecomp(n)
*
spgraph(hfields=3,vfields=3,$
  xlabel=||"Global Risk Factor","Broad Liquidity Factor",$
        "Idiosyncratic Factor"||,$
  ylabel=||"One Step","Five Step","Twenty Step"||)
*
* One step
*
do j=1,n
  set garchdecomp(j) = varmat=%xt(varsums(1),t),$
    100.0*varmat(6,j)/%sum(%xrow(varmat,6))
end do j
graph(max=100.0,min=0.0,row=1,col=1)
# garchdecomp(1)
graph(max=100.0,min=0.0,row=1,col=2)
# garchdecomp(3)
graph(max=100.0,min=0.0,row=1,col=3)
# garchdecomp(6)
*
* Five step
*
do j=1,n
  set garchdecomp(j) = varmat=%xt(varsums(5),t),$
    100.0*varmat(6,j)/%sum(%xrow(varmat,6))
end do j
*
graph(max=100.0,min=0.0,row=2,col=1)
# garchdecomp(1)
graph(max=100.0,min=0.0,row=2,col=2)
# garchdecomp(3)
graph(max=100.0,min=0.0,row=2,col=3)
# garchdecomp(6)
*
* Twenty step
*
do j=1,n
  set garchdecomp(j) = varmat=%xt(varsums(20),t),$
    100.0*varmat(6,j)/%sum(%xrow(varmat,6))
end do j
*
graph(max=100.0,min=0.0,row=3,col=1)
# garchdecomp(1)
graph(max=100.0,min=0.0,row=3,col=2)
# garchdecomp(3)
graph(max=100.0,min=0.0,row=3,col=3)
# garchdecomp(6)
*
spgraph(done)

```

Simulations and Bootstrapping

Only a few types of GARCH processes have closed form (recursive) variance forecast procedures. Univariate and multivariate EGARCH processes don't, DCC multivariate processes don't, multivariate model with asymmetry don't. For models like those, the only way to get approximate variance forecasts is with some type of simulation. Even for models with analytical formulas for forecasts, if something more than simply the mean and variance is needed, again some form of simulation is likely required. In this chapter, we examine how to use random simulation or bootstrapping to generate simulated out-of-sample data governed by a GARCH process.

We provide two examples of calculation of Value at Risk (VaR), one for a univariate and one for a multivariate model. These includes (as a subcalculation) the simulated variances, so if that's all you need, it's easy to adapt these to save those instead of the simulated portfolio returns. The third example shows how to use simulation methods to compute a Volatility Impulse Response Function (VIRF) for a model which doesn't allow a closed-form calculation for that.

9.1 Tools

We will be assuming throughout this chapter that the parameters governing the process, along with the variances and residuals generated by the model in sample, are known. Simulation techniques that include inference about the free parameters will be covered in Chapter 10.

A very general structure for a (multivariate) GARCH model is:

$$E(\mathbf{u}_t | \Omega_{t-1}) = 0$$

$$E(\mathbf{u}_t \mathbf{u}_t' | \Omega_{t-1}) \equiv \mathbf{H}_t = f(\mathbf{H}_{t-1}, \mathbf{H}_{t-2}, \dots, \mathbf{u}_{t-1}, \mathbf{u}_{t-2}, \dots)$$

Given data through T , the calculation of \mathbf{H}_{T+1} requires only sample values for the \mathbf{H}_{T-j} and \mathbf{u}_{T-j} . Given \mathbf{H}_{T+1} , to simulate the process we need to draw (somehow) an appropriate \mathbf{u}_{T+1} with mean 0 and the desired variance. Now \mathbf{H}_{T+1} and \mathbf{u}_{T+1} are "data", and we can use the recursion to compute a value for \mathbf{H}_{T+2} , and use that to draw \mathbf{u}_{T+2} , continuing this process for as long as we want. Once we have the error process (and the variances, if it's an "M" model), we can compute forecasts of the mean model and add to it the draws for the shocks. That will finish one repetition of the simulation process.

The difference between bootstrapping and random simulations comes in the method used to generate u_{T+k} given H_{T+k} —all the other calculations will be identical given the same underlying GARCH model.

When you do simulations outside the original sample, you may need to include some extra **SET** or **GSET** instructions before you start the simulation loop to extend the range of any series (of scalars or matrices) whose elements are being calculated with **COMPUTE**. For efficiency, **COMPUTE** does not try to extend series, so you need to make sure the space is there first. Because **SET** and **GSET** instructions are specifically designed to create series, and have target ranges specified up front, they *do* manage the range of the data.

Gaussian Simulations

If the model is univariate, h_{T+k} will be a scalar, and we can draw an appropriate u_{T+k} with `%RAN(SQRT(h))`. If the model is multivariate, the draw is done with `%RANMVNORMAL(%DECOMP(H))`. The `%DECOMP` is needed because `%RANMVNORMAL` takes as its argument a factor of the covariance matrix—see Appendix A.6.

Student t Simulations

The functions for drawing from a t are `%RANT(nu)` for the univariate process (Appendix A.2) and `%RANMVT(fsigma, nu)` for multivariate (Appendix A.7). Because we're trying to draw to a particular (co)variance, each of these needs a correction for degrees of freedom since the two functions are scaled based upon the (co)variance of the numerator Normal process. To give a univariate draw with variance h , we use

```
sqrt(h*((nu-2.)/nu))*%RANT(nu)
```

and for a multivariate draw with covariance matrix H , we use

```
%RANMVT(%DECOMP(H*((nu-2.)/nu)), nu).
```

Bootstrapping

Bootstrapping a GARCH model is a form of parametric bootstrap—we can't arbitrarily reorder the data, or even the residuals, because of the dynamic relationships among them. Instead, we use the original model to create standardized residuals which should be (if the model is correct) mean zero with unit variance or identity covariance. We draw random entries from those in place of the Normals or t 's in the random simulations, and scale them to create the desired (co)variance.

The process is quite simple for the univariate process and is covered in the *RATS User's Guide*. For a multivariate process, we first need to take the residuals and covariance matrices from the sample period to construct standardized residual vectors. The simplest organization for those is as a `SERIES[VECT]` (rather than `VECT[SERIES]` like `U`) since we will never use them except as a full `VECTOR` across variables:

```
dec series[vect] mvstdu
gset mvstdu gstart gend = inv(%decomp(hh(t)))*%xt(u,t)
```

This uses the simplest factor of the covariance matrix (the Cholesky decomposition) to do the standardizing. A **BOOT** instruction will be used to draw a random set of entry numbers from the range of the source. For instance,

```
boot entries baset+1 baset+nstep gstart gend
```

will make **ENTRIES(T)** from **BASET+1** to **BASET+NSTEP** as random draws of integers in the range from **GSTART** to **GEND**. If **HH(T)** is the desired covariance matrix, then we get a bootstrapped draw (for **U(T)**) by

```
compute %pt(u,t,%decomp(hh(t))*mvstdu(entries(t)))
```

This takes the full n -vector at the sampled entry and premultiplies by a factor of the desired covariance matrix.

9.2 Value-at-risk (VaR) Calculations-Univariate

One important use for GARCH models is to compute the *Value-at-Risk* (VaR) of an asset or portfolio. This is a measure of the risk of a major loss in a holding over a given period of time, typically the loss which is at the .01 quantile. However, there are only a relatively narrow set of GARCH models for which this can even be approximated analytically when the investment horizon is more than a single period.

If we focus first on univariate models, we are typically modeling the one-period (log) return (r_t) on an asset. The multi-period return from T (end-of-data) to $T + h$ is

$$r_T[h] \equiv \sum_{k=1}^h r_{T+k}$$

If we look at a very simple model, suppose

$$r_t = \alpha r_{t-1} + u_t$$

where u_t is governed by a GARCH process. Then, by successive substitution, we get:

$$r_{T+k} = \alpha^k r_T + \sum_{j=0}^{k-1} \alpha^j u_{T+k-j}$$

$$\sum_{k=1}^h r_{T+k} = \sum_{k=1}^h \alpha^k r_T + \sum_{k=1}^h \sum_{j=0}^{k-1} \alpha^j u_{T+k-j} = \sum_{k=1}^h \alpha^k r_T + \sum_{k=1}^h u_{T+k} \left(\sum_{j=0}^{k-1} \alpha^j \right) \quad (9.1)$$

If the u_{T+k} are (by assumption) uncorrelated with each other, then, if the model is one of the forms for which the forecasts of the variance can be computed analytically, then the variance of this can be computed as

$$\sum_{k=1}^h \text{var}(u_{T+k}|T) \left(\sum_{j=0}^{k-1} \alpha^j \right)^2$$

More generally, the weights will depend upon the coefficients of the IRF for the mean model. In applying this idea to the calculation of VaR, there are three problems:

1. It only works if the GARCH is a form for which analytical formulas for the variance forecasts are available.
2. Even if the u are conditionally Gaussian, the multi-period return is *not* because of the non-linear relationship between the shocks and subsequent variances. VaR is defined in terms of the quantiles of the distribution—knowing the mean and variance isn't enough to compute them.
3. It requires the parameters of both the mean model and GARCH model to be treated as fixed.

By contrast, simulating the returns, either by bootstrapping or by random numbers, is quite straightforward and many repetitions can be done in a short period of time. We will deal with item 3 in Chapter 10, so for now we will treat the parameters as known.

We'll demonstrate calculation of VaR using the analytical expression, random simulation and bootstrapping. Example 9.1 is taken from Tsay (2010). The data set is daily log returns on IBM stock from July 1962 to December 1998. Note that the return data are in percentages (returns times 100). That's more convenient for estimation, but we'll have to take that into account when computing the forecasts.

Tsay chooses an AR(2) model on this with the first lag dropped. AIC picks two lags, and the linear regression shows a quite insignificant lag one.

```
@arautolags ibmlog
linreg ibmlog
# constant ibmlog{1 2}
```

Given the size of the data set, even the presence of the lag two is questionable, but we'll go with the selection from the textbook. Because we need the equation for forecasting the mean, we'll define an `EQUATION` and use the `EQUATION` option on `GARCH`:

```
equation ar2 ibmlog
# constant ibmlog{2}
*
garch(p=1,q=1,equation=ar2,hseries=h,resids=res)
```

The desired end result is a 5%-ile VaR at the 15 day horizon after the end of the sample for a \$10,000,000 long position. The analytical calculation requires:

1. The point forecasts of the mean process.
2. The forecasts of the variance.
3. The impulse response weights for the model.

These are computed with

```
compute nstep=15
compute baset=%regend()
*
uforecast(steps=nstep,equation=ar2) fret
@garchfore(steps=nstep) h res
impulse(steps=nstep,model=ar2,shock=1.0,results=irf,noprint)
set psi = irf(1,1)(t)
```

On the **IMPULSE**, we use a unit shock since we need to weight those by variances that change from period to period. Because **IMPULSE** is mainly designed for multiple equation systems, the **RESULTS** option produces a (1×1) **RECT[SERIES]**—for readability, we'll copy that out to a single series named **PSI**. The calculation of the variance for the multi-period return (conditional on data through **BASET**) is done with:

```
compute totalvar=0.0
do k=1,nstep
    sstats 1 nstep+1-k psi(t)>>sumpsi
    compute totalvar=totalvar+sumpsi^2*h(baset+k)
end do k
```

u_{T+k} shows up in fewer and fewer terms in the sum (9.1) as k increases. The **SSTATS** instruction is the most convenient way to add up a certain number of elements of the **PSI** series—the results go into the variable **SUMPSI** whose square is used to weight the conditional variance for K steps ahead.

Because the data have (until now) been in terms of percentage returns, we need to adjust the scale to decimal returns. The mean and standard deviation of the 15-day return are:

```
compute stddev=sqrt(totalvar)*.01
sstats baset+1 baset+nstep fret(t)*.01>>mean
```

where we use **SSTATS** to sum the forecast returns over the forecast period (and scale by .01) to get the mean return.

Using a Gaussian approximation, the VaR is then

```
disp "5% VaR, 15 day horizon on 10000000" $
-10000000*(mean+%invnormal(.05)*stddev)
```

which comes out \$1026353.

When we do simulations, because we need percentiles of the distribution, we need to keep the full set of draws. Since the simulated calculations take very little time, we'll do a large number of them, which we'll put into the series RETSIM.

```
compute ndraws=100000
set retsim 1 ndraws = 0.0
```

To do the variance updates with simulated data, we need to get the GARCH parameters out of the output. Those can be extracted with:

```
dec vect means(%nregmean)
nonlin(parmset=uvgarch) means c0 a1 b1
compute %parmspoke(uvgarch,%beta)
```

using %NREGMEAN to adjust automatically for the number of parameters in the mean model.

The simulations are done with:

```
set u = res
do draw=1,ndraws
  set u baset+1 baset+nstep = h(t)=c0+a1*u{1}^2+b1*h{1},$
    %ran(sqrt(h))
  forecast(steps=nstep,from=baset+1,paths,model=ar2,results=sret)
  # u
  sstats baset+1 baset+nstep sret(1)*.01>>retsim(draw)
end do draw
```

The **SET U** outside the loop copies the residuals from the **GARCH** over into the series **U** which we then extend out-of-sample as part of the recursion.

The **SET U** instruction *inside* the loop simulates the error process. This first uses the GARCH recursion to calculate the variance (H) at t , then draws a mean zero, variance $H(t)$ Normal deviate. The **FORECAST** instruction then does a calculation of the mean process adding in the drawn shocks using the **PATHS** option. The result of that (which will be in the series **SRET(1)**) is a simulated set of daily returns. The (rescaled) sum of that over the forecast horizon gives the simulated 15-day return.

The quantiles of the simulation (in Table 9.1) can be computed using:

```
stats(fractiles) retsim 1 ndraws
```

These results are subject to Monte Carlo errors. The ones shown are generated with a **SEED** of 53531. If you need to reproduce a set of output, you can use **SEED** instruction to do that. See this chapter's Tips and Tricks (Section 9.5) for more on **SEED**. In order to convert this to the desired measure, we need to take

Table 9.1: Multi-Period Returns with Gaussian Simulations

Statistics on Series RETSIM			
Observations	100000		
Sample Mean	0.009571	Variance	0.004636
Standard Error	0.068088	SE of Sample Mean	0.000215
t-Statistic (Mean=0)	44.450828	Signif Level (Mean=0)	0.000000
Skewness	0.014942	Signif Level (Sk=0)	0.053736
Kurtosis (excess)	0.452172	Signif Level (Ku=0)	0.000000
Jarque-Bera	855.636871	Signif Level (JB=0)	0.000000
Minimum	-0.390789	Maximum	0.421172
01-%ile	-0.154201	99-%ile	0.174950
05-%ile	-0.101564	95-%ile	0.120977
10-%ile	-0.075663	90-%ile	0.095209
25-%ile	-0.034883	75-%ile	0.053572
Median	0.009564		

the variable `%FRACT05` defined by **STATISTICS**, change its sign and multiply by the size of the investment:

```
disp "5% VaR, 15 day horizon on 10000000" $
-10000000*%fract05
```

which will give us 1015636. Note that this isn't as extreme as the "analytical" value which used a Gaussian approximation. The true distribution of returns is more heavy-tailed than the Normal, but heavy-tailed (given a specific variance) also means that it has more values near the mean to keep the variance equalized—the point at which the quantiles for the Normal and the fat-tailed distribution are equal depends upon the specific distribution, and here that point is farther out than .05. If we redid this with a 1% VaR, the simulated value would be more extreme than the Gaussian approximation.

We next look at the same basic model, but with errors assumed to be Student-*t*:

```
garch(p=1,q=1,distrib=t,equation=ar2,hseries=h,resids=res)
```

The estimated degrees of freedom is quite small at 6.4 and the likelihood ratio statistic for the restriction of Gaussianity is the extremely significant 660 so clearly the Gaussian processes used above don't seem to match the data very well. The adjustment to the simulation procedure to deal with the *t* errors is trivial: in addition to the three regular GARCH parameters, we need to pull out the estimated degrees of freedom:

```
dec vect means(%nregmean)
nonlin(parmset=uvgarch_w_t) means c0 a1 b1 nu
compute %parmspoke(uvgarch_w_t,%beta)
```

and the instruction which does the actual simulations is adjusted to:

```
set u baset+1 baset+nstep = h(t)=c0+a1*u{1}^2+b1*h{1},$
      sqrt(h*(nu-2)/nu)*%rant(nu)
```

`%RANT(nu)` does a draw from a t with NU degrees of freedom. Because H is the desired *variance* of the Student t process, we have to correct for the fact that a standard t has variance $\nu/(\nu - 2)$.

The resulting VaR estimate is 1024802. The t produces slightly higher estimates, but not by much. Both the Normal and t models have to explain (as best they can) the same data set. The Normal model tends to produce somewhat higher values for the variance to reduce the cost in log likelihood due to outliers. So the higher variance for the Normal tends to cancel out the fatter tails in the t .

Bootstrapping requires a relatively simple adjustment to the simulation loop. We first have to standardize the residuals. Since it fits the data better, we'll use the estimates from the **GARCH** with the t errors. The source for the resampled data will be the standardized residuals over the estimation range:

```
compute gstart=%regstart(),gend=%regend()
set stdu gstart gend = res/sqrt(h)
```

Everything about the simulation loop is the same with two exceptions: inside the loop we need to use a **BOOT** instruction to choose a random set of entry numbers in the range `GSTART` to `GEND`.

```
boot entries baset+1 baset+nstep gstart gend
```

and the **SET** instruction which generates `U` now replaces the unit variance or t with the bootstrapped standardized residual:

```
set u baset+1 baset+nstep = h(t)=c0+a1*u{1}^2+b1*h{1},$
      sqrt(h)*stdu(entries(t))
```

The result is an estimate of 1004358.¹ This is lower than the other two estimates. The explanation for this can be seen by looking at the standardized residuals (Table 9.2):

```
stats(fractiles) stdu
```

The extreme values (particularly the losses) are more extreme but less numerous than would be predicted by a t with 6 degrees of freedom. When you bootstrap, only a few draws will get one of those major outliers, so the 5%-ile isn't as large. Farther out in the left tail, however, the bootstrapped values are higher. An alternative bootstrap procedure which handles the asymmetry

¹All of the simulation results were done using a **SEED** of 53531 immediately before each simulation loop.

Table 9.2: Statistics on Standardized Residuals

Statistics on Series STDU			
Observations	9188		
Sample Mean	0.005668	Variance	1.024496
Standard Error	1.012174	SE of Sample Mean	0.010560
t-Statistic (Mean=0)	0.536761	Signif Level (Mean=0)	0.591445
Skewness	-0.127933	Signif Level (Sk=0)	0.000001
Kurtosis (excess)	5.990224	Signif Level (Ku=0)	0.000000
Jarque-Bera	13762.189870	Signif Level (JB=0)	0.000000
Minimum	-13.432080	Maximum	6.563174
01-%ile	-2.400456	99-%ile	2.601887
05-%ile	-1.539163	95-%ile	1.630596
10-%ile	-1.158434	90-%ile	1.212758
25-%ile	-0.608514	75-%ile	0.588103
Median	-0.017993		

better is a form of “wild bootstrap” which takes the standardized residual and then “flips a coin” to pick a sign. The only change to the previous bootstrap is to replace the **SET** instruction with:

```
set u baset+1 baset+nstep = h(t)=c0+a1*u{1}^2+b1*h{1}, $
      sqrt(h)*stdu(entries(t))*%if(%ranflip(.5),+1,-1)
```

With the same seed, the wild bootstrap gives a VaR of 1031491, almost 3% higher than the standard bootstrap.

9.3 Value-at-risk (VaR) Calculations-Multivariate

The VaR for a univariate process is relatively simple: you’re typically looking at either a long position, which requires looking at the left tail of possible outcomes, or a short position, which requires looking at the right tail. With multiple assets, we have to allow for different portfolio weights. There are two ways to approach this: if you know the set of weights that you want, you can just compute the set of simulated returns to that portfolio. If you don’t, then (as long as the weights are fixed throughout the investment period), you can save the simulated sums of the individual return series and reweight those as needed outside the draw loop. We’ll use the first approach here.

Example 9.2 uses the stock price series from Bauwens & Laurent (2005). However, we will use a DCC with the standard t density rather than the skew- t introduced in the paper. The data set has (percentage daily) returns over the period January 1990 to May 2002 on three stocks: Alcoa (AA), Caterpillar (CAT) and Disney (DIS).

```
open data dj14_jbes.xls
data(format=xls,left=2,org=columns) 1 3112 r_aa r_cat r_dis
```


The portfolio weights that we will use are 1.4, -.2 and -.2 which is one of the three portfolios used in the paper. The paper does a rather elaborate calculation of one-period-ahead VaR with periodic re-estimation of the model to compare its performance against similar models which don't use the skew- t density. Here, we'll just do the same 15 step out-of-sample calculation of the VaR as was done in Example 9.1.

As in the paper, the mean models are univariate AR's with individually-determined lag lengths. Here that has 1 for AA, 0 for both CAT and DIS.

```
linreg(define=aaeq) r_aa / u_aa
# constant r_aa{1}
linreg(define=cateq) r_cat / u_cat
# constant
linreg(define=diseq) r_dis / u_dis
# constant
vcv(matrix=qbar)
# u_aa u_cat u_dis
```

This estimates a DCC model with asymmetric standard univariate GARCH models for the variances and with t -distributed errors. The jointly standardized residuals will be needed for bootstrapping. This saves the number of variables, and the estimation range.

```
group armodel aaeq cateq diseq
garch(model=armodel,mv=dcc,distrib=t,asymm,$
      hmatrices=hh,rvectors=uv,stdresids=mvstdu)
compute n=%nvar,gstart=%regstart(),gend=%regend()
```

These pull out the estimated DCC coefficients and the degrees of freedom. These will always be the final parameters. If you don't use t errors, the DCC parameters will be in positions %NREG-1 and %NREG.

```
compute dcca=%beta(%nreg-2)
compute dccb=%beta(%nreg-1)
compute nu=%beta(%nreg)
```

and this pulls out and organizes the coefficients for the individual GARCH processes. The form of the matrices it creates will be different depending upon the choice for the VARIANCES option, which here is the default of SIMPLE.

```
@MVGARCHVarMats(variances=simple,asymmetric)
```

This creates the fixed set of portfolio weights—long on AA, short on the other two.

```
dec vector weights(n)
compute weights=||1.4,-.2,-.2||
```

There are quite a few pieces that are needed to do the out-of-sample simulations. First, we need a `SERIES[VECT]` for the variances, which are generated from the univariate GARCH models. We need `SERIES[SYMM]` for the full covariance matrices, for the outer product of the residuals and the (signed) outer products of the residuals and for the **Q** matrices in the DCC recursion. The last of those has to be regenerated since it's not directly observable from the other information. These create or copy those over the sample range:

```
dec series[vect] hhd
dec series[symm] hh uu uus qq
*
gset hhd gstart gend = %xdiag(hh(t))
gset uu gstart gend = %outerxx(uv(t))
gset uus gstart gend = %outerxx(%minus(uv(t)))
gset qq gstart gend = qbar
gset qq gstart+1 gend = (1-dcca-dccb)*qbar+dcca*uu{1}+dccb*qq{1}
```

and these extend their range across the forecast interval:

```
gset hh baset+1 baset+nstep = %zeros(n,n)
gset hhd baset+1 baset+nstep = %zeros(n,1)
gset uu baset+1 baset+nstep = %zeros(n,n)
gset uus baset+1 baset+nstep = %zeros(n,n)
gset qq baset+1 baset+nstep = %zeros(n,n)
```

Finally, this will be used for the simulated out-of-sample shocks:

```
dec vect[series] u(n)
clear(zeros) u
```

The pseudo-code for computing the VaR is

```
do draw=1,ndraws
  do t=baset+1,baset+nstep (which is the out-of-sample range)
    <<compute the variances at entry t>>
    <<update Q at entry t>>
    <<combine those two to create the full covariance matrix>>
    <<draw u for entry t (either randomly or by bootstrap)>>
    <<do any bookkeeping required at t to compute t+1>>
  end do t
  <<use the mean model plus the u shocks to create
    out-of-sample data>>
  <<compute and save the return from the desired portfolio>>
end do draw
```

The individual variances are computing one-at-a-time in a loop. The `%%VAR_C`, `%%VAR_A`, `%%VAR_B`, and `%%VAR_D` were created by the earlier `@MVGARCHVarMats` procedure. A, B and D are all $n \times n$ matrices to allow for

more complicated types of variance models (such as `SPILLOVER` and `VARMA`)—with `VARIANCES=SIMPLE`, they'll be diagonal matrices, and we will only need to use the diagonals. We also need only the diagonal elements of the `UU` and `UUS` matrices.

```
do i=1,n
  compute hhd(t) (i)=%%var_c(i)+%%var_a(i,i)*uu(t-1) (i,i)+$
    %%var_d(i,i)*uus(t-1) (i,i)+%%var_b(i,i)*hhd(t-1) (i)
end do i
```

This updates `Q` and uses it, and `HHD`, to generate the full $n \times n$ covariance matrix:

```
compute qq(t)=(1-dcca-dccb)*qbar+dcca*uu(t-1)+dccb*qq(t-1)
compute hh(t)=%corrtocv(%cvtocorr(qq(t)),hhd(t))
```

The random multivariate t simulations, we use the following, which corrects the input covariance matrix so the overall variance of the t is the desired `HH(T)`.

```
compute %pt(u,t,%ranmvt(%decomp(((nu-2)/nu)*hh(t)),nu))
```

Finally, this creates the outer product and signed outer product of the residuals, for use in computing the next entry:

```
compute uu(t)=%outerxx(%xt(u,t))
compute uus(t)=%outerxx(%minus(%xt(u,t)))
```

That completes the generation of the GARCH error process through the forecast period. This uses the estimated mean model and adds in the generated errors (`PATHS` option, plus the `# U` supplementary line) to create simulated data (into the `VECT[SERIES] RFORE`)

```
forecast(model=armodel,from=baset+1,to=baset+nstep,$
  paths,results=rfore)
# u
```

Finally, this computes the return on a \$1 investment in the desired portfolio. (The `.01` is needed because the returns are in percentages):

```
sstats baset+1 baset+nstep $
  %dot(weights,%xt(rfore,t))*0.01>>retsim(draw)
```

For a 1% VaR, we can use `STATISTICS` and pick up minus the 1st percentile:

```
stats(fractiles) retsim 1 ndraws
disp "15-period 1% VaR on $1 with t simulations" -%fract01
```

The code for the bootstrap is almost identical. At the start of each draw, we need to do the shuffled set of entries from the estimation range:

```
boot entries baset+1 baset+nstep gstart gend
```

and, in place of the draw for U from a multivariate t , this takes the standardized residuals and premultiplies by the current factor of the covariance matrix. However, because the Cholesky factor standardizes on positive values down the diagonal, this randomly selects signs of the columns so all possible lower-triangular factors are equally likely. This is another form of “wild bootstrap”. (The `FLIPS VECTOR` is dimensioned before the loop).

```
ewise flips(i)=%if(%ranflip(.5),+1,-1)
compute %pt(u,t,%dmult(%decomp(hh(t)),flips)*%xt(mvstdu,entries(t)))
```

The VaR values with the same seeds we’ve been using are .22816 on a \$1 investment using random draws and 0.22007 using bootstrapping.

9.4 Variance Impulse Responses by Simulation

The method for calculating Variance Impulse Response Functions (VIRF) from Section 6.2 only works for models which can be put into VECM form, and doesn’t extend to asymmetry. A similar type of closed-form calculation can be used for some types of variance models used with CC and DCC models (the `VARIANCES=SIMPLE`, `SPILLOVER` and `VARMA` options) to get responses of the variances only. For any other type of multivariate GARCH model, the only way to get multiple step variance responses is to do some form of simulation.

The calculation is similar to non-linear impulse response functions for the mean.² In general, the model specifies a stochastic process for the data and covariance matrices over the period T_0 to $T_0 + H$ given the data through $T_0 - 1$. The average of the generated covariance matrices across many replications of this will give us their forecasts. Then, at T_0 , we replace the random shock with something of our choosing and redo the simulations over $T_0 + 1$ to $T_0 + H$, whose average will give us forecasts conditional on that initial shock. The difference between the two will be (an estimate of) the VIRF.

If the model is such that the original Hafner-Herwartz calculation could be done, this will (theoretically) give the same results. However, the VIRF calculated by simulations takes a *very* large number of replications to converge, many more than a similar non-linear response for the mean. This is because the variance process itself is random—not just random, but quite heavy-tailed, so even controlling for as much simulation error as possible, there is still considerable randomness in the averages.

²Sometimes known, imprecisely, as generalized impulse responses.

Table 9.3: BEKK Estimates with Asymmetry

MV-GARCH, BEKK - Estimation by BFGS					
Convergence in 58 Iterations. Final criterion was 0.0000067 <= 0.0000100					
Usable Observations		3718			
Log Likelihood		-5253.4448			
	Variable	Coeff	Std Error	T-Stat	Signif
Mean Model(DEMRET)					
1.	Constant	0.0095	0.0073	1.3006	0.1934
2.	DEMRET{1}	0.0037	0.0121	0.3074	0.7585
Mean Model(GBPRET)					
3.	Constant	-0.0021	0.0070	-0.2966	0.7668
4.	GBPRET{1}	0.0178	0.0120	1.4841	0.1378
5.	C(1,1)	0.0911	0.0109	8.3581	0.0000
6.	C(2,1)	0.0634	0.0128	4.9393	0.0000
7.	C(2,2)	-0.0382	0.0081	-4.6936	0.0000
8.	A(1,1)	0.2902	0.0216	13.4119	0.0000
9.	A(1,2)	0.0078	0.0252	0.3112	0.7557
10.	A(2,1)	-0.0493	0.0227	-2.1703	0.0300
11.	A(2,2)	0.2317	0.0244	9.4774	0.0000
12.	B(1,1)	0.9559	0.0066	144.6569	0.0000
13.	B(1,2)	0.0023	0.0071	0.3289	0.7423
14.	B(2,1)	0.0088	0.0068	1.2901	0.1970
15.	B(2,2)	0.9651	0.0069	139.0208	0.0000
16.	D(1,1)	0.0268	0.0405	0.6604	0.5090
17.	D(1,2)	0.1447	0.0305	4.7381	0.0000
18.	D(2,1)	0.0705	0.0366	1.9278	0.0539
19.	D(2,2)	-0.0635	0.0414	-1.5355	0.1247
20.	Shape	4.3608	0.2306	18.9146	0.0000

Example 9.3 will use the same basic model as in Example 6.2, but will add asymmetry (Table 9.3):

```
garch(model=uniar1,mv=bekk,asymmetric,rvectors=rd,hmatrices=hh,$
      distrib=t,pmethod=simplex,piters=20,itors=500)
```

The asymmetry coefficients are jointly significant at conventional levels, though the improvement of the log likelihood from -5259 to -5253 on 3718 observations is well short of the threshold for taking the larger model under BIC.

We need to pull out the degrees of freedom (for use in simulations) and (as with the other example), we need to pull out the matrices for the VEC representation. This will now include the matrix %VECH_D, which takes care of the asymmetry term.

```
compute nu=%beta(%nreg)
compute n=%nvar
*
@MVGARCHtoVECH(mv=bekk,asymmetric)
```

We'll do just the Black Wednesday shock. To handle the simulations, we need to keep track of the VECH forms for the covariance matrices, for the outer products of the residuals and for the outer product of the signed residuals and we need those for both the “baseline” (fully simulated) and the shocked simulations—we will be doing those simultaneously to control the amount of randomness in the estimates by using common random numbers. The 0's below are for the baseline and the 1's are for the shocked. These are all initialized with the appropriate sample values:

```
dec series[vect] hhvech0 uuvech0 uusvech0
*
dec series[vect] hhvech1 uuvech1 uusvech1
*
gset hhvech0 = %vec(hh(t))
gset hhvech1 = %vec(hh(t))
gset uuvech0 = %vec(%outerxx(rd(t)))
gset uuvech1 = %vec(%outerxx(rd(t)))
gset uusvech0 = %vec(%minus(%outerxx(rd(t))))
gset uusvech1 = %vec(%minus(%outerxx(rd(t))))
```

The pseudo-code for the setup is

```
initialize the target for the calculated differences
do draw=1,ndraws
  do t=baset,baset+nstep
    if t==baset, choose random numbers for the baseline given
      the covariance matrix, and the desired initial shocks
      for the shocked history
    otherwise update the variances, choose one set of
      random numbers and use it to create shocks for both
      simulations
    update the UUECH and UUSVECH vectors
    save the difference between the two covariance matrices
  end do t
end do draw
```

The bookkeeping for the VIRF'S will use a `SERIES [VECT]` which will keep track of the sum (and later the mean) of the gaps between the VECH's of the shocked and baseline covariance matrices. This is initialized to zeros.

```
dec series[vect] virfvec
gset virfvec = %zeros(%size(sigma0),1)
```

It's possible (perhaps even probable) that we would be better off saving all the values of this and doing some type of more robust calculation of the mean (trimmed mean for instance—discarding a percentage of extreme values at both ends). That would require more elaborate bookkeeping, but it's certainly quite doable.

Inside the loop, for the case where $T == \text{BASET}$, the baseline case has

```
compute uv0(t)=%ranmvt(%decomp((nu-2)/nu*hh(t)),nu)
```

while the shock case (here) is

```
compute uv1(t)=rd(t)
```

which is the observed residual at the time period of interest. A couple of things to note. First, in this calculation, the covariance matrix used in the baseline matters, while it washes out of the calculation in Hafner-Herwartz. Second, in HH all that matters for computing the VIRF is the outer product of the shock, while here we need to fully specify the shock because the sign matters due to asymmetry.

For the later time periods, we have two effectively parallel calculations:

```
compute hhvech0(t)=%vech_c+%vech_b*hhvech0(t-1)+$
    %%vech_a*uuvech0(t-1)+%%vech_d*uusvech0(t-1)
compute hhvech1(t)=%vech_c+%vech_b*hhvech1(t-1)+$
    %%vech_a*uuvech1(t-1)+%%vech_d*uusvech1(t-1)
compute udraw=%ranmvt(sqrt((nu-2)/nu)*%identity(n),nu)
compute uv0(t)=%decomp(%vectosymm(hhvech0(t),n))*udraw
compute uv1(t)=%decomp(%vectosymm(hhvech1(t),n))*udraw
```

Again, in comparison with the HH calculation, note that we need the variance constant `%%VECH_C` here but it washed out of HH—differentials in variance aren't enough now because we need the two full covariance matrices to do the simulations.

UDRAW is a single multivariate- t draw standardized to a identity matrix as the covariance matrix of the distribution as a whole. UV0 and UV1 are then generated from that to give the correct covariance matrix at this stage for each simulation. This is where the big difference in simulation error comes in compared with non-linear IRF'S in the mean. For a model with a *fixed* covariance matrix (as would be typical in any model with a non-linearity only in the mean), UV0 and UV1 would be identical for every step beyond the initial one, which greatly reduces how different the shock vs baseline versions can be. Here, we have different covariance matrices for each path, so different shocks, so still further different covariance matrices the next period.

This updates the outer products of the shocks:

```
compute uuvech0(t)=%vec(%outerxx(uv0(t)))
compute uusvech0(t)=%vec(%outerxx(%minus(uv0(t))))
compute uuvech1(t)=%vec(%outerxx(uv1(t)))
compute uusvech1(t)=%vec(%outerxx(%minus(uv1(t))))
```

Finally, this adds in the difference between the shocked and baseline paths for (this draw). (This is only done for positive steps).

```

if t>baset
    compute virfvec(t)=virfvec(t)+(hhvech1(t)-hhvech0(t))

```

That finished the simulation loop. The following converts the sums to averages, and shifts the values into separate `SERIES` and shifts them to entry 1.

```

gset virfvec baset+1 baset+nstep = virfvec(t)*(1.0/ndraws)
dec vect[series] virf(%size(sigma0))
*
do i=1,%size(sigma0)
    set virf(i) 1 nstep = virfvec(baset+t)(i)
end do i

```

Finally, as in Example 6.2, we graph the responses of the two variances and the covariance between the series (Figure 9.1):

```

spgraph(vfields=3,hfields=1,footer="VIRF's in Asymmetric BEKK",$
    ylabel=||"DEM/USD Variance","Covariance","GBP/USD Variance"||)
do i=1,%size(sigma0)
    graph(number=1)
    # virf(i)
end do i
spgraph(done)

```

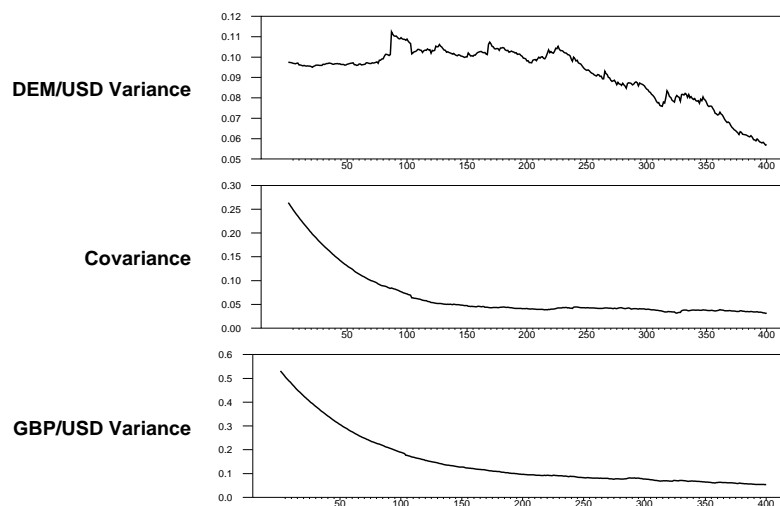


Figure 9.1: Volatility Responses for Asymmetric BEKK

Despite `NDRAWS=100000`, the response of the exchange rate for the DM seems very imprecise. However, it is also rather clearly quite different from the corresponding response in Figure 6.2, while the responses of the covariance and GBP are quite similar, so it would seem that, even though the asymmetry didn't come in that strongly (the coefficient that has the greatest effect on the difference in the behavior of the DM is $D(2, 1)$, at least at the point estimates, there is a decided difference in the long-run dynamics of the variance.

9.5 RATS Tips and Tricks

Random Numbers and SEED

SEED allows you to control the set of random numbers generated by a program. In practice, you would use it mainly in two situations:

1. You're trying to debug a program and need to control the random numbers in order to have results that are comparable from test to test. Note, however, if you make *any* change in the way that the numbers are drawn, the sequence of numbers will change.
2. You've written a "final" version of a paper and want to be able to reproduce the results exactly in case you need to regenerate graphs or tables.

Note that although it seems as if you should want to use a "random-looking" seed, it really doesn't matter: RATS scrambles the value you input quite a bit in generating the internal seed.

Example 9.1 VaR Calculations for a Univariate Model

This is adapted from Tsay (2010), example 7.3. It computes the Value at Risk (VaR) for a series based upon a univariate GARCH model, using analytical calculations, random simulation and bootstrapping. It is covered in detail in Section 9.2.

```
open data d-ibmln98.dat
data(format=free,org=columns) 1 9190 ibmlog
*
@arautolags(crit=bic,table) ibmlog
linreg ibmlog
# constant ibmlog{1 2}
*
* Define the equation for the mean
*
equation ar2 ibmlog
# constant ibmlog{2}
*
* GARCH with normal residuals
*
garch(p=1,q=1,equation=ar2,hseries=h,resids=res)
*
* Forecast the variance using @GARCHFORE and the series itself using
* UFORECAST and compute the impulse responses uses IMPULSE.
*
compute nstep=15
compute baset=%regend()
*
uforecast(steps=nstep,equation=ar2) fret
@garchfore(steps=nstep) h res
impulse(steps=nstep,model=ar2,shock=1.0,results=irf,noprint)
set psi = irf(1,1)(t)
*
compute totalvar=0.0
do k=1,nstep
    sstats 1 nstep+1-k psi(t)>>sumpsi
    compute totalvar=totalvar+sumpsi^2*h(baset+k)
end do k
*
compute stddev=sqrt(totalvar)*.01
sstats baset+1 baset+nstep fret(t)*.01>>mean
*
disp "5% VaR, 15 day horizon on 10000000" $
    -10000000*(mean+%invnormal(.05)*stddev)
*
* VaR computed with Gaussian simulations
*
compute ndraws=100000
set retsim 1 ndraws = 0.0
*
dec vect means(%nregmean)
nonlin(parmset=uvgarch) means c0 a1 b1
```

```

compute %parmspoke (uvgarch, %beta)
*
*seed 53531
set u = res
do draw=1, ndraws
    set u baset+1 baset+nstep = h(t)=c0+a1*u{1}^2+b1*h{1}, $
        %ran(sqrt(h))
    forecast(steps=nstep, from=baset+1, paths, model=ar2, results=sret)
    # u
    sstats baset+1 baset+nstep sret(1)*.01>>retsim(draw)
end do draw
stats(fractiles) retsim 1 ndraws
*
disp "5% VaR, 15 day horizon on 10000000" $
    -10000000*%fract05
*
* GARCH with Student-t residuals
*
garch(p=1, q=1, distrib=t, equation=ar2, hseries=h, resids=res)
dec vect means(%nregmean)
nonlin(parmset=uvgarch_w_t) means c0 a1 b1 nu
compute %parmspoke (uvgarch_w_t, %beta)
*
*seed 53531
set u = res
do draw=1, ndraws
    set u baset+1 baset+nstep = h(t)=c0+a1*u{1}^2+b1*h{1}, $
        sqrt(h*(nu-2)/nu)*%rant(nu)
    forecast(steps=nstep, from=baset+1, paths, model=ar2, results=sret)
    # u
    sstats baset+1 baset+nstep sret(1)*.01>>retsim(draw)
end do draw
stats(fractiles) retsim 1 ndraws
disp "5% VaR, 15 day horizon on 10000000" $
    -10000000*%fract05
*
* Bootstrapping
*
*seed 53531
compute gstart=%regstart(), gend=%regend()
set stdu gstart gend = res/sqrt(h)
*
set u = res
do draw=1, ndraws
    boot entries baset+1 baset+nstep gstart gend
    set u baset+1 baset+nstep = h(t)=c0+a1*u{1}^2+b1*h{1}, $
        sqrt(h)*stdu(entries(t))
    forecast(steps=nstep, from=baset+1, paths, model=ar2, results=sret)
    # u
    sstats baset+1 baset+nstep sret(1)*.01>>retsim(draw)
end do draw
stats(fractiles) retsim 1 ndraws
disp "5% VaR, 15 day horizon on 10000000" $
    -10000000*%fract05

```

```

stats(fractiles) stdu
*
* Wild bootstrap, symmetrizing the draws
*
*seed 53531
compute gstart=%regstart(),gend=%regend()
set stdu gstart gend = res/sqrt(h)
*
set u = res
do draw=1,ndraws
    boot entries baset+1 baset+nstep gstart gend
    set u baset+1 baset+nstep = h(t)=c0+a1*u{1}^2+b1*h{1},$
        sqrt(h)*stdu(entries(t))*%if(%ranflip(.5),+1,-1)
    forecast(steps=nstep,from=baset+1,paths,model=ar2,results=sret)
    # u
    sstats baset+1 baset+nstep sret(1)*.01>>retsim(draw)
end do draw
stats(fractiles) retsim 1 ndraws
disp "5% VaR, 15 day horizon on 10000000" $
    -10000000*%fract05

```

Example 9.2 VaR Calculations for a Multivariate Model

This demonstrates calculation of VaR for a multivariate model. It's adapted from the stock returns example in Bauwens & Laurent (2005), and demonstrates calculations using simulations with a multivariate-*t* and bootstrapping. This is covered in Section 9.3.

```

open data dj14_jbes.xls
data(format=xls,left=2,org=columns) 1 3112 r_aa r_cat r_dis
*
* Equations for the three variables. (AR(1) for AA, constant only
* for CAT and DIS).
*
linreg(define=aaeq) r_aa / u_aa
# constant r_aa{1}
linreg(define=cateq) r_cat / u_cat
# constant
linreg(define=diseq) r_dis / u_dis
# constant
vcv(matrix=qbar)
# u_aa u_cat u_dis
*
group armodel aaeq cateq diseq
garch(model=armodel,mv=dcc,distrib=t,asymm,$
    hmatrices=hh,rvectors=uv,stdresids=mvstdu)
compute n=%nvar,gstart=%regstart(),gend=%regend()
*
* Pull out the DCC and t shape parameters. (These will always be
* the last three.)
*

```

```

compute dcca=%beta(%nreg-2)
compute dccb=%beta(%nreg-1)
compute nu=%beta(%nreg)
*
@MVGARCHVarMats(variances=simple,asymmetric)
*
* VaR calculations
*
* Portfolio weights (+ means long, - means short).
*
dec vector weights(n)
compute weights=||1.4,-.2,-.2||
*
compute nstep=15
compute baset=gend
*
* Random simulations with multivariate t
*
compute ndraws=100000
set retsim 1 ndraws = 0.0
*
* For the variances only
*
dec series[vect] hhd
*
* For the full covariance matrix, the outer product of residuals
* and signed outer product of residuals, the Q recursion
*
dec series[symm] hh uu uus qq
*
gset hhd gstart gend = %xdiag(hh(t))
gset uu gstart gend = %outerxx(uv(t))
gset uus gstart gend = %outerxx(%minus(uv(t)))
gset qq gstart gend = qbar
gset qq gstart+1 gend = (1-dcca-dccb)*qbar+dcca*uu{1}+dccb*qq{1}
*
* Extend these out of sample
*
gset hh baset+1 baset+nstep = %zeros(n,n)
gset hhd baset+1 baset+nstep = %zeros(n,1)
gset uu baset+1 baset+nstep = %zeros(n,n)
gset uus baset+1 baset+nstep = %zeros(n,n)
gset qq baset+1 baset+nstep = %zeros(n,n)
*
dec vect[series] u(n)
clear(zeros) u
*
*seed 53531
do draw=1,ndraws
  do t=baset+1,baset+nstep
    *
    * Compute the variances for entry t
    *
    do i=1,n

```

```

        compute hhd(t) (i)=%var_c(i)+%var_a(i,i)*uu(t-1) (i,i)+$
            %var_d(i,i)*uus(t-1) (i,i)+%var_b(i,i)*hhd(t-1) (i)
    end do i
    *
    * Update Q
    *
    compute qq(t)=(1-dcca-dccb)*qbar+dcca*uu(t-1)+dccb*qq(t-1)
    compute hh(t)=%corrto cv(%cvtocorr(qq(t)),hhd(t))
    *
    * Draw random t's corrected to make HH(T) the covariance matrix
    * of the distribution.
    *
    compute %pt(u,t,%ranmvt(%decomp(( (nu-2)/nu)*hh(t)),nu))
    compute uu(t)=%outerxx(%xt(u,t))
    compute uus(t)=%outerxx(%minus(%xt(u,t)))
end do t
*
* Forecast the data itself conditioned on the simulated paths of
* the shocks.
*
forecast(model=armodel,from=baset+1,to=baset+nstep,$
    paths,results=rfore)
# u
*
* Compute the return on the simulated portfolio
*
sstats baset+1 baset+nstep $
    %dot(weights,%xt(rfore,t))*0.01>>retsim(draw)
end do draw
stats(fractiles) retsim 1 ndraws
disp "15-period 1% VaR on $1 with t simulations" -%fract01
*
* Bootstrapping
*
dec vect flips(n)
*
*seed 53531
do draw=1,ndraws
    boot entries baset+1 baset+nstep gstart gend
    do t=baset+1,baset+nstep
        *
        * Compute the variances for entry t
        *
        do i=1,n
            compute hhd(t) (i)=%var_c(i)+%var_a(i,i)*uu(t-1) (i,i)+$
                %var_d(i,i)*uus(t-1) (i,i)+%var_b(i,i)*hhd(t-1) (i)
        end do i
        *
        * Update Q
        *
        compute qq(t)=(1-dcca-dccb)*qbar+dcca*uu(t-1)+dccb*qq(t-1)
        compute hh(t)=%corrto cv(%cvtocorr(qq(t)),hhd(t))
        *
        * Pick random signs on all columns of the factor
    end do t
end do draw

```

```

*
*   ewise flips(i)=%if(%ranflip(.5),+1,-1)
*   compute %pt(u,t,$
*       %dmult(%decomp(hh(t)),flips)*%xt(mvstdu,entries(t)))
*   compute uu(t)=%outerxx(%xt(u,t))
*   ewise uus(t)(i,j)=uu(t)(i,j)*(u(i)(t)<0)*(u(j)(t)<0)
end do t
*
* Forecast the data itself conditioned on the simulated paths of
* the shocks.
*
forecast(model=armodel,from=baset+1,to=baset+nstep,paths,results=rfore)
# u
*
* Compute the return on the simulated portfolio
*
sstats baset+1 baset+nstep $
    %dot(weights,%xt(rfore,t))*0.01>>retsim(draw)
end do draw
stats(fractiles) retsim 1 ndraws
disp "15-period 1% VaR on $1 with bootstrap" -%fract01

```

Example 9.3 VIRF Calculations for with Asymmetry

This demonstrates calculation of a Volatility Impulse Response Function (VIRF) for an Asymmetric BEKK using simulation methods. This is covered in Section 9.4.

```

open data hhdata.xls
data(format=xls,org=columns) 1 3720 usxjpn usxfra usxsui $
    usxnld usxuk usxbel usxger usxswe usxcan usxita date
*
* This is rescaled to 100.0 compared with data used in the paper.
* The paper also uses local currency/USD, while the data set has
* USD/local currency, so we change the sign on the return.
*
set demret = -100.0*log(usxger/usxger{1})
set gbpret = -100.0*log(usxuk/usxuk{1})
*
* The mean model is a univariate AR on each variable separately.
*
equation demeqn demret
# constant demret{1}
equation gbpeqn gbpret
# constant gbpret{1}
group uniar1 demeqn gbpeqn
*
garch(model=uniar1,mv=bekk,asymmetric,rvectors=rd,hmatrices=hh,$
    distrib=t,pmethod=simplex,piters=20,itors=500)
*
compute nu=%beta(%nreg)

```

```

compute n=%nvar
*
* Transform the BEKK model to its equivalent VEC representation
*
@MVGARCHtoVECH(mv=bekk, asymmetric)
*
* VIRF with historical incidents
*
sstats(max) / %if(date==920916,t,0)>>xblackwed $
               %if(date==930802,t,0)>>xecpolicy
compute blackwed=fix(xblackwed), ecpolicy=fix(xecpolicy)
*
compute nstep=400
*
compute baset=blackwed
compute eps0=rd(blackwed)
compute sigma0=hh(blackwed)
compute shock=%vec(%outerxx(eps0)-sigma0)
*
compute ndraws=100000
*
* These are all in "VECH" form
*
* For the baseline calculation
*
dec series[vect] hhvech0 uuvech0 uusvech0
*
* For the shocked calculation
*
dec series[vect] hhvech1 uuvech1 uusvech1
*
gset hhvech0 = %vec(hh(t))
gset hhvech1 = %vec(hh(t))
gset uuvech0 = %vec(%outerxx(rd(t)))
gset uuvech1 = %vec(%outerxx(rd(t)))
gset uusvech0 = %vec(%minus(%outerxx(rd(t))))
gset uusvech1 = %vec(%minus(%outerxx(rd(t))))
*
dec series[vect] uv0 uv1
gset uv0 = %zeros(n,1)
gset uv1 = %zeros(n,1)
*
dec series[vect] virfvec
gset virfvec = %zeros(%size(sigma0),1)
*
infobox(action=define, progress, lower=1, upper=ndraws) "Calculation of VIRF"
*seed 53531
do draw=1, ndraws
  do t=baset, baset+nstep
    if t==baset {
      *
      * Baseline---draw a value using the current covariance matrix
      *
      compute uv0(t)=%ranmvt(%decomp((nu-2)/nu*hh(t)), nu)
    }
  }

```



```

*
* Shock---use the observed values
*
compute uv1(t)=rd(t)
}
else {
compute hhvech0(t)=%%vech_c+%%vech_b*hhvech0(t-1)+$
%%vech_a*uuvech0(t-1)+%%vech_d*uusvech0(t-1)
compute hhvech1(t)=%%vech_c+%%vech_b*hhvech1(t-1)+$
%%vech_a*uuvech1(t-1)+%%vech_d*uusvech1(t-1)
compute udraw=%ranmvt(sqrt((nu-2)/nu)*%identity(n),nu)
compute uv0(t)=%decomp(%vectosymm(hhvech0(t),n))*udraw
compute uv1(t)=%decomp(%vectosymm(hhvech1(t),n))*udraw
}
compute uuvech0(t)=%vec(%outerxx(uv0(t)))
compute uusvech0(t)=%vec(%outerxx(%minus(uv0(t))))
compute uuvech1(t)=%vec(%outerxx(uv1(t)))
compute uusvech1(t)=%vec(%outerxx(%minus(uv1(t))))
if t>baset
compute virfvec(t)=virfvec(t)+(hhvech1(t)-hhvech0(t))
end do t
infobox(current=draw)
end do draw
infobox(action=remove)
*
gset virfvec baset+1 baset+nstep = virfvec(t)*(1.0/ndraws)
dec vect[series] virf(%size(sigma0))
*
do i=1,%size(sigma0)
set virf(i) 1 nstep = virfvec(baset+t)(i)
end do i
*
spgraph(vfields=3,hfields=1,footer="VIRF's in Asymmetric BEKK",$
ylabels=||"DEM/USD Variance","Covariance","GBP/USD Variance"||)
do i=1,%size(sigma0)
graph(number=1)
# virf(i)
end do i
spgraph(done)

```

Simulation Methods for Model Inference

The methods in Chapter 9 assumed that the parameters of the GARCH process were fixed. Allowing for them to be unknown complicates matters quite a bit. The standard estimation process gives us an approximate distribution for parameters, but it's only an approximation and the covariance matrix is often not that accurately estimated—different methods like BHHH and BFGS can give dramatically different values.

For instance, in Elder & Serletis (2010) (which we'll look at in Section 11.1), the authors describe the following procedure (for computing impulse responses in a VAR-GARCH model):

The Monte Carlo method used to construct the confidence bands is described in Hamilton (1994, p. 337), adopted to our model. That is, the impulse responses are simulated from the maximum likelihood estimates (MLEs) of the model's parameters. Confidence intervals are generated by simulating 1,000 impulses responses, based on parameter values drawn randomly from the sampling distribution of the MLEs, where the covariance matrix of the MLEs is derived from an estimate of Fisher's information matrix.

While not an unreasonable procedure, this really isn't correct technically. For a standard VAR (with residuals assumed Normal and a non-informative prior) the lag coefficients given Σ have an exact Normal distribution and what this describes would be correct. However, for a GARCH model, the asymptotic distribution is, at best, an approximation. Different software, different estimation options, even different guess values will give a different estimate for the approximating covariance matrix even if the point estimates match, and thus (if the procedure above is followed as described) Monte Carlo results which differ by more than simple simulation error. The Monte Carlo procedures described in this chapter make use of the approximating distribution, but then correct for the difference between the true (log) likelihood and the quadratic approximation.

Early work on simulation methods for inference on ARCH and GARCH processes used importance sampling: see, for instance, Geweke (1989). The `GARCHIMPORT.RPF` example demonstrates that. When importance sampling

works, it's generally superior to the Markov Chain Monte Carlo (MCMC) methods since it doesn't rely upon Markov Chain convergence. However, there are a limited number of situations where that is likely to be a successful strategy—once the parameter set gets larger than a small univariate model, it's hard to come up with a good enough approximation to the parameter space to get a high effective yield of draws. The `GARCHGIBBS.RPF` example shows how to use the more modern technique of Metropolis-Hastings (M-H) to analyze another univariate model—it uses Independence Chain Metropolis for the parameters of the mean model (where a fixed distribution based upon the asymptotic distribution of maximum likelihood is likely to be adequate) and Random Walk Metropolis for the GARCH parameters which are less likely to have a distribution well-approximated by the locally quadratic expansion. We'll emphasize the use of M-H techniques in this chapter.

The alternative simulation approach is bootstrapping. However, this isn't a particularly good alternative for GARCH models. It's extremely time-consuming since it requires estimating the model by non-linear methods for each bootstrap replication. If estimating a GARCH model takes even five seconds per model, then 1000 bootstrap replications will take over 80 minutes. Our example of the bootstrap (Example 10.1) uses it in a more limited way to calculate a more conservative estimate for the VaR with a modest number of bootstrap replications in an “outer” loop while doing multiple sets of VaR simulations in an “inner” loop. By contrast, MCMC methods require just one or two function evaluations per sweep. Each function evaluation still requires a calculation through the entire data set, but just one iteration of a 10 parameter GARCH will generally require 12-13 full function evaluations,¹ and you would typically need 50 or more iterations to estimate a model to convergence when using the bootstrap.

10.1 Bootstrapping

Bootstrapping for inference on the model parameters is only slightly different from the out-of-sample calculation done in Chapter 9: the main difference is the range of entries over which the bootstrapping is done.

Example 10.1 works with an asymmetric EGARCH model. It computes a 50 step VaR by doing 250 bootstrap draws for the model, with 100 out-of-sample bootstrap draws for each estimation of the model. This requires an outer loop over the bootstrapped estimations, with an inner loop over the out-of-sample draws. The “pseudocode” (Appendix G) for this is:

¹One for each parameter for numerical derivatives, plus a few extras in the directional search.

```

<<estimate the model on the original data>>
<<prepare for bootstrapping>>
infobox(action=define,progress,lower=0,upper=nouter) $
  "Bootstrapping"
*
do outer=1,nouter
  <<bootstrap full sample and estimate model>>
  do inner=1,ninner
    <<bootstrap out-of-sample, compute and save VaR>>
  end do inner
  infobox(current=outer)
end do outer
infobox(action=remove)

```

The **INFOBOX** instructions (see the *Tips and Tricks*, page 261) are handy when you have a looping calculation like this that is likely to require some calculation time. You should always *start* with a much more modest number of draws (25 and 10 would probably be a good choice here) just to make sure that you get the calculations done correctly and get some idea of how much time a “production” number of simulations will take.

The first step is to read the data and estimate the desired EGARCH model:

```

open data garch.asc
data(format=free,org=columns) 1 1867 bp cd dm jy sf
*
set dlogdm = 100*log(dm/dm{1})
garch(p=1,q=1,exp,asymmetric,hseries=h) / dlogdm

```

As before, we need the standardized residuals as the source for the bootstrap and we need to keep track of the range over which there are defined:

```

compute gstart=%regstart(),gend=%regend()
set stdux = %resids/sqrt(h)

```

Then, we need to pull the coefficients out of the %BETA vector for use in constructing bootstrapped samples:

```

nonlin(parmset=egarchparms) mx cx ax bx dx
compute %parmspoke(egarchparms,%beta)

```

This sets up a similar set for the parameters in the bootstrap estimates:

```

nonlin(parmset=egarchboots) mb cb ab bb db

```

The following sets the number of draws in each part, and creates the target for the simulated returns:

```

compute nouter=250
compute ninner=100
compute ndraws=nouter*ninner
set retsim 1 ndraws = 0.0

```

This creates the range for the VaR bootstraps:

```

compute nsteps=50
compute baset =gend
compute bstart=baset+1
compute bend=baset+nsteps

```

We're now set for the looping structure from above. The draws for the data are done with:

```

boot entries gstart gend gstart gend
set stdu gstart gend = $
    h=exp(cx+bx*log(h{1})+ax*abs(stdu{1})+dx*%max(stdu{1},0.0)), $
    %if(%ranflip(.5),+1,-1)*stdux(entries(t))
set x gstart gend = mx+sqrt(h)*stdu

```

In this case, the **BOOT** instruction uses the **GSTART** to **GEND** range for both the source and the target. Note that **BOOT** (by default) draws by replacement, which is the proper procedure for this. Simulating an EGARCH is actually a bit simpler than other types of models because the recursion uses the standardized residuals themselves. The first **SET** instruction calculates the H_t , but then doesn't need it in calculating **STDU**, which is simply a (symmetrized for sign) resampled value from the **STDUX** series. The second **SET** builds the bootstrapped series X given the mean (here just the value of MX) and the scaled up standardized U .

We then re-estimate the model with the bootstrapped data:

```

garch(p=1,q=1,exp,asymmetric,hseries=h,noprint) gstart gend x

```

We need *these* estimates for doing the out-of-sample bootstrapping, so we pull them out into the separate set of variables created earlier:

```

compute %parmspoke(egarchboots,%beta)

```

The inner loop is basically the same calculation as was done in Example 9.1. The only change (other than the use of the EGARCH recursion) is that the position for the value in the **RETSIM** array now has to be calculated using the two loop indexes. $(OUTER-1)*NINNER+INNER$ will group together all the inner draws generated by a single outer draw. Note that you have to do $OUTER-1$ on the position calculation to get them to start at entry 1.

Table 10.1: VaR with Bootstrapped Coefficients

Statistics on Series RETSIM			
Observations	25000		
Sample Mean	-0.013909	Variance	0.002380
Standard Error	0.048787	SE of Sample Mean	0.000309
t-Statistic (Mean=0)	-45.077284	Signif Level (Mean=0)	0.000000
Skewness	0.044459	Signif Level (Sk=0)	0.004109
Kurtosis (excess)	1.261394	Signif Level (Ku=0)	0.000000
Jarque-Bera	1665.647717	Signif Level (JB=0)	0.000000
Minimum	-0.263011	Maximum	0.277893
01-%ile	-0.137078	99-%ile	0.111267
05-%ile	-0.092721	95-%ile	0.064656
10-%ile	-0.072293	90-%ile	0.045199
25-%ile	-0.043539	75-%ile	0.015570
Median	-0.014056		

```

boot entries bstart bend gstart gend
set stdu bstart bend = $
    h=exp(cb+bb*log(h{1})+ab*abs(stdu{1})+db*%max(stdu{1},0.0)), $
    %if(%ranflip(.5),+1,-1)*stdux(entries(t))
set sret bstart bend = mb+sqrt(h)*stdu
sstats bstart bend sret*.01>>retsim((outer-1)*ninner+inner)

```

The results are in Table 10.1. The VaR values are a bit wider than you get from assuming the coefficients are fixed, though there is still enough variation, even at 25000 total draws, that you would probably want to do quite a few more if this were being used in practice.

10.2 Monte Carlo Methods

Because GARCH models have a complicated log likelihood function, there is no simple distribution from which to draw any important subset of the parameters. The techniques described here are covered in the *RATS User's Guide*, and we provide a quick summary in Appendix H. If you are interested in learning more, you might want to get the RATS e-course on *Bayesian Econometrics*.

10.2.1 Importance Sampling

The most straightforward Monte Carlo method that can be applied to GARCH models is *importance sampling*. This samples the full parameter set from a single fixed distribution (usually the asymptotic distribution or something based upon it) and weights the draws to correct for errors in the shape of the importance function (sampling density) relative to the true likelihood. As mentioned on page 241, this was the first simulation method employed for GARCH models. The problem, in practice, is that it's hard to come up with a good choice for the importance function when the parameter set gets large. However, since many

of the same ideas for choosing, using and adjusting the sampling density are used in other methods, we'll start here.

We'll examine this by looking at the **GARCHGIBBS.RPF** example in some detail. This uses an AR(1) model on the US 3-month T-bill rate. In order to avoid adding an extra complication, all models will have their likelihoods calculated using the same pre-sample value for the variance and squared residuals: the one which comes off the preliminary least squares estimator.

```
linreg ftbs3
# constant ftbs3{1}
compute h0=%sigmasq
*
* Estimate a GARCH with Normally distributed errors
*
garch(p=1,q=1,reg,presample=h0,distrib=normal) / ftbs3
# constant ftbs3{1}
```

The **GARCH** instruction will provide the mean (estimated coefficients) and the “shape” for the importance function. We also save the (log) likelihood maximum to help avoid overflow problems (page 341).

```
compute fxx  =%decomp(%xx)
compute xbase=%beta
compute fbase=%logl
dec vect betau(%nreg)
```

For drawing from a multivariate Normal or multivariate t , we need a *factor* of the desired covariance matrix, so we choose the Cholesky factor computed using %DECOMP.

```
compute ndraws=100000
compute drawdf=5.0
```

We'll do 100000 draws,² using an importance function which is a t with 5 degrees of freedom. Why 5? No special reason. It has nothing to do with the number of observations or the number of parameters being estimated. It's just that, when doing importance sampling, the worst thing you can do (with an otherwise reasonable importance function) is to go too thin in the tails. There's an analytical example of this in the *RATS User's Guide* which shows how a too-thin importance function causes the estimator to have infinite variance. In practice, what can happen is that you can run one set of simulations which looks perfectly reasonable, and then do another that basically ends up with just one effective data point. This is because the thin-tailed importance function undersamples the tails so when you actually *do* draw an “outlier” that's in the direction where the likelihood is high, its relative weight is enormous.

²In practice, you would start with a much smaller number.

We're going to keep track of (weighted) sums of the coefficients and their outer products in order to estimate the mean and the covariance of the distribution. We also need the sum of the weights (to rescale the weighted sums) and the sum of squared weights, which is used to assess the performance of our importance function.

```
compute sumwt=0.0, sumwt2=0.0
compute [vect] b=%zeros(%nreg,1)
compute [symm] bxx=%zeros(%nreg,%nreg)
```

There are other statistics which might be of interest in addition to the parameters. Here, we keep track of the values of the “persistence” (sum of the a_1 and b_1 coefficients) along with the weights. If you want to either compute an empirical density function (as we will do here), or to compute quantiles of any statistic, you need to keep track of the full collection of values, *and* the weights.

```
set persist 1 ndraws = 0.0
set weights 1 ndraws = 0.0
```

The pseudo-code for importance sampling is the following:

```
infobox(action=define,progress,lower=0,upper=ndraws) $
  "Importance Sampling"
do draw=1,ndraws
  << draw coefficients from the importance function >>
  << evaluate the (log) density of the draw >>
  << evaluate log likelihood at the coefficients >>
  << compute the relative weight >>
  << update statistics >>
  infobox(current=draw)
end do draws
infobox(action=remove)
```

The draw for the coefficients is done with:

```
compute betau=xbase+%ranmvt(fxx,drawdf)
```

XBASE (the maximum likelihood estimates) is the mean, FXX is the factor of the covariance of the Normal and DRAWDF is the degrees of freedom.

The log kernel of the density function for the draw can be recovered using the %RANLOGKERNEL function.

```
compute gx=%ranlogkernel()
```

This will only be correct if you do the draw using one of the self-contained random number generation functions. You *could* create the draw by doing a random multivariate Normal and dividing by the square root of an independently drawn gamma (which is what %RANMVT actually does) but if you did, the

value of `%RANLOGKERNEL()` would be the log kernel of whichever of those two draws you did last.

Note that this *doesn't* include the integrating constants. Theoretically, you want the relative weight on a draw to be the ratio of the likelihood to the density, evaluated at the draw. What we will actually compute (to avoid overflow or underflow) is

$$\frac{f(\theta)/f(\theta_{ML})}{g(\theta)/\text{integrating constant}} \quad (10.1)$$

Since $f(\theta_{ML})$ and the integrating constant are both independent of the draw, these have no effect on the relative weights.

The log likelihood is computed using:

```
garch(p=1,q=1,reg,presample=h0,method=eval,initial=betau) / ftbs3
# constant ftbs3{1}
```

This uses **GARCH** with the option `METHOD=EVAL`, which does a single evaluation of the GARCH log likelihood at the parameter vector given by `BETAU`. You have to make sure that `BETAU` has the proper construction (parameters in the correct positions), which is easy here since the draws are based upon information from a **GARCH** instruction with the identical form. This produces no direct output—what we want is the value of `%LOGL` that it produces:

```
compute fx=%logl-fbase
```

`FX` is the difference between the log likelihood at the input parameters and that at the maximum—when `exp'd` this will give us the numerator in (10.1).

The (relative) weight is computed by:

```
compute weight=%if(%valid(%logl),exp(fx-gx),0.0)
```

This allows for the possibility that the log likelihood isn't computable due to explosive behavior, assigning zero weight to any draw for which `%LOGL` isn't valid.

We now create the weighted sums of the coefficients and their outer product:

```
compute sumwt=sumwt+weight,sumwt2=sumwt2+weight^2
compute b=b+weight*betau
compute bxx=bxx+weight*%outerxx(betau)
```

That's all we need if we are computing the first and second moments. We also save each value of the weight and the (unweighted) persistence measure:

```
compute persist(draw)=betau(4)+betau(5)
compute weights(draw)=weight
```

Table 10.2: Relative Weights for Thin-Tailed Importance Function

Statistics on Series WEIGHTS			
Monthly Data From 1957:01 To 10290:04			
Observations	100000		
Sample Mean	1.2808	Variance	30.0109
Standard Error	5.4782	SE of Sample Mean	0.0173
t-Statistic (Mean=0)	73.9324	Signif Level (Mean=0)	0.0000
Skewness	239.4496	Signif Level (Sk=0)	0.0000
Kurtosis (excess)	67977.6377	Signif Level (Ku=0)	0.0000
Jarque-Bera	19254952356833.4260	Signif Level (JB=0)	0.0000
Minimum	0.0000	Maximum	1573.8026
01-%ile	0.0088	99-%ile	6.1929
05-%ile	0.2639	95-%ile	2.5801
10-%ile	0.4997	90-%ile	1.8493
25-%ile	0.8341	75-%ile	1.2614
Median	1.0301		

That completes the loop. The first calculation outside the loop is to check the “effective sample size”:

```
disp "Effective Sample Size" sumwt^2/sumwt2
```

If the weights are identical (which would happen only if we were drawing from the exact density described by the likelihood), this will match the number of draws. The closer this is to the number of draws, the better—if you get below 20%, you might want to look at adjusting the importance function. Before you do 100000 draws, you should try smaller numbers and see how it works. With our standard seed of 53531, we get (on 100000 draws)

Effective Sample Size	38565.55211
-----------------------	-------------

which is a reasonable percentage. If we replace the degrees of freedom on the t with 50 (so that basically we’re drawing from the asymptotic distribution), we end up with just 5183. If you look at the statistics on the `WEIGHTS` series for the latter case (Table 10.2), we see where the problem is—the weighted sum will be dominated by the relatively small number of values that exceed the 99%-ile. The ratio of the maximum to the median is a quick measure of how much the extremes dominate the more typical draws—that’s well over 1000 here.

There are two obvious places to make adjustments: the value of `DRAWDF`, and the scale on the factor of the covariance matrix. If you need to improve the behavior, you can do one or both of these. The latter would be done by giving a value other than one (typically bigger to fatten the tails) to the `scalefactor` in:

```
compute fxx =scalefactor*%decomp(%xx)
```

Assuming we’ve decided that our settings are adequate, the following converts the weighted sums to the mean and variance of the coefficients, and creates a table (Table 10.3) of the means and standard deviations:

Table 10.3: GARCH Estimates by Importance Sampling

a	0.062746	0.021670
b	0.990226	0.004946
gamma	0.001825	0.000708
alpha	0.327805	0.051441
beta	0.713721	0.035013

Table 10.4: GARCH Estimates by Maximum Likelihood

GARCH Model - Estimation by BFGS					
Convergence in 39 Iterations. Final criterion was 0.0000099 <= 0.0000100					
Dependent Variable FTBS3					
Monthly Data From 1957:02 To 2006:12					
Usable Observations		599			
Log Likelihood		-61.3428			
	Variable	Coeff	Std Error	T-Stat	Signif
1.	Constant	0.0631	0.0226	2.7892	0.0053
2.	FTBS3{1}	0.9901	0.0054	184.6694	0.0000
3.	C	0.0015	0.0005	2.7767	0.0055
4.	A	0.3066	0.0444	6.9059	0.0000
5.	B	0.7301	0.0305	23.9360	0.0000

```

compute b=b/sumwt,bxx=bxx/sumwt-%outerxx(b)
report(action=define)
report(atrow=1,atcol=1,fillby=cols) "a" "b" "gamma" "alpha" "beta"
report(atrow=1,atcol=2,fillby=cols) b
report(atrow=1,atcol=3,fillby=cols) %sqrt(%xdiag(bxx))
report(action=show)

```

This compares with the maximum likelihood estimates (Table 10.4). The GARCH coefficients in the simulation estimates shift a bit towards the lagged squared residuals and away from the lagged variance, and have slightly higher standard errors. Note, by the way, that there is nothing wrong with “bootstrapping” (in a different sense) one set of simulation results to improve the behavior of the sampler. The importance function can be *anything* (data-determined or not) that somehow approximates the shape of the likelihood. If we needed to further improve our importance sampler, we could take the mean and variance from one sampler and use that to build the importance function for a second one.

Finally, the following computes and graphs the persistence measure. Note the need to include the `SMPL=WEIGHTS>0.00` option—we don’t want the graph to include values so extreme that their likelihood was uncomputable. Even with that, Figure 10.1 includes a range of very unlikely values. That’s the nature of importance sampling with a fat-tailed proposal function, as those have positive but effectively zero weights.

```

density(weights=weights, smpl=weights>0.00, smoothing=1.5) $
  persist 1 ndraws xx dx
scatter(style=line,$
  header="Density of Persistence Measure (alpha+beta)")
# xx dx

```

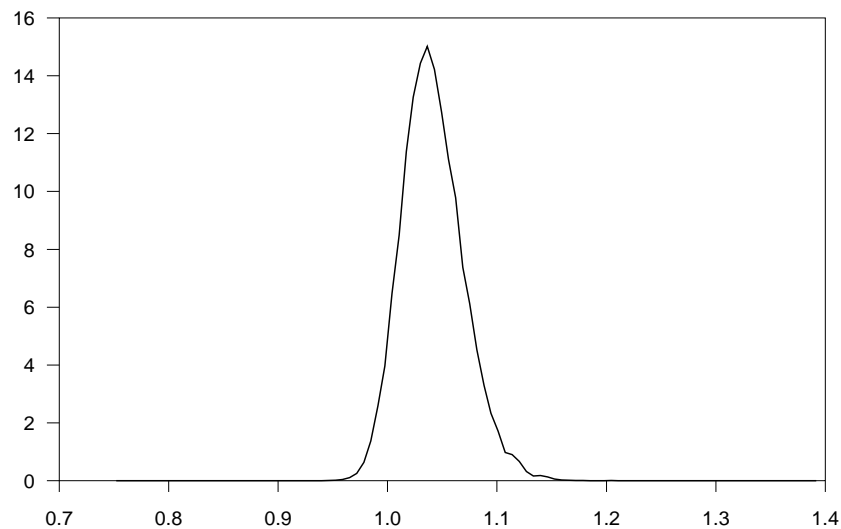


Figure 10.1: Density of Persistence Measure

10.2.2 Markov Chain Monte Carlo

An approach which is more likely to be successful across the broad range of GARCH models is Markov Chain Monte Carlo (MCMC), and, in particular, Metropolis within Gibbs (page 339). An obvious partition of the parameter space is between the mean model parameters and the GARCH parameters.

If there are no “M” effects in the mean model, the distribution of the mean model parameters will probably be reasonably well approximated by the asymptotic distribution given by **GARCH** or it could be approximated more accurately by re-computing the GLS estimator given an updated set of GARCH variances at each sweep. The Normal distribution with the GLS estimates as the mean and the standard GLS covariance matrix would be *exactly* the conditional distribution if it weren’t for the feedback from the residuals to the variance through the GARCH recursion. However, since it isn’t exact, we would have to do the special calculation for GLS each sweep, and *still* do the Metropolis jump calculations. The calculation with the one fixed distribution at each sweep is much simpler, and so should be preferred unless it doesn’t work adequately. We’ll do the mean model using an Independence Chain draw (page 340).

For the GARCH parameters, we could probably also successfully handle this with Independence Chain M-H for a univariate model, since the parameter set is relatively small. Instead, we’ll demonstrate the use of Random Walk M-

H draws (page 341) which is likely to work better in a much wider range of situations.

Example 10.2 is the same data set and same GARCH model as Example 10.1, except with an AR(3) mean model instead of just an intercept. The pseudo-code for the MCMC loop is:

```
infobox(action=define,progress,lower=-nburn,upper=ndraws) $
  "Random Walk Metropolis"
do draw=-nburn,ndraws

  << evaluate the log likelihood at the current settings for the
    mean and GARCH parameters >>
  << draw a test vector from the proposal density for the mean>>
  << evaluate the log likelihood with the test mean vector >>
  << compute the jump probability >>
  << if we jump, replace the mean parameters >>
  << draw an increment for the GARCH parameters >>

  << evaluate the log likelihood for the test GARCH vector >>
  << compute the jump probability >>
  << if we jump, replace the GARCH parameters >>

  infobox(current=draw)
  if draw<=0
    next

  << do whatever bookkeeping is necessary >>
end do draw
infobox(action=remove)
```

Obviously, there are quite a few steps—fortunately, each one is relatively short, and none are hard to understand if you are familiar with the general methods of Metropolis-Hastings.

The basic GARCH model is set up and estimated with:

```
linreg(define=meaneq) dlogdm
# constant dlogdm{1 2 3}
compute h0=%seesq
garch(p=1,q=1,exp,asymmetric,hseries=h,presample=h0,$
equation=meaneq)
```

As before, we use the PRESAMPLE option to control the pre-sample value for the variance throughout the calculations. The output is in Table 10.5. The need for the AR parameters seems obvious from this, and the diagnostics on the standard residuals would tend to show the model is adequate.³

³The Q has a significance level of .08 and the McLeod-Li has .67.

Table 10.5: Standard EGARCH Model with AR(3) Mean

GARCH Model - Estimation by BFGS					
Convergence in 29 Iterations. Final criterion was 0.0000096 <= 0.0000100					
Dependent Variable DLOGDM					
Usable Observations		1863			
Log Likelihood		-2053.7695			
	Variable	Coeff	Std Error	T-Stat	Signif
1.	Constant	-0.0297	0.0141	-2.1100	0.0349
2.	DLOGDM{1}	-0.0733	0.0235	-3.1166	0.0018
3.	DLOGDM{2}	0.0446	0.0217	2.0566	0.0397
4.	DLOGDM{3}	0.0645	0.0189	3.4118	0.0006
5.	C	-0.1845	0.0239	-7.7339	0.0000
6.	A	0.2147	0.0268	8.0270	0.0000
7.	B	0.9677	0.0099	98.0438	0.0000
8.	D	-0.0190	0.0128	-1.4846	0.1376

This extracts the vector of labels for the final output from the ones used in the **GARCH** output:

```
compute alllabels=%reglabels()
```

We need to pull the following information out of the **GARCH** estimation:

1. The mean model coefficients and a factor of their covariance matrix.
2. The GARCH coefficients and a factor of *their* covariance matrix.

The first block of coefficients (for the mean) are handled with

```
compute xbeta0=%xsubvec(%beta,1,%nregmean)
compute fbeta=%decomp(%xsubmat(%xx,1,%nregmean,1,%nregmean))
compute xbeta=xbeta0
compute gx=0.
```

The **XBETA** will be used for the “working” copy of the mean parameters, while **XBETA0** will be used as the mean of the distribution from which we will draw the coefficient vectors. Independence Chain M-H is very similar to Importance Sampling, so we might find it necessary to scale up the factor in **FBETA** if we don’t get the acceptance rate that we would like. **GX** will be used to keep track of the log kernel density at the current working vector; this is zero, since we’re starting at the mean.

The information for the second block is:

```
compute xgarch=%xsubvec(%beta,%nregmean+1,%nreg)
compute fgarch=.5*%decomp(%xsubmat($
    %xx,%nregmean+1,%nreg,%nregmean+1,%nreg))
```

XGARCH will be the working vector of GARCH parameters. We're using Random Walk M-H on this block. This is more flexible and can handle even fairly oddly shaped likelihood surfaces, but there's much more of an art to picking the density for the test changes, and what works in one situation may fail rather badly in another. Here, we'll do mean zero multivariate Normals with covariance matrix a medium-sized multiple of the curvature estimate off maximum likelihood. In most cases where the likelihood isn't *too* badly shaped, this will work reasonably well—it tends to get the relative scales of the movements in the parameters correct and also tends to have them move in the proper direction in multivariate space.⁴ The other difficulty is that, unlike Independence Chain M-H (or Importance Sampling) where the ideal is to get 100% acceptance (or equal weights), there really is no “ideal” acceptance rate for Random Walk M-H. You can push it very close to 100% by making the size of the increment very small (for instance, making the multiplier in the FGARCH calculation equal to .01). However, that defeats the purpose of Random Walk M-H by never giving the sampler a chance to explore values away from the initial settings. Somewhere in the 20%-40% range is a realistic target.

As the first step inside the loop, we evaluate the log likelihood at the current settings of the parameters, saving it into FX. We could avoid this step by carefully keeping track of the calculations which are being retained, but it's generally simpler to start each sweep with a fresh computation.

```
garch(p=1,q=1,exp,asymmetric,presample=h0,equation=meaneq,$
      method=eval,initial=xbeta~~xgarch)
set u = %resids
compute fx=%logl
```

The ~~ operator does a concatenation of the two working vectors. We also keep track of the residuals.

The next step is to draw a test vector for the mean parameters (into YBETA) using the fixed proposal density determined by the mean of XBETA0 and covariance factor FBETA. The GARCH model is evaluated at parameter vector formed by combining YBETA (the test means) with XGARCH (the current GARCH parameters).

```
compute ybeta=xbeta0+%ranmvnormal(fbета)
compute gy=%ranlogkernel()
garch(p=1,q=1,exp,asymmetric,presample=h0,equation=meaneq,$
      method=eval,initial=ybeta~~xgarch)
compute fy=%logl
```

At this point, FX and GX are the log likelihood and log kernel density at the last accepted set of parameters and FY and GY are the same at the test set. Using these, the jump probability is:

⁴If two parameters are positively correlated, the increment will tend to have them move in the same direction, and the opposite direction if they are negatively correlated.

```
compute alpha=exp(fy-fx-gy+gx)
```

The test for acceptance and handling in case of acceptance is:

```
if %ranflip(alpha) {
  compute bjumps=bjumps+1
  compute xbeta=ybeta,gx=gy
  set u = %resids
}
```

You don't have to do a separate test for whether $\alpha > 1$ as %RANFLIP will always come out 1 when the "probability" is greater than 1. If we accept, we copy the values for BETA and Y from the "test" G variables to the "accepted" F's.

If we change the mean coefficients, we need to recompute the likelihood with the new set. For simplicity, we'll just compute it regardless. Again, we can avoid this when we don't move; it's just a bit more straightforward if we recompute it either way. If there are no "M" terms, then this can be computed using the residuals alone and the NOMEAN option, which is why we've been saving the residuals. If you have M effects, you have to repeat the full calculation at XBETA~~XGARCH.

```
garch(p=1,q=1,exp,asymmetric,presample=h0,nomean,$
  method=eval,initial=xgarch) / u
compute fx=%logl
```

The test vector (YGARCH) for the GARCH parameters is generated by adding a draw from a mean zero multivariate Normal draw with covariance factor FGARCH. Again, this is done on the residuals from earlier since (if there are no "M" effects), those won't have changed.

```
compute ygarch=xgarch+%ranmvnormal(fgarch)
garch(p=1,q=1,exp,asymmetric,presample=h0,nomean,$
  method=eval,initial=ygarch) / u
compute fy=%logl
```

The jump probability is computed with

```
compute alpha=%if(%valid(fy),exp(fy-fx),0.0)
```

The check for a valid FY is needed here, but not when we drew the mean parameters, since it's only the GARCH parameters that can cause the recursion to become explosive.

```
if %ranflip(alpha) {
  compute gjumps=gjumps+1
  compute xgarch=ygarch
}
```


Table 10.6: AR(3) EGARCH Done by MCMC

Constant	-0.0283	0.0154
DLOGDM{1}	-0.0739	0.0242
DLOGDM{2}	0.0463	0.0232
DLOGDM{3}	0.0672	0.0246
C	-0.1953	0.0243
A	0.2256	0.0271
B	0.9625	0.0106
D	-0.0173	0.0140

In this case, the bookkeeping done on positive values for `DRAW` (thus ignoring the burn-in draws) is:

```
compute betau=xbeta~~xgarch
compute b=b+betau,bxx=bxx+%outerxx(betau)
```

which combines the working parameters into a single vector, and updates the sum and the outer product.

We're now done with the draws. The first calculation after that is the percentage of jumps (across the full set of draws, both the burn-in and the kept ones):

```
disp "Percentage of jumps for mean parms" $
100.0*float(bjumps)/(nburn+ndraws+1)
disp "Percentage of jumps for GARCH parms" $
100.0*float(gjumps)/(nburn+ndraws+1)
```

The results (with a `SEED` of 53531) are

Percentage of jumps for mean parms	62.80338
Percentage of jumps for GARCH parms	63.92146

The first is fine (for Independence Chain), the second a bit higher than we would like (for the Random Walk), so a higher scaling than .5 on the calculation of `EGARCH` is worth a try. The mean and standard errors from the estimator are in Table 10.6. Again, they're fairly similar to the results from maximum likelihood (Table 10.5), with the main difference being somewhat higher standard errors, particularly on the lag coefficients in the mean model.

10.2.3 MCMC Estimation of DCC Correlations

The final example uses Random Walk M-H to give confidence bands around the time-varying correlations produced by DCC (Section 5.7). Maximum likelihood gives us point estimates of the correlations, but there is no good way to apply the delta method or anything like it to give any error bands on those since they are computed using a rather complicated recursive process.

Example 10.3 uses the same exchange rate data set as the `GARCHMV.RPF` program distributed with RATS. We look at a three variable model, with returns on Japanese, French and Swiss currencies versus the U.S. dollar.

```

set xjpn = 100.0*log(usxjpn/usxjpn{1})
set xfra = 100.0*log(usxfra/usxfra{1})
set xsui = 100.0*log(usxsui/usxsui{1})
*
compute n=3
dec vect[strings] vlabels(n)
compute vlabels=||"Japan", "France", "Switzerland"||

```

As written, this program requires a great deal of memory. It's 6237 data points and the number of statistics of interest (the time-varying correlations) are of that same large size. The program is computing the quantiles of those (median, and the 5th and 95th percentiles). Since there are no sufficient statistics for quantiles, we have to save all the draws for each of those correlations. To keep the memory requirements down, we only do 1000 (keeper) draws, which is probably enough in practice. With more data points, more variables and more draws, you could run into memory problems on the 32-bit version of RATS. However, if you want to do this type of analysis with a larger model, you can also just save the first and second moments of the correlations and use those to get the mean and error bands. This takes much less memory since you only need two values for each correlation data point. The difference between quantiles and moment-based bands is small if the bands are tight and the distribution relatively symmetrical.

As before, we start with the maximum likelihood estimates:

```

garch(p=1,q=1,mv=dcc,rvectors=rv,hmatrices=h) / xjpn xfra xsui
compute gstart=%regstart(),gend=%regend()

```

We'll do Random Walk M-H across the entire parameter set. (There is no mean "model", just the intercepts). The increment will be similar to the random walk block in Example 10.2, except that we will use a t rather than a Normal:

```

compute fxx =.5*%decomp(%xx)
compute nuxx=10.0

```

We'll use 1000 burn-in draws followed by 1000 keeper draws.

```

compute nburn    =1000
compute nkeep    =1000
compute accept   =0

```

This organizes the rather complicated structure for saving the DCC calculations. It's a SYMMETRIC of SERIES of VECTORS. The inner VECTOR is the collection across draws, the SERIES is across entries and the outer SYMMETRIC is across the country pairs. While we could also keep track of the model's coefficients, in this example, we'll just be saving the correlations.

```

dec symm[series[vect]] dcccrr(n,n)
do i=1,n
  do j=1,i
    gset dcccrr(i,j) gstart gend = %zeros(nkeep,1)
  end do j
end do i

```

We start at the ML estimates. Since we need to extract the correlations for each draw, we save the current estimates of the covariances in a `SERIES[SYMM]` called `HLAST`.

```

dec series[symm] hlast
compute logplast=%logl
compute blast    =%beta
gset hlast gstart gend = h

```

Inside the draw loop, we draw a test set of parameters by adding a multivariate- t increment to the last set of parameters and evaluate the GARCH model there. Note that we save the covariance matrices into the separate `SERIES[SYMM]` called `HTEST`, since we don't know yet if we want to use them.

```

compute btest=blast+%ranmvt(fxx,nuxx)
garch(p=1,q=1,mv=dcc,initial=btest,method=eval,$
      rvector=rv,hmatrices=htest) / xjpn xfra xsui
compute logptest=%logl

```

For the random walk increment, the probability of moving is just the ratio of the likelihoods—as before we include a test for whether the GARCH model has failed to be computable. In case we move, we need to save the test values for the coefficients, the log likelihood and the series of covariance matrices.

```

compute alpha=%if(%valid(logptest),exp(logptest-logplast),0.0)
if %ranflip(alpha) {
  compute accept  =accept+1
  compute blast   =btest
  compute logplast=logptest
  gset hlast gstart gend = htest
}

```

In this example, the `INFOBOX` is set to provide a running statistic on the percentage of draws that have been accepted. This allows us to cancel the operation relatively quickly if the acceptance percentage is running too high or low, make an adjustment to the increment and try again.

```

infobox(current=draw) %strval(100.0*accept/(draw+nburn+1),"##.##")

```

When we're outside the burn-in period, we need to convert the sequence of current covariance matrices (`HLAST(T)`) to correlations, and put the values into the proper slots:

```

do i=1,n
  do j=1,i
    do t=gstart,gend
      compute dcccorr(i,j)(t)(draw)=%cvtocorr(hlast(t))(i,j)
    end do t
  end do j
end do i

```

Again, DCCORR is a SYMMETRIC[SERIES[VECTORS]] so the first set of subscripts are for the country pair, the second for the time period, and the third for the draw number.

That finishes the work inside the loop. Outside, we need to analyze each set of draws for the correlations. That's organized using the following:

```

dec vect xcorr(nkeep)
dec symm[series] lower(n,n) upper(n,n) median(n,n)
clear(zeros) lower upper median

```

XCORR is just used as a temporary when we are working with the draws for a single correlation. The LOWER, UPPER and MEDIAN have (respectively) the 5%-ile, 95%-ile and medians for the combination of countries.

While the first set of graphs does the entire range, there's really nothing to see, because the spread is small relative to the movement of the series themselves. So we also do a shorter period. The full range is done with:

```

do j=2,n
  do k=1,j-1
    do t=gstart,gend
      ewise xcorr(i)=dcccorr(j,k)(t)(i)
      compute frac=%fractiles(xcorr,||.05,.50,.95||)
      compute lower(j,k)(t)=frac(1)
      compute upper(j,k)(t)=frac(3)
      compute median(j,k)(t)=frac(2)
    end do tme

    graph(header=$
      "DCC Estimates for "+vlabels(k)+" with "+vlabels(j)) 3
      # median(j,k) gstart gend
      # lower(j,k) gstart gend 2
      # upper(j,k) gstart gend 2
    end do k
  end do j

```

The EWISE instruction extracts the draws for the combination of countries J, K at period T. That's put through the %FRACTILES function to get the desired three quantiles. Note that this calculation *can* take quite a while if the number

of draws is high since the complexity of the sorting operation goes up more than proportionately with the number of draws.

One of the graphs from a shorter range (just from entry 1000 to 1200) is shown in Figure 10.2. As you can see, it's not even close to being constant width, with a high-low gap of around .05 for the first two-thirds of the range, and effectively zero for the remainder. The data are noisier in the second half, with several large outliers at similar times—those tend to dominate the DCC recursion for a considerable period of time.

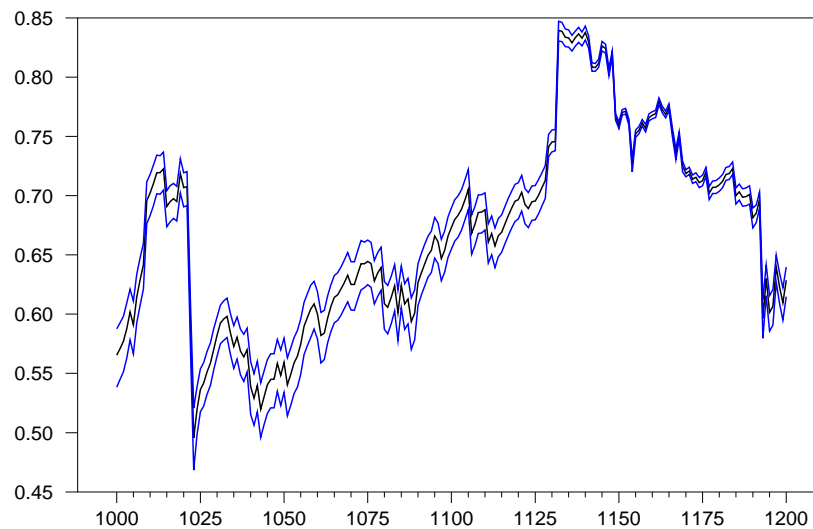


Figure 10.2: DCC Estimates for France with Switzerland

10.3 RATS Tips and Tricks

The INFOBOX instruction

You can use **INFOBOX** if you want feedback about what is happening during a lengthy operation. As always, the first step is to see that the loop is doing the calculation correctly with a small number of draws. When you ramp up to a large number of draws, however, it's generally a good idea to include the **INFOBOX**. This brings up a dialog box which sits on top of the other RATS windows. You can display a graphical *progress bar* and up to two lines of text in the box. There are three stages to this: one before the loop to set up the dialog box, one inside it to update the progress, and one at the end to remove it from the screen:

```
infobox(action=define,other options) "top messagestring"
infobox(other options) "bottom messagestring"
infobox(action=remove)
```

In our typical case, these three lines are:

```
infobox(action=define,progress,lower=-nburn,upper=ndraws) $
  "Random Walk MH"
infobox(current=draw) %strval(100.0*accept/(draw+nburn+1),"##.##")
infobox(action=remove)
```

The first defines the progress bar as running from `-nburn` to `ndraws` which is the range of the `do draw` loop. The messagestring on this is the top line on the dialog, and cannot be changed later. The **INFOBOX** in the middle of the loop updates the progress bar with the current value of `draw`. It also (in this case), shows what our running acceptance probability is for M-H. The progress bar has a *Cancel* button. If the acceptance rate is running too low, you can kill the process and try retuning your settings. Note that `lower`, `upper` and `current` must be integers. The final **INFOBOX** outside the loop gets rid of the progress bar.

Example 10.1 Bootstrapping

This demonstrates re-estimation of a model using bootstrapped data. It calculates the VaR using an EGARCH model with a combination of coefficients estimated with bootstrapped data and bootstrapped out-of-sample residuals. The details are in Section 10.1.

```

open data garch.asc
data(format=free,org=columns) 1 1867 bp cd dm jy sf
*
set dlogdm = 100*log(dm/dm{1})
garch(p=1,q=1,exp,asymmetric,hseries=h) / dlogdm
*
* Create the standardized residuals
*
compute gstart=%regstart(),gend=%regend()
set stdux = %resids/sqrt(h)
*
* Pull out the parameters for the EGARCH model. Because the mean
* model is just a constant, mx is just a scalar real.
*
nonlin(parmset=egarchparms) mx cx ax bx dx
compute %parmspoke(egarchparms,%beta)
*
* These will be used for the parameters from the bootstrap estimates
*
nonlin(parmset=egarchboots) mb cb ab bb db
*
* Set the number of draws of each type, and prepare the RETSIM
* series for the simulated returns.
*
compute nouter=250
compute ninner=100
compute ndraws=nouter*ninner
set retsim 1 ndraws = 0.0
*
* Set the number of VaR steps and set up the range for the VaR
* bootstrap.
*
compute nsteps=50
compute baset =gend
compute bstart=baset+1
compute bend =baset+nsteps
*
* Initialize the H and STDU series from start to the end of the VaR
* interval. The lagged (pre-sample) values are needed in generating
* the full bootstrap sample.
*
stats dlogdm
set h 1 bend = %variance
set stdu 1 bend = 0.0
*
*seed 53531

```

```

infobox(action=define,progress,lower=0,upper=nouter) $
  "Bootstrapping"
*
do outer=1,nouter
  *
  * In the outer loop, we draw a full set from the estimation range
  *
  boot entries gstart gend gstart gend
  *
  * Rebuild the data using the "x" parameters.
  *
  set stdu gstart gend = $
    h=exp(cx+bx*log(h{1})+ax*abs(stdu{1})+dx*%max(stdu{1},0.0)), $
    %if(%ranflip(.5),+1,-1)*stdux(entries(t))
  set x gstart gend = mx+sqrt(h)*stdu
  *
  * Estimate the model on the bootstrapped data
  *
  garch(p=1,q=1,exp,asymmetric,hseries=h,noprint) gstart gend x
  *
  * Pull out the coefficients on the bootstrapped estimates
  *
  compute %parmspoke(egarchboots,%beta)
  do inner=1,ninner
    *
    * In the inner loop, we draw a set only over the forecast
    * range. These use the "b" parameters.
    *
    boot entries bstart bend gstart gend
    set stdu bstart bend = $
      h=exp(cb+bb*log(h{1})+ab*abs(stdu{1})+db*%max(stdu{1},0.0)), $
      %if(%ranflip(.5),+1,-1)*stdux(entries(t))
    set sret bstart bend = mb+sqrt(h)*stdu
    sstats bstart bend sret*.01>>retsim((outer-1)*ninner+inner)
  end do inner
  infobox(current=outer)
end do outer
infobox(action=remove)
*
stats(fractiles) retsim 1 ndraws
disp "1% VaR over " nsteps "periods on $1 investment" -%fract01

```

Example 10.2 Univariate Markov Chain Monte Carlo Estimation

This estimates a univariate EGARCH model by Markov Chain Monte Carlo (MCMC) methods. This is covered in Section 10.2.2.

```

open data garch.asc
data(format=free,org=columns) 1 1867 bp cd dm jy sf
*

```



```

set dlogdm = 100*log(dm/dm{1})
linreg(define=meaneq) dlogdm
# constant dlogdm{1 2 3}
compute h0=%seesq
garch(p=1,q=1,exp,asymmetric,hseries=h,presample=h0,$
      equation=meaneq)
compute alllabels=%reglabels()
*
* Do diagnostics on standardized residuals
*
set ustd = %resids/sqrt(h)
@regcorrs(number=10,dfc=3,qstat) ustd
@mcleodli(number=10,dfc=3) ustd
*
* Extract the mean model coefficients and (factor) of the covariance
* matrix. We'll use independence chain M-H to draw this block.
*
compute xbeta0=%xsubvec(%beta,1,%nregmean)
compute fbeta =%decomp(%xsubmat(%xx,1,%nregmean,1,%nregmean))
compute xbeta =xbeta0
compute gx     =0.
*
* Pull out the GARCH coefficients. These will be drawn as a second
* block using Random Walk M-H.
*
compute xgarch=%xsubvec(%beta,%nregmean+1,%nreg)
compute fgarch=.5*%decomp(%xsubmat($
      %xx,%nregmean+1,%nreg,%nregmean+1,%nreg))
*
compute nburn=1000,ndraws=10000
*
* Initialize vectors for the first and second moments of the
* coefficients.
*
compute [vect] b=%zeros(%nreg,1)
compute [symm] bxx=%zeros(%nreg,%nreg)
dec vect betau(%nreg) betadraw ygarch
*
* Counters for the number of jumps
*
compute bjumps=0,gjumps=0
*
* In practice, this should be commented out. We use it to control
* the results for demonstration purposes
*
seed 53531
*
infobox(action=define,progress,lower=-nburn,upper=ndraws) $
  "Random Walk Metropolis"
do draw=-nburn,ndraws
  *
  * Drawing regression parameters. Evaluate the log likelihood
  * (into FX) at the values for XBETA and XGARCH.
  *

```

```

garch(p=1,q=1,exp,asymmetric,presample=h0,equation=meaneq,$
      method=eval,initial=xbeta~~xgarch)
set u = %resids
compute fx=%logl
*
* Draw a test vector from the (fixed) proposal density for the
* mean model. Recalculate the log likelihood (into FY) and fetch
* the log density at the draw (into GY).
*
compute ybeta=xbeta0+%ranmvnormal(fbета)
compute gy=%ranlogkernel()
garch(p=1,q=1,exp,asymmetric,presample=h0,equation=meaneq,$
      method=eval,initial=ybeta~~xgarch)
compute fy=%logl
*
* Compute the jump alpha. If we need to move, reset xbeta and
* gx, and copy the residuals from Y into u.
*
compute alpha=exp(fy-fx-gy+gx)
if %ranflip(alpha) {
  compute bjumps=bjumps+1
  compute xbeta=ybeta,gx=gy
  set u = %resids
}
*
* Evaluate the log likelihood (into FX) on the residuals at the
* current settings for XGARCH.
*
garch(p=1,q=1,exp,asymmetric,presample=h0,nomean,$
      method=eval,initial=xgarch) / u
compute fx=%logl
*
* Draw from the proposal distribution centered around XGARCH and
* evaluate the log likelihood into FY.
*
compute ygarch=xgarch+%ranmvnormal(fgarch)
garch(p=1,q=1,exp,asymmetric,presample=h0,nomean,$
      method=eval,initial=ygarch) / u
compute fy=%logl
*
* Compute the jump alpha. (Because draws for the GARCH
* parameters can turn the model explosive, check for an invalid
* FY).
*
compute alpha=%if(%valid(fy),exp(fy-fx),0.0)
if %ranflip(alpha) {
  compute gjumps=gjumps+1
  compute xgarch=ygarch
}
infobox(current=draw)
if draw<=0
  next
*
* Update the accumulators if we're past the burn-in

```

```

*
  compute betau=xbeta~~xgarch
  compute b=b+betau,bxx=bxx+%outerxx(betau)
end do draw
infobox(action=remove)
*
disp "Percentage of jumps for mean parms" $
  100.0*float(bjumps)/(nburn+ndraws+1)
disp "Percentage of jumps for GARCH parms" $
  100.0*float(gjumps)/(nburn+ndraws+1)
*
* Transform the accumulated first and second moments into mean and
* variance.
*
compute b=b/ndraws,bxx=bxx/ndraws-%outerxx(b)
report(action=define)
report(atrow=1,atcol=1,fillby=cols) alllabels
report(atrow=1,atcol=2,fillby=cols) b
report(atrow=1,atcol=3,fillby=cols) %sqrt(%xdiag(bxx))
report(action=show)

```

Example 10.3 Markov Chain Monte Carlo Estimation: Multivariate Model

This computes confidence bands on the DCC estimates of the time-varying correlation using Random Walk M-H. See Section 10.2.3 for more.

```

open data g10xrate.xls
data(format=xls,org=columns) 1 6237 usxjpn usxfra usxsui
*
set xjpn = 100.0*log(usxjpn/usxjpn{1})
set xfra = 100.0*log(usxfra/usxfra{1})
set xsui = 100.0*log(usxsui/usxsui{1})
*
compute n=3
dec vect[strings] vlabels(n)
compute vlabels=||"Japan","France","Switzerland"||
*****
garch(p=1,q=1,mv=dcc,rvectors=rv,hmatrices=h) / xjpn xfra xsui
compute gstart=%regstart(),gend=%regend()
*
* Proposal density for the increment in random walk MH is based
* upon the covariance matrix from the GARCH estimates. This is a
* multivariate t with <<nuxx>> degrees of freedom. The scale on FXX
* and the value for NUXX may need to be adjusted in different
* applications to get a reasonable acceptance rate.
*
compute fxx =.5*%decomp(%xx)
compute nuxx=10.0
*
compute nburn    =1000
compute nkeep    =1000
compute accept   =0
*
* This keeps track of the DCC correlations estimates. This is a
* complicated structure since we have, for each draw, one full
* series for each off-diagonal. (For simplicity, this is also
* saving the diagonals, which are, of course, 1's). With a bigger
* model (longer time series or more variables) or more draws, you
* could run into memory issues.
*
dec symm[series[vect]] dcccorr(n,n)
do i=1,n
  do j=1,i
    gset dcccorr(i,j) gstart gend = %zeros(nkeep,1)
  end do j
end do i
*
* Start at the ML estimates
*
dec series[symm] hlast
compute logplast=%logl
compute blast    =%beta
gset hlast gstart gend = h

```

```

*
infobox(action=define,progress,lower=-nburn,upper=nkeep) $
  "Random Walk M-H"
do draw=-nburn,nkeep
  compute btest=blast+%ranmvt(fxx,nuxx)
  *
  * Evaluate the GARCH model at the proposal
  *
  garch(p=1,q=1,mv=dcc,initial=btest,method=eval,$
    rvector=rv,hmatrices=htest) / xjpn xfra xsui
  compute logptest=%logl
  *
  * Carry out the Metropolis test
  *
  compute alpha=%if(%valid(logptest),exp(logptest-logplast),0.0)
  if %ranflip(alpha) {
    compute accept =accept+1
    compute blast  =btest
    compute logplast=logptest
    gset hlast gstart gend = htest
  }
  *
  infobox(current=draw) %strval(100.0*accept/(draw+nburn+1),"##.##")
  if draw<=0
    next
  *
  do i=1,n
    do j=1,i
      do t=gstart,gend
        compute dcccrr(i,j)(t)(draw)=%cvtocorr(hlast(t))(i,j)
      end do t
    end do j
  end do i
end do draw
infobox(action=remove)
*
dec vect xcorr(nkeep)
dec symm[series] lower(n,n) upper(n,n) median(n,n)
clear(zeros) lower upper median
*
* Generate the median and 90% confidence band for the DCC
* correlations. For this data set, the period to period movement is
* large enough relative to the spread that the three lines
* basically end up on top of each other. If you graph over a
* shorter period, you can see at least some spread between the
* upper and lower bounds.
*
do j=2,n
  do k=1,j-1
    do t=gstart,gend
      ewise xcorr(i)=dcccrr(j,k)(t)(i)
      compute frac=%fractiles(xcorr,||.05,.50,.95||)
      compute lower(j,k)(t)=frac(1)
      compute upper(j,k)(t)=frac(3)
    end do t
  end do k
end do j

```

```

        compute median(j,k) (t)=frac(2)
    end do tme
    graph(header="DCC Estimates for "+vlabels(k)+" with "+vlabels(j)) 3
    # median(j,k) gstart gend
    # lower(j,k) gstart gend 2
    # upper(j,k) gstart gend 2
end do k
end do j
*
* Over a narrower range. The calculations of the quantiles are
* already done above.
*
do j=2,n
    do k=1,j-1
        graph(header="DCC Estimates for "+vlabels(k)+" with "+vlabels(j)) 3
        # median(j,k) 1000 1200
        # lower(j,k) 1000 1200 2
        # upper(j,k) 1000 1200 2
    end do k
end do j

```

GARCH Models with Macroeconomic Data

The overwhelming majority of applications of GARCH models are to financial returns data, and most multivariate GARCH models apply to a collection of similar returns, such as a set of exchange rates, or set of stocks or bonds. Applications to macroeconomic data are not as common. The UK inflation example in Engle (1982) wasn't very convincing and interest rates seem too dominated by policy to show the "endogenous" variance behavior of the typical GARCH. Aside from a question as to whether the typical macro variances would have GARCH effects at *any* recording rate, there is also the fact that these would be much less apparent at coarser frequencies. For process *means*, if you have a random walk, it's still a random walk at a lower sampling rate, and it's still a unit root process (with added short-run dynamics) if it's time-averaged. However, even a very persistent GARCH process very quickly loses the variance clustering when sampled at a lower frequency, and even more quickly when time-averaged.

And unlike financial return data where the mean model is often quite trivial, with macro data, there is no reason to expect lag variables to be tiny at best. In fact, with macro data, the primary interest is likely to *be* the mean model, rather than the variance model—allowing for GARCH effects should improve the efficiency of the estimates of lag coefficients and change the definition of a "standard" shock.

The example used in this Chapter is based upon the model in Elder & Serletis (2010). This is a bivariate SVAR-GARCH model on oil price growth and a growth measure for some macroeconomic aggregate—we'll use GDP. The goal is to study the effect of an oil price shock on GDP, which would ordinarily be handled using a standard SVAR. However, by embedding this in a GARCH-M model, it's possible to also look at whether the *variance* of changes in oil price have an effect on growth.

We will make certain changes to their analysis—lengthening the data set, and making a change to the construction of the oil price series. One thing we will *not* do is change the restriction that the only "M" effect is on the oil price variance—there is no reason believe that variance in GDP will cause a mean shift in oil price growth. If you try to apply the same type of model to a different pair of variables, that might not be a reasonable restriction.

Elder and Serletis use the composite refiners' acquisition cost (RAC) data for

the oil price¹ which is a monthly series available from 1974 on. If you want to analyze oil price effects in the U.S. and would like to include the mid 1970's in your data set, you need to be careful about your measure of oil price, since U.S. (domestically produced) crude oil prices were controlled during that period. The RAC includes a mix of prices on foreign- and domestic-sourced oil. Where we will differ from them is that they compact it from monthly to quarterly by taking the final value in each quarter. They then convert this to “real” oil prices by dividing by the GDP deflator. This creates a subtle timing problem since the GDP deflator is basically an average across the whole quarter, as is GDP itself. Instead, our programs will compact the oil price as quarterly averages of the monthly values. While an apparently trivial change, it actually corrects a problem in their estimates because, with the timing problem, the oil price series shows at best a very weak GARCH effect (both in the bivariate specification and also just in a univariate model), while it shows more standard GARCH behavior when the timing issue is eliminated.

11.1 An SVAR-GARCH-M model

We looked at five-variable SVAR-GARCH model in Section 8.4. That used a “B” type structural model, with

$$\mathbf{u}_t = \mathbf{B}(\theta)\mathbf{v}_t$$

where \mathbf{u} are the VAR residuals and \mathbf{v} are the structural shocks, which are assumed to follow independent univariate GARCH processes. Elder and Serletis use an “A” model:

$$\mathbf{A}(\theta)\mathbf{u}_t = \mathbf{v}_t \quad (11.1)$$

If we write the VAR in the form of the very general multivariate regression:

$$\mathbf{Y}_t = \mathbf{G}\mathbf{Z}_t + \mathbf{u}_t \quad (11.2)$$

then we can combine this with (11.1) as

$$\mathbf{A}\mathbf{Y}_t = (\mathbf{A}\mathbf{G})\mathbf{Z}_t + \mathbf{v}_t \quad (11.3)$$

If there were no restrictions on \mathbf{G} , then (11.3) and (11.2) would be equivalent. Their model does, however, have a restriction on one of the coefficient pairs: that the “M” term doesn’t enter the first (oil price) equation. However, the \mathbf{A} matrix is a 2×2 lower triangular matrix

$$\begin{bmatrix} 1 & 0 \\ b & 1 \end{bmatrix} \quad (11.4)$$

which is a standard recursive (Cholesky) model ordering the variables: oil price then GDP. If you apply this to the restricted coefficients:

$$\begin{bmatrix} 1 & 0 \\ b & 1 \end{bmatrix} \begin{bmatrix} 0 \\ g_2 \end{bmatrix} = \begin{bmatrix} 0 \\ g_2 \end{bmatrix}$$

¹Available from the U.S. Department of Energy web site.

so the restriction is unaffected by multiplying through.

For the simple two-variable recursive structural model, the A and B forms are equivalent:

$$\begin{bmatrix} 1 & 0 \\ b & 1 \end{bmatrix}^{-1} = \begin{bmatrix} 1 & 0 \\ -b & 1 \end{bmatrix}$$

We'll do the first example program using the A-form model from the paper, and the second showing the (more generally applicable) B-model.

As mentioned earlier, the M effect only uses the variance on the oil price shock. Because of the recursive structure, for the oil price $u = v$ so we can use whichever form seems most convenient.

The data are read with:

```
open data oildata.rat
calendar(q) 1974:1
data(format=rats,compact=average) 1974:01 2013:01 ipdm rac gdpmc1
```

RAC and GDPMC1 are monthly and quarterly series as downloaded from their sources.² IPDM is a monthly disaggregation of the quarterly implicit price deflator, but all that matters is that the disaggregation maintained the quarterly average. The difference in data preparation between this and the original paper was that they (in effect) did

```
data(format=rats,compact=last) 1974:01 2013:01 rac
data(format=rats,compact=average) 1974:1 2013:1 ipdm gdpmc1
```

The data are transformed to annualized growth rates, where the nominal oil price is deflated first:

```
set realoil = rac/ipdm
set oilgrow = 400.0*log(realoil/realoil{1})
set gdpgrow = 400.0*log(gdpmc1/gdpmc1{1})
```

If we look at the lag length for the VAR³

```
@varlagselect(crit=aic,lags=8)
# oilgrow gdpgrow
```

we find that AIC picks 3, with 4 a close second.

²GDPMC1 was obtained from FRED.

³With the caveat that this assumes a constant covariance matrix.

VAR Lag Selection	
Lags	AICC
0	2371.28852
1	2348.14845
2	2339.83192
3	2337.82515*
4	2338.79800
5	2345.29634
6	2348.94857
7	2355.71840
8	2362.06212

The authors chose to use a full year's worth, going with 4, which was standard in the VAR literature on this topic.

The VAR system is set up with four lags on each variable, and including a slot for the square root of the oil price variance which will be generated as part of the likelihood evaluation:

```
compute nlags=4
set sqrthoil = 0.0
*
system(model=basevar)
variables oilgrow gdpgrow
lags 1 to nlags
det constant sqrthoil
end(system)
```

The M term could have been something other than the current standard deviation—it could have been the variance instead, or some lag of one function or another. However, the work in the paper was done with the standard deviation, and that *does* fit somewhat better than the variance.

Because we will need it later to impose a restriction on the M effect, we figure out the position of that coefficient in each equation:

```
compute meffectpos=2*nlags+2
```

The following will estimate an OLS VAR with constant covariance matrices:

```
estimate(resids=u)
```

Because `SQRTHOIL` is (at this point) just all zeros, that regressor has no effect on any of the results (other than the *count* of the number of regressors).

To compare the SVAR-GARCH with the standard VAR, the authors compute the Schwarz or Bayesian Information Criterion. The two models actually nest⁴ so a formal likelihood ratio test *could* be used if desired. The Schwarz criterion for the VAR is computed with:

⁴If the lagged terms in the two GARCH processes are zeroed out, we have identical representations for the covariance matrix.

```
compute varparms=%nfree-%nvar
compute varsic=-2.0*%logl+varparms*log(%nobs)
```

Usually, when you compute one of the information criteria for a VAR you count only the total number of lag parameters (provided by %NREGSYSTEM). However, **ESTIMATE** is also estimating the covariance matrix, and if we're comparing fit with a GARCH model that explicitly models the covariance matrix, we have to count the number of free parameters in the covariance matrix for the VAR, which are included in the %NFREE variable. The -%NVAR adjusts for the inclusion (in the %NFREE count) of the zeroed-out SQRTHOIL terms.

One big difference between this example and the one in Section 8.4 is that here the VAR coefficients are estimated along with the GARCH parameters, while there, a two-step process was employed, with the GARCH model being applied only to the residuals. That's not an option here because the M term enters the VAR. So we will have to be able to compute the residuals (including the updated value for SQRTHOIL) as part of the calculation. Much of the setup will be similar to before, but there will be some additions.

```
dec vect[series] y(2)
set y(1) = oilgrow
set y(2) = gdpgrow
*
dec series[symm] vv hhv
gset vv = %sigma
gset hhv = %sigma
*
dec vect[equation] garchmeqns(%nvar)
dec vect[vector] bvec(%nvar)
dec vect[vector] g(%nvar)
do i=1,%nvar
  compute garchmeqns(i)=%modeleqn(basevar,i)
  compute bvec(i)=%eqncoeffs(garchmeqns(i))
  compute g(i)=||%sigma(i,i)*(1-.20-.60),.20,.60||
end do i
```

GARCHMEQNS has the equations for the means, which here have identical forms. As we've done before, the coefficients themselves are organized as a VECT[VECT] with one (inner) VECTOR for each equation. Those are initialized with the OLS VAR estimates. The SQRTHOIL coefficients will be zero, as that's how **ESTIMATE** will handle a redundant (in this case all-zero) regressor.⁵

The following function is now needed—it computes the VECTOR of right-hand-side values for the equations at the current settings for the BVEC coefficients.

⁵In fact, they will have zero coefficients with zero standard errors.

```

function SVARRHSVector time
type vector SVARRHSVector
type integer time
*
dim SVARRHSVector(%nvar)
do i=1,%nvar
    compute SVARRHSVector(i)=%eqnvalue(garchmeqns(i),time,bvec(i))
end do i
end

```

The function for computing H is similar to Section 8.4, except that we can't do any calculation that uses current residuals, since we can't compute them until we have the variance done due to the presence of the M term:

```

function SVARHVMatrix time
type symm SVARHVMatrix
type integer time
*
compute SVARHVMatrix=%zeros(%nvar,%nvar)
do i=1,%nvar
    compute SVARHVMatrix(i,i)=g(i)(1)+$
        g(i)(2)*vv(time-1)(i,i)+g(i)(3)*hhv(time-1)(i,i)
end do i
end

```

Next is the function called at the start of each function evaluation. In this example, we'll do the pre-sample using the stationary variance given the model coefficients. The difference between this and computing it from %SIGMA and the BB matrix as we did in Example 8.3 (and will do again in Example 11.2) is likely to be trivial.

```

compute b=0.0
*
function SVARHVModelStart
dec vect statvar(2)
compute bb=||1.0,0.0|b,1.0||
ewise statvar(i)=g(i)(1)/(1-g(i)(2)-g(i)(3))
gset hhv 1 gstart-1 = %diag(statvar)
gset vv 1 gstart-1 = hhv(t)
end

```

The log likelihood is computed using:

```

frml garchmlogl = hhv=SVARHVMatrix(t),sqrthoil=sqrt(hhv(t)(1,1)), $
    vx=bb*%xt(y,t)-SVARRHSVector(t),vv=%outerxx(vx), $
    %logdensity(hhv,vx)

```

In order, this

Table 11.1: Estimates of SVAR-GARCH-M Model

MAXIMIZE - Estimation by BFGS					
Convergence in 46 Iterations. Final criterion was 0.0000033 <= 0.0000100					
Quarterly Data From 1975:02 To 2013:01					
Usable Observations		152			
Function Value		-1156.3688			
	Variable	Coeff	Std Error	T-Stat	Signif
1.	B	-0.0068	0.0029	-2.3254	0.0201
2.	BVEC(1)(1)	0.4082	0.1083	3.7682	0.0002
3.	BVEC(1)(2)	-0.3801	0.0825	-4.6056	0.0000
4.	BVEC(1)(3)	0.2305	0.0850	2.7108	0.0067
5.	BVEC(1)(4)	-0.1662	0.0702	-2.3666	0.0180
6.	BVEC(1)(5)	0.8743	0.8899	0.9825	0.3258
7.	BVEC(1)(6)	-0.3520	1.0179	-0.3459	0.7295
8.	BVEC(1)(7)	-1.6248	0.8188	-1.9843	0.0472
9.	BVEC(1)(8)	1.4147	0.8210	1.7231	0.0849
10.	BVEC(1)(9)	3.2320	5.8950	0.5483	0.5835
11.	BVEC(1)(10)	0.0000	0.0000	0.0000	0.0000
12.	BVEC(2)(1)	-0.0098	0.0037	-2.6306	0.0085
13.	BVEC(2)(2)	0.0031	0.0037	0.8342	0.4042
14.	BVEC(2)(3)	-0.0075	0.0037	-2.0206	0.0433
15.	BVEC(2)(4)	-0.0024	0.0035	-0.7057	0.4804
16.	BVEC(2)(5)	0.3208	0.0815	3.9340	0.0001
17.	BVEC(2)(6)	0.1629	0.0844	1.9298	0.0536
18.	BVEC(2)(7)	-0.0098	0.0888	-0.1101	0.9123
19.	BVEC(2)(8)	-0.0030	0.0699	-0.0424	0.9662
20.	BVEC(2)(9)	2.8073	0.8940	3.1401	0.0017
21.	BVEC(2)(10)	-0.0240	0.0145	-1.6621	0.0965
22.	G(1)(1)	698.6057	296.8902	2.3531	0.0186
23.	G(1)(2)	0.3327	0.1081	3.0770	0.0021
24.	G(1)(3)	0.4150	0.1539	2.6968	0.0070
25.	G(2)(1)	0.4185	0.2505	1.6707	0.0948
26.	G(2)(2)	0.2361	0.0646	3.6520	0.0003
27.	G(2)(3)	0.7320	0.0712	10.2869	0.0000

1. computes the (diagonal) covariance matrix of the structural residuals.
2. extracts the current value for `SQRTHOIL` using the variance of the first structural component.
3. computes the structural residuals using the “pre-determined” right-side variables and the structural matrix `BB` applied to the dependent variables.
4. computes the outer product of the structural residuals for use in next period’s GARCH recursion.
5. computes the log likelihood assuming Normal residuals.

Note that this last step is only correct because the determinant of `BB` is 1—if it weren’t we would need to allow for the (log) Jacobian of the transformation.

The model estimated in the paper constrains the `M` term on the oil price in the oil (1st) equation to zero. Table 11.1 shows the result of:

```

nonlin b bvec g bvec(1) (meffectpos)=0.0
maximize(start=SVARHVModelStart(),pmethod=simplex,$
    pitters=5, iters=500) garchmlogl gstart gend

```

The M effect is the `BVEC(2)(10)` coefficient, which has the expected sign, though isn't strongly significant. The B has what might be considered an unexpected sign, as that means (once the equation is rearranged) that the effect of current oil price changes on current GDP growth has a positive sign.

The information criterion for the GARCH model can be computed easily, just requiring an adjustment of 1 to the number of parameters because of the M effect that was pegged:

```

compute garchmsic=-2.0*%logl+(%nreg-1)*log(%nobs)
disp "SIC for VAR" varsic
disp "SIC for GARCH-M" garchmsic

```

The GARCH model is clearly superior on that basis:

SIC for VAR	2468.13993
SIC for GARCH-M	2443.35856

11.2 Impulse Responses in an SVAR-GARCH-M Model

If it weren't for the M term, there would be no real difference between computing impulse responses (for the mean) in this model versus a standard VAR other than deciding upon the appropriate size for the shocks. The M effect makes things quite different, however, since the volatility response now feeds into the mean effect as well. As we saw in Section 6.2, because of non-linearity, the volatility response is dependent upon the context in which the shock is made. Because we're interested here in both the size of the shock (for the volatility effects) and the sign (for the mean effects), what we want to calculate is the effect of replacing a zero shock with either a positive or negative "standard size" shock. One complication to computing the response is that the M effect uses the square root rather than just the variance. The incremental effect of one of the shocks on the variance itself is exactly computable, but the effect on the standard deviation will require linearization. The linearization requires an expansion point, and the most obvious choice for that is the variance used in determining the standard size shock.

As promised, Example 11.2 will use the B-model form similar to what was done in Section 8.4. The first part of the programs (reading data, setting up the VAR, initializing the VAR and GARCH coefficients) are the same. We start to split off when we define the loading matrix from the structural shocks to residuals:

```

dec frml[rect] bfrml
dec real b
*
frml bfrml = ||$
    1.0,0.0|$
    b ,1.0||
compute b=0.0
*
nonlin(parmset=svarparms) bvec b
nonlin(parmset=garchparms) g

```

As before, we need to save series of the outer products of the structural shocks (the v), the vectors of structural variances, and the full covariance matrix of the VAR residuals.

```

dec series[symm] vv
dec series[vect] hhd
dec series[symm] hh
dec rect binv
*
gset hhd 1 gend = %zeros(n,1)
gset hh 1 gend = %zeros(n,n)
gset vv 1 gend = %zeros(n,n)

```

The **SVARStart** function is identical to the one in Example 8.3:

```

function SVARStart
local symm vsigma
*
compute bmat=bfrml(1)
compute binv=inv(bmat)
compute vsigma=binv*%sigma*tr(binv)
gset hhd 1 gstart-1 = %xdiag(vsigma)
gset vv 1 gstart-1 = vsigma
end

```

The **SVARHMatrix** function is slightly different because we can't do anything using the current residuals yet, since we don't know what they are and can't compute them until we have the variances calculated. So it just updates the **HHD** and **HH**:

```

function SVARHMatrix t
type symm SVARHMatrix
type integer t
*
local integer i
*
do i=1,n
    compute hhd(t)(i)=$
        g(i)(1)+g(i)(2)*vv(t-1)(i,i)+g(i)(3)*hhd(t-1)(i)
end do i
*
compute hh(t)=bmat*%diag(hhd(t))*tr(bmat)
compute SVARHMatrix=hh(t)
end

```

What we need now, and didn't before, is a separate function which calculates the residuals.⁶ This also computes (into $VV(T)$) the outer product of the structural residuals for use in next period's GARCH recursion.

```

function SVARUVector t
type vect SVARUVector
type integer t
*
local integer i
*
do i=1,%nvar
    compute u(i)(t) = y(i)(t) - %eqnvalue(garchmeqns(i), t, bvec(i))
end do i
compute vv(t)=%outerxx(binv*%xt(u,t))
compute SVARUVector=%xt(u,t)
end

```

The sequence of calculations for the log likelihood is:

```

frml SVARLogl = hh=SVARHMatrix(t), sqrthoil=sqrt(hh(t)(1,1)), $
    ux=SVARUVector(t), %logdensity(hh,ux)

```

The H matrix is computed first, the current value for `SQRTHOIL` is computed using that, and then the residual vector. The log likelihood (assuming multivariate Normals) is calculated in the standard way, since we aren't using transformed data at this point.

For this paper, we need to estimate the model both with and without the M terms to illustrate how the variance terms enter the responses. This estimates with the M coefficients shut down:

⁶Example 8.3 took the residuals from an OLS VAR as the input to the GARCH model.


```

compute allparms=garchparms+svarparms
nonlin(parmset=noMeffect) bvec(1)(meffectpos)=0 $
                                bvec(2)(meffectpos)=0
*
maximize(start=SVARStart(),parmset=allparms+noMeffect,$
          pitters=10,pmethod=simplex,itors=500) SVARLog1 gstart gend
compute noMParms=%parmspeek(allparms)

```

The final line saves the parameters into a VECTOR called `noMParms`. `%PARMSPEEK`, and its “inverse” `%PARMSPOKE`, will be used quite a bit to keep the full set of parameters organized. This is much simpler than trying to manage the three separate parts (the lag coefficients, the GARCH coefficients and the structural model coefficient).

Estimates *with* the M effects are done with:

```

maximize(start=SVARStart(),parmset=allparms,$
          pitters=10,pmethod=simplex,itors=500) SVARLog1 gstart gend
compute withMParms=%parmspeek(allparms)

```

We’re varying from the model in the article by allowing the M effect in *both* equations. With the longer sample and change in the timing, the coefficient on the standard deviation of oil in the oil equation is quite significant and positive.

As mentioned before, if it weren’t for the M effect, we could use standard methods to compute the impulse response function. The VAR coefficients are in `BVEC`; we can transfer them into the `BASEVAR` model with

```

compute varcoeffs=%modelgetcoeffs(basevar)
ewise varcoeffs(i,j)=bvec(j)(i)
compute %modelsetcoeffs(basevar,varcoeffs)

```

`%MODELSETCOEFFS` and `%MODELGETCOEFFS` move coefficient matrices into and out of the model framework—since we’re using a `VECT[VECT]` structure for organizing the lag coefficients, we need to copy them into a `RECT` matrix first. The first line is there just to give `VARCOEFFS` the correct dimensions so the **EWISE** will work correctly. With the coefficients put into the model, we can use **IMPULSE** to compute the responses. The only question is what shock to use since there’s no fixed covariance matrix to provide a natural size and shape.

However, *with* the M effect, there’s no good way to avoid having to do the impulse response calculations “by hand”. In the program, the calculations of the impulse response function are organized into the procedure **CalcIRF** which takes the initial shock (vector) as its only parameter. Because we need to be able to compute the IRF’s for many different sets of parameters (unconstrained ML, constrained ML and random draws), **CalcIRF** assumes only that the desired parameters have been copied into their locations in the parameter sets.

To compute the IRF’s, it’s necessary to split the coefficients of the VAR up into lag matrices. That’s done with:

```

ewise varcoeffs(i,j)=bvec(j) (i)
compute %modelsetcoeffs (basevar,varcoeffs)
dec vect[rect] phi(nlags)
ewise phi(i)=%modellagmatrix(basevar,i)

```

`%MODELLAGMATRIX(model, lag)` extracts the $n \times n$ matrix of coefficients for the endogenous variables (only) at the indicated lag. To get the loadings from the standard deviation onto the variables, we can use

```

dec vector pi0(%nvar)
ewise pi0(i)=bvec(i) (%size(bvec(i)))

```

which extracts the final component of each coefficient vector, since that's where we positioned the M coefficient.⁷

We also must evaluate the B matrix for the current parameter set:

```

compute bmat=bfrml(1)

```

We need to compute both impulse responses and volatility responses. For the latter, we only need the responses in the oil shock component, but the impulse responses themselves need to be computed for the pair of variables. For a horizon of `NSTEPS` (+1 for the initial shock), we set these up (*outside* the procedure) with:

```

dec vect[vect] irf(nsteps+1)
dec vect hirf(nsteps+1)

```

The impact responses go into the first slot in these. For (structural) shocks given by the `VECTOR EPS0`, the impacts are

```

compute irf(1) =bmat*eps0
compute hirf(1)=0.0

```

The impact on the variance is zero in the first period since the variance is only affected with a lag. We then do the following (pseude-code) loop:

```

do i=1,nsteps
  << compute IRF(i+1) using just the VAR coefficients
    and IRF(1), ..., IRF(i)>>
  << compute HIRF(i+1) using HIRF(i) and
    the GARCH coefficients >>
  << add the (linearized) impatch of HIRF(i+1) to
    IRF(i+1) >>
end do i

```

⁷`%SIZE` returns the number of elements when it's applied to a `VECTOR`. `%ROWS` could also be used here.

The first calculation requires a loop over the preceding steps:

```
compute irf(i+1)=%zeros(%nvar,1)
do k=1,%imin(i,nlags)
    compute irf(i+1)=irf(i+1)+phi(k)*irf(i+1-k)
end do k
```

The %IMIN on the loop index is to make sure we don't try to use "pre-shock" information.

The VIRF calculation is much simpler since it uses only a one-period lag. We do, however, need to treat the case when $T=1$ differently, since that's where the initial shock actually comes into play. Again, this is specific to the M effect only operating through the first structural component.

```
compute hirf(i+1)=g(1)(3)*hirf(i)+$
                g(1)(2)*%if(i==1,eps0(1)^2,hirf(i))
```

The adjustment for the (mean) response to the variance response is done with:

```
compute irf(i+1)=irf(i+1)+pi0*.5/sqrt(meanhoil)*hirf(i+1)
```

where MEANHOIL is the expansion point for the linearization. In the paper, the authors use for MEANHOIL the unconditional sample variance for the oil growth series. Since it's being used as the expansion point for the GARCH recursion, we will use the sample average of the GARCH variance (from the maximum likelihood estimates of the full model):

```
sstats(mean) gstart gend hh(t)(1,1)>>meanhoil
```

The square root of MEANHOIL is used as the shock size. Because the M effect depends only upon the magnitude and not the sign, the responses to positive and negative shocks *aren't* mirror images of each other.

The first "experiment" is positive and negative shocks to the parameter set with the M terms included. The shock is only to the oil price growth (only the first component in the shock vector is non-zero), and we only are interested in the response of GDP growth (taking just the (2) component out of $IRF(T)$):

```
compute shock=sqrt(meanhoil)
compute %parmspoke(allparms,withMParms)
@CalcIRF ||shock,0.0||
set gdptoplus 1 nsteps+1 = irf(t)(2)
@CalcIRF ||-shock,0.0||
set gdptominus 1 nsteps+1 = irf(t)(2)
```

By comparison, we can do the responses using the VAR without the M terms with:

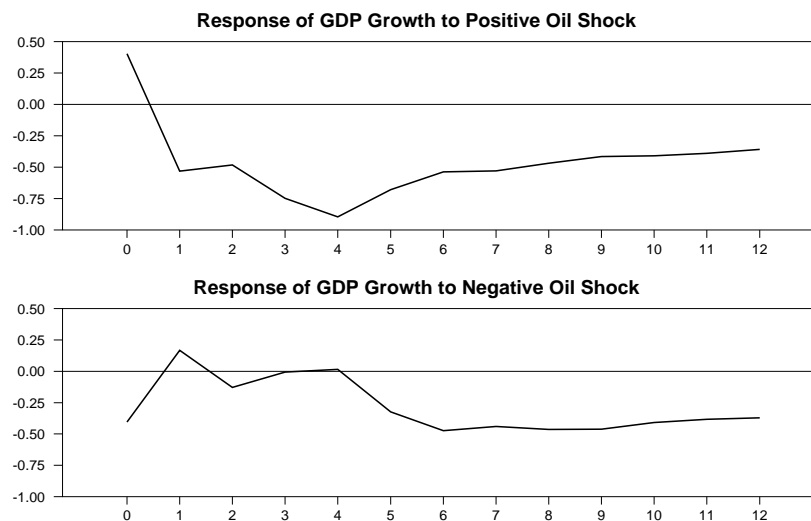


Figure 11.1: GDP Responses to Oil Shocks

```
compute %parmspoke(allparms,noMParms)
@CalcIRF ||shock,0.0||
set gdptoplusnom 1 nsteps+1 = irf(t) (2)
@CalcIRF ||-shock,0.0||
set gdptominusnom 1 nsteps+1 = irf(t) (2)
```

These are organized into two two-pane graphs. The first compares the response of positive shock to the negative one in the full model (Figure 11.1)⁸

```
spgraph(vfields=2)
graph(number=0,min=-1.0,max=0.5,$
  header="Response of GDP Growth to Positive Oil Shock")
# gdptoplus      1 nsteps+1
graph(number=0,min=-1.0,max=0.5,$
  header="Response of GDP Growth to Negative Oil Shock")
# gdptominus     1 nsteps+1
spgraph(done)
```

One thing to note is that the shock size is quite large (56.5). Although that's an annualized growth rate, that still means quarter to quarter changes of around 14% on average. An interesting question is whether the "structural" part of the SVAR may be incorrect. It may be that, in the short-term (reflected in the contemporaneous relationship) GDP growth drives the oil price more than the reverse. Switching the order isn't quite as simple as it is with a standard VAR, as one would have to decide whether the M effect would be on the variance of \mathbf{u} or of \mathbf{v} . With oil price ordered first, the two are the same, but not if it's second.

⁸The MIN and MAX options are specific to this example, and were added after looking at the graphs.

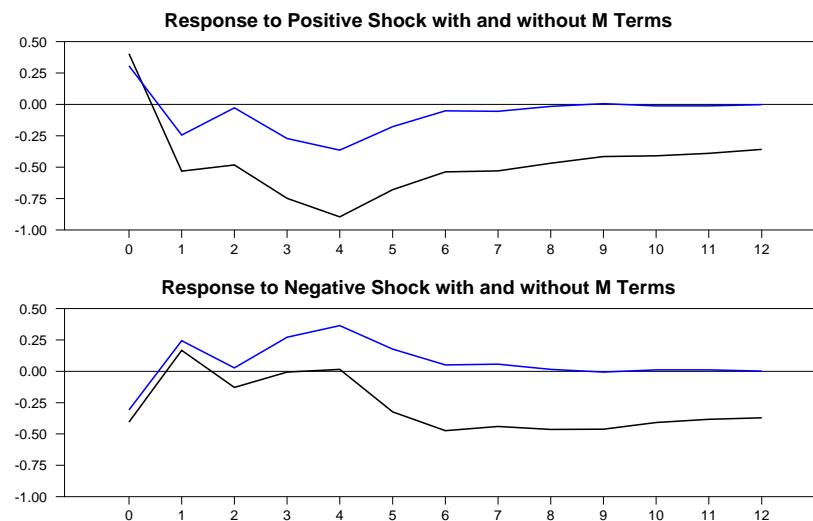


Figure 11.2: Responses to Oil Shocks With and Without M

At any rate, the “positive” effect of a negative oil shock is very short-lived, and the effect of the variance dominates both the positive and negative IRF’s after a year.

The second looks at the two shocks in the models with and without the M terms (Figure 11.2):

```
spgraph(vfields=2)
graph(number=0,min=-1.0,max=0.5,$
  header="Response to Positive Shock with and without M Terms") 2
# gdptoplus      1 nsteps+1
# gdptoplusnom   1 nsteps+1
graph(number=0,min=-1.0,max=0.5,$
  header="Response to Negative Shock with and without M Terms") 2
# gdptominus     1 nsteps+1
# gdptominusnom  1 nsteps+1
spgraph(done)
```

The “without” responses are sign flips of each other by construction. As constructed, the full model seems to show an almost permanent 1/4 point (annualized) hit to GDP growth from price shocks in either direction.

11.2.1 Error Bands

We’ll compute error bands using Random Walk Metropolis. With standard IRF’s, the impact responses are randomized as well (as you get new draws for the covariance matrix), but here we will continue to use the same fixed size shock. We want to keep track of responses to both negative and positive shocks, so we will use the standard bookkeeping for impulse responses (using the %%RESPONSES arrays) as described in Chapter 16 of the *RATS User’s Guide*:

```

compute nburn=2500,nkeep=10000
declare vect[rect] %%responses
dim %%responses(nkeep)

```

Because the data set is so short (compared to typical finance data), we can do a large number of draws in a relatively short time. We'll start the sampler at the maximum likelihood estimates (which is the last one we did), and use multivariate Normal increments using a scale multiple of the (Cholesky factor of the) estimated covariance matrix. The %PARMSPOKE function makes sure that we start out with the correct set of parameters since we reset them in computing the responses earlier. The .5 gives an acceptance rate in the low 20's, which is reasonable, particularly given the size of the parameter set.

```

compute logplast=%logl
compute %parmspoke(allparms,withMParms)
compute fxx      =.5*%decomp(%xx)
compute accept   =0

```

The sampling procedure (inside the draw loop) is actually fairly simple. We get a test set of parameters by adding an increment to the last set:

```

compute blast=%parmspeek(allparms)
compute btest=blast+%ranmvnormal(fxx)

```

We “poke” that back into ALLPARMS so we evaluate the formulas at these values.

```

compute %parmspoke(allparms,btest)

```

You don't even need to use **MAXIMIZE** to get the value of the likelihood, just:

```

compute SVARStart()
sstats gstart gend SVARLogl(t)>>logptest

```

which resets the function start information, then sums up the values of the log likelihood, producing the value LOGPTEST.

The Metropolis test is done with

```

compute alpha=%if(%valid(logptest).and.%nobs==gnobs,$
                  exp(logptest-logplast),0.0)
if %ranflip(alpha) {
  compute accept =accept+1
  compute logplast=logptest
}
else {
  compute %parmspoke(allparms,blast)
}

```

The added step of checking whether the **SSTATS** instruction uses the full set of observations (the `%NOBS==GNOBS` clause in the `%IF`) is needed because the a setting of parameters could go explosive which produces NA values, which are simply skipped by **SSTATS**). If we don't accept the new parameters, we reset the parameters to the **BLAST** values—if we *do* accept, the **BTEST** values are already in **ALLPARMS**).

The most complicated part of the process is the bookkeeping on the responses. In the typical VAR, the responses are an $n \times n$ collection of series, with n target variables and n structural shocks. Here, we aren't interested in the responses to the two structural shocks that the model defines, but instead want two separately defined shocks. To get the input correct for **@MCGraphIRF**, we have to put the “negative” shocks into the locations where the second structural shocks would go. That's what this is doing:

```
compute shock=sqrt(meanhoil)
*
* Interleave results for the positive and negative shocks.
*
dim %%responses(draw) (2*2,nsteps+1)
@CalcIRF ||shock,0.0||
do j=1,nsteps+1
    compute %%responses(draw) (1,j)=irf(j) (1)
    compute %%responses(draw) (2,j)=irf(j) (2)
end do j
@CalcIRF ||-shock,0.0||
do j=1,nsteps+1
    compute %%responses(draw) (3,j)=irf(j) (1)
    compute %%responses(draw) (4,j)=irf(j) (2)
end do j
```

`%%RESPONSES` *must* include responses for all variables, though in actually graphing, we will only include the GDP responses, which is we save the responses of both the first and second variables. The responses are grouped by shock, which is why we use 1 and 2 for the positive shock and 3 and 4 for the negative one.

The final act, once we're done with the loop is to do the **@MCGraphIRF** (Figure 11.3).

```
@mcgraphirf(model=basevar,include=||2||,center=median,$
    shocklabels=||"Positive Oil Shock","Negative Oil Shock"||)
```

If you don't like the look of the labeling (which is designed for a much larger matrix of graphs than this), you can use **@MCProcessIRF** instead to do the calculations of the bands, then do the graphs that you would like.

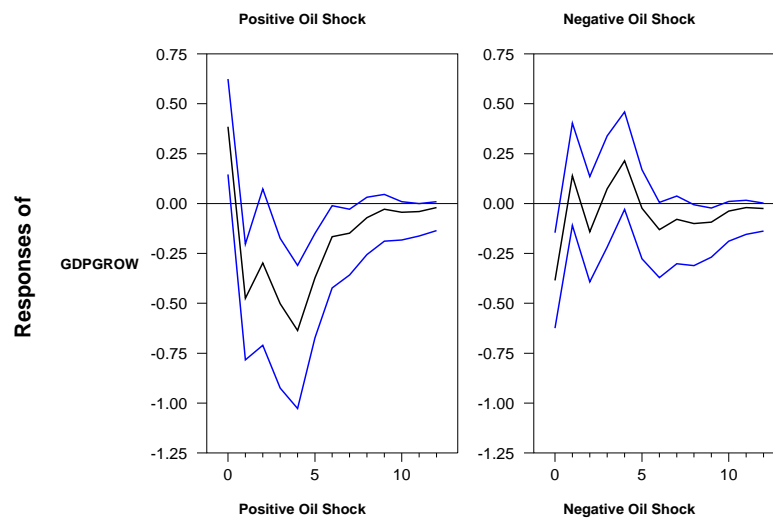


Figure 11.3: Responses to Oil Shocks, with Error Bands

Example 11.1 SVAR-GARCH-M, Estimation

This estimates the model used in Elder & Serletis (2010) with a lengthened and re-defined data set. This is discussed in detail in Section 11.1.

```
open data oildata.rat
calendar(q) 1974:1
data(format=rats,compact=average) 1974:01 2013:01 ipdm rac gdpmc1
*
* The difference in data preparation is that Elder and Serletis use
* the final period within the quarter for the (monthly source) RAC,
* while this uses the quarterly average. The quarterly average
* matches the timing on GDP and the deflator.
*
* Data are annualized growth rates
*
set realoil = rac/ipdm
set oilgrow = 400.0*log(realoil/realoil{1})
set gdpgrow = 400.0*log(gdpmc1/gdpmc1{1})
*
@varlagselect(crit=aic,lags=8)
# oilgrow gdpgrow
*
compute nlags=4
set sqrthoil = 0.0
*
* Set up the VAR including the square root of oil (which will be
* generated recursively) as one of the regressors.
*
system(model=basevar)
variables oilgrow gdpgrow
lags 1 to nlags
det constant sqrthoil
```



```

end(system)
*
* This is the position of the "M" effect coefficient
*
compute meffectpos=2*nlags+2
*
* Estimate the OLS VAR
*
estimate(resids=u)
*
* This is the SIC (Schwarz Information Criterion) including the
* coefficients in the estimated covariance matrix, but deleting the
* unnecessary (at this point) coefficients on the standard
* deviation of oil.
*
compute varparms=%nfree-%nvar
compute varsic=-2.0*%logl+varparms*log(%nobs)
compute gstart=%regstart(),gend=%regend()
*
dec vect[series] y(2)
set y(1) = oilgrow
set y(2) = gdpgrow
*
dec series[symm] vv hhv
gset vv = %sigma
gset hhv = %sigma
*
dec vect[equation] garchmeqns(%nvar)
dec vect[vect] bvec(%nvar)
dec vect[vect] g(%nvar)
do i=1,%nvar
    compute garchmeqns(i)=%modeleqn(basevar,i)
    compute bvec(i)=%eqncoeffs(garchmeqns(i))
    compute g(i)=||%sigma(i,i)*(1-.20-.60),.20,.60||
end do i
*****
function SVARRHSVector time
type vector SVARRHSVector
type integer time
*
dim SVARRHSVector(%nvar)
do i=1,%nvar
    compute SVARRHSVector(i)=%eqnvalue(garchmeqns(i),time,bvec(i))
end do i
end
*****
function SVARHVMatrix time
type symm SVARHVMatrix
type integer time
*
compute SVARHVMatrix=%zeros(%nvar,%nvar)
do i=1,%nvar
    compute SVARHVMatrix(i,i)=g(i)(1)+$
        g(i)(2)*vv(time-1)(i,i)+g(i)(3)*hhv(time-1)(i,i)

```

```

end do i
end
*****
*
* Rotation coefficient. The log det of the Jacobian for
*  $||1.0, 0.0|b, 1.0||$  is zero, so we don't need the added term for that.
*
compute b=0.0
*
function SVARHVModelStart
dec vect statvar(2)
compute bb= $||1.0, 0.0|b, 1.0||$ 
ewise statvar(i)=g(i) (1) / (1-g(i) (2)-g(i) (3))
gset hhv 1 gstart-1 = %diag(statvar)
gset vv 1 gstart-1 = hhv(t)
end
*
frml garchmlogl = hhv=SVARHVMatrix(t), sqrthoil=sqrt(hhv(t) (1,1)), $
    vx=bb*%xt(y,t)-SVARRHSVector(t), vv=%outerxx(vx), $
    %logdensity(hhv, vx)
*
* The model estimated in the paper constrains the "M" term on the oil
* price in the oil (1st) equation to zero.
*
nonlin b bvec g bvec(1) (meffectpos)=0.0
maximize(start=SVARHVModelStart(), pmethod=simplex, $
    pitters=5, iters=500) garchmlogl gstart gend
*
* Correct %NREG for the one pegged zero regressor
*
compute garchmsic=-2.0*%logl+(%nreg-1)*log(%nobs)
disp "SIC for VAR" varsic
disp "SIC for GARCH-M" garchmsic

```

Example 11.2 SVAR-GARCH, Impulse Responses

This does impulse response analysis for the SVAR “B” model equivalent to Example 11.1, including calculation of error bands. This is covered in Section 11.2.

```
open data oildata.rat
calendar(q) 1974:1
data(format=rats,compact=average) 1974:01 2013:01 ipdm rac gdpmc1
*
* The difference in data preparation is that Elder and Serletis use
* the final period within the quarter for the (monthly source) RAC,
* while this uses the quarterly average. The quarterly average
* matches the timing on GDP and the deflator.
*
* Data are annualized growth rates
*
set realoil = rac/ipdm
set oilgrow = 400.0*log(realoil/realoil{1})
set gdpgrow = 400.0*log(gdpmc1/gdpmc1{1})
*
compute nlags=4
set sqrthoil = 0.0
*
* Set up the VAR including the square root of oil (which will be
* generated recursively) as one of the regressors.
*
system(model=basevar)
variables oilgrow gdpgrow
lags 1 to nlags
det constant sqrthoil
end(system)
*
* This is the position of the "M" effect coefficient
*
compute meffectpos=2*nlags+2
*
compute n=2
dec vect[series] y(n)
set y(1) = oilgrow
set y(2) = gdpgrow
*
* Estimate the OLS VAR
*
estimate(resids=u)
compute gstart=%regstart(),gend=%regend()
*
* Parameters for the GARCH processes for the orthogonal V
*
dec vect[vect] g(n)
dec vect[vect] bvec(n)
dec vect[equation] garchmeqns(n)
do i=1,n
```

```

        compute garchmeqns(i)=%modeleqn(basevar,i)
        compute bvec(i)=%eqncoeffs(garchmeqns(i))
        compute g(i)=||sigma(i,i)*(1-.20-.60),.20,.60||
    end do i
*****
*
* Through here is the same as 11.1
*
*****
*
* B-matrix loading for the SVAR
*
dec frml[rect] bfrml
dec real b
*
frml bfrml = ||$
    1.0,0.0|$
    b ,1.0||
compute b=0.0
*
nonlin(parmset=svarparms) bvec b
nonlin(parmset=garchparms) g
*
* Series for recursions on the variances
*
dec series[symm] vv
dec series[vect] hhd
dec series[symm] hh
dec rect binv
*
gset hhd 1 gend = %zeros(n,1)
gset hh 1 gend = %zeros(n,n)
gset vv 1 gend = %zeros(n,n)
*****
*
* This is a "startup" function which needs to be executed at the
* start of a function evaluation. The B matrix and its inverse
* change with the parameters, but not with time. And we need to get
* the transformed pre-sample values.
*
function SVARStart
local symm vsigma
*
compute bmat=bfrml(1)
compute binv=inv(bmat)
compute vsigma=binv*%sigma*tr(binv)
gset hhd 1 gstart-1 = %xdiag(vsigma)
gset vv 1 gstart-1 = vsigma
end
*****
*
* This is the function evaluated at each entry to do the recursion
* in the variance of the structural residuals, then maps it back to
* the standard residuals.

```

```

*
function SVARHMatrix t
type symm SVARHMatrix
type integer t
*
local integer i
*
do i=1,n
    compute hhd(t)(i)=$
        g(i)(1)+g(i)(2)*vv(t-1)(i,i)+g(i)(3)*hhd(t-1)(i)
end do i
*
compute hh(t)=bmat*%diag(hhd(t))*tr(bmat)
compute SVARHMatrix=hh(t)
end
*****
*
* This is the function evaluated at each entry to compute the
* residuals. It also saves the outer product of the structural
* residuals for use in the GARCH recursion.
*
function SVARUVector t
type vect SVARUVector
type integer t
*
local integer i
*
do i=1,%nvar
    compute u(i)(t) = y(i)(t)-%eqnvalue(garchmeqns(i),t,bvec(i))
end do i
compute vv(t)=%outerxx(binv*%xt(u,t))
compute SVARUVector=%xt(u,t)
end
*****
*
* This computes the logl likelihood by computing the covariance
* matrix, then the square root of the oil shock variance, then the
* residuals. It has to be done in that order since current sqrthoil
* is needed in the formula for the residuals.
*
frml SVARLogl = hh=SVARHMatrix(t),sqrthoil=sqrt(hh(t)(1,1)), $
    ux=SVARUVector(t),%logdensity(hh,ux)
*
* Estimate first with the sqrt(H) terms constrained to zero
*
compute allparms=garchparms+svarparms
nonlin(parmset=noMeffect) bvec(1)(meffectpos)=0 $
    bvec(2)(meffectpos)=0
*
maximize(start=SVARStart(),parmset=allparms+noMeffect,$
    pitters=10,pmethod=simplex,itors=500) SVARLogl gstart gend
compute noMParms=%parmspeek(allparms)
*
* Then with the sqrt(H) terms unconstrained

```

```

*
maximize(start=SVARStart(),parmset=allparms,$
  pitters=10,pmethod=simplex,itors=500) SVARLog1 gstart gend
compute withMParms=%parmspeek(allparms)
*****
*
* Responses to replacing 0 oil shock with historically typical size.
* (Because of the non-linearity, responses don't scale).
*
* Get the mean level for the conditional variance of oil.
*
sstats(mean) gstart gend hh(t) (1,1)>>meanhoil
*
* Get locations prepared for doing the IRF's
*
compute varcoeffs=%modelgetcoeffs(basevar)
dec vect[rect] phi(nlags)
dec vector pi0(%nvar)
*
compute nsteps=12
*
* Because the "M" term in the GARCH-M is the square root rather
* than the variance, the partial derivative depends upon the level
* of the variance. We'll linearize the derivative about the average
* conditional variance, which is also being used for the typical
* shock size.
*
dec vect[vect] irf(nsteps+1)
dec vect hirt(nsteps+1)
*****
procedure CalcIRF eps0
type vector eps0
*
local integer i j k
*
* Copy the VAR coefficients into the model
*
ewise varcoeffs(i,j)=bvec(j)(i)
compute %modelsetcoeffs(basevar,varcoeffs)
*
* Extract the coefficients from the VAR by lag.
*
ewise phi(i)=%modellagmatrix(basevar,i)
*
* Extract the VAR loadings on the conditional standard deviation of
* oil (final entry in the BVEC's).
*
ewise pi0(i)=bvec(i)(%size(bvec(i)))
*
compute bmat=bfrml(1)
*
* eps0 is the standardized shock size. The impact response will be
* BMAT times that.
*

```

```

compute irf(1) =bmat*eps0
compute hirtf(1)=0.0
*
do i=1,nsteps
  *
  * Take care of the standard VAR terms.
  *
  compute irf(i+1)=%zeros(%nvar,1)
  do k=1,%imin(i,nlags)
    compute irf(i+1)=irf(i+1)+phi(k)*irf(i+1-k)
  end do k
  *
  * Take care of the ARCH terms. This uses the square of eps0 for
  * the lagged squared residual for step 1, and feeds through the
  * previous forecasted variance for steps higher than that.
  *
  * Note, this is specific to the case where M effects only
  * originate in shock 1.
  *
  compute hirtf(i+1)=g(1) (3)*hirtf(i)+$
                    g(1) (2)*%if(i==1,eps0(1)^2,hirtf(i))
  *
  * Include the effect of the "M" term. This is where the
  * linearization comes in.
  *
  compute irf(i+1)=irf(i+1)+pi0*.5/sqrt(meanhoil)*hirtf(i+1)
end do i
end
*****
*
* Compute responses (of GDP, the second variable) to positive and
* negative average-sized shocks in oil (the first variable) for the
* model with M effects
*
compute shock=sqrt(meanhoil)
compute %parmspoke(allparms,withMParms)
@CalcIRF ||shock,0.0||
set gdptoplus 1 nsteps+1 = irf(t) (2)
@CalcIRF ||-shock,0.0||
set gdptominus 1 nsteps+1 = irf(t) (2)
*
* And without M effects
*
compute %parmspoke(allparms,noMParms)
@CalcIRF ||shock,0.0||
set gdptoplusnom 1 nsteps+1 = irf(t) (2)
@CalcIRF ||-shock,0.0||
set gdptominusnom 1 nsteps+1 = irf(t) (2)
*
spgraph(vfields=2)
graph(number=0,min=-1.0,max=0.5,$
  header="Response of GDP Growth to Positive Oil Shock")
# gdptoplus      1 nsteps+1
graph(number=0,min=-1.0,max=0.5,$

```

```

    header="Response of GDP Growth to Negative Oil Shock")
# gdptominus      1 nsteps+1
spgraph(done)
*
spgraph(vfields=2)
graph(number=0,min=-1.0,max=0.5,$
    header="Response to Positive Shock with and without M Terms") 2
# gdptoplus      1 nsteps+1
# gdptoplusnom   1 nsteps+1
graph(number=0,min=-1.0,max=0.5,$
    header="Response to Negative Shock with and without M Terms") 2
# gdptominus     1 nsteps+1
# gdptominusnom  1 nsteps+1
spgraph(done)
*****
*
* Get error bands using Random Walk Metropolis
*
compute nburn=2500,nkeep=10000
declare vect[rect] %%responses
dim %%responses(nkeep)
*
compute logplast=%logl
compute %parmspoke(allparms,withMParms)
compute fxx      =.25*%decomp(%xx)
compute accept   =0
compute gnobs    =gend-gstart+1
*
infobox(action=define,progress,lower=-nburn,upper=nkeep) $
    "Random Walk Metropolis"
do draw=-nburn,nkeep
    *
    * Use the asymptotic Normal for the estimates as the proposal
    * density.
    *
    compute blast=%parmspeek(allparms)
    compute btest=blast+%ranmvnormal(fxx)
    *
    * "poke" the draw into the parameter set.
    *
    compute %parmspoke(allparms,btest)
    *
    * Compute the log likelihood for the model given the new
    * coefficients and test the Metropolis condition.
    *
    compute SVARStart()
    sstats gstart gend SVARLogl(t)>>logptest
    compute alpha=%if(%valid(logptest).and.%nobs==gnobs,$
        exp(logptest-logplast),0.0)
    if %ranflip(alpha) {
        compute accept =accept+1
        compute logplast=logptest
    }
    else {

```



```

        compute %parmspoke(allparms,blast)
    }
    infobox(current=draw) $
        "Acceptance Rate "+%strval(100.0*accept/(draw+nburn+1),"###.##")
    *
    * What's left is only needed on keeper draws
    *
    if draw<=0
        next
    compute shock=sqrt(meanhoil)
    *
    * Interleave results for the positive and negative shocks.
    *
    dim %%responses(draw) (2*2,nsteps+1)
    @CalcIRF ||shock,0.0||
    do j=1,nsteps+1
        compute %%responses(draw) (1,j)=irf(j) (1)
        compute %%responses(draw) (2,j)=irf(j) (2)
    end do j
    @CalcIRF ||-shock,0.0||
    do j=1,nsteps+1
        compute %%responses(draw) (3,j)=irf(j) (1)
        compute %%responses(draw) (4,j)=irf(j) (2)
    end do j
end do draw
infobox(action=remove)
@mcgraphirf(model=basevar,include=||2||,center=median,$
    shocklabels=||"Positive Oil Shock","Negative Oil Shock"||)

```

Stochastic volatility models

Stochastic Volatility Models are a conceptually simple class of model for volatility which are, however, technically quite complicated. The basic model is

$$\log h_t = \gamma + \varphi \log h_{t-1} + w_t \quad (12.1)$$

$$y_t = \sqrt{h_t} v_t \quad (12.2)$$

$$v_t \sim N(0, 1) \quad (12.3)$$

An important special case has $\gamma = 0$ and $\varphi = 1$, when $\log h_t$ follows a random walk. w_t and v_t are assumed to be jointly and serially independent.

The observation equation ((12.2) combined with (12.3)) is basically the same as for an ARCH or GARCH model. We've simplified it here to remove the “mean” model, but that could easily be included. The difference is with the evolution of the variance (12.1). In the stochastic volatility model, this is an exogenous process—large residuals tend to cluster because the variance process randomly moves between higher and lower variance regions and the larger residuals are created by that. By contrast, in the ARCH and GARCH models, the variance is driven up by large residuals, which then tend to produce more large residuals.

The technical problems come about because the variance isn't “observable”—in the GARCH model, the variance is an exact recursive function of the data (other than the somewhat arbitrary initial conditions), so the log likelihood can be computed recursively and maximized. While models with latent variables like $\log h$ are common (state-space and factor models, for instance), the form of the interaction between the two equations doesn't easily fit into any standard setting.

There are a number of methods of estimating the free parameters of this (γ , φ and the variance of w), most of which use (rather complicated) simulations. A simpler technique is to approximate it with a standard state-space model.

12.1 State-Space Approximation

If we square and take logs of both sides of (12.2), we get

$$\log y_t^2 = \log h_t + \log v_t^2$$

$\log y_t^2$ is a perfectly good observable (aside from not being defined when $y_t = 0$). Combined with (12.1), we have a state-space model with $\log h_t$ as the only state.

Table 12.1: E-GARCH Model for Exchange Rate

GARCH Model - Estimation by BFGS					
Convergence in 26 Iterations. Final criterion was 0.0000000 <= 0.0000100					
Dependent Variable DLOGP					
Usable Observations		945			
Log Likelihood		-935.0288			
	Variable	Coeff	Std Error	T-Stat	Signif
1.	C	-0.1663	0.0334	-4.9793	0.0000
2.	A	0.1948	0.0374	5.2120	0.0000
3.	B	0.9802	0.0099	99.0250	0.0000

However, the measurement error in this ($\log v_t^2$) isn't Normal, and isn't even mean zero. It is, however, a known (though not heavily used) distribution: the $\log \chi^2$, which is a special case of the extreme value or Gumbel distribution. This has mean $\psi(1/2) + \log(2) \approx -1.27$ and variance $\pi^2/2$.¹ We can approximate the stochastic volatility model with an observation equation of

$$\log y_t^2 - E(\log \chi^2) = \log h_t + \tilde{v}_t; \tilde{v}_t \sim N(0, \pi^2/2) \quad (12.4)$$

Kalman filtering and smoothing with the combination of this and (12.1) will get the mean and variance correct, just not the higher moments that are due to the non-normal errors. This is an example of Quasi-Maximum Likelihood Estimation (QMLE)—estimating the model with a convenient likelihood function which you know (or at least suspect) to be not the truth. If you're unfamiliar with this, you can read more about it in Appendix B.

Example 12.1 is based upon Harvey et al. (1994). This looks at the US/UK exchange rate. The raw data are the exchange rates themselves, which are transformed to log differences² and de-meaned. Because there's effectively zero chance that the de-meaned data will have any exact zero values, we won't be concerned about the possibility of a observation entry being undefined.

```
set dlogp = 100.0*log(usxuk{0}/usxuk{1})
diff(center) dlogp / demean
```

The GARCH model closest to the SV model is an EGARCH, so we estimate that with:

```
garch(exp,nomean,hseries=hgarch) / demean
```

The output is in Table 12.1.

The mean and variance of the $\log \chi^2$ can be computed exactly with:

```
compute meanx2=%digamma(0.5)-log(0.5)
compute varx2 =%trigamma(0.5)
```

¹ $\psi(x)$ is the *digamma* function, computable in RATS using %DIGAMMA(x).

²We include the recommended scaling up by 100, which isn't done in the paper.

Table 12.2: ARMA Estimates for Guess Values

	Variable	Coeff	Std Error	T-Stat	Signif
1.	CONSTANT	-0.9393	0.2806	-3.3478	0.0008
2.	AR{1}	0.9878	0.0107	92.5506	0.0000
3.	MA{1}	-0.9539	0.0189	-50.3753	0.0000

In applications, the mean is often approximated as -1.27, but we'll use the exact value. The following transforms `YSQ` to remove the mean from the error process:

```
set ysq = log(demean^2)-meanx2
```

The γ and φ parameters are very highly correlated when φ is near 1 (which we would expect given the EGARCH estimates). This makes estimation by general hill-climbing somewhat complicated. Instead, we can reparameterize this to use $\gamma^* = \gamma/(1 - \varphi)$, which makes γ^* the mean of the log h process.

We can get guess values by using the fact that (12.4) combined with (12.1) means that `YSQ` will be an ARMA(1,1) process, as it's an AR(1) plus uncorrelated white noise.³

```
boxjenk(ar=1,ma=1,constant) ysq
```

The coefficients from the estimates are in Table 12.2, though the calculations of the guesses can be done just using the variables defined by `BOXJENK`.

The φ is read off as the AR coefficient and the γ^* as the `CONSTANT`. The variance of the MA part is $\sigma_w^2 + (1 - \varphi^2)\sigma_v^2$ and the first lag covariance is $-\varphi\sigma_v^2$. However, we know that $\sigma_v^2 = \pi^2$ and we can get φ from the AR coefficient, so we just need to back out σ_w^2 . Since the ratio of the first lag covariance to the variance for an MA process is $\theta/(1 + \theta^2)$ where θ is the MA coefficient, with some simple algebra we can isolate σ_w^2 . One minor complication is that there is no guarantee that the indirect estimate of σ_w^2 will be positive—if there's a problem, we'll reset it to a small positive number:

```
nonlin phi sw gammax
compute gammax=%beta(1)
compute phi=%beta(2)
compute sw=-phi*varx2*(1+%beta(3)^2)/%beta(3)-(1+phi^2)*varx2
compute sw=%if(sw<0,.1,sw)
```

With everything set, we can now estimate the (approximate) model with:

```
dln(y=ysq,sw=sw,sv=varx2,c=1.0,a=phi,z=gammax*(1-phi),$
presample=ergodic,method=bfgs,type=filter) 2 * states
```

³Apply $(1 - \varphi L)$ to both equations and use (12.1) to simplify (12.4).

Table 12.3: QMLE Estimates for Stochastic Volatility Model

DLM - Estimation by BFGS					
Convergence in 9 Iterations. Final criterion was 0.0000041 <= 0.0000100					
Usable Observations		945			
Rank of Observables		945			
Log Likelihood		-2081.2196			
	Variable	Coeff	Std Error	T-Stat	Signif
1.	PHI	0.9912	0.0084	117.6210	0.0000
2.	SW	0.0070	0.0053	1.3226	0.1860
3.	GAMMAX	-0.7931	0.3336	-2.3774	0.0174

This is a relatively simple state-space model once everything has been defined. The state and measurement equations in matrix form are:

$$\begin{aligned} [\log h_t] &= [\gamma^*(1 - \varphi)] + [\varphi][\log h_{t-1}] + w_t \\ \text{YSQ}_t &= [1][\log h_t] + v_t \end{aligned}$$

where the variance of v (handled by the `SV` option) is known to be π^2 . The “intercept” in the state equation is handled with the `Z` option. The output is shown in Table 12.3. Note that the log likelihood is not at all comparable to that of the GARCH model, since this is the density for $\log y^2$ rather than y .

We can get the filtered⁴ estimates of the variances with

```
set hsv = exp(states(t)(1))
```

and do a comparison graph (Figure 12.1) with

```
graph(footer="Estimates of Variance from SV and EGARCH Models", $
  key=upleft, klabels=| | "SV", "EGARCH" | |) 2
# hsv
# hgarch
```

Obviously, the stochastic volatility model estimates a much smoother variance process. If we focus in on a smaller segment near an outlier (entry 863), we can see that more clearly (Figure 12.2):

The EGARCH variance jumps rather dramatically after 863, while the SV variance barely reacts at all. One of the main problems with the approximate Gaussian state-space representation is that it deals rather poorly with “inliers”. Near-zero values for y are converted into “left-tail” outliers by the $\log y^2$ transformation. Since these aren’t *that* uncommon (zero, after all, is the modal value no matter what the variance), the only way for the estimates to avoid overreacting to those is to also not react as strongly to the actual “right-tail” outliers. Thus the more persistent estimates. The method in the next section doesn’t have this problem.

⁴The filtered estimates will be more comparable to the ones we got with `GARCH` than smoothed estimates would be.

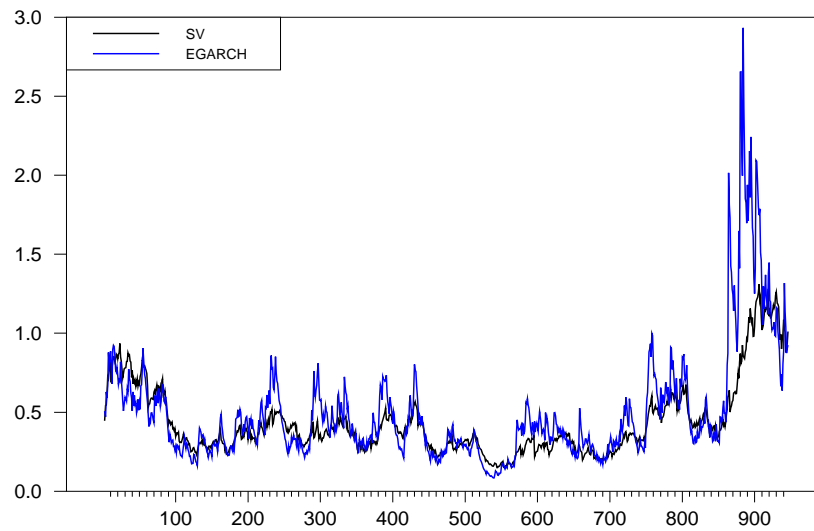


Figure 12.1: Estimates of Variance from SV and EGARCH Models

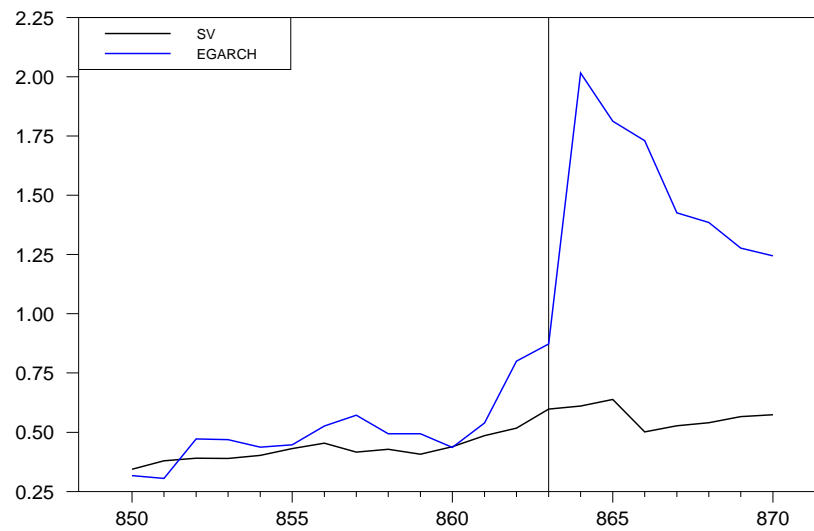


Figure 12.2: Estimates of Variance from SV and EGARCH Models

12.2 Gibbs Sampling

Jacquier et al. (1994) (henceforward JPR) describe an approach for estimating a stochastic volatility model using Gibbs Sampling. There have been several later papers by other authors which have provided refinements to improve the performance of the sampler.

The idea behind this is to treat the unobservable variances as extra parameters. The Gibbs Sampler then has to be set up to sample those (as well as the other unknowns). If the $\log h$ is treated as known, (12.1) is a straightforward autoregression with Normal errors which is easily sampled. The complicated part of this is sampling the variances.

A sampling “sweep” for the variances requires sampling each, conditional on the data and the other variances, which means we need to sample from the density:

$$f(h_t | y_1, \dots, y_T, h_1, \dots, h_{t-1}, h_{t+1}, \dots, h_T)$$

where conditioning on the standard model parameters is suppressed for simplicity. The relative simplicity of the model helps to simplify this rather considerably—because (12.1) is a one-lag autoregression, there is no additional information about h_t in h_{t+j} and h_{t-j} for any $j > 1$. And since y_s provides (direct) information only about x_s , the only y that matters is the current one, so the conditioning simplifies to

$$f(h_t | y_t, h_{t-1}, h_{t+1})$$

This is actually significant—there are many Gibbs samplers in situations like this where you have to condition on the whole sample, which means that an evaluation across the whole sample needs to be made for each t in each pass, making each Gibbs sweep an $O(T^2)$ operation. Here, it’s just $O(T)$.

Using standard conditioning arguments,

$$f(h_t | y_t, h_{t-1}, h_{t+1}) \propto f(h_t, y_t, h_{t-1}, h_{t+1}) = f(y_t, h_{t+1} | h_{t-1}, h_t) f(h_{t-1}, h_t)$$

y_t and h_{t+1} are independent conditional on h_t and are each conditionally independent of h_{t-1} so this simplifies further to

$$f(y_t | h_t) f(h_{t+1} | h_t) f(h_{t-1}, h_t)$$

If h_{t-1} exists, then the last factor converts to

$$f(h_{t-1}, h_t) \propto f(h_t | h_{t-1})$$

so we have

$$f(h_t | y_t, h_{t-1}, h_{t+1}) \propto f(y_t | h_t) f(h_{t+1} | h_t) f(h_t | h_{t-1})$$

where all the densities on the right side are given directly by the model. If $t = T$ so there is no h_{t+1} , we have

$$f(h_t | y_t, h_{t-1}) \propto f(y_t | h_t) f(h_t | h_{t-1})$$

If $t = 1$, so there is no h_{t-1} , we end up with

$$f(h_t|y_t, h_{t+1}) \propto f(y_t|h_t)f(h_{t+1}|h_t)f(h_t) \quad (12.5)$$

This requires that h_1 have an unconditional distribution, which rules out the random walk—to handle inference with a random walk in (12.1) you would have to treat h_1 (or h_0) in some other fashion. We'll assume that we have a stationary representation, so $|\varphi| < 1$.

So there are three different forms depending upon which neighbors entry t has. However, in all three cases, the densities involving just the h 's are Normals in $\log h$, just with different means and variances. In the most complicated case (with two neighbors), we have the quadratic part of the log of $f(h_{t+1}|h_t)f(h_t|h_{t-1})$ as

$$\sigma^{-2}(\log h_{t+1} - \gamma - \varphi \log h_t)^2 + \sigma^{-2}(\log h_t - \gamma - \varphi \log h_{t-1})^2$$

If this is expanded to isolate $\log h_t$, we get

$$\{\sigma^{-2}(1 + \varphi^2)(\log h_t)^2 - 2(\log h_t)\sigma^{-2}(\varphi \log h_{t+1} - \varphi\gamma + \gamma + \varphi \log h_{t-1}) + \dots\}$$

where the omitted terms don't include $\log h_t$. From the result in Appendix F, we can read off that this means that

$$\log h_t \sim N\left(\frac{\varphi \log h_{t+1} + \varphi \log h_{t-1} + \gamma(1 - \varphi)}{(1 + \varphi^2)}, \frac{\sigma^2}{(1 + \varphi^2)}\right)$$

If we don't have h_{t+1} , it's immediate that we have

$$\log h_t \sim N(\gamma + \varphi \log h_{t-1}, \sigma^2)$$

If we don't have h_{t-1} , then $f(h_{t+1}|h_t)f(h_t)$ in (12.5) can be recast using the “backwards” representation of the stationary autoregression to give

$$\log h_t \sim N(\gamma + \varphi \log h_{t+1}, \sigma^2)$$

Those would all be easy to handle if that were it—whether the Gibbs procedure draws $\log h$ or h doesn't matter. The technical problem comes from the one factor that's common to all three branches: $f(y_t|h_t)$. The density is proportional to

$$h_t^{-1/2} \exp\left(-\frac{y_t^2}{2h_t}\right)$$

which makes h an inverse gamma (Appendix A.4). Thus, we have one factor with h as an inverse gamma and one where it's log normal. The combination isn't a known density, or anything even close. So the technical question is how to draw h from this, and do it efficiently, since each t generates a very different density.

Tsay (2010) recommends “griddy Gibbs”. The latter approximates the density on a grid—however, it's very computationally intensive and it's very hard to

set up an appropriate grid when each time through the situation changes. It's almost always the case that there is a more practical alternative to this.

JPR used Metropolis within Gibbs (Appendix H). While this can be made to work (and work better than the JPR method with some “tuning”), Metropolis within Gibbs has a special problem in this situation. The draw from (12.2) is known as “single-move” sampling. We have an entire history for h , but change them one at a time. However, h_t depends (rather strongly in many cases) on h_{t-1} and h_{t+1} . If you have a relatively low draw from h_t for one pass, that will tend to lead to relatively low draws for its neighbors h_{t-1} and h_{t+1} , which tends on a future sweep to lead to relatively low draws for h_t (and similarly with low replaced by high). Thus, there is quite a bit of inertia in the draws for h . Because Metropolis “corrects” for an incorrect draw distribution by staying put rather than jumping, in this application, using Metropolis adds an extra layer of inertia to the Gibbs sampler.

What we'll describe is a refinement on the rejection method (Appendix I) used in Kim et al. (1998). The rejection method requires more care setting up than Metropolis, but it gives an exact procedure for each draw for h_t , thus eliminating one layer of inertia in the chain.

The following idea is based upon John Geweke's discussion to Jacquier et al. (1994). This works in quite a few situations where a density is formed by multiplying a Normal by a non-Normal continuous density, which we have here if we define $x = \log h_t$. The log density for a $N(\mu, \sigma^2)$ can be written

$$\log p(x) = -\frac{1}{2}(ax^2 - 2bx + c) \quad \text{where} \quad \sigma^2 = 1/a, \mu = b/a$$

If the non-Normal density can be approximated with an expansion around x_0 as

$$\log q(x) \approx -\frac{1}{2}(d(x - x_0)^2 - 2e(x - x_0) + f) = -\frac{1}{2}(dx^2 - 2(e + dx_0)x + f^*) \quad (12.6)$$

then

$$\log p(x)q(x) = \log p(x) + \log q(x) \approx -\frac{1}{2}(a + d)x^2 + (b + e + dx_0)x + (f^* + c) \quad (12.7)$$

which (by inspection) is (using Appendix F)

$$N\left(\frac{b + e + dx_0}{a + d}, \frac{1}{a + d}\right) \quad (12.8)$$

We can write the inverse gamma factor as

$$\log q(x) = -\frac{1}{2}x - \frac{y^2}{2}\exp(-x) \quad (12.9)$$

which is a globally concave function. As such, it lies below its tangent line at *any* expansion point, so

$$\log q(x) \leq \log q^*(x) \equiv -\frac{1}{2}x_0 - \frac{y^2}{2}\exp(-x_0) + \left(-\frac{1}{2} + \frac{y^2}{2}\exp(-x_0)\right)(x-x_0) \quad (12.10)$$

and

$$p(x)q(x) = p(x)q^*(x) (q(x)/q^*(x)) \quad (12.11)$$

Thus, we can use $p(x)q^*(x)$ as the proposal density with acceptance function $q(x)/q^*(x)$. In (12.2) and (12.7), p and q^* have

$$a = \frac{1}{\sigma^2}, b = \frac{\mu}{\sigma^2}, d = 0, e = \left(-\frac{1}{2} + \frac{y^2}{2}\exp(-z_0)\right) \quad (12.12)$$

so the proposal density is

$$N\left(\mu - \frac{\sigma^2}{2} + \frac{\sigma^2 y^2}{2}\exp(-x_0), \sigma^2\right) \quad (12.13)$$

This will work (theoretically) regardless of the choice of x_0 , but might be quite inefficient if the expansion point is chosen poorly. Kim, Shephard and Chib suggest using $x_0 = \mu$. If the shape of the combined density were dominated by the Normal factor (as it will often be if it were a Normal likelihood combined with a non-Normal *prior*), that would probably work quite well. However, that's not necessarily the case here. Both densities are functions of the *data*, and an outlier value for y can make the mean of the Normal factor an inefficient expansion point.⁵ With a bit of extra calculation, it's possible to refine the expansion by moving closer to the mode of the final density. This is done by taking a few Newton-Raphson steps, using a quadratic expansion, before using the linear expansion when actually drawing random numbers.⁶

This is done using the `@KSCPostDraw` procedure. This is a special-purpose procedure for doing the draws (for the log) in this context. This works on just a single observation—you have to loop over this since the information from the neighbors is different at each data point.

```
@KSCPostDraw y zterms zsqterms zdraw
```

y	observed data
$zterms$	linear term in the log density for the Gaussian distribution from the neighbors
$zsqterms$	quadratic term in the log density

⁵We've found that 1000 tries might not be enough for some data points.

⁶You can't use the quadratic expansion for the rejection method itself because the approximation won't necessarily lie above the true density.

`zdraw` (output) draw from posterior for $\log h$

It has one option: `ITERS` gives the number of preliminary iterations to refine the expansion point. `ITERS=0` gives the original KSC proposal. We recommend `ITERS=5`.

Example 12.2 uses the same data set and again uses the de-meaned log returns. We'll use a GARCH model to get a preliminary estimate of the h process and will use a linear regression on the log of that to initialize the parameters. Since this is just the start of a Gibbs sampling process, it's not important to be extremely careful with how this is done.

```
garch(p=1,q=1,hseries=h) / demean
set z = log(h)
linreg z
# constant z{1}
```

The `NONLIN` instruction here doesn't ever get used in a non-linear optimization—instead, it's just a convenient way to organize the parameters that we will be estimating. Because the `GAMMA` and `SIGW` parameters are fairly sensitive to the value of φ when the last is near 1, we also keep track of the overall mean and variance for the $\log h$ process, calling them `GAMMAX` and `SIGWX`.

```
compute gamma=%beta(1),phi=%beta(2),sigw=sqrt(%sigmasq)
compute gammax=gamma/(1-phi),sigwx=sigw/sqrt(1-phi^2)
```

JPR use a very diffuse (though very slightly informative) prior for the γ and φ . That's certainly reasonable for γ given that it's so dependent upon φ . Obviously, we need φ to satisfy stationarity assumptions, which we'll handle by making its prior relatively diffuse but zero outside $(-1,1)$, which is accomplished by just rejecting any draws that stray out of bounds. As this is set up, the means for both are zero, and they're independent each with a standard deviation of 10—`HGAMMA0` is the precision.⁷

```
compute [symm] hgamma0=%diag(||1.0/100.0,1.0/100.0||)
compute [vect] gamma0=%zeros(2,1)
```

The prior for the variance of the $\log h$ has to be at least *somewhat* informative. With models like this, a zero variance (in the regression for $\log h$) is an absorbing state in the chain, so for safety, you have to have something to prevent the variance from collapsing to zero. Here, we're about as diffuse as can be, with just 1 degree of freedom. (The scale value comes from looking at the preliminary estimates).

```
compute pscale=.007
compute pdf =1.0
```

⁷The precision is the reciprocal of the variance.

This sets the number of burn-in and keeper draws for the chain. Because there are so many parameters in the chain (each h_t separately, plus the three structural parameters), and because the h_t are sampled in a highly correlated fashion, it's likely to take a long time for the chain to settle down.

```
compute nkeep=10000
compute nburn=10000
```

We'll keep track of the full set of draws for the structural parameters (the original three plus the two derived ones), the sum (to produce the average) for the h_t process across the data set, and the full set of draws for just the entry 863 that was the focus above. Obviously, this last is specific to this application.

```
dec series[vect] stats
gset stats 1 nkeep = %parmspeek(svparms)
set hstats gstart gend = 0.0
set hout 1 nkeep = 0.0
```

We have the usual structure for a MCMC draw procedure:

```
infobox(action=define,progress,lower=-nburn,upper=nkeep) $
  "Gibbs Sampler"
do draws=-nburn,nkeep

  DO CALCULATIONS

  *
  * Skip the bookkeeping if we're still in the burn-in
  *
  infobox(current=draws)
  if draws<=0
    next

  SAVE STATISTICS

end do draws
infobox(action=remove)
```

Within that, we have the following: γ and φ are drawn jointly as a linear regression since the $\log h_t$ are being treated as “data” at this stage. If `PHI` ends up out of range, we just re-draw.

```

cmom(noprint)
# constant z{1} z
:redraw
compute dgamma=%ranmvpostcmom(%cmom,1.0/sigw^2,hgamma0,gamma0)
compute gamma=dgamma(1),phi=dgamma(2)
if abs(phi)>1.0 {
    disp "Problem with phi"
    goto redraw
}

```

SIGW (or actually the variance) is drawn using the standard method for regression variance.

```

compute %rss=%qform(%cmom,dgamma~-1.0)
compute sigmasq=(pscale*pdf+%rss)/%ranchisqr(%nobs+pdf)
compute sigw=sqrt(sigmasq)

```

What's unique to this model is drawing the $\log h_t$. We loop over the sample and draw the value for TIME given the values for TIME-1 and TIME+1, and the regression parameters. This requires dealing with the three branches (first time period, last time period, and the ones in the middle), and, to use @KSCPostDraw, computing the linear and quadratic terms from the expansions. The only part of this loop that is application-dependent is the reference to `demean(time)` on the @KSCPostDraw—that needs to be replaced with your data series.

```

do time=gstart,gend
    if time==gstart
        compute zsqterms=1.0/sigmasq,$
                zterms=1.0/sigmasq*(gamma+phi*z(time+1))
    else
        if time==gend
            compute zsqterms=1.0/sigmasq,$
                    zterms=1.0/sigmasq*(gamma+phi*z(time-1))
        else
            compute zsqterms=(1+phi^2)/sigmasq,$
                    zterms=(1/sigmasq)*(phi*(z(time+1)+z(time-1))+gamma*(1-phi))

        @KSCPostDraw(iters=5) demean(time) zterms zsqterms z(time)
        if failed
            disp time demean(time) zterms zsqterms
    end do time

```

The `if failed` is there in case something goes badly wrong. The procedure does 1000 attempts on the rejection algorithm before it gives up—in practice, it rarely takes more than 3. However, if you change the `ITERS=5` to `ITERS=0`, you will get the original KSC proposal, and you may very well see the rejection algorithm fail.

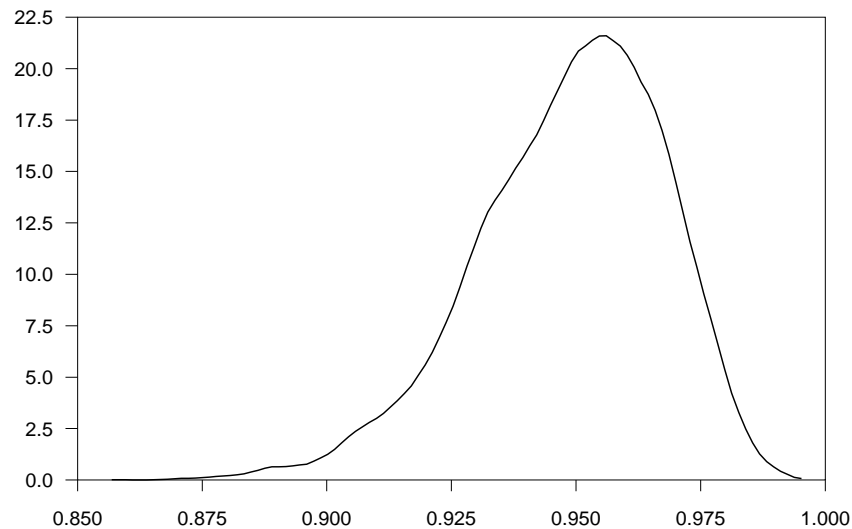


Figure 12.3: Posterior Density for φ

The bookkeeping requires computing GAMMAX and SIGWX, saving the contents of the SVPARMS parameter set, saving the estimated variance at entry 863 and adding in the current draw to the HSTATS series:

```
compute gammax=gamma/(1-phi), sigwx=sigw/sqrt(1-phi^2)
compute stats(draws)=%parmspeek(svparms)
compute hout(draws)=exp(z(863))
set hstats gstart gend = hstats+exp(z)
```

The post-processing draws graphs of the five parameters (or transformations). The most interesting of these is for φ (Figure 12.3):

```
set phis 1 nkeep = stats(t)(2)
density(grid=automatic,maxgrid=100,smoothing=1.5) phis / xphi fphi
scatter(style=line,vmin=0.0,footer="Posterior Density for phi")
# xphi fphi
```

which shows quite a bit less persistence than we got with the approximate state-space model. The mean estimate (Figure 12.4), computed using

```
set hstats gstart gend = hstats/nkeep
graph(footer="Mean Estimate of Volatility")
# hstats
```

is much more similar to the GARCH model than what we got with the state-space approximation. Note, however, that this is a “smoothed” estimate of the variance. That this is a smoothed estimate is more obvious if we look at a small segment (around the outlier, Figure 12.5):

```
graph(footer="Estimates of Variance from SV with MCMC",grid=(t==863))
# hstats 850 870
```

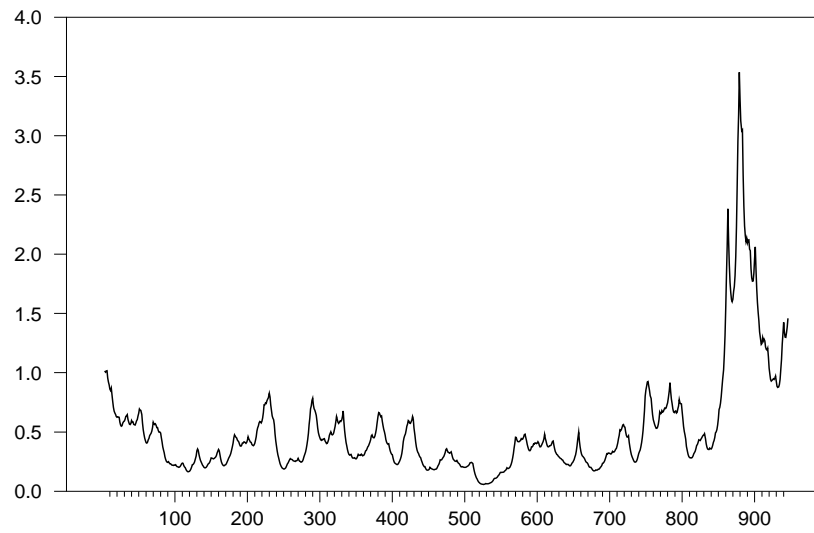


Figure 12.4: Mean Estimate of Volatility

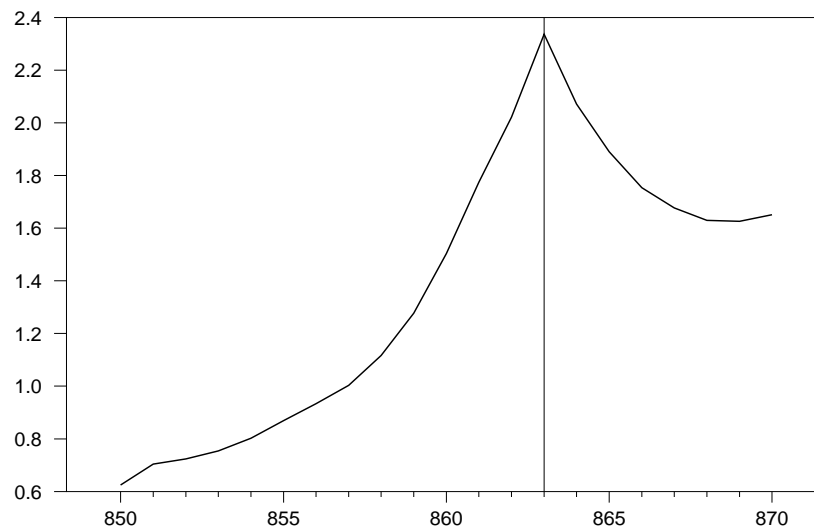


Figure 12.5: Mean Estimate of Volatility near Outlier

Example 12.1 Stochastic Volatility: State-Space Approximation

This estimates a stochastic volatility model by means of state-space approximation. The data set is from Harvey et al. (1994). The technical details are in Section 12.1.

```
open data xrates.xls
data(format=xls,org=columns) 1 946 usxuk usxger usxjpn usxsui
*
* meanx2=mean of log chi-square,varx2=variance of log chi-square.
* These are exact (rather than rounded) values.
*
compute meanx2=%digamma(0.5)-log(0.5)
compute varx2 =%trigamma(0.5)
*
set dlogp = 100.0*log(usxuk{0}/usxuk{1})
diff(center) dlogp / demean
*
garch(exp,nomean,hseries=hgarch) / demean
*
set ysq = log(demean^2)-meanx2
*
* Get initial guess values from an ARMA(1,1) model
*
boxjenk(ar=1,ma=1,constant) ysq
*
* phi is taken directly as the AR(1) coefficient. The variance sw is
* backed out by matching first order autocorrelations in the MA
* term. gamma is chosen to reproduce the mean of the ysq series
*
nonlin phi sw gammax
compute gammax=%beta(1)
compute phi=%beta(2)
compute sw=-phi*varx2*(1+%beta(3)^2)/%beta(3)-(1+phi^2)*varx2
compute sw=%if(sw<0,.1,sw)
*
* Estimate the unconstrained model
*
dlm(y=ysq,sw=sw,sv=varx2,c=1.0,a=phi,z=gammax*(1-phi),$
    presample=ergodic,method=bfgs,type=filter) 2 * states
*
set hsv = exp(states(t)(1))
*
* Graph comparing variance estimates
*
graph(footer="Estimates of Variance from SV and EGARCH Models",$
    key=upleft,klabels=|"SV","EGARCH"|) 2
# hsv
# hgarch
*
graph(footer="Estimates of Variance from SV and EGARCH Models",$
    key=upleft,klabels=|"SV","EGARCH"|,grid=(t==863)) 2
```



```
# hsv      850 870
# hgarch   850 870
*
* Compute smoothed estimates of the (log) variances
*
dln(y=ysq, sw=sw, sv=varx2, c=1.0, a=phi, z=gammax*(1-phi), $
    presample=ergodic, type=smooth) 2 * states
*
set hsvsmooth = exp(states(t)(1))
graph(footer="Filtered/Smoothed Estimates of Variance from SV Model", $
    key=upleft, klabels=||"Filtered", "Smoothed"||, grid=(t==863)) 2
# hsv      850 870
# hsvsmooth 850 870
```

Example 12.2 Stochastic Volatility: Gibbs Sampling

This estimates a stochastic volatility model by means of Gibbs sampling. It does a refinement on the sampler with rejection method from Kim et al. (1998). The details are in Section 12.2.

```
open data xrates.xls
data(format=xls, org=columns) 1 946 usxuk usxger usxjpn usxsui
*
set dlogp = 100.0*log(usxuk{0}/usxuk{1})
diff(center) dlogp / demean
*
* Get initial guess values by running a GARCH model to get the h's,
* then running a regression on that to get the other parameters.
*
garch(p=1, q=1, hseries=h) / demean
set z = log(h)
linreg z
# constant z{1}
*
nonlin(parmset=svparms) gamma phi sigw gammax sigwx
*
compute gamma=%beta(1), phi=%beta(2), sigw=sqrt(%sigmasq)
compute gammax=gamma/(1-phi), sigwx=sigw/sqrt(1-phi^2)
*
* Prior for gamma/phi
*
compute [symm] hgamma0=%diag(||1.0/100.0, 1.0/100.0||)
compute [vect] gamma0=zeros(2,1)
*
* Prior for sigw^2
*
compute pscale=.007
compute pdf =1.0
*
* This is a very high ratio of burn-in draws to keepers. Because
* each h(t) is a separate parameter for the sampler, there are many
* parameters, all sampled independently.
```

```

*
compute nkeep=10000
compute nburn=10000
*
compute gstart=%regstart(),gend=%regend()
dec vect dphi
*
* Initialize bookkeeping. We're keeping five complete sets (for
* gamma, phi, sigw, gammax and sigwx), the sum of the h series and
* the full set of h for the outlier observation.
*
dec series[vect] stats
gset stats 1 nkeep = %parmspeek(svparms)
set hstats gstart gend = 0.0
set hout 1 nkeep = 0.0
*
infobox(action=define,progress,lower=-nburn,upper=nkeep) $
"Gibbs Sampler"
do draws=-nburn,nkeep
  *
  * Draw gamma and phi given h's and sigw
  *
  cmom(noprint)
  # constant z{1} z
:redraw
  compute dgamma=%ranmvpostcmom(%cmom,1.0/sigw^2,hgamma0,gamma0)
  compute gamma=dgamma(1),phi=dgamma(2)
  if abs(phi)>1.0 {
    disp "Problem with phi"
    goto redraw
  }
  *
  * Draw sigw given gamma and phi
  *
  compute %rss=%qform(%cmom,dgamma~-1.0)
  compute sigmasq=(pscale*pdf+%rss)/%ranchisqr(%nobs+pdf)
  compute sigw=sqrt(sigmasq)
  *
  * Draw z's given everything else
  *
  do time=gstart,gend
    if time==gstart
      compute zsqterms=1.0/sigmasq,$
      zterms=1.0/sigmasq*(gamma+phi*z(time+1))
    else
      if time==gend
        compute zsqterms=1.0/sigmasq,$
        zterms=1.0/sigmasq*(gamma+phi*z(time-1))
      else
        compute zsqterms=(1+phi^2)/sigmasq,$
        zterms=(1/sigmasq)*(phi*(z(time+1)+z(time-1))+gamma*(1-phi))

    @KSCPostDraw(iters=5) demean(time) zterms zsqterms z(time)
  if failed

```

```

        disp time demean(time) zterms zsqterms
    end do time
    *
    * Skip the bookkeeping if we're still in the burn-in
    *
    infobox(current=draws)
    if draws<=0
        next

        compute gammax=gamma/(1-phi),sigwx=sigw/sqrt(1-phi^2)
        compute stats(draws)=%parmspeek(svparms)
        compute hout(draws)=exp(z(863))
        set hstats gstart gend = hstats+exp(z)

    end do draws
    infobox(action=remove)
    *
    set gammas 1 nkeep = stats(t) (1)
    density(grid=automatic,maxgrid=100,smoothing=1.5) $
        gammas / xgamma fgamma
    scatter(style=line,vmin=0.0,footer="Posterior Density for gamma")
    # xgamma fgamma
    *
    set phis 1 nkeep = stats(t) (2)
    density(grid=automatic,maxgrid=100,smoothing=1.5) $
        phis / xphi fphi
    scatter(style=line,vmin=0.0,footer="Posterior Density for phi")
    # xphi fphi
    *
    set sigws 1 nkeep = stats(t) (3)
    density(grid=automatic,maxgrid=100,smoothing=1.5) $
        sigws / xsigma fsigma
    scatter(style=line,vmin=0.0,footer="Posterior Density for Sigma")
    # xsigma fsigma
    *
    set gammaws 1 nkeep = stats(t) (4)
    density(grid=automatic,maxgrid=100,smoothing=1.5) $
        gammaws / xgammaw fgammaw
    scatter(style=line,vmin=0.0,$
        footer="Posterior Density for Process Mean")
    # xgammaw fgammaw
    *
    set sigwxs 1 nkeep = stats(t) (5)
    density(grid=automatic,maxgrid=100,smoothing=1.5) $
        sigwxs / xsigmax fsigmax
    scatter(style=line,vmin=0.0,$
        footer="Posterior Density for Process Variance")
    # xsigmax fsigmax
    *
    set hstats gstart gend = hstats/nkeep
    graph(footer="Mean Estimate of Volatility")
    # hstats
    *
    density(grid=automatic,maxgrid=100,smoothing=1.5) $

```

```
      hout / xout fout
scatter(style=line,vmin=0.0,$
      footer="Posterior Density for Variance at Outlier Point")
# xout fout
*
graph(footer="Estimates of Variance from SV with MCMC",grid=(t==863))
# hstats 850 870
```

Probability Distributions

A.1 Univariate Normal

Parameters	Mean (μ), Variance (σ^2)
Kernel	$\sigma^{-1} \exp \left(-\frac{(x - \mu)^2}{2\sigma^2} \right)$
Support	$(-\infty, \infty)$
Mean	μ
Variance	σ^2
Main uses	Distribution of error terms in univariate processes. Asymptotic distributions. Prior, exact and approximate posteriors for parameters with unlimited ranges.
Density Function	<code>%DENSITY(x)</code> is the non-logged standard Normal density. More generally, <code>%LOGDENSITY(variance,u)</code> . Use <code>%LOGDENSITY(sigmasq,x-mu)</code> to compute $\log f(x \mu, \sigma^2)$.
CDF	<code>%CDF(x)</code> is the standard Normal CDF (running from 0 in the left tail to 1 in the right). To get $F(x \mu, \sigma^2)$, use <code>%CDF((x-mu)/sigma)</code> . <code>%ZTEST(z)</code> gives the two-tailed tail probability (probability a $N(0,1)$ exceeds z in absolute value).
Inverse CDF	<code>%INVNORMAL(p)</code> is the standard Normal inverse CDF.
Random Draws	<code>%RAN(s)</code> draws one or more (depending upon the target) independent $N(0, s^2)$. <code>%RANMAT(m,n)</code> draws a matrix of independent $N(0, 1)$.

A.2 Univariate Student (*t*)

Parameters	Mean (μ), Variance of underlying Normal (σ^2) or of the distribution itself (s^2), Degrees of freedom (ν)
Kernel	$\left(1 + (x - \mu)^2 / (\sigma^2 \nu)\right)^{-(\nu+1)/2} \text{ or } \\ \left(1 + (x - \mu)^2 / (s^2 (\nu - 2))\right)^{-(\nu+1)/2}$
Support	$(-\infty, \infty)$
Mean	μ
Variance	$\sigma^2 \nu / (\nu - 2)$ or s^2
Main uses	Small sample distributions of univariate statistics. Fatter-tailed alternative to Normal for error processes. Prior, exact and approximate posteriors for parameters with unlimited ranges.
Density Function	<p><code>%TDENSITY(x, nu)</code> is the (non-logged) density function for a standard ($\mu = 0, \sigma^2 = 1$) t.</p> <p><code>%LOGTDENSITY(ssquared, u, nu)</code> is the log density based upon the s^2 parameterization.</p> <p>Use <code>%LOGTDENSITY(ssquared, x-mu, nu)</code> to compute $\log f(x \mu, s^2, \nu)$ and</p> <p><code>%LOGTDENSITYSTD(sigmasq, x-mu, nu)</code> to compute $\log f(x \mu, \sigma^2, \nu)$.</p>
CDF	<p><code>%TCDF(x, nu)</code> is the CDF for a standard t.</p> <p><code>%TTEST(x, nu)</code> is the two-tailed probability of a standard t</p>
Inverse CDF	<p><code>%INVTCDF(p, nu)</code> is the inverse CDF for a standard t.</p> <p><code>%INVTTEST(p, nu)</code> is the critical value for a two-tailed standard t test.</p>
Random Draws	<code>%RANT(nu)</code> draws one or more (depending upon the target) standard t 's with independent numerators and a <i>common</i> denominator. To get a draw from a t density with variance <code>ssquared</code> and <code>nu</code> degrees of freedom, use <code>%RANT(nu)*sqrt(ssquared*(nu-2.)/nu)</code> .
Notes	With $\nu = 1$, this is a Cauchy (no mean or variance); with $\nu \leq 2$, the variance doesn't exist. $\nu \rightarrow \infty$ tends towards a Normal.

A.3 Gamma Distribution

Parameters	shape (a) and scale (b), alternatively, degrees of freedom (ν) and mean (μ). The RATS functions use the first of these. The relationship between them is $a = \nu/2$ and $b = \frac{2\mu}{\nu}$. The chi-squared distribution with ν degrees of freedom is a special case with $\mu = \nu$.
Kernel	$x^{a-1} \exp\left(-\frac{x}{b}\right)$ or $x^{(\nu/2)-1} \exp\left(-\frac{x\nu}{2\mu}\right)$
Support	$[0, \infty)$
Mean	ba or μ
Variance	b^2a or $\frac{2\mu^2}{\nu}$
Main uses	Prior, exact and approximate posterior for the precision (reciprocal of variance) of residuals or other shocks in a model
Density function	<code>%LOGGAMMADENSITY(x, a, b)</code> . For the $\{\nu, \mu\}$ parameterization, use <code>%LOGGAMMADENSITY(x, .5*nu, 2.0*mu/nu)</code>
Random Draws	<code>%RANGAMMA(a)</code> draws one or more (depending upon the target) independent Gammas with unit scale factor. Use <code>b*%RANGAMMA(a)</code> to get a draw from $Gamma(a, b)$. If you are using the $\{\nu, \mu\}$ parameterization, use <code>2.0*mu*%RANGAMMA(.5*nu)/nu</code> . You can also use <code>mu*%RANCHISQR(nu)/nu</code> .
Moment Matching	<code>%GammaParms(mean, sd)</code> (external function) returns the 2-vector of parameters ((a, b) parameterization) for a gamma with the given mean and standard deviation.

A.4 Inverse Gamma Distribution

Parameters	shape (a) and scale (b). An inverse gamma is the reciprocal of a gamma. The special case is the scaled inverse chi-squared (Appendix A.5 with parameters ν (degrees of freedom) and τ^2 (scale parameter) which has $a = \nu/2$ and $b = \nu\tau^2/2$).
Kernel	$x^{-(a+1)} \exp(-bx^{-1})$
Integrating Constant	$b^a/\Gamma(a)$
Support	$[0, \infty)$
Mean	$\frac{b}{(a-1)}$ if $a > 1$
Variance	$\frac{b^2}{(a-1)^2(a-2)}$ if $a > 2$
Main uses	Prior, exact and approximate posterior for the variance of residuals or other shocks in a model. For these purposes, it's usually simpler to directly use the scaled inverse chi-squared variation.
Density Function	<code>%LOGINVGAMMADENSITY(x, a, b)</code> . Added with RATS 9.1.
Moment Matching	<code>%InvGammaParms(mean, sd)</code> (external function) returns the 2-vector of parameters ((a, b) parameterization) for the parameters of an inverse gamma with the given mean and standard deviation. If <code>sd</code> is the missing value, this will return $a = 2$, which is the largest value of a which gives an infinite variance.
Random draws	You can use <code>b/%rangamma(a)</code> . A draw from a scaled inverse chi-squared is typically done with <code>nu*tausqr/%ranchisqr(nu)</code> .

A.5 (Scaled) Inverse Chi-Squared Distribution

Parameters	Degrees of freedom (ν) and scale (τ^2). An inverse chi-squared is the reciprocal of a chi-squared combined with scaling factor which represents a “target” variance that the distribution is intended to represent. (Note that the mean is roughly τ^2 for large degrees of freedom.)
Kernel	$x^{-(a+1)} \exp(-bx^{-1})$
Integrating Constant	$b^a / \Gamma(a)$
Support	$[0, \infty)$
Mean	$\frac{\nu\tau^2}{\nu-2}$ if $\nu > 2$
Variance	$\frac{2\nu^2\tau^4}{(\nu-2)^2(\nu-4)}$ if $\nu > 4$
Main uses	Prior, exact and approximate posterior for the variance of residuals or other shocks in a model. The closely-related inverse gamma (Appendix A.4) can be used for that as well, but the scaled inverse chi-squared tends to be more intuitive.
DensityFunction	<code>%LOGINVCHISQRDENSITY(x, nu, tausq)</code> . Added with RATS 9.1.
Moment Matching	<code>%InvChisqrParms(mean, sd)</code> (external function) returns the 2-vector of parameters ((ν, τ^2) in that order) for the parameters of an inverse chi-squared with the given mean and standard deviation. If <code>sd</code> is the missing value, this will return $\nu = 4$, which is the largest value of ν which gives an infinite variance.
Random draws	You can use <code>(nu*tausq)/%ranchisqr(nu)</code> . Note that you divide by the random chi-squared.

A.6 Multivariate Normal

Parameters	Mean (μ), Covariance matrix (Σ) or precision (H)
Kernel	$ \Sigma ^{-1/2} \exp \left(-\frac{1}{2} (x - \mu)' \Sigma^{-1} (x - \mu) \right)$ or $ H ^{1/2} \exp \left(-\frac{1}{2} (x - \mu)' H (x - \mu) \right)$
Support	\mathbb{R}^n
Mean	μ
Variance	Σ or H^{-1}
Main uses	Distribution of multivariate error processes. Asymptotic distributions. Prior, exact and approximate posteriors for a collection of parameters with unlimited ranges.
Density Function	<code>%LOGDENSITY(sigma,u)</code> . To compute $\log f(x \mu, \Sigma)$ use <code>%LOGDENSITY(sigma,x-mu)</code> . (The same function works for univariate and multivariate Normals).
Distribution Function	<code>%BICDF(x,y,rho)</code> returns $P(X \leq x, Y \leq y)$ for a bivariate Normal with mean zero, variance 1 in each component and correlation rho.
Random Draws	<code>%RANMAT(m,n)</code> draws a matrix of independent $N(0,1)$. <code>%RANMVNORMAL(F)</code> draws an n -vector from a $N(0, FF')$, where F is any factor of the covariance matrix. This setup is used (rather than taking the covariance matrix itself as the input) so you can do the factor just once if it's fixed across a set of draws. To get a single draw from a $N(\mu, \Sigma)$, use <code>MU+%RANMVNORMAL(%DECOMP(SIGMA))</code> <code>%RANMVPOST</code> , <code>%RANMVPOSTCMOM</code> , <code>%RANMVKRON</code> and <code>%RANMVKRONCMOM</code> are specialized functions which draw multivariate Normals with calculations of the mean and covariance matrix from other matrices.

A.7 Multivariate Student (*t*)

Parameters	Mean (μ), covariance matrix of the underlying multivariate Normal (Σ) or of the distribution itself (S), Degrees of freedom (ν)
Kernel	$(1 + (x - \mu)' \Sigma^{-1} (x - \mu) / \nu)^{-(\nu+n)/2}$ or $(1 + (x - \mu)' S^{-1} (x - \mu) / (\nu - 2))^{-(\nu+n)/2}$
Support	\mathbb{R}^n
Mean	μ
Variance	$\Sigma \frac{\nu}{(\nu - 2)}$ or S
Main uses	Prior, exact and approximate posteriors for sets of parameters with unlimited ranges.
Density Function	<p><code>%LOGTDENSITY(s, u, nu)</code> is the log density based upon the S parameterization.</p> <p>Use <code>%LOGTDENSITY(s, x-mu, nu)</code> to compute $\log f(x \mu, S, \nu)$ and <code>%LOGTDENSITYSTD(sigma, u, nu)</code> to compute $\log f(x \mu, \Sigma, \nu)$.¹ The same functions work for both univariate and multivariate distributions.</p>
Random Draws	<p><code>%RANT(nu)</code> or <code>%RANMVT(fsigma, nu)</code> <code>%RANT(nu)</code> draws one or more (depending upon the target) standard <i>t</i>'s with independent numerators and a common denominator. <code>%RANMVT(fsigma, nu)</code> draws a multivariate <i>t</i> with <i>fsigma</i> as a “square root” of the Σ matrix. You would typically compute <i>fsigma</i> as <code>%DECOMP(sigma)</code>. The function is parameterized this way in case you need to do many draws with a single <i>sigma</i>, as you can do the decomposition in advance.</p> <p>To draw a multivariate <i>t</i> with covariance matrix <i>S</i> and <i>nu</i> degrees of freedom, use <code>%RANMVT(%DECOMP(s*((nu-2.)/nu)))</code>.</p>

¹`%LOGTDENSITYSTD` and `%RANMVT` were added with RATS 7.3. Before that, use `%LOGTDENSITY(sigma*nu/(nu-2), x-mu, nu)` and `mu+%decomp(sigma)*(u=%rant(nu))`

A.8 GED distribution

Generalized Error Distribution, also known as simply the *error distribution*, or as the *exponential power distribution*.

Parameters	2 (c (shape) and either b (scale) or σ^2 (variance))
Kernel	$\exp\left(-(x /b)^{2/c}/2\right)$
Support	$(-\infty, \infty)$
Mean	0 (a mean shift is simple, but rarely needed)
Variance	$\frac{2^c b^2 \Gamma(3c/2)}{\Gamma(c/2)}$
Other Moments	$E x ^r = \frac{2^{rc/2} b^r \Gamma((r+1)c/2)}{\Gamma(c/2)}$ $E\frac{ x }{\sigma} = \frac{\Gamma(c)}{\sqrt{\Gamma(3c/2)\Gamma(c/2)}}$
Kurtosis	$\frac{\Gamma(5c/2)\Gamma(c/2)}{[\Gamma(3c/2)]^2}$ <p>This is greater than 3 (thus fat-tailed) if $c > 1$ and less than 3 if $c < 1$.</p>
Main uses	In financial econometrics, as an alternative to the t to provide different tail behavior from the Normal.
Density function	<code>%LOGGEDDENSITY (x, c, variance)</code>
Distribution Function	<code>%GEDCDF (x, c)</code> with variance pegged to 1.0
Inverse CDF	<code>%INVGED (p, c)</code> with variance pegged to 1.0
Random Draws	<code>%RANGED (c, variance)</code> (external function)
Special Cases	$c = 1$ is Normal, $c = 2$ is Laplace (two-tailed exponential)

Quasi-Maximum Likelihood Estimations (QMLE)

The main source for results on QMLE is White (1994). Unfortunately, the book is so technical as to be almost unreadable. We'll try to translate the main results as best we can.

Suppose that $\{x_t\}, t = 1, \dots, \infty$ is a stochastic process and suppose that we have observed a finite piece of this $\{x_1, \dots, x_T\}$ and that the true (unknown) log joint density of this can be written

$$\sum_{t=1}^T \log g_t(x_t, \dots, x_1)$$

This is generally no problem for either cross section data (where independence may be a reasonable assumption) or time series models where the data can be thought of as being generated sequentially. Some panel data likelihoods will not, however, be representable in this form.

A (log) quasi likelihood for the data is a collection of density functions indexed by a set of parameters θ of the form

$$\sum_{t=1}^T \log f_t(x_t, \dots, x_1; \theta)$$

which it is hoped will include a reasonable approximation to the true density. In practice, this will be the log likelihood for a mathematically convenient representation of the data such as joint Normal. The QMLE is the (or more technically, a, since there might be non-uniqueness) $\hat{\theta}$ which maximizes the log quasi-likelihood.

Under the standard types of assumptions which would be used for actual maximum likelihood estimation, $\hat{\theta}$ proves to be consistent and asymptotically Normal, where the asymptotic distribution is given by $\sqrt{T}(\hat{\theta} - \theta) \xrightarrow{d} N(0, \mathbf{A}^{-1} \mathbf{B} \mathbf{A}^{-1})$, where \mathbf{A} is approximated by

$$\mathbf{A}_T = \frac{1}{T} \sum_{t=1}^T \frac{\partial^2 \log f_t}{\partial \theta \partial \theta'}$$

and \mathbf{B} by (if there is no serial correlation in the gradients)

$$\mathbf{B}_T = \frac{1}{T} \sum_{t=1}^T \left(\frac{\partial \log f_t}{\partial \theta} \right)' \left(\frac{\partial \log f_t}{\partial \theta} \right) \quad (\text{B.1})$$

with the derivatives evaluated at $\hat{\theta}$.¹ Serial correlation in the gradients is handled by a Newey-West type calculation in (B.1). This is the standard “sandwich” estimator for the covariance matrix. For instance, if $\log f_t = -(x_t - z_t\theta)^2$, (with z_t treated as exogenous), then

$$\frac{\partial \log f_t}{\partial \theta} = 2(x_t - \theta z_t) z_t'$$

and

$$\frac{\partial^2 \log f_t}{\partial \theta \partial \theta'} = -2z_t' z_t$$

and the asymptotic covariance matrix of $\hat{\theta}$ is

$$\left(\sum z_t' z_t \right)^{-1} \left(\sum z_t' u_t^2 z_t \right) \left(\sum z_t' z_t \right)^{-1}$$

the standard Eicker-White robust covariance matrix for least squares. Notice that, when you compute the covariance matrix this way, you can be somewhat sloppy with the constant multipliers in the log quasi likelihood—if this were the actual likelihood for a Normal, $\log f_t$ would have a $\frac{1}{2\sigma^2}$ multiplier, but that would just cancel out of the calculation since it gets squared in the center factor and inverted in the two ends.

This is very nice, but what is the θ_0 to which this is converging? After all, nothing above actually required that the f_t even approximate g_t well, much less include it as a member. It turns out that this is the value which minimizes the Kullback-Liebler Information Criterion (KLIC) discrepancy between f and g which is (suppressing various subscripts) the expected value (over the density g) of $\log(g/f)$. The KLIC has the properties that it’s non-negative and is equal to zero only if $f = g$ (almost everywhere), so the QMLE will at least asymptotically come up with the member of the family which is closest (in the KLIC sense) to the truth.

Again, closest might not be close. However, in practice, we’re typically less interested in the complete density function of the data than in some aspects of it, particularly moments. A general result is that if f is an appropriate selection from the linear exponential family, then the QMLE will provide asymptotically valid estimates of the parameters in a conditional expectation. The linear exponential family are those for which the density takes the form

$$\log f(x; \theta) = a(\theta) + b(x) + \theta' t(x)$$

This is a very convenient family because the interaction between the parameters and the data is severely limited.² This family includes the Normal, gamma (chi-squared and exponential are special cases), Weibull and beta distributions

¹The formal statement of this requires pre-multiplying the left side by a matrix square root of $AB^{-1}A$ and having the target covariance matrix be the identity.

²The exponential family in general has $d(\theta)$ entering into that final term, though if d is invertible, it’s possible to reparameterize to convert a general exponential to the linear form.

among continuous distributions and binomial, Poisson and geometric among discrete ones. It *does not* include the logistic, t , F , Cauchy and uniform.

For example, suppose that we have “count” data—that is, the observable data are nonnegative integers (number of patents, number of children, number of job offers, etc.). Suppose that we posit that the expected value takes the form $E(y_t|w_t) = \exp(w_t\theta)$. The Poisson is a density in the exponential family which has the correct support for the underlying process (that it, it has a positive density only for the non-negative integers). Its probability distribution (as a function of its single parameter λ) is defined by $P(x; \lambda) = \frac{\exp(-\lambda)\lambda^x}{x!}$. If we define $\omega = \log(\lambda)$, this is linear exponential family with $a(\omega) = -\exp(\omega)$, $b(x) = \log x!$, $t(x) = x$. There’s a very good chance that the Poisson will *not* be the correct distribution for the data because the Poisson has the property that both its mean and its variance are λ . Despite that, the Poisson QMLE, which maximizes $\sum -\exp(w_t\theta) + x_t(w_t\theta)$, will give consistent, asymptotically Normal estimates of θ .

It can also be shown that, under reasonably general conditions, if the “model” provides a set of moment conditions (depending upon some parameters) that match up with QMLE first order conditions from a linear exponential family, then the QMLE provides consistent estimates of the parameters in the moment conditions.

Diagnostic Tests on Large Data Sets

If you do quite a bit of work with large data sets (typically financial models), you may find it frustrating to get a model which “passes” a standard set of diagnostic tests for model adequacy. This, however, is a well-known (though rarely discussed) problem when dealing with real-world data. In Berkson (1938), the author points out that tests for Normality will almost always reject in very large data sets with p -values “beyond any usual limit of significance”. Now, the key is that we’re talking about “real-world” data. There have been dozens, if not hundreds, of papers published in the literature which propose a test, demonstrate that it has asymptotically the correct size when simulated under the null, demonstrate it has power against simulated alternatives, and ... that’s the end of the paper. What’s missing in many of these papers is any attempt to demonstrate how it works with real data, not simulated data. If the new test doesn’t produce qualitatively different results from an existing (probably simpler) test, is there any real point to it?

A very readable discussion of what’s happening with tests in large data sets is provided in Leamer (1974), Chapter 4. Effectively no model is “correct” when applied to real world data. Consider, for instance, Berkson’s example of a test for Normality. In practice, actual data is likely to be bounded—and the Normal distribution isn’t. Various Central Limit Theorems show that the sums of many relatively small independent (or weakly correlated) components approach the Normal, but in practice, some of the components probably will be a bit more dominant than the CLT’s expect.

While Berkson was talking about using a chi-squared test, it’s simpler to look at what happens with the commonly used Jarque-Bera test. Suppose we generate a sample from a t with 50 degrees of freedom (thus close to, but not quite, Normal). The following is a not atypical set of results from applying the Jarque-Bera test to samples of varying sizes:

N	Kurtosis	JB	Signif
100	0.243	0.448	0.799469
500	0.287	4.071	0.130635
1000	0.230	2.638	0.267441
2000	0.208	3.676	0.159127
5000	0.201	8.507	0.014212
10000	0.201	16.883	0.000216
100000	0.152	96.544	0.000000

The $N = 100000$ line would be an example of the “beyond any usual level of significance”. The theoretical excess kurtosis for this is roughly .13—not a

particularly heavy-tailed distribution, and one which is barely distinguishable in any meaningful way from the Normal. If we stick with the conventional significance level of .05, regardless of sample size, we are choosing to allow the probability of Type I errors to remain at .05 while driving the probability of Type II errors down effectively to zero. As we get more and more data, we should be able to push down the probabilities of *both* types of errors, and a testing process which doesn't is hard to justify.

The JB test has an asymptotic χ^2_2 distribution. That has a .05 critical value of 5.99, a .01 critical value of 9.21 and a .001 critical value of 13.81. In effect, the JB estimates two extra parameters (the skewness and excess kurtosis) and sees whether they are different from the 0 values that they would have if the distribution were Normal. If we look at how the SBC (or BIC) would deal with the decision about allowing for those extra 2 parameters, its “critical values” (the point at which we would choose the larger model) is $2\log T$, which is 9.2 for $T = 100$, 13.3 for $T = 1000$, 18.4 for $T = 10000$ to 23.0 for $T = 100000$. Eventually (by $T = 100000$), the data evidence in favor of the (correct) non-normal distribution is strong enough to cause us to choose it, at $T = 10000$ the decision somewhat marginal (16.883 vs a critical value of 18.4), but at the smaller sample sizes, we would rather strongly favor the simpler model—the difference between the $t(50)$ and Normal just isn't apparent enough until we get a truly enormous amount of data.

Now JB has degrees of freedom fixed at 2. Let's look at a typical diagnostic test in time series analysis like the Ljung-Box Q test for serial correlation.

$$Q = T(T+2) \sum_{k=1}^h \frac{\hat{\rho}_k^2}{(T-k)} \quad (\text{C.1})$$

where $\hat{\rho}_k$ is the lag k autocorrelation (typically of residuals in some form). This is easier to examine more carefully if we take out some of the small-sample corrections and look at the simpler (asymptotically equivalent) Box-Pierce test

$$Q = T \sum_{k=1}^h \hat{\rho}_k^2 \quad (\text{C.2})$$

In practice, h is often fairly large (10 or more) and, in fact, the recommendation is that it increase (slowly!) with T . However, let's fix on $h = 25$. Suppose we have $T = 2500$ (a not uncommon size for a GARCH model). The .05 critical value for a χ^2_{25} is 37.7, the .01 is 44.3. Because of the T multiplier in (C.2), the size of correlations that triggers a “significant” result is quite small—if the typical autocorrelation is a mere .03, it would generate a Q of 56.3, which has a p -value of .0003. That's despite the fact that .03 autocorrelations are probably not correctable by any change you could make to the model: they're statistically significant, but practically insignificant. This is a common problem with high degrees of freedom tests. Now the $h \log T$ “critical value” suggested by the SBC (which would be 195.6 here) is a bit extravagant, since we're aren't

really estimating a model with 25 extra parameters, but we should not be at all surprised to see a fairly reasonable-looking model on this size data set produce a Q statistic at least 2 or 3 times h , without the test suggesting *any* change to improve the model.

This is not to suggest that you simply brush aside results of the diagnostic tests. For instance, if residual autocorrelations are large on the small lags (1 and 2, particularly), that suggests that you need longer lags in the mean model—it's a “significant” result triggered by (relatively) large correlations at odd locations like 7 and 17 that is unlikely to be improved by a change to the model. One simple way to check whether there is anything to those is to compute the diagnostic on a split sample. What you would typically see is that the pattern of “large” autocorrelations is completely different on the subsamples and thus isn't a systematic failure of the model. Remember that the point of the diagnostics is to either suggest a better model or warn you of a serious problem with the one you're using. This type of “significant” Q test does neither.

Delta method

The *delta method* is used to estimate the variance of a non-linear function of a set of already estimated parameters. The basic result is that if θ are the parameters and we have

$$\sqrt{T} \left(\hat{\theta} - \theta \right) \xrightarrow{d} N(0, \Sigma_{\theta}) \quad (\text{D.1})$$

and if $f(\theta)$ is continuously differentiable, then, by using a first order Taylor expansion

$$\left(f(\hat{\theta}) - f(\theta) \right) \approx f'(\theta) \left(\hat{\theta} - \theta \right)$$

Reintroducing the \sqrt{T} scale factors and taking limits gives

$$\sqrt{T} \left(f(\hat{\theta}) - f(\theta) \right) \xrightarrow{d} N \left(0, f'(\theta) \Sigma_{\theta} f'(\theta)' \right)$$

In practice, this means that if we have

$$\hat{\theta} \approx N(\theta, \mathbf{A}) \quad (\text{D.2})$$

then

$$f \left(\hat{\theta} \right) \approx N \left(f(\theta), f'(\hat{\theta}) \mathbf{A} f'(\hat{\theta})' \right) \quad (\text{D.3})$$

(D.1) is the type of formal statement required, since the \mathbf{A} in (D.2) collapses to zero as $T \rightarrow \infty$. It's also key that (D.1) implies that $\hat{\theta} \xrightarrow{p} \theta$, so $f'(\hat{\theta}) \xrightarrow{p} f'(\theta)$ allowing us to replace the unobservable $f'(\theta)$ with the estimated form in (D.3). So the point estimate of the function is the function of the point estimate, at least as the center of the asymptotic distribution. If $\hat{\theta}$ is unbiased for θ , then it's almost certain that $f(\hat{\theta})$ will *not* be unbiased for $f(\theta)$ so this is *not* a statement about expected values.

To compute the asymptotic distribution, it's necessary to compute the partial derivatives of f . For scalar functions of the parameters estimated using a RATS instruction, that can usually be most easily done using the instruction **SUMMARIZE**.

Central Limit Theorems with Dependent Data

The simplest form of Central Limit Theorem (CLT) assumes a sequence of i.i.d. random variables with finite variance. Under those conditions, regardless of the shape of the distributions (anything from 0-1 Bernoullis to fat-tailed variables with infinite fourth moments),

$$\sqrt{T}(\bar{x} - \mu) \xrightarrow{d} N(0, \sigma^2) \quad (\text{E.1})$$

Those were extended to allow independent, but non-identically distributed, random variables as long as there was some control on the tail behavior and the relative variances to prevent a small percentage of the summands from dominating the result. The assumption of independence serves two purposes:

1. It makes it much easier to prove the result, since it's relatively easy to work with characteristic functions of independent random variables.
2. Independence helps to restrict the influence of each element.

In time series analysis, independence is too strong an assumption. However, it's still possible to construct CLT's with weaker assumptions as long as the influence of any small number of elements is properly controlled.

One type of useful weakening of independence is to assume a sequence is a *martingale difference sequence* (m.d.s.). $\{u_t\}$ is an m.d.s. if

$$E(u_t | u_{t-1}, u_{t-2}, \dots) = 0$$

It's called this because a martingale is a sequence which satisfies

$$E(x_t | x_{t-1}, x_{t-2}, \dots) = x_{t-1}$$

so, by the Law of Iterated Expectations (conditioning first on a superset)

$$\begin{aligned} E(x_t - x_{t-1} | x_{t-1} - x_{t-2}, x_{t-2} - x_{t-3}, \dots) = \\ E(E(x_t - x_{t-1} | x_{t-1}, x_{t-2}, \dots) | x_{t-1} - x_{t-2}, x_{t-2} - x_{t-3}, \dots) = 0 \end{aligned}$$

thus the first difference of a martingale is an m.d.s. An i.i.d. mean zero process is trivially an m.d.s. A non-trivial example is $u_t = \varepsilon_t \varepsilon_{t-1}$, where ε_t is an i.i.d. mean zero process. u_t isn't independent of u_{t-1} because they share a ε_{t-1} factor; as a result, the *variances* of u_t and u_{t-1} will tend to move together.

The ergodic martingale CLT states that if u_t is a stationary ergodic m.d.s. and $Eu_t^2 = \sigma^2$, then

$$\frac{1}{\sqrt{T}} \sum_{t=1}^T u_t \xrightarrow{d} N(0, \sigma^2)$$

We can write this (somewhat informally) as

$$\frac{1}{\sqrt{T}} \sum_{t=1}^T u_t \xrightarrow{d} N(0, Eu_t^2)$$

and very informally, this is used as

$$\sum_t u_t \approx N\left(0, \sum_t u_t^2\right) \quad (\text{E.2})$$

This is the form that is useful when we have serially uncorrelated (though not necessarily serial independent) summands. However, it won't handle serial correlation. A basic CLT which can be applied more generally is the following: if

$$x_t = \sum_{s=0}^q c_s \varepsilon_{t-s}$$

where ε_t has assumptions which generate a standard $N(0, \sigma^2)$, then

$$\frac{1}{\sqrt{T}} \sum_{t=1}^T x_t \xrightarrow{d} N\left(0, \left(\sum c_s\right)^2 \sigma^2\right)$$

If we write $x_t = C(L)\varepsilon_t$, then we can write $C(1) = \sum_{s=0}^q c_s$, so the limiting distribution can be written $C(1)^2 \sigma^2$. This is known as the long-run variance of x : if ε_t were subject to a permanent shift generated by a random variable with variance σ^2 , the variance that would produce in x_t is $C(1)^2 \sigma^2$.

The somewhat informal restatement of this is

$$\frac{1}{\sqrt{T}} \sum_{t=1}^T x_t \xrightarrow{d} N(0, lvar(x))$$

where $lvar(x)$ is the long-run variance of the x process, and in practice we use

$$\sum_t x_t \approx N\left(0, \sum_t \sum_{l=-L}^L w_l x_t x'_{t-l}\right) \quad (\text{E.3})$$

where the variance in the target distribution uses some feasible estimator for the long-run variance (such as Newey-West).

The approximating covariance matrix in (E.2) can be computed using the instruction **CMOMENT** (applied to u), or with **MCOV** without any **LAG** options, and

that in (E.3) can be computed using **MCOV** using `LAG` and `LWINDOW` options. Note that both these are written using sums (not means) on both sides. That tends to be the most convenient form in practice—when you try to translate a result from the literature, you need to make sure that you get the factors of T correct.

Properties of Multivariate Normals

The density function for a jointly Normal random n -vector \mathbf{x} with mean μ and covariance matrix Σ is

$$(2\pi)^{-n/2} |\Sigma|^{-1/2} \exp \left(-\frac{1}{2} (\mathbf{x} - \mu)' \Sigma^{-1} (\mathbf{x} - \mu) \right)$$

Manipulations with this can quickly get very messy. Fortunately, the exponent in this (which is just a scalar – the result of the quadratic form) contains enough information that we can just read from it μ and Σ . In particular, if we can determine that the kernel of the density of \mathbf{x} (the part of the density which includes all occurrences of \mathbf{x}) takes the form

$$\exp \left(-\frac{1}{2} Q(\mathbf{x}) \right) \quad , \text{where} \quad Q(\mathbf{x}) = \mathbf{x}' \mathbf{A} \mathbf{x} - 2\mathbf{x}' \mathbf{b} + c$$

then, by completing the square, we can turn this into

$$Q(\mathbf{x}) = (\mathbf{x} - \mathbf{A}^{-1} \mathbf{b})' \mathbf{A} (\mathbf{x} - \mathbf{A}^{-1} \mathbf{b}) + (c - \mathbf{b}' \mathbf{A}^{-1} \mathbf{b})$$

Thus $\Sigma = \mathbf{A}^{-1}$ and $\mu = \mathbf{A}^{-1} \mathbf{b}$. The final part of this, $(c - \mathbf{b}' \mathbf{A}^{-1} \mathbf{b})$, doesn't involve \mathbf{x} and will just wash into the constant of integration for the Normal. We might need to retain it for analyzing other parameters (such as the residual variance), but it has no effect on the conditional distribution of \mathbf{x} itself.

One standard manipulation of multivariate Normals comes in the Bayesian technique of combining a prior and a likelihood to produce a posterior density. In the standard Normal linear model, the data have density

$$f(\mathbf{Y}|\beta) \sim N(\mathbf{X}\beta, \sigma^2 \mathbf{I})$$

The (log) kernel of the density is

$$-\frac{1}{2\sigma^2} (\mathbf{Y} - \mathbf{X}\beta)' (\mathbf{Y} - \mathbf{X}\beta) = -\frac{1}{2} (\beta' (\sigma^{-2} \mathbf{X}' \mathbf{X}) \beta - 2\beta' \sigma^{-2} \mathbf{X}' \mathbf{Y} + \sigma^{-2} \mathbf{Y}' \mathbf{Y})$$

Looking at this as a function of β , we read off

$$\beta|\mathbf{Y} \sim N \left((\mathbf{X}' \mathbf{X})^{-1} \mathbf{X}' \mathbf{Y}, \sigma^2 (\mathbf{X}' \mathbf{X})^{-1} \right) = N \left(\hat{\beta}, \sigma^2 (\mathbf{X}' \mathbf{X})^{-1} \right)$$

Now, suppose that, in addition to the data, we have the prior

$$\beta \sim N(\beta^*, \mathbf{H}^{-1})$$

and we write $\hat{\mathbf{H}} = \sigma^{-2} (\mathbf{X}'\mathbf{X})$ (the inverse of the least squares covariance matrix). If we multiply the densities, the only parts that include β will be the two quadratic parts, which we can add in log form to get

$$Q(\beta) = (\beta - \hat{\beta})' \hat{\mathbf{H}} (\beta - \hat{\beta}) + (\beta - \beta^*)' \mathbf{H} (\beta - \beta^*) \quad (\text{F.1})$$

Expanding this gives us

$$Q(\beta) = \beta' (\hat{\mathbf{H}} + \mathbf{H}) \beta - 2\beta' (\hat{\mathbf{H}}\hat{\beta} + \mathbf{H}\beta^*) + \dots$$

where the extra terms don't involve β . Thus, the posterior for β is

$$\beta \sim N \left((\hat{\mathbf{H}} + \mathbf{H})^{-1} (\hat{\mathbf{H}}\hat{\beta} + \mathbf{H}\beta^*), (\hat{\mathbf{H}} + \mathbf{H})^{-1} \right)$$

Notice that the posterior mean is a matrix weighted average of the two input means, where the weights are the inverses of the variances. The inverse of the variance (of a Normal) is known as the *precision*. Notice that precision is additive: the precision of the posterior is the sum of the precisions of the data information and prior.

If we need to keep track of the extra terms, there's a relatively simple way to evaluate this. If we write the posterior mean and precision as:

$$\bar{\beta} = (\hat{\mathbf{H}} + \mathbf{H})^{-1} (\hat{\mathbf{H}}\hat{\beta} + \mathbf{H}\beta^*), \bar{\mathbf{H}} = \hat{\mathbf{H}} + \mathbf{H}$$

then we have

$$Q(\beta) = (\beta - \bar{\beta})' \bar{\mathbf{H}} (\beta - \bar{\beta}) + Q(\bar{\beta}) \quad (\text{F.2})$$

The first term in (F.2) has value 0 at $\bar{\beta}$, so using this and (F.1) gives

$$Q(\bar{\beta}) = (\bar{\beta} - \hat{\beta})' \hat{\mathbf{H}} (\bar{\beta} - \hat{\beta}) + (\bar{\beta} - \beta^*)' \mathbf{H} (\bar{\beta} - \beta^*)$$

As another example, consider the partitioned process

$$\begin{bmatrix} \mathbf{Y}_1 \\ \mathbf{Y}_2 \end{bmatrix} \sim N \left(\begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix}, \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma'_{12} & \Sigma_{22} \end{bmatrix} \right)$$

The Q function takes the form

$$\begin{bmatrix} \mathbf{Y}_1 - \mu_1 \\ \mathbf{Y}_2 - \mu_2 \end{bmatrix}' \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma'_{12} & \Sigma_{22} \end{bmatrix}^{-1} \begin{bmatrix} \mathbf{Y}_1 - \mu_1 \\ \mathbf{Y}_2 - \mu_2 \end{bmatrix}$$

For now, let's just write the inverse in partitioned form without solving it out, thus

$$\begin{bmatrix} \mathbf{Y}_1 - \mu_1 \\ \mathbf{Y}_2 - \mu_2 \end{bmatrix}' \begin{bmatrix} \Sigma^{11} & \Sigma^{12} \\ \Sigma^{12'} & \Sigma^{22} \end{bmatrix} \begin{bmatrix} \mathbf{Y}_1 - \mu_1 \\ \mathbf{Y}_2 - \mu_2 \end{bmatrix}$$

This expands to

$$(\mathbf{Y}_1 - \mu_1)' \Sigma^{11} (\mathbf{Y}_1 - \mu_1) + 2(\mathbf{Y}_1 - \mu_1)' \Sigma^{12} (\mathbf{Y}_2 - \mu_2) + (\mathbf{Y}_2 - \mu_2)' \Sigma^{22} (\mathbf{Y}_2 - \mu_2)$$

where the cross terms are scalars which are transposes of each other, and hence equal, hence the 2 multiplier. If we now want to look at $\mathbf{Y}_1|\mathbf{Y}_2$, we get immediately that this has covariance matrix

$$(\Sigma^{11})^{-1}$$

and mean

$$\mu_1 - (\Sigma^{11})^{-1} \Sigma^{12} (\mathbf{Y}_2 - \mu_2)$$

If we want to reduce this to a formula in the original matrices, we can use partitioned inverse formulas to get

$$(\Sigma^{11})^{-1} = \Sigma_{11} - \Sigma_{12} \Sigma_{22}^{-1} \Sigma_{21}$$

and

$$(\Sigma^{11})^{-1} \Sigma^{12} = -\Sigma_{12} \Sigma_{22}^{-1}$$

thus

$$\mathbf{Y}_1|\mathbf{Y}_2 \sim N(\mu_1 + \Sigma_{12} \Sigma_{22}^{-1} (\mathbf{Y}_2 - \mu_2), \Sigma_{11} - \Sigma_{12} \Sigma_{22}^{-1} \Sigma_{21})$$

Note that the covariance matrix of the conditional distribution satisfies $\Sigma_{11} - \Sigma_{12} \Sigma_{22}^{-1} \Sigma_{21} \leq \Sigma_{11}$ and that it doesn't depend upon the actual data observed, even if those data are seriously in conflict with the assumed distribution.

Pseudo-Code

Pseudo-coding is a useful way to organize a set of calculations before you start to write actual instructions. In many cases, if you try to jump straight into “coding”, you will get things out of order, or skip a key step. In pseudo-coding, you start by writing what you need to do in plain English (or French or whatever). For instance,

```

Loop over possible breaks
  Do unit root test given the breaks
  If this is least favorable to the unit root hypothesis
    Save it
End

```

which is a fairly general description of a unit root test allowing for breaks. You then start to replace the plain language with actual code to do the required calculations and tests. It’s usually a good idea to convert the plain text to a comment so you can remember what a section is doing.

It’s often possible for the first attempt to include at least some actual code, particularly for the control instructions. For instance, in

```

do time=1871:1,1970:1
  <<update particles at time given time-1>>
end do time

```

the first and last lines are legal RATS codes, and we just need to flesh out the <<update...>>part. What’s important here is that we’ve made clear that we’ve set the loop so that it will compute the particles at <<time>>given <<time-1>>. One common problem that many people have is to write a loop without having a clear idea of what is supposed to be done on a trip through it.

It’s usually a good idea to replace the sections of pseudo-code in stages. For instance, the following is the setup generated by the simplest case of the Monte Carlo/Gibbs Sampling generator wizard (for simple Monte Carlo):

```
COMPUTE NDRAWS=100
*
* Do any calculations that aren't specific to a draw.
* Initialize anything needed for holding results of draws
*
DO DRAW=1,NDRAWS
  *
  * Do new draw from distribution
  *
  * Do bookkeeping here. Save results for entry DRAW
  *
END DO
```

At the moment, this does nothing other than run an empty loop 100 times. In general, you shouldn't start with the "bookkeeping" since until you actually have the draws and related calculations working, there's really nothing to save. Note that the wizard intentionally (by default) generates code for a rather small number of draws. A common error is to start with a "production" number of draws (like 10000), which typically requires shutting off the output from any calculations done inside the loop. As a result, you might miss the fact that (for instance), a nonlinear estimation fails to converge or a series being generated produces NA's where it shouldn't. With a small number of draws you can use `PRINT` options or `PRINT` or `DISPLAY` instructions to check that you're getting the types of results that you need. Once you're satisfied that you are generating proper results, you can turn off the "debugging" output and increase the number of draws. (One of the reasons for the wizard writing the program with a `COMPUTE` instruction for `NDRAWS` is to make it easier to change that).

Gibbs Sampling and Markov Chain Monte Carlo

Markov Chain Monte Carlo (MCMC) techniques allow for generation of draws from distributions which are too complicated for direct analysis. The simplest form of this is Gibbs sampling. This simulates draws from the density by means of a correlated Markov Chain. Under the proper circumstances, estimates of sample statistics generated from this converge to their true means under the actual posterior density.

Gibbs Sampling

The idea behind Gibbs sampling is that we partition our set of parameters into two or more groups: for illustration, we'll assume we have two, called Θ_1 and Θ_2 . We want to generate draws from a joint density $f(\Theta_1, \Theta_2)$, but don't have any simple way to do that. We can always write the joint density as $f(\Theta_1|\Theta_2)f(\Theta_2)$ and as $f(\Theta_2|\Theta_1)f(\Theta_1)$. In a very wide range of cases, these conditional densities *are* tractable. It's the unconditional densities of the *other* block that are the problem.

The intuition behind Gibbs sampling is that if we draw Θ_1 given Θ_2 , and then Θ_2 given Θ_1 , the pair should be closer to their unconditional distribution than before. Each combination of draws is known as a *sweep*. Repeat sweeps often enough and it should converge to give draws from the joint distribution. This turns out to be true in most situations. Just discard enough at the beginning (called the *burn-in* draws) so that the chain has had time to converge to the unconditional distribution. Once you *have* a draw from $f(\Theta_1, \Theta_2)$, you (of necessity) have a draw from the marginals $f(\Theta_1)$ and $f(\Theta_2)$, so now each sweep will give you a new draw from the desired distribution. They're just not *independent* draws. With enough "forgetfulness", however, the sample averages of the draws will converge to the true mean of the joint distribution.

Gibbs sampling works best when parameters which are (strongly) correlated with each other are in the same block, otherwise it will be hard to move both since the sampling procedure for each is tied to the other.

Metropolis-Hastings/Metropolis within Gibbs

Metropolis within Gibbs is a more advanced form of MCMC. Gibbs sampling requires that we be able to generate the draws from the conditional densities. However, there are only a handful of distributions for which we can do that—things like Normals, gammas, Dirichlets. In many cases, the desired density

is the likelihood for a complicated non-linear model which doesn't have such a form.

Suppose, we want to sample the random variable θ from $f(\theta)$ which takes a form for which direct sampling is difficult.¹ Instead, we sample from a more convenient density $q(\theta)$. Let $\theta^{(i-1)}$ be the value from the previous sweep. Compute

$$\alpha = \frac{f(\theta)}{f(\theta^{(i-1)})} \times \frac{q(\theta^{(i-1)})}{q(\theta)} \quad (\text{H.1})$$

With probability α , we accept the new draw and make $\theta^{(i)} = \theta$, otherwise we stay with our previous value and make $\theta^{(i)} = \theta^{(i-1)}$. Note that it's possible to have $\alpha > 1$, in which case we just accept the new draw.

The first ratio in (H.1) makes perfect sense. We want, as much as possible, to have draws where the posterior density is high, and not where it's low. The second counterweights (notice that it's the ratio in the opposite order) for the probability of drawing a given value. Another way of looking at the ratio is

$$\alpha = \frac{f(\theta)}{q(\theta)} \bigg/ \frac{f(\theta^{(i-1)})}{q(\theta^{(i-1)})}$$

f/q is a measure of the relative desirability of a draw. The ones that really give us a strong "move" signal are where the target density (f) is high and the proposal density (q) is low—we may not see those again, so when we get a chance we should move. Conversely, if f is low and q is high, we might as well stay put—we may revisit that one at a later time.

What this describes is *Independence Chain Metropolis*, where the proposal density doesn't depend upon the last draw. Where a set of parameters is expected to have a single mode, a good proposal density often can be constructed from maximum likelihood estimates, using a Normal or multivariate t centered at the ML estimates, with the covariance matrix some scale multiple (often just 1.0) of the estimate coming out of the maximization procedure.

If the actual density has a shape which *isn't* a good match for a Normal or t , this is unlikely to work well. If there are points where f is high and q is relatively low, it might take a very large number of draws before we find them, and once we get there we'll likely stay for a while. A more general procedure has the proposal density depending upon the last value: $q(\theta|\theta^{(i-1)})$. The acceptance criterion is now based upon:

$$\alpha = \frac{f(\theta)}{q(\theta|\theta^{(i-1)})} \bigg/ \frac{f(\theta^{(i-1)})}{q(\theta^{(i-1)}|\theta)} \quad (\text{H.2})$$

¹What we're describing here is Metropolis-Hastings or M-H for short. In all our applications, the densities will be conditional on the other blocks of parameters, which is formally known as Metropolis within Gibbs.

Note that the counterweighting is now based upon the ratio of the probability of moving from $\theta^{(i-1)}$ to θ to the probability of moving back. The calculation simplifies greatly if the proposal is a mean zero Normal or t added to $\theta^{(i-1)}$. Because of symmetry, $q(\theta|\theta^{(i-1)}) = q(\theta^{(i-1)}|\theta)$, so the q cancels, leaving just:

$$\alpha = f(\theta) / f(\theta^{(i-1)})$$

This is known as *Random Walk Metropolis*, which is probably the most common choice for Metropolis within Gibbs. Unlike Independence Chain, when you move to an isolated area where f is high, you can always move back by, in effect, retracing your route.

Avoiding Overflows

The f that we've been using in this is either the sample likelihood or the posterior (sample likelihood times prior). With many data sets, the sample *log* likelihood is on the order of 100's or 1000's (positive or negative). If you try to convert these large log likelihoods by taking the exp, you will likely overflow (for large positive) or underflow (for large negative), in either case, losing the actual value. Instead, you need to calculate the ratios by adding and subtracting in log form, and taking the exp only at the end.

Diagnostics

There are two types of diagnostics: those on the behavior of the sampling methods for subsets of the parameters, and those on the behavior of the overall chain. When you use Independence Chain Metropolis, you would generally like the acceptance rate to be fairly high. In fact, if $q = f$ (which means that you're actually doing a simple Gibbs draw), everything cancels and the acceptance is $\alpha = 1$. Numbers in the range of 20-30% are generally fine, but acceptance rates well below 10% are often an indication that you have a bad choice for q —your proposal density isn't matching well with f . When you use Random Walk Metropolis, an acceptance probability near 100% *isn't* good. You will almost never get rates like that unless you're taking very small steps and thus not moving around the parameter space sufficiently. Numbers in the 20-40% range are usually considered to be desirable. You can often tweak either the variance or the degrees of freedom in the increment to move that up or down. Clearly, in either case, you need to count the number of times you move and compare with the total number of draws.

If you're concerned about the overall behavior of the chain, you can run it several times and see if you get similar results. If you get decidedly different results, it's possible that the chain hasn't run long enough to converge, requiring a greater number of burn-in draws. The `@MCMCPOSTPROC` procedure also computes a CD measure which does a statistical comparison of the first part of the accepted draws with the last part. If the burn-in is sufficient, these should be asymptotically standard Normal statistics (there's one per parameter). If you

get values that have absolute values far out in the tails for a Normal (such as 4 and above), that's a strong indication that you either have a general problem with the chain, or you just haven't done a long enough burn-in.

Pathologies

Most properly designed chains will *eventually* converge, although it might take many sweeps to accomplish this. There are, however, some situations in which it won't work no matter how long it runs. If the Markov Chain has an *absorbing state* (or set of states), once the chain moves into this, it can't get out. This is particularly common with switching models, where once the probability of a regime is low enough you get no data points which actually are considered to fall into it, therefore the probability is pushed even more towards zero.

Rejection Method

Sometimes, distributions can't be derived as simple functions of the elementary distributions such as normal, gamma and beta. For instance, posterior densities can be the product of two distributions that can't easily be combined. The Rejection method is a tool which can often be used in univariate cases to generate draws. The Rejection method (or Acceptance-Rejection or Acceptance, all three terms are used) is used within RATS to generate draws from the normal and gamma distributions.

Suppose that we need draws from a $N(\mu, \sigma^2)$ truncated to the interval $[a, b]$. An obvious way to do this is to draw $N(\mu, \sigma^2)$ deviates, then reject any that fall outside of $[a, b]$. The accepted draws would have the desired distribution, but this process will be inefficient if the probability of the Normal draw falling in $[a, b]$ is fairly low.

An alternative is to draw a random number x from $[a, b]$. Also draw a random number z from $U(0, 1)$. Compare z with the ratio between $f_N(x|\mu, \sigma^2)$ and the maximum that $f_N(x|\mu, \sigma^2)$ achieves on $[a, b]$. If z is less, accept x ; if not, reject it. This trims away the uniform draws to match the shape of the Normal. This will be more efficient if the density function is fairly constant on $[a, b]$, so the comparison ratio is close to one and almost all draws are accepted.

A stylized procedure for doing the rejection method is

```
loop
  compute x = draw from the proposal density
  compute a = acceptance function
  if %ranflip(a)
    break
end loop
```

The value of x when the loop breaks is the draw. If you know that you have set this up correctly, and you know that the probability of accepting a draw is high, then you can adopt this code as is. However, this will loop until a draw is accepted, and, if you make a bad choice for the proposal density, you might end up with an acceptance function which produces values very close to zero, so this could loop effectively forever. A "fail-safed" alternative is


```

do try=1,maximum_tries
  compute x = draw from the proposal density
  compute a = acceptance function
  if %uniform(0.0,1.0)<a
    break
end do try

```

This will quit the loop if it fails to accept a draw in `MAXIMUM_TRIES`. How that limit should be set will depend upon the situation. If you set it to 100 and find that you're routinely hitting the limit, there is probably a better way to generate the draws.

To apply the rejection method to the density f , you need to be able to write

$$f(x) = \alpha(x)g(x)$$

where the proposal density $g(x)$ is one from which we can conveniently draw and

$$0 \leq \alpha(x) \leq 1 \quad (\text{I.1})$$

In our two examples for drawing from the truncated Normal, these are (in order):

$$g = N(\mu, \sigma^2), \alpha = I_{[a,b]} \quad (\text{I.2})$$

$$g = U(a, b), \alpha = f_N(x|\mu, \sigma^2) / \max_{[a,b]} f_N(\bullet|\mu, \sigma^2)$$

Any density for which $f(x)/g(x)$ is bounded will (theoretically) work as a proposal. However, as a general rule, you want to choose a distribution which has tails as thick as (or thicker) than the target density. For instance, if you use a Cauchy for g , you will get a small percentage of very extreme draws. If f is a thin-tailed distribution, f/g will be very small out there, and the extreme draws will be rejected. In other words, the rejection procedure thins out the Cauchy tails by rejecting most tail draws. If, however, f/g is quite *large* in the tails, the only way we can “thicken” the tails is by accepting the tail draws and rejecting most of the draws near the mode.

So long as $f(x)/g(x)$ is bounded, we can always choose

$$\alpha(x) = (f(x)/g(x)) / \max(f(\bullet)/g(\bullet))$$

(where the max is taken over the support of f) and that will give the most efficient rejection system for that particular choice of g . However, the rejection method is often applied in situations where the distribution changes slightly from one draw to the next. It's quite possible that you would spend more time computing the maximum in order to achieve “efficient” draws than it would take to use a simpler but less efficient choice of $\alpha(x)$. For instance, if we look at (I.2) above, we could also use

$$g = U(a, b), \alpha = \exp(-(x - \mu)^2 / 2\sigma^2)$$

If $\mu \in [a, b]$, this will be exactly the same as (I.2). If it isn't, then this will reject more draws than (I.2), since $\alpha(x)$ is strictly less than one for all the x 's that will be generated by the proposal density. Whether the extra work of finding the normalizing constant in (I) is worth it will depend upon the situation. Here, the added cost isn't high because the maximum will be at one of $\{\mu, a, b\}$. If, however, f and g are non-trivial density functions, the maximum might very well only be found by some iterative root-finding technique like Newton's method. Since finding the normalizing constant needs to be done just once per set of draws, the extra work will generally pay off when you need many draws from the same distribution, but not if you need just one.

While we've used a truncated Normal as our example, neither of these rejection systems is actually used by the `%RANTRUNCATE` function. Because `%RANTRUNCATE` can handle intervals that are unbounded, the second method wouldn't apply, and the first can be quite inefficient. Instead, it uses a different rejection system, using a truncated logistic as the proposal density.

GNU Free Documentation License

Version 1.3, 3 November 2008

Copyright © 2000, 2001, 2002, 2007, 2008 Free Software Foundation, Inc.

`<https://fsf.org/>`

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The purpose of this License is to make a manual, textbook, or other functional and useful document “free” in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of “copyleft”, which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated

herein. The “**Document**”, below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as “**you**”. You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A “**Modified Version**” of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A “**Secondary Section**” is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document’s overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The “**Invariant Sections**” are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The “**Cover Texts**” are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A “**Transparent**” copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not “Transparent” is called “**Opaque**”.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or

XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The “**Title Page**” means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, “Title Page” means the text near the most prominent appearance of the work’s title, preceding the beginning of the body of the text.

The “**publisher**” means any person or entity that distributes copies of the Document to the public.

A section “**Entitled XYZ**” means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as “**Acknowledgements**”, “**Dedications**”, “**Endorsements**”, or “**History**”.) To “**Preserve the Title**” of such a section when you modify the Document means that it remains a section “Entitled XYZ” according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document’s license notice requires Cover Texts, you must enclose the copies in covers that

carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together

with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

- C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
- D. Preserve all the copyright notices of the Document.
- E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- H. Include an unaltered copy of this License.
- I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
- J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
- K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled “Endorsements” or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version’s license notice. These titles must be distinct from any other section titles.

You may add a section Entitled “Endorsements”, provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled “History” in the various original documents, forming one section Entitled “History”; likewise combine any sections Entitled “Acknowledgements”, and any sections Entitled “Dedications”. You must delete all sections Entitled “Endorsements”.

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an “aggregate” if the copyright resulting from the compilation is not used to limit the legal rights of the compilation’s users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document’s Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a

translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled “Acknowledgements”, “Dedications”, or “History”, the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See <https://www.gnu.org/licenses/>.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License “or any later version” applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document

does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

“Massive Multiauthor Collaboration Site” (or “MMC Site”) means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A “Massive Multiauthor Collaboration” (or “MMC”) contained in the site means any set of copyrightable works thus published on the MMC site.

“CC-BY-SA” means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

“Incorporate” means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is “eligible for relicensing” if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

Bibliography

- Aielli G P 2013 *Journal of Business & Economic Statistics* **31**(3), 282–299.
- Andersen T & Bollerslev T 1998 *International Economic Review* **39**, 885–905.
- Bauwens L & Laurent S 2005 *Journal of Business and Economic Statistics* **23**, 346–354.
- Berkson J 1938 *Journal of American Statistical Association* **33**(303), 526–536.
- Bollerslev T 1986 *Journal of Econometrics* **31**(3), 307–327.
- Bollerslev T 1990 *Review of Economics and Statistics* **72**(3), 498–505.
- Bollerslev T, Engle R F & Wooldridge J 1988 *Journal of Political Economy* **96**(1), 116–131.
- Cermeño R & Grier K 2006 in B Baltagi, ed., ‘Panel Data Econometrics: Theoretical Contributions and Empirical Applications’ Springer pp. 259–278.
- Elder J & Serletis A 2010 *Journal of Money, Credit and Banking* **42**(6), 1137–1159.
- Enders W 2010 *Applied Econometric Time Series* 3rd edn Wiley.
- Engle R F 1982 *Econometrica* **50**(4), 987–1007.
- Engle R F 2002 *Journal of Business and Economic Statistics* **20**(3), 339–350.
- Engle R F & Kroner K F 1995 *Econometric Theory* **11**(1), 122–150.
- Engle R F, Lilien D M & Robins R P 1987 *Econometrica* **55**(2), 391–407.
- Engle R F & Ng V K 1993 *Journal of Finance* **48**(5), 1749–1778.
- Geweke J 1989 *Journal of Econometrics* **40**(1), 63–86.
- Glosten L R, Jagannathan R & Runkle D E 1993 *Journal of Finance* **48**(5), 1779–1801.
- Gospodinov N 2005 *Journal of Financial Econometrics* **3**(3), 344–371.
- Hacker, R. S. & Hatemi-J A 2005 *Applied Economics Letters* **12**, 411–417.
- Hafner C M & Herwartz H 2006 *Journal of International Money and Finance* **25**(5), 719–740.

- Hamilton J & Susmel R 1994 *Journal of Econometrics* **vol 64**(1-2), 307–333.
- Harvey A C, Ruiz E & Shepard N 1994 *Review of Economic Studies* **61**(2), 247–264.
- Inclan C & Tiao G 1994 *Journal of American Statistical Association* **89**(427), 913–923.
- Jacquier E, Polson N G & Rossi P E 1994 *Journal of Business and Economic Statistics* **12**(4), 371–389.
- Kim S, Shepard N & Chib S 1998 *Review of Economic Studies* **65**(3), 361–393.
- Leamer E 1974 *Specification Searches* New York: Wiley.
- Lee T H 1994 *Journal of International Money and Finance* **13**(3), 375–383.
- Martin V, Hurn S & Harris D 2012 *Econometric Modelling with Time Series: Specification, Estimation and Testing* Cambridge: Cambridge University Press.
- McLeod A I & Li W K 1983 *Journal of Time Series Analysis* **4**(4), 269–273.
- Muthen B 1990 *British Journal of Mathematical and Statistical Psychology* **43**(1), 131–143.
- Nelson D B 1990 *Econometric Theory* **6**(3), 1990.
- Nelson D B 1991 *Econometrica* **59**(2), 347–370.
- Newey W & West K 1987 *Econometrica* **55**(3), 703–708.
- Nyblom J 1989 *Journal of American Statistical Association* **84**(405), 223–230.
- Patton A J 2011 *Journal of Econometrics* **160**(1), 246–256.
- Perron P 1989 *Econometrica* **57**(6), 1361–1401.
- Sadorsky P 2012 *Energy Economics* **34**, 2.
- Sanso A, Arago V & Carrion-i Silvestre J 2004 *Spanish Review of Financial Economics* **4**(32-53).
- Tsay R 2010 *Analysis of Financial Time Series* Wiley.
- Tse Y K 2000 *Journal of Econometrics* **98**(1), 107–127.
- West K & Cho D 1995 *Journal of Econometrics* **69**(2), 367–391.
- White H 1994 *Estimation, Inference and Specification Analysis* Cambridge: Cambridge University Press.

Index

- ~~ operator, 254
- Absorbing state, 342
- Acceptance method, 343
- Acceptance-rejection method, 343
- Aielli, G. P., 108
- Andersen, T., 57
- Arago, V., 29
- @**ARAUTOLAGS** procedure, 32
- ARCH model, 29
- @**ARCHTEST** procedure, 36
- Bauwens, L., 190, 209, 223
- @**BDSTEST** procedure, 9
- BEKK model, 96
 - with asymmetry, 135
 - with variance regressors, 138
- Berkson, J., 327
- BFGS algorithm, 91
- BHHH algorithm, 91
- %**BICDF** function, 133, 321
- @**BICORRTEST** procedure, 9
- Bivariate Normal distribution, 133
- Bollerslev, T., 30, 47, 57, 90, 103
- BOOT** instruction, 217
- Bootstrapping, 216
 - drawbacks, 242
 - wild, 223
- Brownian motion, 1
- Burn-in, 339
- Carrion-i-Silvestre, J., 29
- Causality
 - in mean, 102
 - in variance, 100
- CC model, 103
 - with asymmetry, 134
- %**CDF** function, 316
- Cermeño, R., 183
- Chi-squared distribution, 320
- Chib, S., 304
- Cho, D., 4, 22
- Conditional volatility profile, 127
- Constant correlation, *see* CC model
- CRSPW.TXT data set, 31
- %**CVTOCORR** function, 94
- D-IBMLN98.DAT data set, 218
- DCC model, 106
- Delta method, 11, 330
- %**DENSITY** function, 316
- Dependence tests, 9
- Diagonal VECH model, 91
 - with restrictions, 184
- DJ14_JBES.XLS data set, 223
- DLM** instruction, 299
- DOFOR** instruction, 13
- DVECH model, *see* Diagonal VECH model
- Dynamic Conditional Correlation, *see* DCC model
- EGARCH model, 69
 - multivariate, 105
- Eigen decomposition, 88
- Elder, J., 241, 270, 287
- Elliptically symmetrical density, 190
- Enders, W., 82
- Engle, R., 29, 35, 66, 72, 90, 96
- ESMOOTH** instruction, 3
- EXRATES(DAILY).XLS data set, 82
- Extreme value distribution, 298
- FAMA5497.DAT data set, 167
- FILTER** instruction, 2
- @**FLUX** procedure, 62
- FORECAST** instruction, 86
 - PATHS option, 220
- Forecasts
 - for mean, 86
 - for variance, 56, 125
- FUNCTION** statement, 182
- Gamma distribution, 318, 319

GARCH instruction

ASYMMETRIC option, 74
 CONDITION option, 38
 DERIVES option, 62
 DISTRIB=GED option, 70
 DISTRIB=T option, 44
 EQUATION option, 55
 EXP option, 70
 HADJUST option, 67
 HMATRICES option, 86
 HSERIES option, 41
 I=DRIFT option, 60
 I=NODRIFT option, 60
 METHOD=EVAL option, 248
 MODEL option, 84, 86
 MV=BEKK option, 97
 MV=CC option, 103
 MV=DBEKK option, 125
 MV=DIAG option, 86
 MV=STANDARD option, 92
 MV=VECH option, 90
 output, 39, 86
 PMETHOD option, 92
 PRESAMPLE option, 48
 REGRESSORS option, 39
 ROBUSTERRORS option, 46
 RVECTORS option, 86
 sample range parameters, 39
 SIGNS option, 135
 START option, 187
 STDRESIDS option, 88
 UADJUST option, 67
 VARIANCES=EXP option, 105
 VARIANCES=SPILLOVER option, 105
 VARIANCES=VARMA option, 131, 134
 XBEKK option, 138
 XREGRESSORS option, 63, 138
 GARCH model, 30, 46
 integrated, 60
 SVAR, 195, 271
 GARCH-M model, 66
 multivariate, 271
 @GARCHFORE procedure, 56

GARCHGIBBS .RPF example, 242, 246

GARCHIMPORT .RPF example, 241

%GARCHV variable, 67

GARCHVIRFDIAG .RPF example, 132

GED distribution, 70, 323

Geweke, J., 241, 304

Gibbs sampling, 339

Glosten, L., 75

GMM estimation, 167

Goldfeld-Quandt test, 28

Gospodinov, N., 167, 178, 179

GRAPH instruction

 PICTURE option, 113

@GRIDSERIES procedure, 72

Grier, K., 183

GROUP instruction, 124

Gumbel distribution, 298

HAC standard errors, 10

Hacker, R. S., 85

Hadamard product, 91

Hafner, C., 124, 127

Hamilton, J., 31

Handle

 series, 15

Harris, R., 210

Harvey, A., 298

Hatemi-J, A., 85

Hedge ratio, 146

Herwartz, H., 124, 127

HHDATA .XLS data set, 124

Hurn, S., 210

@ICSS procedure, 28

IGARCH model, 60

Importance sampling, 241, 245

Impulse response function

 for mean, 277

 for variance, *see* Volatility impulse response function

Inclan, C., 28

Independence chain M-H, 251, 340

INFOBOX instruction, 261

INQUIRE instruction, 165

%INVNORMAL function, 316

- `%INVTCDF` function, 317
- `%INVTTEST` function, 317
- Jacquier, E., 302
- Jagannathan, R., 75
- Jarque-Bera test, 9, 42
- Kim, S., 304
- KLIC, 325
- Kroner, K., 96
- Kullback-Liebler Information Criterion, 325
- Laurent, S., 190, 209, 223
- Leamer, E., 327
- Lee, T.-H., 139, 156
- Li, W.K., 12
- LIBOR_DATA.XLS data set, 195
- Lilien, D., 66
- Ljung-Box Q test, 10
- LOCAL** statement, 199
- `%LOGDENSITY` function, 164, 316, 321
- `%LOGMVSKEWT` function, 192
- `%LOGTDENSITY` function, 317, 322
- `%LOGTDENSITYSTD` function, 317, 322
- M-H, *see* Metropolis-Hastings
- M-IBMSPLN.DAT data set, 163
- Markov Chain Monte Carlo, 242, 251, 339
- Martin, V., 210
- Martingale difference sequence, 331
- MAXIMIZE** instruction
 - RECURSIVE option, 165
- McLeod, A., 12
- McLeod-Li test, 12
- @MCLEODLI** procedure, 9, 12
- MCMC, *see* Markov Chain Monte Carlo
- @MCMCPOSTPROC** procedure, 341
- Metropolis-Hastings, 242, 340
- MHCDATA.XLS data set, 133
- `%MODELGETCOEFFS` function, 280
- `%MODELLAGMATRIX` function, 281
- `%MODELSETCOEFFS` function, 280
- Multivariate Normal distribution, 321, 334
- Multivariate t distribution, 322
- Muthen, B., 133
- @MVARCHTEST** procedure, 85
- @MVGARCHFORE** procedure, 126
- @MVGARCHTOVECH** procedure, 98
- @MVGARCHVarMats** procedure, 132
- @MVJB** procedure, 136
- @MVQSTAT** procedure, 89
- Nelson, D., 59, 69
- Newey, W., 10
- News Impact Curve, 72
- `%NFREE` variable, 274
- Ng, V., 72
- NLSYSTEM** instruction, 11, 169
- Normal distribution, 316
 - multivariate, 321, 334
- `%NREGMEAN` variable, 72
- `%NVAR` variable, 88
- Nyblom, J., 61
- OILDATA.RAT data set, 272
- `%OUTERXX` function, 199
- Overflow, 341
- Panel GARCH, 183
- `%PARMSPEEK` function, 280
- `%PARMSPOKE` function, 72, 280
- Patton, A., 58
- Perron, P., 38
- PICTURE option, 113
- PITERS option, 165
- PMETHOD option, 165
- Polson, N., 302
- Portfolio weights, 146, 148
- Precision, 335
- Probability distributions
 - GED, 323
 - bivariate normal, 133
 - gamma, 318
 - inverse chi-squared, 320
 - inverse gamma, 319
 - multivariate normal, 321, 334
 - multivariate t, 322
 - normal, 316
 - t, 317

Progress bar, 261

QMLE, 46, 298, 324

Quasi-Maximum Likelihood Estimation, 298, 324

%RAN function, 316

Random walk M-H, 252, 341

%RANFLIP function, 255

%RANLOGKERNEL function, 247

%RANMAT function, 316, 321

%RANMVNORMAL function, 216, 321

%RANMVT function, 216, 322

%RANT function, 216, 317, 322

@REGCORRS procedure, 37, 42, 166

@REGCRITS procedure, 143

%REGENDF function, 56

Rejection method, 343

REPORT instruction, 15

COL=NEW option, 18

FILLBY=COLS option, 17

ROW=NEW option, 18

%RESIDS series, 41

Residuals

multivariate standardized, 88

standardized, 41, 87

Robins, R., 66

Rossi, P., 302

Ruiz, E., 298

Runkle, D., 75

%S function, 14

Sadorsky, P., 148

Sanso, A., 29

SEED instruction, 232

Series handle, 15

Serletis, A., 241, 270, 287

Shephard, N., 298, 304

Simplex iterations, 92

SPGRAPH instruction, 113

Spillover, 96

variance, 100

SSTATS instruction, 2, 7, 219

Standardized residuals, 41, 87

multivariate, 88

START option, 187

STATISTICS instruction, 10

FRACTILES option, 17

Stochastic volatility model, 3, 297

SUMMARIZE instruction, 11

Susmel, R., 31

SVAR-GARCH, 195, 271

Sweep

in MCMC, 339

t distribution, 317

%TCDF function, 317

%TDENSITY function, 317

Test

for ARCH, 35

for CC, 104

dependence, 9

fluctuations, 61

Goldfeld-Quandt, 28

Jarque-Bera, 9, 42

Ljung-Box Q, 10

McLeod-Li, 12

multivariate ARCH, 85

West-Cho, 12

TeX tables, 20

Tiao, G., 28

Tsay, R., 163, 177, 218, 233

Tse, Y.K., 104

@TSECCTEST procedure, 104

%TTEST function, 317

Underflow, 341

Value-at-risk, 217

@VARLAGSELECT procedure, 84

%VEC function, 129

VECH model, 90

VIRF, *see* Volatility impulse response function

Volatility impulse response function, 127

by simulation, 227

%WEEKDAY function, 138

West, K., 4, 10, 22

West-Cho test, 12

WESTCHO-XRATE.XLS data set, 4

@WESTCHOTEST procedure, 12

White, H., 324

Wild bootstrap, 223

Wooldridge, J., 90

XBEKK option, 138

%XT function, 199

%ZTEST function, 316