Chapter 4

TAR Models

Figure 4.1 shows data on the U.S. Civilian Unemployment Rate. Note that the behavior of the series is asymmetrical—the increases are more rapid than the decreases. This is a problem for "linear" time series models like autoregressions or ARIMA models, which, by their nature, have up and down phases which have to have similar properties.

In Example **4.1**, we estimate a linear AR(4) model on the first difference of this series. There's no obvious problem with the estimates, but if we simulate a realization of the estimated model for the unemployment rate, we get Figure 4.2, which, as you can see, shows very similar upwards and downwards movements, and thus hasn't really captured the dynamics of the actual data.

The Threshold Autoregression (TAR) model is an autoregression allowing for two or more branches governed by the values for a threshold variable. This allows for asymmetric behavior that can't be explained by a single ARMA model. For a two branch model, one way to write this is:

$$y_t = \begin{cases} \phi_{11} y_{t-1} + \ldots + \phi_{1p} y_{t-p} + u_t & \text{if } z_{t-d} \leq c \\ \phi_{21} y_{t-1} + \ldots + \phi_{2q} y_{t-q} + u_t & \text{if } z_{t-d} > c \end{cases} \tag{4.1}$$

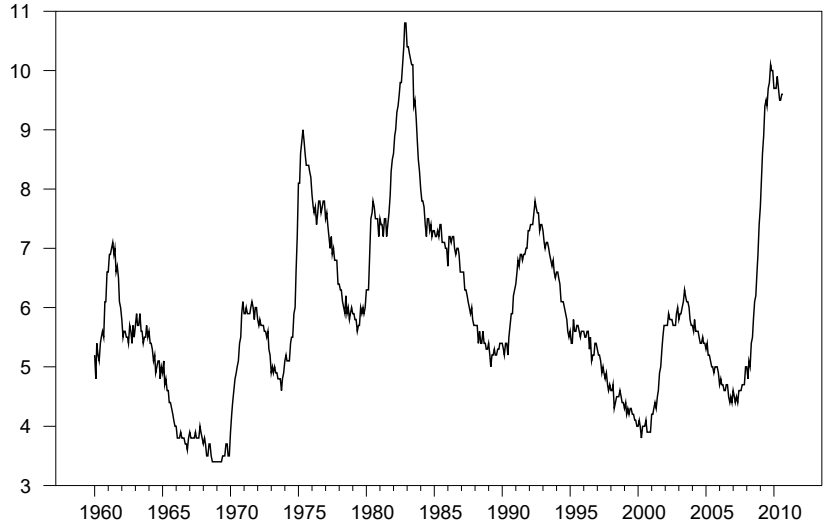

Figure 4.1: U.S. Civilian Unemployment Rate

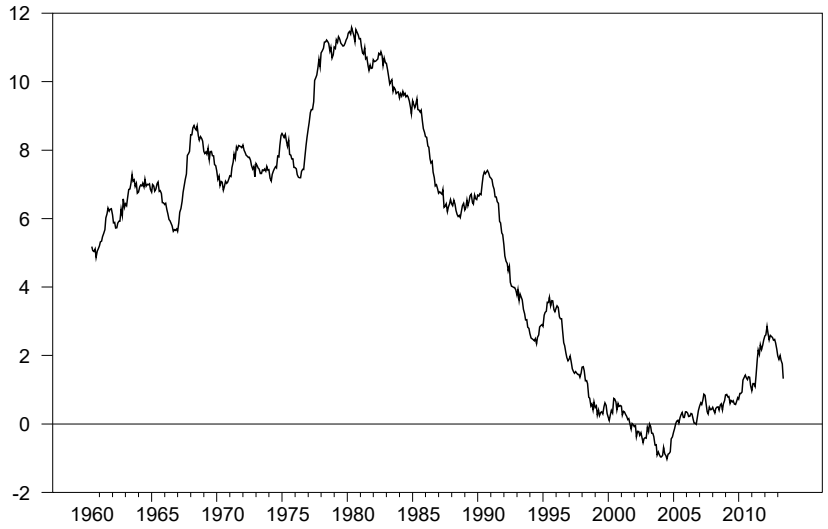

Figure 4.2: Simulated Unemployment Rate from Linear AR

A SETAR model (Self-Exciting TAR) is a special case where the threshold variable is y itself.

In this chapter, we'll work with two data series. Example **4.1** uses the U.S. unemployment rate and employs a standard TAR model. In Example **4.2**, we'll look at the spread between U.S. short and long-run interest rates (Figure 4.3), where we will be using the data set from Enders and Granger (1998). This will use what the authors call the "linear attractor" model:

$$\Delta y_t = \begin{cases} \beta \Delta y_{t-1} + \rho_2 \left(y_{t-1} - a_0 \right) + u_t & \text{if } y_{t-1} \leq a_0 \\ \beta \Delta y_{t-1} + \rho_1 \left(y_{t-1} - a_0 \right) + u_t & \text{if } y_{t-1} > a_0 \end{cases}$$

Unlike (4.1), where the two branches are completely separate, this has β in common, which requires estimating the model with dummies over the full sample, and will thus require some special purpose programming.

Throughout the chapter, we'll switch between discussions of the two examples, since each is very similar analysis applied to different models.

Note that each of these examples might potentially have a problem with round-off error in the data. "Headline" unemployment in the U.S. is usually is presented with just one decimal place, so period to period differences will usually be one of a small number of values (-.2,-.1,0,.1,.2). However, in machine-arithmetic, all .1's are not "equal" because of rounding, and that could cause a problem for "sharp cutoff" models. In this case, we avoid that by using a three decimal place version; alternatively, you can use the `%ROUND` function to force the differenced values to match. See page 65 for more on that.

4.1 Estimation

The first question is how to pick p in (4.1). If there really is a strong break at an unknown value, then standard methods for picking p based upon information

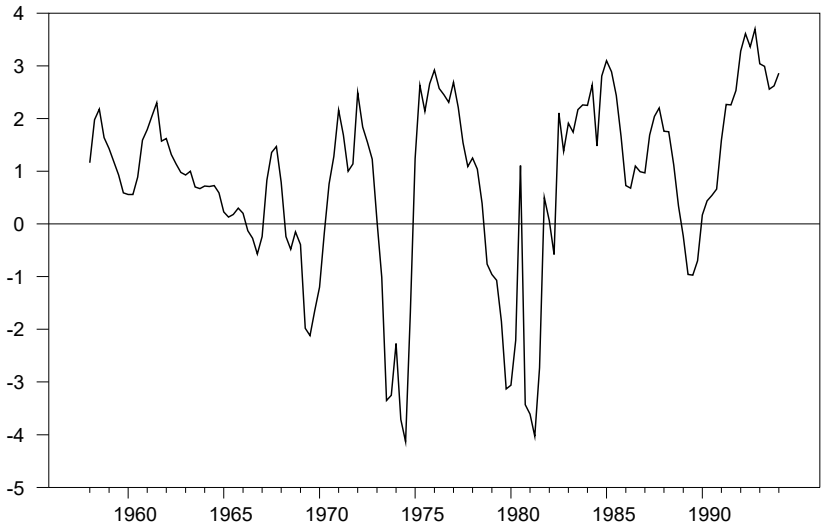

Figure 4.3: U.S. Interest Rate Spread

criteria won't work well, since the full data series doesn't follow an $AR(p)$, and you can't easily just apply those to a subset of the data, like you could if the break were based upon time and not a threshold value. The standard strategy is to overfit to start, then prune out unnecessary lags. While (4.1) is written with the same structure in each branch, in practice they can be different.

The second question is how to pick the threshold z and delay d. For a SETAR model, z is y. Other variables might be suggested by theory, as, for instance, in the Enders-Granger paper, which includes an M-TAR (Momentum TAR) model where z is Δy. The delay is obviously a discrete parameter, so estimating it requires a search over the possible values. c is less obviously discrete, but, in fact, the only values of c at which (4.1) changes are observed values for z_{t-d}, so it's impossible to use variational methods to estimate c. Instead, that will *also* require a search. If both d and c are unknown, a nested search over d and, given a test value of d, over the values of z_{t-d} is necessary.

4.2 Testing

The test which suggests itself is a likelihood ratio test or something similar which compares the one-branch model with the best two-branch model. The problem with that approach is that the test statistic will not have a standard asymptotic distribution because of both the lack of differentiability with respect to c, plus the lack of identification of c if there is no difference between the branches. The fixed regressor bootstrap (Section 3.3) can be used to approximate the p-value for the standard test. First, however, we'll look at another procedure isn't quite as powerful, but is much quicker.

4.2.1 Arranged Autoregression Test

The Arranged Autoregression Test (Tsay (1989)) first runs recursive estimates of the autoregression with the data points added, not in the conventional time series order, but in the order of the test threshold variable. Under the null of no break, the residuals should be fairly similar to the least squares residuals, so there should be no correlation between the recursive residuals and the regressors, and Tsay shows that under the null, an exclusion test for the full coefficient vector in a regression of the recursive residuals on the regressor set is asymptotically a chi-square (or approximated by a small-sample F).[1] If there *is* a break, and you have the correct threshold variable, then the recursive estimates will be expected to be quite different for the entries with low values of the threshold than they will be for the high values, and we would likely see some correlation in regressing the recursive residuals on the regressors, which would show as a significant exclusion test. In effect, this does the testing procedure in reverse order—rather than do the standard regression first and then see if there is a correlation between the residuals and regressors across subsamples, this does the "subsamples" first, and sees if there is a correlation with the full sample.

This is quite simple to do with RATS because the **RLS** instruction allows you to provide an ORDER series which does the estimates in the given order, but keeps the original alignment of the entries. This is already implemented in the **@TSAYTEST** procedure, which, if you look at it, is rather short. **@TSAYTEST** can be applied with any threshold series, not just a lag of the dependent variable—use the THRESHOLD option to provide the threshold series:

```
@tsaytest(thresh=dur,d=1) dur
# constant dur{1 to 4}
```

which produces:

```
TSAY Arranged Autoregression Test
F(5,627) 3.071 Signif 0.010
```

4.2.2 Direct Threshold Tests

The direct test for the threshold can be done with either of two procedures. **@THRESHTEST** is the more general procedure, which can be used for any linear regression with any threshold series. Both of these use the fixed regressor bootstrap to approximate the p-values.

```
@threshtest(thresh=dur,d=1,nreps=500) dur
# constant dur{1 to 4}
```

[1] Note that this isn't the same as the conventional regression F statistic, which doesn't test the intercept. It's a test on the full vector, including the constant.

```
Test of Null of No Threshold Against Alternative of Threshold
Under Maintained Assumption of Homoskedastic Errors

Dependent Variable      DUR
Threshold Variable      DUR{1}
Observations               637
From                    1960:06
To                      2013:06
Maximum F-Test           5.2149
Achieved at              0.0250
Bootstrap Replications      500
Bootstrap P-Value        0.0020
```

The other is the **@TAR** procedure, which is specifically designed for working with (self-exciting) TAR models. It tests *all* the lags as potential thresholds and reports the one which gives the most significant break test.

```
@tar(p=4,nreps=500) dur
```

For the unemployment rate, this gives a rather unambiguous result supporting a threshold effect:

```
Threshold Autoregression
Threshold is DUR{2}=0.0220
Tests for Threshold Effect use 500 draws
SupLM           25.746219 P-value          0.002000
ExpLM            9.382281 P-value          0.000000
AveLM           11.404145 P-value          0.004000
```

The three variations of the test statistic are the maximum LM, the Andrews-Ploberger exponential average (page 35) and the arithmetic average. The **@TAR** procedure chooses lag 2 as the threshold series with the most significant test, however, for further work, we'll use lag 1.

These procedures won't work with the linear attractor model of the Enders-Granger paper since it has a common regressor. In addition, the presence of a_0 in the regression function itself, not just in the regime-selection criterion, means that the function changes, not just at the observed values of y_{t-1}, but also in between—it will be discontinuous at the observed values (since the subsamples change) and continuous but not constant in between. Instead of searching over the observed values of the threshold variable, knowing that the optimum has to be at one of them, we'll use a fine grid of values, hoping that we'll be close to the optimum. (We can't use a variational method like non-linear least squares because of the discontinuities at the observed threshold values).

This generates a grid with 501 points running from the 15%-ile to the 85%-ile of the threshold series. The search is initialized at the sum of squared residuals for a restricted model (symmetrical adjustments).

```
set thresh = spread{1}
*
compute trim=.15
compute startl=%regstart(),endl=%regend()
set copy startl endl = thresh
compute limits=%fractiles(copy,||trim,1-trim||)
compute grid=%seqrange(limits(1),limits(2),501)
compute ssqmin=%rss
```

This next part runs over the test values in the grid, generating the "plus" and "minus" versions of the difference between the threshold series and the test. Given that, the estimates can be done with a **LINREG**. Inside the loop, we find the smallest sum of squares and save the threshold value which gives that.

```
dofor [real] test = grid
   set splus  = %max(thresh-test,0.0)
   set sminus = %min(thresh-test,0.0)
   linreg(noprint) ds
   # splus sminus ds{1}
   if %rss<ssqmin
      compute ssqmin=%rss,breakvalue=test
end do time
disp "For Linear Attractor model, best SSQ is " ssqmin $
       " at " breakvalue
```

The results are:

```
For Linear Attractor model, best SSQ is      101.81867  at       -0.27393
```

which is a bit disappointing, since the restricted model has a sum of squares of just 103.74 so the improvement with the TAR model is fairly minor. The F or LR tests for asymmetry have the typically non-standard distributions, but here they end up being insignificant at *conventional* levels. However, for illustration, we'll stick with this model.

4.3 Forecasting

A TAR model is a self-contained dynamic model for a series. However, to this point, there has been almost no difference between the analysis with a truly exogenous threshold and a threshold formed by lags of the dependent variable—whether the threshold is endogenous or exogenous, for estimation purposes it's observable. Once we start to forecast, that's no longer the case—we need the link between the threshold and the dependent variable to be included in the model.

While the branches may very well be (and probably are) linear, the connection isn't, so we need to use a non-linear system of equations. For the unemployment series, the following estimates the two branches, given the

identified break (computed using the **@THRESHTEST** procedure, which defines %%BREAKVALUE as the value giving the most extreme break test statistic—BREAK1 is a copy of that).

```
linreg(smpl=dur{1}<=break1,frml=branch1) dur
# constant dur{1 to 4}
compute rss1=%rss,ndf1=%ndf
linreg(smpl=dur{1}>break1,frml=branch2) dur
# constant dur{1 to 4}
compute rss2=%rss,ndf2=%ndf
compute seesq=(rss1+rss2)/(ndf1+ndf2)
```

The forecasting equation is created by gluing these two branches together with:

```
frml(variance=seesq) tarfrml dur = $
       %if(dur{1}<=break1,branch1,branch2)
```

The following is for accumulating back the unemployment rate from the change, and combines the non-linear equation plus the identity into a model:

```
frml(identity) urid unrate = unrate{1}+dur
group tarmodel tarfrml urid
```

Because of the non-linearity of the model, the minimum mean square forecast can't be computed using the point values done by **FORECAST**, but need the average across simulations.

```
compute fstart=2009:1,fend=2010:06
set urfore fstart fend = 0.0
compute ndraws=5000
do draw=1,ndraws
   simulate(model=tarmodel,from=fstart,to=fend,$
              results=sims,noprint)
   set urfore fstart fend = urfore+sims(2)
end do draw
set urfore fstart fend = urfore/ndraws
```

This produces Figure 4.4, which also includes actual data through the end of 2008.

This next estimates the rate spread model at the chosen break value, and converts it into a non-linear FRML using an explicit reference to spread1 as the threshold. (B rather than %BETA is used for the coefficients so the FRML won't be damaged if you do another estimation later). As with the unemployment model, we have a separate identity in the model which accumulates the changes in the spread to create the spread itself.

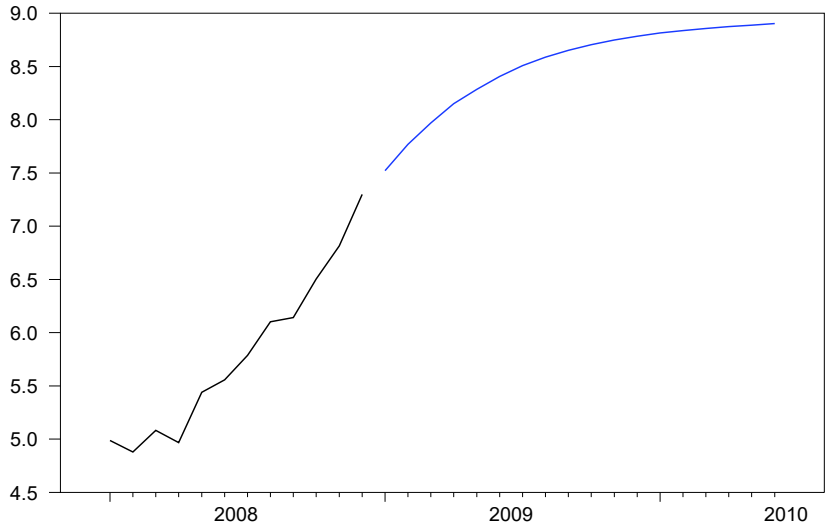

Figure 4.4: Unemployment Forecast from TAR Model

```
set splus  = %max(thresh-breakvalue,0.0)
set sminus = %min(thresh-breakvalue,0.0)
linreg(print) ds
# splus sminus ds{1}
*
compute b=%beta
frml adjust ds = b(3)*ds{1}+$
                 b(1)*%max(spread{1}-breakvalue,0.0)+$
                 b(2)*%min(spread{1}-breakvalue,0.0)
frml(identity) spreadid spread = spread{1}+ds
*
group latarmodel adjust spreadid
```

This does the forecasts by the same process, here starting at the end of the sample, where the spread is relatively high (Figure 4.5). Notice that the forecasts are converging to roughly .85, *not* the estimated "attractor" value of -0.27393. Both branches of the model push the process towards the attractor, but the "minus" branch has a substantially more negative coefficient (-.286 vs -.065). If the residual standard deviation were very small, the mean would indeed converge to the attractor. Here, however, it's nearly .9. Shocks which push the process below the attractor are much more quickly reversed than shocks which push the process above it, so the process mean is above the attractor (fairly substantially so). How *far* above it depends upon the variance, which is another major difference with linear models, where the variance doesn't affect the mean at all.

```
compute ndraws=5000
compute nsteps=60
compute istart=1994:2,iend=istart+nsteps-1
set highspread istart iend = 0.0
do draw=1,ndraws
   simulate(model=latarmodel,from=istart,to=iend,$
       results=sims,cv=%seesq)
   set highspread istart iend = highspread+sims(2)
end do draw
set highspread istart iend = highspread/ndraws
graph(footer="Forecasts from High Spread Initial Conditions")  2
# spread istart-12 istart-1
# highspread
```

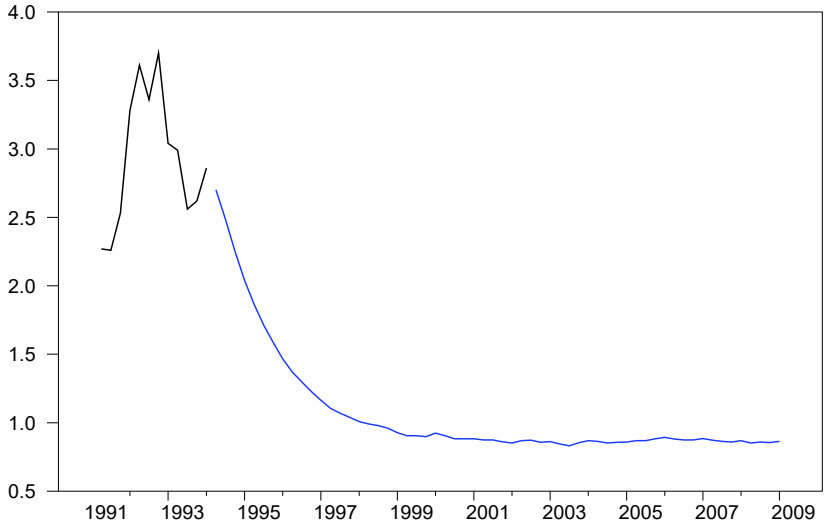

Figure 4.5: Forecasts of Spread from Linear Attractor TAR

4.4 Non-linear Impulse Responses

All threshold models are non-linear. As a result, there is no single "impulse response function". For a linear model (like a VAR), the effect of superimposing a shock is the same regardless of the past values for the data—the response to a linear combination of initial shocks is the same linear combination of responses from the component shocks. Now, in some ways, this is unrealistic—the true response of the economy to a 100 point shock in the interest rate is unlikely to be anything like 100 times the response to a 1 point shock, that is, the linearity of a standard VAR is only approximate and doesn't extend to atypical behavior. But for a *non-linear* model, the effects of a shock can be quite different even with historically typical shock sizes in historically typical situations.

The linear impulse response function is computed by zeroing out the initial conditions and simulating the model with a fixed set of first period shocks,

whether unit shocks or some other pattern. Neither part of this is going to be a proper practice with a TAR or similar model. First, the response, in general, will be very sensitive to the initial conditions—if we're far from the threshold in either direction, any shock of reasonable size will be very unlikely to cause us to shift from one branch to the other, so the differential effect will be whatever the dynamics are of the branch on which we started. On the other hand, if we're near the threshold, a positive shock will likely put us into one branch, while a negative shock of equal size would put us into the other, and (depending upon the dynamics), we might see the branch switch after a few periods of response.

The *non-linear impulse response* is computed by simulating the model with and without a fixed initial shock, and averaging the difference between those across a large number of simulations. This is often called the *generalized impulse response* as the definition is from Koop, Pesaran, and Potter (1996), however, we'll use the term non-linear rather than generalized as the latter term is related more specifically to the implications of the technique to VAR's rather than non-linear models.

Non-linear IRF's depend upon the initial conditions. Quite a few different methods have been proposed to deal with that, and you should never simply say that you did non-linear IRF's without being clear about this. With few exceptions, the initial conditions are based upon histories observed in the actual data. With these you can:

1. Pick specific entries which have properties of interest.
2. Average across all possible histories, which will give an "average" response. This can either be done as a simple average, or by bootstrap draws from all possible histories.
3. Similar to #2, but restricting the histories to those which have certain properties. This will give you an average response for that subset of entries.

#2 is the closest to the IRF in a linear model, since you're not singling out any special conditions. However, it does lose some of the special nature of the non-linear model.

There's also a question of whether you should use Gaussian draws or bootstrapping to do the simulations—the difference between the two should not be all that great, and Gaussian draws are generally substantially simpler. And, because the shape (and not just the size) of the IRF depend upon the size and sign of the shocks, there are different ways to present that information—you can show responses to + and - shocks, or + shocks of different scales, or even + and - shocks of different scales. With the shocks of different scales, one possibility is to divide those through by the size of the shock, to allow a more direct comparison of the *shapes*. Again, it's important to document exactly what you

do, and it's important to read information very carefully in papers to see what was done (assuming the paper is careful to document it).

Computing non-linear impulse responses in very similar to the forecast procedure, but the random draws must be controlled better, so, instead of **SIMULATE**, you use **FORECAST** with PATHS.

For the spread model, this computes the non-linear IRF to a positive unit shock (the delta variable) based on 1974:3, where the spread is large and negative. When doing non-linear IRF's, it's important to use shock sizes that are reasonable for the model—here, the unit shock isn't just a convenient number, but also a typical size.

```
compute ndraws=5000
compute nsteps=60
compute stddev=sqrt(%seesq)
compute delta=1.0
compute istart=1974:4,iend=istart+nsteps-1
set nlirf istart iend = 0.0
*
do draw=1,ndraws
   set shocks = %ran(stddev)
   forecast(paths,model=latarmodel,from=istart,to=iend,$
      results=basesims)
   # shocks
   compute shocks(istart)=delta
   forecast(paths,model=latarmodel,from=istart,to=iend,$
      results=sims)
   # shocks
   set nlirf istart iend = nlirf+(sims(2)-basesims(2))
end do draw
*
set nlirf istart iend = nlirf/ndraws
graph(number=0,footer="NL-IRF for Spread at 1974:3")
# nlirf istart iend
```

The first **FORECAST** will be identical to what you would get with **SIMULATE**. The second replaces the first period shock with a shock of a specific size. The NL-IRF's at two different sets of initial conditions are shown in Figures 4.6 and 4.7 (1974:3 has a historically low value of the spread and 1994:1 a historical high value). Remember that these are *not* forecasts—they are the differences between forecasts with and without the added initial $+1$ shock to the spread. With a stationary process, these will have to converge to zero.

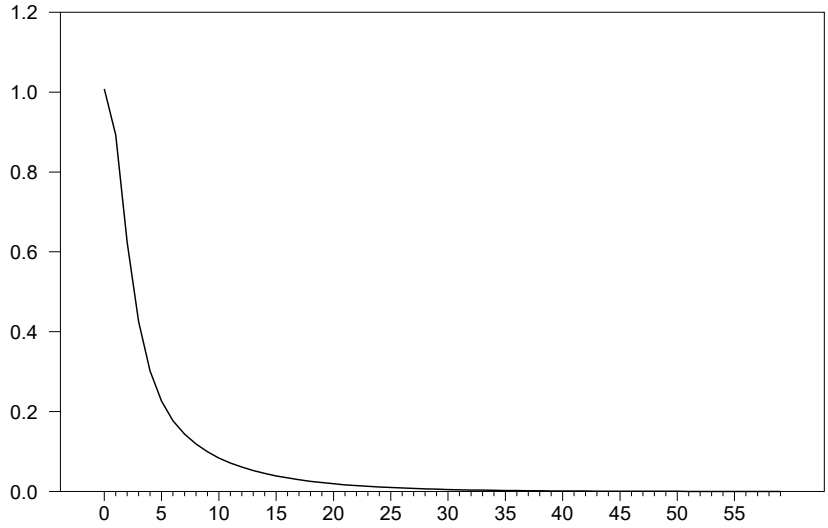

Figure 4.6: NL-IRF for Spread at 1974:3

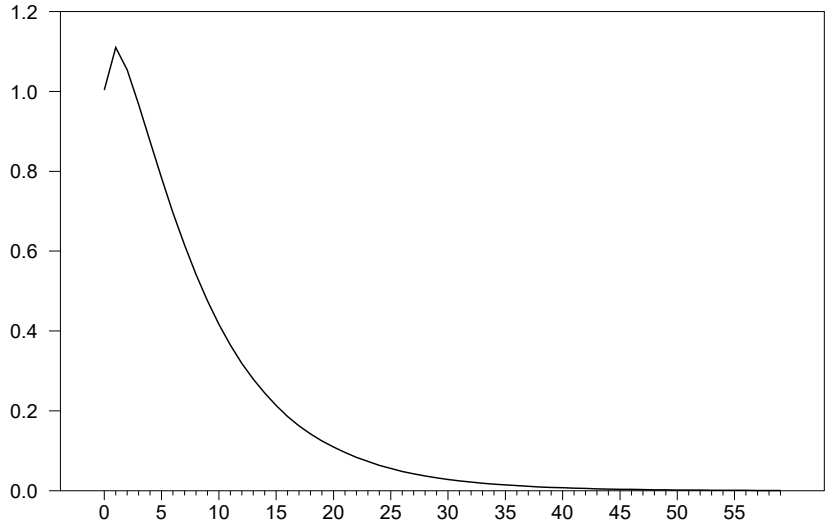

Figure 4.7: NL-IRF for Spread at 1994:1

4.5 Tips and Tricks

Rounding the Threshold Series

You may need to be careful creating a threshold series such as the difference in the unemployment rate or interest rate differentials. The (headline) U.S. unemployment rate is reported at just one decimal digit, and it's a relatively slow-moving series. As a result, almost all the values for the difference are one of -.2, -.1, 0, .1 or .2. However, in computer arithmetic, all .1's are not the same—due to machine rounding there can be several values which disagree in the 15th digit when you subtract. (Fortunately, zero differences will be correct). Because the test in (4.1) has a sharp cutoff, it's *possible* for the optimal threshold to come out in the middle of (for instance) the .1's, since some will be (in computer arithmetic) larger than others. While this is unlikely (since the rounding errors are effectively random with respect to time), you can protect against it by using the `%ROUND` function to force all the similar values to map to exactly the same result. We're using a different reporting of the unemployment rate which uses three (rather than one) decimal digits, so it doesn't have this issue. The interest rate spread is reported at two digits, and takes many more values, so while we include the rounding in our program, it is quite unlikely that rounding error will cause a problem. This forces all two digit differences to have exactly the same value:

```
set spread = %round(r_10-r_short,2)
```

Example 4.1 TAR Model for Unemployment

This is an example of a SETAR model applied to the U.S. unemployment rate series. It is based upon an example from Enders (2015).

```
seed 550103
*
calendar(m) 1960:1
open data unrate.xls
data(format=xls,org=columns) 1960:01 2013:6
*
graph(footer="The U.S. Unemployment Rate",vlabel="Percent")
# unrate
*
@bjident(diffs=1) unrate
*
* Note well that this data set has unemployment calculated to three
* decimal places, not the "headline" US unemployment numbers which
* just use one. At one decimal place, there are too few distinct
* values to use the difference as a threshold variable (almost all
* differences are -.2, -.1, 0, .1 or .2).
*
set dur = unrate-unrate{1}
*
@arautolags(crit=hq) dur
@arautolags(crit=bic) dur
*
* Estimate the linear AR model suggested by @ARAutoLags
*
linreg dur
# constant dur{1 2 3 4}
*
* Simulate and graph an example with the linear AR model
*
uforecast(simulate) sdur %regstart() %regend()
*
* Transform back to levels from differences
*
set sur %regstart() %regend() = $
      %if(t==%regstart(),unrate{1},sur{1})+sdur
graph(footer="Simulated UR with Linear AR")
# sur
*
* Do Tsay arranged regression test
*
@tsaytest(thresh=dur,d=1) dur
# constant dur{1 to 4}
@threshtest(thresh=dur,d=1,nreps=500) dur
# constant dur{1 to 4}
compute break1=%%breakvalue
*
* Do TAR test
*
```

```
@tar(p=4,nreps=500) dur
*
* Estimate the two branches and define equations
*
linreg(smpl=dur{1}<=break1,frml=branch1) dur
# constant dur{1 to 4}
compute rss1=%rss,ndf1=%ndf
linreg(smpl=dur{1}>break1,frml=branch2) dur
# constant dur{1 to 4}
compute rss2=%rss,ndf2=%ndf
compute seesq=(rss1+rss2)/(ndf1+ndf2)
*
* Define the forecasting model
*
frml(variance=seesq) tarfrml dur = $
      %if(dur{1}<=break1,branch1,branch2)
frml(identity) urid unrate = unrate{1}+dur
group tarmodel tarfrml urid
*
* Compute average across simulations
*
compute fstart=2009:1,fend=2010:06
set urfore fstart fend = 0.0
compute ndraws=5000
do draw=1,ndraws
   simulate(model=tarmodel,from=fstart,to=fend,$
        results=sims,noprint)
   set urfore fstart fend = urfore+sims(2)
end do draw
set urfore fstart fend = urfore/ndraws
graph(footer="Out-of-sample forecasts") 2
# unrate fstart-12 fstart-1
# urfore
*
* Average "impulse response"
* We need to simulate over the forecast range, but also need to
* control the initial shock. The results of this will depend upon
* the starting point (ISTART).
*
* A 0.25 shock is not atypical at the start of a recession.
*
compute delta=0.25
compute stddev=sqrt(seesq)
compute ndraws=5000
compute nsteps=36
*
compute istart=2009:1,iend=istart+nsteps-1
set nlirf istart iend = 0.0
*
do draw=1,ndraws
   *
   * Do a simulation with fully randomized shocks. These are
   * controlled so we can reuse them.
   *
```

```
   set shocks istart iend = %ran(stddev)
   forecast(paths,model=tarmodel,from=istart,to=iend,results=basesims)
   # shocks
   *
   * Do a second simulation using the previous shocks beyond the
   * initial period and the controlled shock in the first period of
   * forecast.
   *
   compute shocks(istart)=delta
   forecast(paths,model=tarmodel,from=istart,to=iend,results=sims)
   # shocks
   *
   * Add the difference between the two simulations to the NLIRF
   * series. The unemployment rate itself is the dependent variable
   * of the second equation.
   *
   set nlirf istart iend = nlirf+(sims(2)-basesims(2))/delta
end do draw
*
set nlirf istart iend = nlirf/ndraws
graph(number=0,footer="NL-IRF for unemployment at "+%datelabel(istart))
# nlirf
*
compute stddev=sqrt(%seesq)
clear nlirf
*
* Same thing, different initial time period
*
compute istart=1984:1,iend=istart+nsteps-1
set nlirf istart iend = 0.0
do draw=1,ndraws
   set shocks istart iend = %ran(stddev)
   forecast(paths,model=tarmodel,from=istart,to=iend,results=basesims)
   # shocks
   compute shocks(istart)=delta
   forecast(paths,model=tarmodel,from=istart,to=iend,results=sims)
   # shocks
   set nlirf istart iend = nlirf+(sims(2)-basesims(2))/delta
end do draw
*
set nlirf istart iend = nlirf/ndraws
graph(number=0,footer="NL-IRF for unemployment at "+%datelabel(istart))
# nlirf istart iend
```

Example 4.2 TAR Model for Interest Rate Spread

This is an example of a threshold autoregressive model with a restriction to require a common coefficient between the regimes. This makes analysis more difficult because the two regimes have to be estimated together. This is based upon Enders and Granger (1998).

```
open data granger.xls
```

```
calendar(q) 1958:1
data(format=xls,org=columns) 1958:01 1994:01 date r_short r_10
*
* The %ROUND function is used to make sure all spreads which should
* be the same actually are. If you don't use this, various
* differences which should be .01 will come out slightly different
* due to round off error.
*
set spread = %round(r_10-r_short,2)
graph(footer="Interest rate spread")
# spread
*
* ADF Test. @ADFAutoSelect picks either 0 or 1 augmenting lags for
* most of the criteria. The test statistic shown on @ADFAutoSelect
* is slightly different from that on @DFUnit with the same number
* of lags because the first procedure uses a common range for all
* lags tested.
*
@adfautoselect(print,det=constant) spread
@dfunit(lags=1,det=constant) spread
*
* RESET tests of the ADF regression.
*
set ds = spread-spread{1}
*
* This is equivalent to the restricted form of the model (with
* "symmetrical adjustment").
*
linreg ds
# constant spread{1} ds{1}
*
* Do RESET tests
*
@regreset(h=3)
@regreset(h=4)
*
set thresh = spread{1}
*
compute trim=.15
compute startl=%regstart(),endl=%regend()
set copy startl endl = thresh
compute limits=%fractiles(copy,||trim,1-trim||)
compute grid=%seqrange(limits(1),limits(2),501)
*
compute ssqmin=%rss
dofor [real] test = grid
   set splus  = %max(thresh-test,0.0)
   set sminus = %min(thresh-test,0.0)
   linreg(noprint) ds
   # splus sminus ds{1}
   if %rss<ssqmin
      compute ssqmin=%rss,breakvalue=test
end do time
disp "For Linear Attractor model, best SSQ is " ssqmin $
```

```
        " at " breakvalue
*
* Further analysis of the model.
* Generate a self-contained model of the process, estimated at the
* least squares fit.
*
set splus  = %max(thresh-breakvalue,0.0)
set sminus = %min(thresh-breakvalue,0.0)
linreg(print) ds
# splus sminus ds{1}
*
* To do any forecasting, we need to write this in terms of
* spread{1}, which will be generated by the model.
*
compute b=%beta
frml adjust ds = b(3)*ds{1}+$
                 b(1)*%max(spread{1}-breakvalue,0.0)+$
                 b(2)*%min(spread{1}-breakvalue,0.0)
frml(identity) spreadid spread = spread{1}+ds
*
group latarmodel adjust spreadid
*
* Starting from end of sample, where spread is relatively high
*
compute ndraws=5000
compute nsteps=60
compute istart=1994:2,iend=istart+nsteps-1
set highspread istart iend = 0.0
do draw=1,ndraws
   simulate(model=latarmodel,from=istart,to=iend,$
       results=sims,cv=%seesq)
   set highspread istart iend = highspread+sims(2)
end do draw
set highspread istart iend = highspread/ndraws
graph(footer="Forecasts from High Spread Initial Conditions")  2
# spread istart-12 istart-1
# highspread
*
* Starting from 1974:3, where the spread is large negative.
*
compute istart=1974:4,iend=istart+nsteps-1
set lowspread istart iend = 0.0
compute ndraws=5000
do draw=1,ndraws
   simulate(model=latarmodel,from=istart,to=iend,$
      results=sims,cv=%seesq)
   set lowspread istart iend = lowspread+sims(2)
end do draw
set lowspread istart iend = lowspread/ndraws
graph(footer="Forecasts from Low Spread Initial Conditions") 2
# spread istart-12 istart-1
# lowspread
*
* Non-linear impulse responses
```

```
* We need to simulate over the forecast range, but also need to
* control the initial shock. The results of this will depend upon
* the starting point.
*
compute ndraws=5000
compute nsteps=60
compute stddev=sqrt(%seesq)
compute delta=1.0
compute istart=1974:4,iend=istart+nsteps-1
set nlirf istart iend = 0.0
*
do draw=1,ndraws
   set shocks = %ran(stddev)
   forecast(paths,model=latarmodel,from=istart,to=iend,$
      results=basesims)
   # shocks
   compute shocks(istart)=delta
   forecast(paths,model=latarmodel,from=istart,to=iend,$
      results=sims)
   # shocks
   set nlirf istart iend = nlirf+(sims(2)-basesims(2))
end do draw
*
set nlirf istart iend = nlirf/ndraws
graph(number=0,footer="NL-IRF for Spread at 1974:3")
# nlirf istart iend
*
compute istart=1994:2,iend=istart+nsteps-1
clear nlirf
set nlirf istart iend = 0.0
do draw=1,ndraws
   set shocks istart iend = %ran(stddev)
   forecast(paths,model=latarmodel,from=istart,to=iend,$
      results=basesims)
   # shocks
   compute shocks(istart)=delta
   forecast(paths,model=latarmodel,from=istart,to=iend,$
      results=sims)
   # shocks
   set nlirf istart iend = nlirf+(sims(2)-basesims(2))
end do draw
*
set nlirf istart iend = nlirf/ndraws
graph(number=0,footer="NL-IRF for Spread at 1994:1")
# nlirf
```