

RATS Handbook for State-Space 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 State-Space Models	1
1.2 Kalman Filtering with the Local Linear Model	4
1.3 Kalman Filtering with the Time-Varying Coefficients Model	9
1.4 Kalman Smoothing	12
1.5 Kalman Smoothing with the Local Level Model	13
1.6 Forecasts and Missing Data	14
1.7 RATS Tips and Tricks	16
1.1 Kalman Filter: Nile Data	20
1.2 Kalman Filter: Drug Sales Data	21
1.3 Kalman Filter: Time-Varying Coefficients Model	22
1.4 Kalman Smoothing: Nile Data	23
1.5 Kalman Smoothing: Estimated Errors	23
1.6 Missing Data	24
1.7 Out of Sample Forecasting	25
2 More States	26
2.1 Kalman Filter in Matrix Form	26
2.2 ARMA Processes	27
2.3 Local Trend Model	29
2.4 Seasonals	32
2.1 Local Level vs Local Trend Model	36
3 Estimating Variances	37
3.1 The Likelihood Function	37
3.2 Estimating the Local Level Model	39
3.3 Estimating the Local Trend Model	40

3.4	Diagnostics	42
3.1	Estimating the Local Level Model	43
3.2	Estimating the Local Trend Model	44
3.3	Diagnostics	45
4	Initialization	47
4.1	Ergodic Solution	47
4.2	Diffuse Prior	48
4.3	Mixed Stationary and Non-Stationary Models.	51
5	Practical Examples with a Single Observable	53
5.1	Basic Structural Models.	53
5.2	Trend plus Stationary Cycle.	56
5.3	Gap Models	66
5.4	Time-Varying Coefficients in a Linear Model	81
5.4.1	Fixed Coefficients	81
5.4.2	Time-varying coefficients	84
5.4.3	TVC with empirical drift variance	89
5.1	Airline Data	91
5.2	Clark Model	93
5.3	Trend plus Stationary Cycle Model	94
5.4	Gap Model with Unrestricted Estimates	96
5.5	Gap Model with Restrictions	100
5.6	Linear regression with fixed coefficients	102
5.7	Linear regression with TVC, diagonal drift variance	103
5.8	Linear regression with TVC, empirical drift variance	104
6	Practical Examples with Multiple Observables	106
6.1	Indicator Models	106
6.2	Multivariate H-P Filter	111
6.3	Dynamic (Multiple) Factor Models	113
6.4	Gap Models	123
6.1	Stock-Watson Indicator Model	131

6.2	Bivariate H-P Filter	133
6.3	Dynamic Factor Model with Theoretical Loadings	135
6.4	Dynamic Factor Model, Principal Components	136
6.5	Bivariate Gap model	138
7	Interpolation and Distribution	143
7.1	Linear Model.	144
7.2	Log-Linear Model	145
7.3	Proportional Denton Method	146
7.1	Proportional Denton method	148
8	Simulation Methods	149
8.1	Conditional Simulation	149
8.2	Gibbs Sampling	150
8.2.1	Local Level Model	151
8.2.2	Time-varying Coefficients	156
8.2.3	Models with Regressions/Autoregressions	164
8.1	Simulations	173
8.2	Gibbs Sampling with Local Level Model	174
8.3	TVC Model, Gibbs Sampling with Independent Drift Variances	175
8.4	TVC Model, Gibbs Sampling with Correlated Empirically-Based Drift	177
8.5	Gibbs Sampling with Linear Regression with AR errors	180
9	Non-Normal Errors and Non-linear Models	184
9.1	Stochastic volatility model	184
9.2	t Distributed Errors	186
9.3	Non-linearities in the Equations	188
9.4	Particle filtering	196
9.5	Restrictions on the States	205
9.1	Stochastic Volatility Model	212
9.2	Fat-Tailed Errors	213
9.3	Non-Linear Kalman Filter	215
9.4	Particle Filtering	221
9.5	Constrained Kalman Filter	225

10 DSGE: Setting Up and Solving Models	228
10.1 Requirements	230
10.2 Adapting to different information sets	231
10.3 Non-linear models	232
10.4 Unit Roots	234
10.5 Dynare scripts	235
11 DSGE: Applications	237
11.1 Simulations	237
11.2 Impulse Responses	239
11.1 DSGE Simulation	241
11.2 DSGE Impulse Response Functions	242
12 DSGE: Estimation	243
12.1 Maximum Likelihood	244
12.2 Bayesian Methods	248
12.3 Tips and Tricks	255
12.1 Maximum Likelihood: Hyperinflation Model	256
12.2 Maximum Likelihood: Hansen RBC	257
12.3 Bayesian Estimation: Hyperinflation Model	259
A Probability Distributions	263
A.1 Uniform	263
A.2 Univariate Normal	264
A.3 Beta distribution	265
A.4 Gamma Distribution	266
A.5 Inverse Gamma Distribution	267
A.6 Chi-Squared Distribution	268
A.7 (Scaled) Inverse Chi-Squared Distribution	269
A.8 Bernoulli Distribution	270
A.9 Multivariate Normal	271
A.10 Wishart Distribution	272
A.11 Inverse Wishart Distribution	273

Contents	v
B Properties of Multivariate Normals	275
C Non-Standard Matrix Calculations	278
D A General Result on Smoothing	281
E Generalized Ergodic Initialization	282
F Gibbs Sampling and Markov Chain Monte Carlo	285
G Importance Sampling	289
H VAR Likelihood Function	292
I Quasi-Maximum Likelihood Estimations (QMLE)	295
J GNU Free Documentation License	298
1. APPLICABILITY AND DEFINITIONS	298
2. VERBATIM COPYING	300
3. COPYING IN QUANTITY.	300
4. MODIFICATIONS	301
5. COMBINING DOCUMENTS	303
6. COLLECTIONS OF DOCUMENTS	304
7. AGGREGATION WITH INDEPENDENT WORKS	304
8. TRANSLATION	304
9. TERMINATION	305
10. FUTURE REVISIONS OF THIS LICENSE	305
11. RELICENSING	306
Bibliography	307
Index	310

Preface

The presentation will cover most of Durbin & Koopman's (2012) *Time Series Analysis by State Space Methods, 2nd Edition*, with some additions from Harvey's (1989) *Forecasting, structural time series and the Kalman filter* and West & Harrison's (1997) *Bayesian Forecasting and Dynamic Models*. The textbook replication examples of all of those have many other examples of state-space models done using **DLM**. In addition to those, Commandeur & Koopman (2007) is an introductory book on state-space modeling, Kim & Nelson (1999) is a mix of state-space modeling and switching models (where the combined models are covered as part of the *Structural Breaks and Switching Models* e-course, and Novales et al. (2009) extensively uses **DSGE** to solve growth models and then **DLM** to simulate them.

This second edition is greatly expanded compared with the first, with

- new sections on “gap” models with univariate models (5.3), bivariate models for gap plus Phillips curve (6.4), and non-linear dynamic models (9.3)
- greatly increased coverage of time-varying coefficients models (Sections 5.4 and 8.2.2)
- a new section on Dynamic (Multiple) Factor Models (6.3)
- a detailed discussion of Gibbs sampling methods (8.2)
- a new section on particle filters (9.4)
- a new section on Kalman filtering with constraints on the states (9.5)

We use bold-faced Courier (for instance, **DLM**) for any use of RATS instruction names within the main text, and non-bolded Courier (%SCALAR) 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 also available.

Introduction

1.1 State-Space Models

State-space models are dynamic linear models, which have a broad usefulness in many areas of econometrics, either directly in their linear form, or indirectly, as an approximation to non-linear dynamics.

The general form of a state-space model has quite a few components: unobservable states, observable data, shocks, mapping matrices. The RATS **DLM** instruction has as many as twelve inputs, and almost as many potential outputs.

Unfortunately, almost every description that you will run across chooses its own scheme for labeling these components. The one that we use in the RATS manual, which we'll use here as well, uses names of the corresponding options on the **DLM** instruction:

$$\mathbf{X}_t = \mathbf{A}_t \mathbf{X}_{t-1} + \mathbf{Z}_t + \mathbf{F}_t \mathbf{W}_t \quad (1.1)$$

$$\mathbf{Y}_t = \mu_t + \mathbf{C}_t' \mathbf{X}_t + \mathbf{V}_t \quad (1.2)$$

The \mathbf{X} 's are the unobservable *state variables*. The \mathbf{Y} 's are the observable data. The first equation is the *state* or *transition equation*, the second is the *observation* or *measurement equation*. The \mathbf{Z} 's, if present, are exogenous variables in the evolution of the states. The \mathbf{W} 's are shocks to the states; the \mathbf{F} matrix has the loadings from those shocks to states, which allows for there to be fewer shocks than states (which will generally be the case). The μ are any components in the description of the observable which depend upon exogenous variables (such as a mean in the form of a linear regression). The \mathbf{V} 's are measurement errors. The \mathbf{W} 's and \mathbf{V} 's are assumed to be mean zero, Normally distributed, independent across time and independent of each other at time t as well (though there may be correlation within the elements of \mathbf{W} or \mathbf{V}).

The **DLM** instruction was introduced with RATS version 5.0 in 2001. There have been many improvements to **DLM**, and additions to RATS generally, that have made it much easier to work with state-space models in RATS. The “Tips and Tricks” section (1.7) can help you with translations of older examples to use the new features.

The model (1.1) and (1.2) is too broad for many purposes, and too narrow for others. It is broader than often needed since many of the components will be

either absent or at least won't be time-varying. For instance, in (1.1), the transition matrices A_t are usually time-invariant, as are the variances of shocks W_t . And most models don't need the exogenous shift Z_t .

It is too *narrow* since it allows for only one lag of the state, and both equations are linear. The first restriction is easily overcome; allowing for non-linearity is harder, but doable, at least approximately, and will be covered in section 7.2. Contemporaneous correlation between the shocks in the state equation (W_t) and those in the measurement equation (V_t) can be handled by adding V_t to the set of states; serial correlation in V_t is handled the same way.

What can state-space models do for us? In some cases, our main interest is in the unobservable states. In the typical application, the state-space model generates a decomposition of the observable Y_t into two or more *unobservable components*. The state-space model (known as a UC model) allows estimates of these. In other cases, the states are only a means to an end—we're interested in doing inference on some parameters governing the process (such as variances of the shocks, or free parameters in the A or C matrices), and the state-space model is the most convenient way to analyze this. In either case, we need to be able to estimate the states given the data, so we'll start with that.

Let our data be represented as $\{Y_1, Y_2, \dots, Y_T\}$. For any random variable ξ , we'll abbreviate the conditional density $f(\xi|Y_1, Y_2, \dots, Y_t)$ as $f(\xi|t)$. There are three “natural” types of inference for X_t :

1. *Prediction*: $f(X_t|t-1)$
2. *Filtering*: $f(X_t|t)$
3. *Smoothing*: $f(X_t|T)$

When our main interest is in the X_t themselves, we'll usually want to do smoothing, putting *all* the information to use. Prediction and filtering are most useful when the states are only of indirect interest—they maintain the sequencing of the data.

Note, by the way, that there is no reason that we need a value of Y_t for every $t = 1, \dots, T$. The state-space framework can handle missing data in a very simple fashion—it's one of its greatest advantages as a computational tool.

The simplest non-trivial state-space model is the *local level* model, or random walk with noise. The single state variable follows the random walk:

$$x_t = x_{t-1} + w_t \tag{1.3}$$

and the measurement equation is

$$y_t = x_t + v_t \tag{1.4}$$

The interpretation of this model is that x_t is an (unobservable) local level or mean for the process. It's *local* in the sense that it can evolve from period to

period because of the shock process w_t . The observable y_t is the underlying process mean contaminated with the measurement error v_t .

If we compare this with (1.1) and (1.2), we can read off the following:

- $\mathbf{X}_t = [x_t]$
- $\mathbf{A}_t = [1]$
- $\mathbf{Z}_t = [0]$
- $\mathbf{F}_t = [1]$
- $\mathbf{W}_t = [w_t]$
- $\mathbf{Y}_t = [y_t]$
- $\mu_t = [0]$
- $\mathbf{C}_t = [1]$
- $\mathbf{V}_t = [v_t]$

Let's look at the most basic of the inference problems: prediction for \mathbf{X}_1 . There are no data before $t = 1$, so where will we get $f(x_{1|0})$? A random walk like x_t has no “natural” level. RATS provides a method to handle the initial conditions for non-stationary models like this (the `EXACT` or `PRESAMPLE=DIFFUSE` option), but we'll get to that in Chapter 4. For now, we'll assume that our “pre-data” information can be written

$$x_0 \sim N(\mu_0, \sigma_0^2)$$

where this is assumed to be independent of the v and w processes. Using that and (1.3), we get the prediction density of

$$x_{1|0} \sim N(\mu_0, \sigma_0^2 + \sigma_w^2) \quad (1.5)$$

To further simplify the notation, we will use $x_{t|s}$ to represent the mean of x_t given information through s and $\sigma_{t|s}^2$ to be the corresponding variance. We thus have

$$x_{1|0} = \mu_0; \sigma_{1|0}^2 = \sigma_0^2 + \sigma_w^2$$

We now combine (1.5) with (1.4). Because v_1 is independent of $x_{1|0}$, we get the joint predictive density:

$$\begin{bmatrix} x_1 \\ y_1 \end{bmatrix} \sim N \left(\begin{bmatrix} x_{1|0} \\ x_{1|0} \end{bmatrix}, \begin{bmatrix} \sigma_{1|0}^2 & \sigma_{1|0}^2 \\ \sigma_{1|0}^2 & \sigma_{1|0}^2 + \sigma_v^2 \end{bmatrix} \right)$$

The predictive density of the observable is Normal with mean $x_{1|0}$ and variance $\sigma_{1|0}^2 + \sigma_v^2$.

To get the filtered density for x_1 , we use standard results for multivariate Normals (Appendix B) to get

$$x_{1|1} = x_{1|0} + \frac{\sigma_{1|0}^2}{\sigma_{1|0}^2 + \sigma_v^2} (y_1 - x_{1|0})$$

and

$$\sigma_{1|1}^2 = \sigma_{1|0}^2 - \frac{(\sigma_{1|0}^2)^2}{\sigma_{1|0}^2 + \sigma_v^2} = \sigma_{1|0}^2 \left(\frac{\sigma_v^2}{\sigma_{1|0}^2 + \sigma_v^2} \right)$$

We correct our prediction ($x_{1|0}$), adjusting for the error in the prediction for y_1 : $y_1 - x_{1|0}$. The filtered state adjusts towards the new data point with percentage adjustment depending upon the ratio of the variances:

$$\frac{\sigma_{1|0}^2}{\sigma_{1|0}^2 + \sigma_v^2}$$

This adjustment factor is known as the *Kalman gain*. The closer this is to 1.0, the greater we adjust; the closer to zero, the less we adjust. That condition depends upon how large the measurement equation variance σ_v^2 is relative to the uncertainty in the states. A large value (noisy data), makes for a small adjustment; a small value makes for the large adjustment.

The filtered variance can be no larger than the predicted variance. *Note that this doesn't depend upon the observed data.* Whether what we observe hits the prediction exactly, or is five standard deviations away, the filtered variance always is reduced in a systematic way. This is a property of the multivariate Normal—we can only produce different behavior for this by using a different (and more difficult) distribution.

We now have a Normal distribution for $x_{1|1}$. We can repeat the prediction and correction steps above for $t = 2$, and continue on through the data set to $t = T$. This combination of prediction and correction is known as *Kalman filtering*.

1.2 Kalman Filtering with the Local Linear Model

We introduce the use of the **DLM** instruction using examples of filtering from Durbin and Koopman (DK) and from West and Harrison (WH). The DK example (Example 1.1) uses a well-known (though non-economic) series of flow data from the Nile River. The data are annual from 1871 to 1970.

What do we need in order to run the filter? The only input components that aren't fixed by the structure of the model are the pre-sample mean and variance, and the variances of the w and v processes. DK choose to make the pre-sample data approximately “diffuse” by choosing mean 0 and variance 10^7 . This appears to be nonsensical (negative flow is as likely as positive flow), but it just means that the first update will be dominated by the data, not the prior. The variance of v is taken to be 15099.0 and the variance of w is 1469.1. Those are estimated values from later in the book (our section 3.2).

A and C are input using the A and C options on **DLM**, so here they will be A=1 . 0 and C=1 . 0. The default for almost all the options is zero,¹ so we can just leave

¹Other than for A and F, where it's the identity matrix, or, here with just one state, 1.0.

out the `Z` and `MU` options. The variance for w is input using the `SW` option, so that will be `SW=1469.1`. The variance for v uses the `SV` option, so `SV=15099.0`. The data use the `Y` option— that’s expecting a series or a formula, which will here be `Y=NILE`.

The only inputs left are the initial mean and variance. The initial mean uses the `X0` option and the initial variance uses `SX0`, so `X0=0.0`, `SX0=1.e+7`. There’s one minor complication here. If you look at the local level model as described in DK (their equation 2.3):

$$y_t = \alpha_t + \varepsilon_t$$

$$\alpha_{t+1} = \alpha_t + \eta_t$$

there’s a slight timing difference on the two shocks compared with our model. The state equation shock dated time t is applied to the state (their α) at t to produce the prediction for $t + 1$, while in our model, the state equation shock dated time t is added onto the state at $t - 1$ to produce the prediction for t . As long as

1. the state and measurement shocks are uncorrelated
2. the data are the only time-varying input

there’s no effective difference between the two models. If either of those fails to be true, it’s just very important to get the timing right no matter which form is used. While it doesn’t really affect the outcome, the real difference between the two timing schemes is that under the one used by RATS, the calculations done at time t are: predict t given $t - 1$, correct t given time t data. For the DK scheme, it’s: correct t given time t data, predict $t + 1$ given t . Because of the ordering used, the pre-sample information for RATS (and most, but certainly not all, presentations of state-space models) comes as information dated at $t = 0$. With the DK scheme, you start with information at $t = 1$ (given 0). In order to get the calculations to match exactly, you need one additional option: `PRESAMPLE=X1`. This says the pre-sample information (`X0` and `SX0` options) is information dated $t = 1$. The default for this option is `PRESAMPLE=X0`, which means that `X0` and `SX0` are dated $t = 0$.

If we just look at the inputs, we now have the instruction:

```
d1m(a=1.0,c=1.0,x0=0.0,sx0=1.e+7,sv=15099.0,sw=1469.1,$
presample=x1,y=nile)
```

What about outputs? DK graph the following:

1. the filtered mean of the state ($x_{t|t}$)
2. the filtered variance of the state ($\sigma_{t|t}^2$)
3. the prediction error ($y_t - x_{t|t-1}$)
4. the variance of the prediction error ($\sigma_{t|t-1}^2 + \sigma_v^2$)

Each of these is a complete series. And, in practice, most models will have more than one state, so the first of these will generally be an n vector at each data point while the second will be an $n \times n$ matrix. RATS also allows for more than one observable series, so the third and fourth might, in other applications, be vectors and matrices as well. Because of this, the outputs are in the form of SERIES of VECTORS (for the means and errors) and SERIES of SYMMETRICS for the variances. You then extract the information you need using a **SET** instruction.

The first two outputs are by far the most commonly used, and are handled as parameters (after the estimation range). The options for the prediction error and its variance are **VHAT** and **SVHAT**, respectively. Our finished instruction is:

```
d1m(a=1.0,c=1.0,x0=0.0,sx0=1.e+7,sv=15099.0,sw=1469.1,$
    presample=x1,y=nile,vhat=vhat,svhat=fhat) / xstates vstates
```

The instructions for pulling information out of the output series are:

```
set a = %scalar(xstates(t))
set p = %scalar(vstates(t))
set v = %scalar(vhat(t))
set f = %scalar(fhat(t))
set lower = a-sqrt(p)*%invnormal(.95)
set upper = a+sqrt(p)*%invnormal(.95)
```

These are switching to the names used by DK, so **A** is the state, **P** the variance of the state, **V** the prediction error and **F** the prediction error variance. **XSTATES** is a SERIES of VECTORS, so **XSTATES(T)** is a VECTOR (here a “1” vector): the filtered mean of the state at t . The **%SCALAR** function takes the first element (here the only element) out of a vector or matrix. The same is applied to the other four outputs.

Because of the choice for the pre-sample information (0 mean), the prediction error for the first data point is very large and not at all typical of the model, so that’s left out of the graph. Similarly, the prediction error variance is a major outlier for that first data point, so its graph also starts at the second point. The first few observations for filtered data can, in many models, still be very imprecise; this, however, has only the one state to resolve, so it looks reasonable from $t = 1$.

This does a multiple-pane graph (Figure 1.1). The first of the four (top left) shows the estimate of the filtered state (solid black), upper and lower bounds for the estimates and the actual data.²

²The footer on the graph is for the numbering scheme in DK.

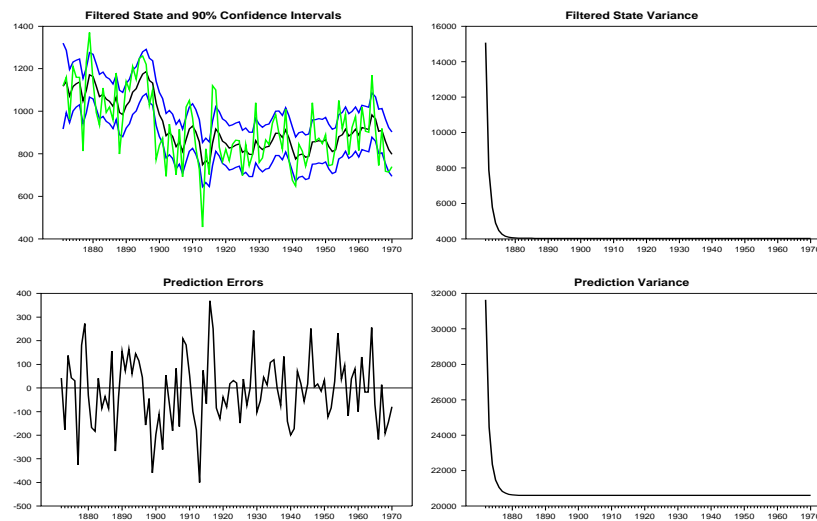


Figure 1.1: Kalman filter output

```

spgraph(vfields=2,hfields=2,$
  footer="Figure 2.1. Nile data and output of Kalman filter")
graph(header="Filtered State and 90% Confidence Intervals") 4
# a
# lower / 2
# upper / 2
# Nile / 3
graph(header="Prediction Errors")
# v 1872:1 *
graph(header="Filtered State Variance")
# p
graph(header="Prediction Variance")
# f 1872:1 *
spgraph(done)

```

The West and Harrison example (Example 1.2) is a better example of a use of Kalman *filtering*. In the DK example, we would almost certainly have a greater interest in the *smoothed*, rather than filtered, estimates of the states.

The situation in the WH example is that a (mythical) drug, called KURIT, has been selling at roughly 100 units per month. The local level model is considered to be appropriate, because the drug is used by individual patients for many months. At $t = 1$, a new formulation is being introduced which is expected to increase demand. The best guess from the marketing department is that sales will go up to 130 with a 95% confidence band running from 90 to 170. This means roughly a standard deviation of 20 or variance of 400. It's assumed that the underlying demand (the state) changes with a very low variance of 5, while the measurement error variance is the much higher 100. (Due to, for instance, the timing of when patients refill prescriptions). Because this is more of a forecasting exercise than the DK example, this uses the `YHAT` option to obtain

the predicted values for Y , rather than the $VHAT$, which gives prediction *errors*. (The $SVHAT$ variance applies to both). Other than that, the instruction is quite similar:

```
d1m(a=1.0,c=1.0,sv=100.0,sw=5.0,x0=130.0,sx0=400.0,y=kurit,$
    yhat=yhat,svhat=svhat) 1 9 xstate vstate
```

West and Harrison now add a twist, which **DLM** can handle with ease. At $t = 10$, a competitor drug is being pulled from the market. That drug had a roughly 50% market share, so if KURIT and all the other remaining drugs keep their relative market shares, each would be expected to double their sales. However, there's a good deal of uncertainty regarding the stability of the market shares. This is modeled as a one-shot shift to the state variable, with a sizeable increase in the variance in the state equation.

The exogenous shift in the state requires our first use of the **Z** option. And that, and the change in the variance, mean that we need a time-varying rather than fixed input for **W**. The input options will accept either fixed inputs or formulas, and the formulas can be either be put directly into the instruction, or defined separately as **FRMLs**. We'll do the latter, for readability (the formulas aren't short). The shift in mean at $t = 10$ is 143 (roughly double the $t = 9$ estimate) and the variance is $30^2 = 900$.

You'll note the decided difference in the handling of the input variances between this example and the previous one from **DK**. The error variances in the **DK** example are chosen as the maximum likelihood estimates (again, we'll discuss estimation in chapter 3) and the initial variance is chosen as a huge number representing "ignorance". Here, everything is chosen based upon judgment. Clearly, part of the difference is due to the size of the data sets—the KURIT data has just nine data points before the market sees a major change, while the **DK** data set has 100 observations. Another major difference, however, is that the KURIT example is designed to show an (artificial) example of real-time use of the Kalman filter. (The **DK** example is more of an *ex post* analysis of the data set). If the manufacturer wants a predicted range for what sales will be at $t = 10$, that will have to be done based upon information and belief that exists at that time. You convert a guess as to the likely range (interpreted as 95% confidence) into a standard deviation using $(upper - lower)/4$.

```
frml swf = %if(t<>10,5.0,900.0)
frml zf  = %if(t<>10,0.0,143.0)
d1m(a=1.0,c=1.0,sv=100.0,sw=swf,x0=130.0,sx0=400.0,$
    y=kurit,z=zf,yhat=yhat) 1 15
set predict 1 15 = %scalar(yhat(t))
graph(footer="Figure 2.3 KURIT examples",grid=(t==9),$
    key=upleft,klab=||"Actual","Predicted"||) 2
# kurit
# predict
```

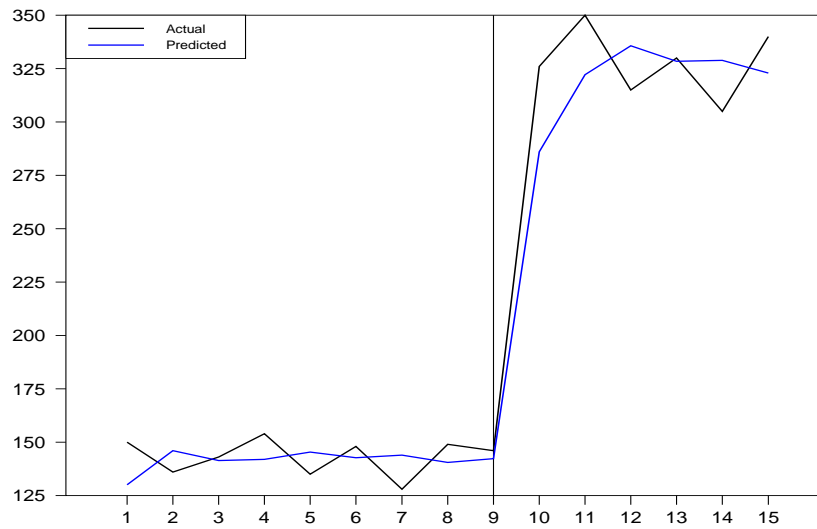


Figure 1.2: KURIT example

Note that z_F is non-zero only for $t = 10$. Because the state follows a random walk, the single impulse will push the mean up permanently (Figure 1.2).

1.3 Kalman Filtering with the Time-Varying Coefficients Model

The Kalman filter can also be applied to the time-varying coefficients (TVC) model, where a linear regression has coefficients which can vary from period to period. We'll look at a simple example from West and Harrison (Example 1.3). The data set has a time series for milk production as the dependent variable, with number of cows as the independent. The production function is assumed to be linear (rather clearly through the origin), but the production per cow is allowed to change as farming techniques change. The model is thus:

$$M_t = \beta_t C_t + \varepsilon_t$$

$$\beta_t = \beta_{t-1} + \eta_t$$

This is in state-space form with:

- $\mathbf{X}_t = \beta_t$
- $\mathbf{A}_t = 1$
- $\mathbf{W}_t = \eta_t$
- $\mathbf{Y}_t = M_t$
- $\mathbf{C}_t = C_t$
- $\mathbf{V}_t = \varepsilon_t$

Note that unlike the local level model (and almost all state-space models which decompose a series) the \mathbf{C} is time-varying, representing the explanatory variable.

Again, as the state equation is non-stationary, there is no obvious starting value, though this would be a situation where anyone working with this would likely have some fairly good information about production in a dairy herd. The pre-sample information is chosen as $X_0=10.0$, $SX_0=100.0$ —that variance is quite high relative to the values that will be observed. The measurement variance is set to be $SV=1.0$. West and Harrison fit two models, one with $SW=0.05$, the other the “fixed coefficient” model with $SW=0$.

Note that the fixed coefficients model doesn’t produce estimates which are “fixed” (see Figure 1.3). The filtered estimates at the start of the data set are based only upon the early data and are different (and much less precise) than the estimates for later periods.³ What’s fixed is the target coefficient vector—with enough data, the estimates will converge to a single value. With the non-zero SW , there’s a moving target—no matter how much data you get, the estimates would almost certainly keep moving around.

There are several special issues which come up in TVC models that aren’t important for UC models. The first is that the appropriate scale for the coefficient variance(s) isn’t as clear. In a UC model, the standard deviations are all in the same scale as the data. Here, they depend upon both the scale of the dependent variable and of the independent variables. The second is that it’s very easy to allow far too much time variation. What the Kalman filter does is divide up the prediction error in the dependent variable into a measurement error and a coefficient adjustment. When you have a model with K explanatory variables, there’s quite a bit of freedom to explain the data (almost perfectly) by adjusting the coefficients. We’ll take up TVC for multiple regressions in Section 5.4.

Here, with just one explanatory variable, it’s a bit easier to come up with a reasonable choice for the variance of the coefficient drift. The coefficient follows a random walk. The variance of a random walk over T periods is T times the variance in its increment. If you can come up with a guess as to how much the coefficient might range over a fixed length of time, you can back out a value for the variance from that. If a top to bottom range of 3.0 over 12 periods seems reasonable and you translate that to a range of four standard deviations (roughly 95% coverage), you would have $(3.0/4)^2/12 \approx .05$ for the incremental variance.

With multiple explanatory variables, that logic won’t work as well, since it doesn’t take into account the correlation among the variables themselves. For instance, if you have one slope and also an intercept, how much the intercept drifts will depend upon the range of values of the explanatory variable. The results of a TVC estimation will also depend upon the variance of the measurement error. If you make it too small, the only way to fit the data well is to shift the coefficients; make it too large, and movements in the data will be assigned to measurement error.

³*Smoothed* estimates (Section 1.4), however, will be the same for all periods.

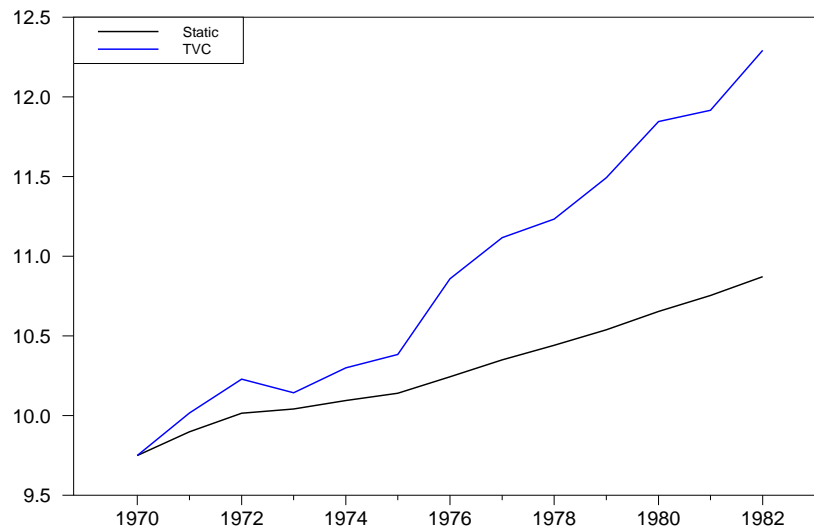


Figure 1.3: Coefficient Estimates

If you get the impression from this that TVC models are hard, you're correct. There are many (published) examples of estimated models which clearly used bad choices for variances. West and Harrison offer what may be a better way to handle these models, although it doesn't seem to have really caught on. Instead of adding variance to the state equation at each time period, it multiplies the last period variance by a constant somewhat larger than 1. (It's specified as a "discount" parameter, the reciprocal of that multiplier). This reduces the "tuning" parameters to just two—the discount parameter and the measurement error variance—even with many explanatory variables.

In this example, the dynamic (TVC) and static (fixed) coefficient models are estimated with:

```
dlim(y=milk,a=1.0,c=cows,sv=1.00,x0=10.0,sx0=100.0,sw=.05) / $
  xstates vstates
set dynfit = %scalar(xstates(t))
set dynvar = %scalar(vstates(t))
*
dlim(y=milk,a=1.0,c=cows,sv=1.00,x0=10.0,sx0=100.0) / $
  xstates vstates
set statfit = %scalar(xstates(t))
set statvar = %scalar(vstates(t))
```

The `C` option here specifies the `COW` series as its values. In the more common situation where there's also an intercept, the option would be `C=||1.0,COWS||` which evaluates to a 2-vector at each time period. See page 16 for a more general way to specify the explanatory variables in a model like this.

1.4 Kalman Smoothing

In an unobservable components model like the local level model, we're generally more interested in smoothing rather than filtering. Our estimates of the components are so imprecise early in the data set that we couldn't even graph the variances. It makes more sense to put *all* the data to work in estimating the components.

Kalman filtering gives us the smoothed value for precisely one data point: the last one. Because, under the assumptions, the data and states are jointly Normal, we could, theoretically, just generate a huge joint covariance matrix and use the formulas in Appendix B. That would, however, require inverting a (possibly huge) $T \times T$ matrix. One of the great advantages of the Kalman techniques is that they are designed to break big problems like that down to more manageable ones by doing calculations sequentially.

If you read the description of the Kalman smoothing algorithm in Durbin and Koopman and compare it with the more standard derivation in (for instance) Hamilton (1994), you might wonder if they are actually doing the same thing. The standard derivation directly goes after the calculation of $\mathbf{X}_{t|T}$, while DK use an indirect calculation. The advantage of the DK calculation (which RATS uses internally) is that it avoids an $n \times n$ inversion at each time period, and it also generalizes more easily to the exact diffuse calculations (Section 4.2).

Even though RATS doesn't use it, we'll explain briefly how the standard formula for Kalman smoothing works. We've included in Appendix D a simple proposition which underlies Kalman smoothing. In our situation, the two variables of interest are $x = \mathbf{X}_t$ and $y = \mathbf{X}_{t+1}$ and the two information sets (\mathcal{I} and \mathcal{J}) are through t and T respectively. Kalman smoothing works because the state-space model satisfies the proposition's condition:

$$f(\mathbf{X}_t | \mathbf{X}_{t+1}, T) = f(\mathbf{X}_t | \mathbf{X}_{t+1}, t)$$

Note that this reverses the timing—it asks what \mathbf{X}_{t+1} tells us about \mathbf{X}_t . If you tell me what \mathbf{X}_{t+1} is, there is no information in any $\{y_{t+1}, \dots, y_T\}$ which will help figure out \mathbf{X}_t , since the additional information will just be \mathbf{W} and \mathbf{V} dated $t + 1$ and later. Thus, if we've computed $f(\mathbf{X}_{t+1} | T)$ (the smoothed density for $t + 1$), that, combined with $f(\mathbf{X}_{t+1} | t)$ (the predicted density for $t + 1$ given t) and $f(\mathbf{X}_t | t)$ (the filtered density for t) is enough to compute $f(\mathbf{X}_t | T)$ (the smoothed density at t). The last two of these are done as part of the Kalman filter. Since we have the smoothed value at T as the result of the filter, we can just calculate backwards a step at a time to get the full set of smoothed estimates.

In Kalman filtering, the usual behavior for the uncertainty regarding the states is that it starts high and decreases through the data set. If the state equation is time-invariant (that is, \mathbf{A} and \mathbf{S}_W don't change), and there are no missing values, it *has to* decrease and will eventually approach a limit. In Kalman smoothing, the variance tends instead to be U-shaped. It's highest at the end

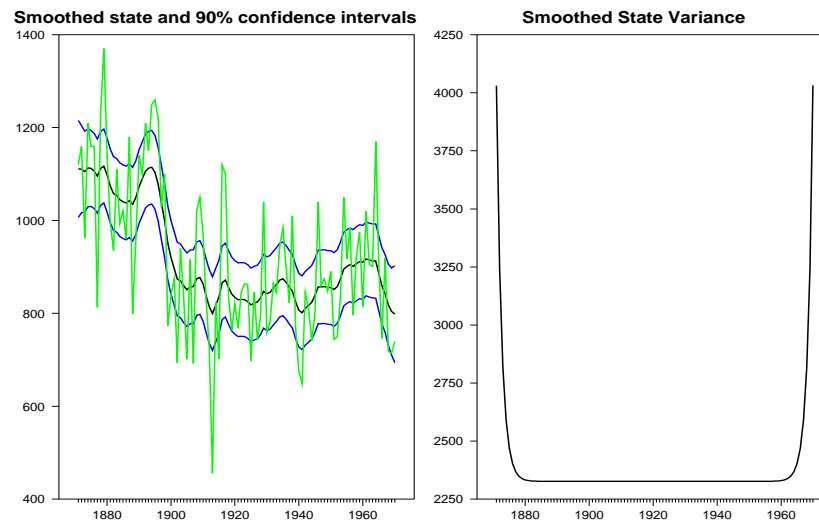


Figure 1.4: Kalman smoother output

points, and lowest in the middle—see the right pane in Figure 1.4. Data points in the middle have information in both directions to help with estimation, while those at the start and end have mainly one direction only.

1.5 Kalman Smoothing with the Local Level Model

The inputs to **DLM** are the same whether you’re filtering or smoothing. In order to do smoothing, you need to use the option `TYPE=SMOOTH`. This is from Example 1.4, which smooths, rather than filters, the Nile flow data.

```
dln(a=1.0,c=1.0,sx0=1.e+7,sv=15099.0,sw=1469.1,presample=x1,$
    type=smooth,y=nile) / xstates vstates
set a      = %scalar(xstates(t))
set p      = %scalar(vstates(t))
set lower  = a-sqrt(p)*%invnormal(.95)
set upper  = a+sqrt(p)*%invnormal(.95)
```

The parameters for the estimated states and the variances have the same structure as with the filter. Most of the other output options also work, but now have smoothed rather than filtered values. One that doesn’t change is `YHAT`, since the smoothed version of `Y` is trivially itself.

There are two additional output options that are available: `WHAT` (read W-hat) is the smoothed estimate of the shock in the state equation and `SWHAT` is its variance. This will have the dimensions of the `W` as defined by your model, so if you have an `F` matrix, the estimated disturbances to the states will be $F \times \text{WHAT}$.

Note that in the Kalman filter, the `VHAT` are (by construction) independent of each other, and so can be used easily for diagnostics. The smoothed estimates are *not* independent.

The following is from Example 1.5. This is one place where the timing difference in the state equation shock between our state equation and Durbin and Koopman's matters. Their $W(1)$ will be our $W(2)$, etc.

```
dlim(c=1.0, sx0=1.e+7, sv=15099.0, sw=1469.1, presample=x1, y=nile, $
    type=smooth, vhat=vhat, svhat=fhat, what=what, swhat=swhat) / $
    xstates vstates
set v = %scalar(vhat(t))
set sv = %scalar(fhat(t))
set w = %scalar(what(t))
set sw = %scalar(swhat(t))
```

1.6 Forecasts and Missing Data

As mentioned before, one great advantage of the Kalman filter is that it can easily handle missing data. How easily? At each time period, the Kalman filter does a prediction, then a correction. To handle missing data, it just doesn't do the correction. If y_t is missing, then $x_{t|t} = x_{t|t-1}$ and $\sigma_{t|t}^2 = \sigma_{t|t-1}^2$, that is, the filtered values are the same as the predicted ones. The predicted variance will go up for the next data point and keep going up until you start seeing data again.

DLM checks two places for missing data: the **Y** value and the **C** value. The missing value will almost always be the **Y** value, but as we saw in the TVC model, it's possible for **C** to represent data (such as explanatory variables) as well.

Example 1.6 artificially withholds some data, patching parts of the input with the %NA missing value code.

```
set withmiss = %if(t>=21.and.t<=40.or.t>=61.and.t<=80,%na,nile)
```

While this is quite artificial here, it actually has some real value. If you want to forecast out of sample, the way to do that is to extend the range of the filter beyond the end of your data. Note, by the way, that that will give you “dynamic” (multiple step) forecasts. The “static” (one step ahead) forecasts can be done by processing the **YHAT** option with the full **Y** series as input. This is from Example 1.7, which forecasts out 30 years beyond the end of the data. Note that there are two “forecasts” generated by this: **XSTATES** has the forecasts of the state (the local level), and **YHAT** has the forecasts of the observable. Those are (in this model) the same. What are different are their variances: **VSTATES** for the state and **FHAT** (from the **SVHAT** option) for the observable, since the observable has the extra component of the measurement error. See Figure 1.5.

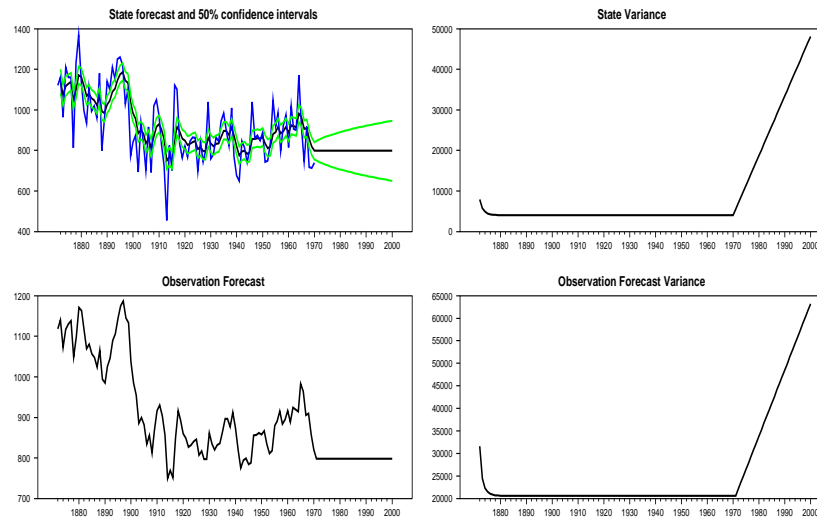


Figure 1.5: Out-of-Sample Forecasts

```
dln(a=1.0,c=1.0,sx0=1.e+7,sv=15099.0,sw=1469.1,presample=x1,$
    type=filter,y=nile,yhat=yhat,svhat=fhat) 1871:1 2000:1 $
    xstates vstates
set a      * 2000:1 = %scalar(xstates(t))
set p      * 2000:1 = %scalar(vstates(t))
set f      * 2000:1 = %scalar(fhat(t))
set y      * 2000:1 = %scalar(yhat(t))
```

1.7 RATS Tips and Tricks

There have been many changes to the **DLM** instruction since it was introduced. We have updated our textbook and published paper examples to show the “best” (simplest or fastest) ways to use these. This section provides a guide to these updates.

In-Line Formulas

The inputs to **DLM** can take many forms. The observable **Y** can be a single series, but also can be a calculated formula, and if you have more than one observable, it needs to evaluate to a **VECTOR** at each data point. In the first models in this chapter, the variances (**SW** and **SV**) were fixed constants, but in most cases, **SV** will be a variable that needs to be estimated, and **SW** is usually a matrix.

DLM will take the input information in whatever form is needed. You can include in-line matrices and formulas, such as `A=||1.0,1.0|0.0,1.0||` or `SW=exp(LSW)`. As mentioned below (in the discussion of the **START** option), this flexibility can come with a bit of a cost in time, but generally you should choose the form of the input based upon what’s easiest to read and understand. In our examples, we’ve generally tried to keep the inputs on a single line, and put other options (such as output options and control options for estimation) on a second line.

%EQNXVECTOR function

`%EQNXVECTOR(eqn,entry)` can be helpful in organizing some of the options: most commonly **C** (for instance, in time-varying parameter situations) and **Y** (when there are a large number of observables). For instance, if in example 1.3, we had a **CONSTANT** as well as **COWS** as explanatory variables, we could use something like

```
equation tvceq *
# constant cows
```

and use the option `c=%eqnxvector(tvceq,t)` rather than `c=||1,cows||`.⁴ In this case, this probably would make little sense, but if you wanted to be able to be flexible with the form of the regression, defining the equation once and then referring to it using `%EQNXVECTOR` later is a superior design.

Bernanke et al. (2005) estimates a very large state-space model with over 100 observables. The key instructions for setting this up early on are the following **EQUATION** definitions:

⁴Because the equation is used only to supply the “right-side” variables, there is no need for a dependent variable, hence the `*`.

```

*
* Variables excluded from VAR. Separated into slow- and fast-moving.
* Slow-moving are used to identify the loadings.
*
equation sloweqn *
# ip lhur punew ipp ipf ipc ipcd ipcn ipe ipi ipm ipmd ipmnd ipmfg $
  ipd ipn ipmin iput ipxmca pmi pmp gmpyq gmyxpq lhel lhelx lhem $
  lhnag lhu680 lhu5 lhu14 lhu15 lhu26 lpnag lp lpgd lpmi lpcc lpem $
  lped lpen lpsp lptu lpt lpfr lps lpgov lphrm lpmosa pmemp gmcq $
  gmcqdq gmcnq gmcsq gmcanq pwfsa pwfcsa pwimsa pwcmsa psm99q pu83 $
  pu84 pu85 puc pucd pus puxf puxhs puxm lehcc lehm
equation fasteqn *
# hsfh hsne hsmw hssou hswst hsbr hmob pmnv pmno pmdel mocmq $
  msondq fsncom fspcom fspin fspcap fsput fsdxdp fspxe exrsw exrjan $
  exruk exrcan fygm3 fygm6 fygt1 fygt5 fygt10 fyaaac fybaac sfygm3 $
  sfygm6 sfygt1 sfygt5 sfygt10 sfyaaac sfybaac fm1 fm2 fm3 fm2dq $
  fmfba fmrra fmrrba fclnq fclbmc ccinrv pmcp hhsntn
*
* Variables included in the VAR
*
equation yeqn *
# fyff

```

which define three subsets of the list of observables. `%eqnvector(sloweqn,t)` will produce a VECTOR with the 70 elements of the first list at time T. Change the list on the supplementary card, and the rest of the program goes through. To make the program flexible, you need to also use some other EQUATION utility functions like `%EQNSIZE(sloweqn)`, which will return the number of variables in the equation (here 70).

EXACT OR PRESAMPLE=DIFFUSE option

Most state-space models used in economics have at least one non-stationary (unit root) state. In some important models, *all* the states are non-stationary. With no natural (pre-data) value for these states, these were generally handled the way they are in the first DK example, with a very large (effectively “infinite”) variance. The problem with that approach is that what is sufficiently large in one situation might not be in another. Some software (early versions of STAMP, for instance) deal with this by trying several big values and checking the sensitivity. The EXACT option (or equivalently PRESAMPLE=DIFFUSE), which we’ll discuss in greater detail in Chapters 2 and 4, does an exact (limit) calculation, working with a pair of reduced-rank matrices, rather than a full-rank matrix with a mix of very large and very small elements.

F Option

Most state-space models have fewer fundamental shocks than states. If we don't have the `F` option, the state equation becomes:

$$\mathbf{X}_t = \mathbf{A}_t \mathbf{X}_{t-1} + \mathbf{Z}_t + \mathbf{W}_t$$

The `SW` matrix will have to be an $n \times n$ matrix even if it only has a few non-zero elements. If, on the other hand, the shock term is written as $\mathbf{F}\mathbf{W}_t$, where \mathbf{F} is $n \times r$, then the `SW` matrix is a (possibly much) smaller $r \times r$ matrix. This greatly simplifies the setup of many important models.

START Option

Most inputs to `DLM` aren't time-varying, but some may depend upon parameters that we need to estimate, and some (most commonly `A` and `SW`) can be quite complicated, being built from several parts. The `START` option allows you to create a function or formula which will be calculated *before* anything else in a function evaluation. Suppose that you need one of these complicated `A` matrices. The best way to handle this is to create a function which puts together your matrix:

```
function ADLM
type rect ADLM
...
end ADLM
dlm(A=ADLM(), ...)
```

However, as written, this will redo the calculation of the `A` matrix at every data point. The cost of the flexibility described at the start of the section is that anything that isn't a constant number (or a fixed variable or matrix) is assumed to be time-varying. For small models and small data sets, that won't matter. It could make a big difference in execution time if the model is larger.

This is where the `START` option can help. `START` does whatever calculations you describe, and does so before starting each Kalman filter pass. If we change the code above to:

```
dec rect A
function ADLM
type rect ADLM
...
end ADLM
dlm(start=(A=ADLM()), A=A, ...)
```

it will compute the `ADLM` function once per evaluation, put the result into the matrix `A`, and the (now fixed) `A` will be used at each data point in the filter pass.

Matrix concatenation operators

Many important state-space models are built out of components. A common model, for instance, has a trend component and independent seasonal component. The model is that the observable is the sum of the two (plus noise). If the two models can be written separately as:

$$\begin{aligned} \mathbf{T}_t &= \mathbf{A}_1 \mathbf{T}_{t-1} + \mathbf{F}_1 \mathbf{W}_{1t} \\ \mathbf{S}_t &= \mathbf{A}_2 \mathbf{S}_{t-1} + \mathbf{F}_2 \mathbf{W}_{2t} \end{aligned}$$

then the overall state equation is:

$$\mathbf{X}_t = \begin{bmatrix} \mathbf{T}_t \\ \mathbf{S}_t \end{bmatrix} = \begin{bmatrix} \mathbf{A}_1 & 0 \\ 0 & \mathbf{A}_2 \end{bmatrix} \begin{bmatrix} \mathbf{T}_{t-1} \\ \mathbf{S}_{t-1} \end{bmatrix} + \begin{bmatrix} \mathbf{F}_1 & 0 \\ 0 & \mathbf{F}_2 \end{bmatrix} \begin{bmatrix} \mathbf{W}_{1t} \\ \mathbf{W}_{2t} \end{bmatrix}$$

The “grand” \mathbf{A} and \mathbf{F} matrices can be constructed from the components with the diagonal concatenation operator `~\` with something like

```
compute a=a1~\a2
compute f=f1~\f2
```

In some earlier codings of models like this, we used the `%BLOCKGLUE` and `%BLOCKDIAG` functions. The concatenation operators are clearly much simpler. The other two are `~`, which does horizontal (side by side) concatenation and `~~`, which does vertical (one over the other).

Example 1.1 Kalman Filter: Nile Data

This is an example of Kalman filtering using the local trend model. This is the illustration from section 2.2.5 of Durbin & Koopman (2012).

```
open data nile.dat
calendar 1871
data(format=free,org=columns,skips=1) 1871:1 1970:1 nile
*
* Note that there is difference between DLM and the description in the
* book which will affect some of the calculations. The RATS definition
* of the state-space model is
*
*  $x(t) = A x(t-1) + w(t)$ 
*  $y(t) = c x(t) + v(t)$ 
*
* (The options on the DLM instruction take their names from the
* component names in this description)
*
* The book uses
*
*  $a(t+1) = T a(t) + \eta(t)$ 
*  $y(t) = Z a(t) + \epsilon(t)$ 
*
* Note the difference in the subscripts in the state equation (the first
* in each pair). In the textbook, the shock to the state equation dated
*  $t$  is used in moving to  $t+1$  given  $t$ , while in DLM it's used in moving
* from  $t$  given  $t-1$ . This has no practical difference for estimating  $x$  or
* predicting  $y$  as long as the shocks to the state equation are
* uncorrelated with those to the measurement equation (which is the
* assumption built in to DLM).
*
* However, because of this, the standard timing for pre-sample data is
* different. For DLM, the initial mean and variance are considered to be
* for  $t=0$ . In the text, they are for  $t=1$  given no data. In order to
* match the results, you need to add the PRESAMPLE=X1 option. This
* "re-dates" that initial data so it is treated as information at  $t=1$ .
*
* This is a very simple set up for DLM since there is only one state
* variable, and one observable, and all the variances are treated as
* known.
*
dln(a=1.0,c=1.0,x0=0.0,sx0=1.e+7,sv=15099.0,sw=1469.1,presample=x1,$
    y=nile,vhat=vhat,svhat=fhat) / xstates vstates
*
* Because DLM can handle multiple inputs (vector Y) and because most
* models have more than one state, the outputs generally are series of
* VECTORS for states, errors and observables and series of SYMMETRICs
* for variances. To use these in further calculations, you usually will
* have to extract the information into simpler series of real numbers.
* That's what the SET instructions are doing. The %SCALAR function pulls
* the first element (in all cases here, the only element) out of a
* vector or matrix.
```

```

*
set a = %scalar(xstates(t))
set p = %scalar(vstates(t))
set v = %scalar(vhat(t))
set f = %scalar(fhat(t))
set lower = a-sqrt(p)*%invnormal(.95)
set upper = a+sqrt(p)*%invnormal(.95)
*
* The graphs for the prediction error and prediction error variance
* leave out the first data point (1871) which would show up as a huge
* outlier on both graphs.
*
spgraph(vfields=2,hfields=2,$
  footer="Figure 2.1. Nile data and output of Kalman filter")
graph(header="Filtered State and 90% Confidence Intervals") 4
# a
# lower / 2
# upper / 2
# nile / 3
graph(header="Prediction Errors")
# v 1872:1 *
graph(header="Filtered State Variance")
# p
graph(header="Prediction Variance")
# f 1872:1 *
spgraph(done)

```

Example 1.2 Kalman Filter: Drug Sales Data

This is an example of Kalman filtering (with intervention) from West & Harrison (1997), example 2.1.

```

data(format=free,unit=input) 1 10 kurit
150 136 143 154 135 148 128 149 146 .
*
dlm(a=1.0,c=1.0,sv=100.0,sw=5.0,x0=130.0,sx0=400.0,y=kurit,$
  yhat=yhat,svhat=svhat) 1 9 xstate vstate
set postmean 1 9 = %scalar(xstate(t))
set postvar 1 9 = %scalar(vstate(t))
set predict 1 9 = %scalar(yhat(t))
set predvar 1 9 = %scalar(svhat(t))
print(picture="*.#") / predvar predict kurit postmean postvar
*
* Intervention
* This is done with a time-varying <<sw>> formula and a state mean
* shifter, both of which take a different value for t==10 than for other
* data points.
*
frml swf = %if(t<>10,5.0,900.0)
frml zf = %if(t<>10,0.0,143.0)
*
* Other than the 326, these numbers are guessed from the graph

```

```

*
data(unit=input) 10 15 kurit
326 350 315 330 305 340
dlm(a=1.0,c=1.0,sv=100.0,sw=swf,x0=130.0,sx0=400.0,$
    y=kurit,z=zf,yhat=yhat) 1 15
set predict 1 15 = %scalar(yhat(t))
graph(footer="Figure 2.3 KURIT examples",grid=(t==9),$
    key=upleft,klabels=||"Actual","Predicted"||) 2
# kurit
# predict

```

Example 1.3 Kalman Filter: Time-Varying Coefficients Model

This is an example of a simple time-varying coefficients model. It's adapted from West & Harrison (1997), Example 3.1 from pp 81-82. Details are covered in Section 1.3.

```

cal(a) 1970
open data table3_1.xls
data(format=xls,org=columns) 1970:1 1982:1 milk cows
*
* Dynamic model
*
dlm(y=milk,a=1.0,c=cows,sv=1.00,x0=10.0,sx0=100.0,sw=.05) / $
    xstates vstates
set dynfit = %scalar(xstates(t))
set dynvar = %scalar(vstates(t))
*
* Static model
*
dlm(y=milk,a=1.0,c=cows,sv=1.00,x0=10.0,sx0=100.0) / $
    xstates vstates
set statfit = %scalar(xstates(t))
set statvar = %scalar(vstates(t))
*
print(picture="*.###") / cows milk dynfit dynvar statfit statvar
*
graph(footer="Coefficient estimates",$
    key=upleft,klabels=||"Static","TVC"||) 2
# statfit
# dynfit
*
set foredyn = dynfit*cows
set upperdyn = (dynfit+2.0*sqrt(dynvar))*cows
set lowerdyn = (dynfit-2.0*sqrt(dynvar))*cows
graph(footer="Dynamic Forecasts with Actual Data") 4
# milk
# foredyn
# upperdyn / 3
# lowerdyn / 3

```

Example 1.4 Kalman Smoothing: Nile Data

This is an example of Kalman smoothing in a local level model. It's taken from Durbin and Koopman, illustration from section 2.4.3. See Section 1.4 for details.

```
open data nile.dat
calendar 1871
data(format=free,org=columns,skips=1) 1871:1 1970:1 nile
*
* This is the same model as the previous example, but with type=smooth
* used to get smoothed rather than filtered estimates of the states and
* variances.
*
dlm(a=1.0,c=1.0,sx0=1.e+7,sv=15099.0,sw=1469.1,presample=x1,$
    type=smooth,y=nile) / xstates vstates
set a      = %scalar(xstates(t))
set p      = %scalar(vstates(t))
set lower  = a-sqrt(p)*%invnormal(.95)
set upper  = a+sqrt(p)*%invnormal(.95)
*
* With the smoothed estimates, the first data point can be included in the
* graphs without distorting the picture.
*
spgraph(hfields=2,footer=$
    "Figure 2.2. Nile data and output of state smoothing recursion")
graph(header="Smoothed state and 90% confidence intervals") 4
# a      1871:1 *
# lower 1871:1 * 2
# upper 1871:1 * 2
# nile   1871:1 * 3
graph(header="Smoothed State Variance")
# p 1871:1 *
spgraph(done)
```

Example 1.5 Kalman Smoothing: Estimated Errors

This example computes and graphs smoothing errors. It's adapted from Durbin & Koopman (2012), illustration from section 2.5.3.

```
open data nile.dat
calendar 1871
data(format=free,org=columns,skips=1) 1871:1 1970:1 nile
*
dlm(c=1.0,sx0=1.e+7,sv=15099.0,sw=1469.1,presample=x1,y=nile,$
    type=smooth,vhat=vhat,svhat=fhat,what=what,swhat=swhat) / $
    xstates vstates
set v = %scalar(vhat(t))
set sv = %scalar(fhat(t))
set w = %scalar(what(t))
set sw = %scalar(swhat(t))
```

```

*
* Because of the timing difference between the state disturbance in the
* text and that used by DLM, the w and sw series will be shifted one
* period. (the book's w(1) will be our w(2)). The v and sv will match.
*
spgraph(hfields=2,vfields=2,$
  footer="Figure 2.3. Output of disturbance smoothing recursion")
graph(header="Observation error")
# v
graph(header="State error")
# w
graph(header="Observation error variance")
# sv
graph(header="State error variance")
# sw
spgraph(done)

```

Example 1.6 Missing Data

This demonstrates filtering and smoothing with missing data. It's adapted from Durbin & Koopman (2012), illustration from section 2.7.1.

```

open data nile.dat
calendar 1871
data(format=free,org=columns,skips=1) 1871:1 1970:1 nile
*
* Create the series with missing values for two entry ranges
*
set withmiss = %if(t>=21.and.t<=40.or.t>=61.and.t<=80,%na,nile)
*
* Get filtered estimates
*
dlm(a=1.0,c=1.0,sx0=1.e+7,sv=15099.0,sw=1469.1,presample=x1,$
  type=filter,y=withmiss,vhat=vhat,svhat=fhat) / xstates vstates
set a = %scalar(xstates(t))
set p = %scalar(vstates(t))
*
* Get smoothed estimates
*
dlm(a=1.0,c=1.0,sx0=1.e+7,sv=15099.0,sw=1469.1,presample=x1,$
  type=smooth,y=withmiss,vhat=vhat,svhat=fhat) / xstates vstates
set as = %scalar(xstates(t))
set ps = %scalar(vstates(t))
*
*
spgraph(vfields=2,hfields=2,footer=$
  "Figure 2.5. Filtering and smoothing output with missing observations")
graph(header="Filtered state (extrapolation)") 2
# a 1871:1 *
# withmiss 1871:1 *
graph(header="Smoothed state (interpolation)") 2
# as 1871:1 *

```

```
# withmiss 1871:1 *
graph(header="Filtered state variance")
# p 1871:1 *
graph(header="Smoothed state variance")
# ps 1871:1 *
spgraph(done)
```

Example 1.7 Out of Sample Forecasting

This shows out-of-sample forecasting. It's adapted from Durbin & Koopman (2012), illustration from section 2.8.1.

```
open data nile.dat
calendar 1871
data(format=free,org=columns,skips=1) 1871:1 1970:1 nile
*
* To forecast out of sample, just run the filter beyond the range of the
* existing data.
*
dlim(a=1.0,c=1.0,sx0=1.e+7,sv=15099.0,sw=1469.1,presample=x1,$
     type=filter,y=nile,yhat=yhat,svhat=fhat) 1871:1 2000:1 $
     xstates vstates
set a      * 2000:1 = %scalar(xstates(t))
set p      * 2000:1 = %scalar(vstates(t))
set f      * 2000:1 = %scalar(fhat(t))
set y      * 2000:1 = %scalar(yhat(t))
set lower  * 2000:1 = a-%invnormal(.75)*sqrt(p)
set upper  * 2000:1 = a+%invnormal(.75)*sqrt(p)
*
spgraph(vfields=2,hfields=2,$
       footer="Figure 2.6. Nile data and output of forecasting")
graph(header="State forecast and 50% confidence intervals") 4
# a      1872:1 2000:1
# nile   1871:1 *
# upper  1872:1 2000:1 3
# lower  1872:1 2000:1 3
graph(header="Observation Forecast")
# y 1872:1 2000:1
graph(header="State Variance")
# p 1872:1 2000:1
graph(header="Observation Forecast Variance")
# f 1872:1 2000:1
spgraph(done)
```


More States

2.1 Kalman Filter in Matrix Form

Once we add more states, we need to generalize the Kalman filter to do inference on a vector. The same strategy is used in developing the algorithm: get a joint Normal predictive density for $\{\mathbf{X}_t, \mathbf{Y}_t\}$ given $t-1$, then condition on \mathbf{Y}_t to get the filtered distribution for \mathbf{X}_t . In our most general case (also allowing \mathbf{Y}_t to be a vector), the Kalman filter can be written:

$$\Sigma_{t|t-1} = \mathbf{A}_t \Sigma_{t-1|t-1} \mathbf{A}_t' + \mathbf{M}_t$$

$$\mathbf{X}_{t|t-1} = \mathbf{A}_t \mathbf{X}_{t-1|t-1} + \mathbf{Z}_t$$

$$\mathbf{X}_{t|t} = \mathbf{X}_{t|t-1} + \Sigma_{t|t-1} \mathbf{c}_t (\mathbf{C}_t' \Sigma_{t|t-1} \mathbf{C}_t + \mathbf{N}_t)^{-1} (\mathbf{Y}_t - \mu_t - \mathbf{C}_t' \mathbf{X}_{t|t-1})$$

$$\Sigma_{t|t} = \Sigma_{t|t-1} - \Sigma_{t|t-1} \mathbf{c}_t (\mathbf{C}_t' \Sigma_{t|t-1} \mathbf{C}_t + \mathbf{N}_t)^{-1} \mathbf{C}_t' \Sigma_{t|t-1}$$

where $\Sigma_{t|s}$ is the covariance matrix of \mathbf{X}_t given s , \mathbf{N}_t is the covariance matrix of \mathbf{V}_t and \mathbf{M}_t is the covariance matrix of \mathbf{W}_t .

The filtered (or smoothed) states and their covariance matrices can be obtained from **DLM** with the state and variances parameters. In order to pull information out, you'll need something like

```
set rate    = xstates(t)(2)
set ratesd  = sqrt(vstates(t)(2,2))
```

which extracts the second state out of the state vector series and the square root of the 2,2 element from the variance matrix.

As with the one-state case, if we don't observe \mathbf{Y}_t , we just do the first two equations (prediction) and skip the last two (correction). Note that it's possible now to have a partial observation—if \mathbf{Y}_t is designed to be a 2-vector, we might have some data points where we can observe only one component. The formula above can handle this by removing the parts of \mathbf{Y}_t , μ_t , \mathbf{C}_t and \mathbf{N}_t corresponding to the missing value. The **DLM** instruction will make that adjustment automatically if it detects a missing value in components of \mathbf{Y}_t or \mathbf{C}_t .

And also as with the one-state case, the estimate of the mean $\mathbf{X}_{t|t}$ is unaffected if all the input variances (\mathbf{N}_t , \mathbf{M}_t , $\Sigma_{0|0}$) are multiplied by the same constant.

The predictive density for \mathbf{Y}_t given $t-1$ is given by:

$$\mathbf{Y}_t \sim N(\mu_t + \mathbf{C}'_t \mathbf{X}_{t|t-1}, \mathbf{C}'_t \Sigma_{t|t-1} \mathbf{C}_t + \mathbf{N}_t)$$

Because of the sequential conditioning, the predictive errors:

$$\hat{\mathbf{V}}_t = \mathbf{Y}_t - (\mu_t + \mathbf{C}'_t \mathbf{X}_{t|t-1})$$

form an independent sequence. This will be important for estimation and diagnostics.

2.2 ARMA Processes

How can we convert a standard ARMA model such as

$$y_t = \varphi_1 y_{t-1} + \dots + \varphi_p y_{t-p} + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \dots + \theta_q \varepsilon_{t-q}$$

into state-space form? The pure AR(p)

$$y_t = \varphi_1 y_{t-1} + \dots + \varphi_p y_{t-p} + \varepsilon_t$$

is fairly simple. We *augment* the state vector to include $p-1$ lags—combined with the natural single lag from the standard state-space form, that will give us p lags in all. The state equation is:

$$\mathbf{X}_t = \begin{bmatrix} y_t \\ y_{t-1} \\ y_{t-2} \\ \vdots \\ y_{t-(p-1)} \end{bmatrix} = \begin{bmatrix} \varphi_1 & \varphi_2 & \varphi_3 & \dots & \varphi_p \\ 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & \ddots & \dots & 0 \\ 0 & \dots & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} y_{t-1} \\ y_{t-2} \\ y_{t-3} \\ \vdots \\ y_{t-p} \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \varepsilon_t \quad (2.1)$$

and the measurement equation is

$$y_t = [1, 0, \dots, 0] \mathbf{X}_t \quad (2.2)$$

Note that there is no “measurement” error. We can, in fact, *always* arrange a state-space model to have all the shocks in the state vector. For instance, the local level model can also be written:

$$\mathbf{X}_t = \begin{bmatrix} x_t \\ v_t \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x_{t-1} \\ v_{t-1} \end{bmatrix} + \begin{bmatrix} w_t \\ v_t \end{bmatrix}$$

$$y_t = [1, 1] \mathbf{X}_t$$

This would, in most cases, be a pointless complication, but if we want to allow for a correlation between w_t and v_t , this is the way to do it.

Now the combination of (2.1) and (2.2) might *also* seem a bit pointless in state-space form. After all, in this case, the states are completely observable. Almost. The state is only fully observable for $t = p+1$ and later—until then, it includes pre-sample values y_0, y_{-1}, \dots which *aren't* observable. And, if we have

any missing values in the middle of the data set, again, we can only estimate those. Kalman filtering will give us a one-sided estimate (past only) for the missing values; Kalman smoothing will give us a full-sample estimate.

In addition to being useful for handling (properly) the pre-sample for an AR(p) process and for treating embedded missing values, it can also be useful as an unobserved component. Suppose, for instance, that we're assuming the data can reasonably be modeled using the local level, but that the measurement error is serially correlated. If the serial correlation is a stationary process, we can distinguish between it and the "permanent" moves associated with the changes in the local level. A random walk local level plus an AR(2) noise can be written:

$$\mathbf{X}_t = \begin{bmatrix} x_t \\ u_t \\ u_{t-1} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \varphi_1 & \varphi_2 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_{t-1} \\ u_{t-1} \\ u_{t-2} \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} w_t \\ \varepsilon_t \end{bmatrix}$$

$$y_t = [1, 1, 0] \mathbf{X}_t$$

Pure MA(q) processes are also fairly simple:

$$\mathbf{X}_t = \begin{bmatrix} \varepsilon_t \\ \varepsilon_{t-1} \\ \vdots \\ \varepsilon_{t-q} \end{bmatrix} = \begin{bmatrix} 0 & 0 & \cdots & 0 \\ 1 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 1 & 0 \end{bmatrix} \begin{bmatrix} \varepsilon_{t-1} \\ \varepsilon_{t-2} \\ \vdots \\ \varepsilon_{t-(q+1)} \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \varepsilon_t$$

$$y_t = [1, \theta_1, \dots, \theta_q] \mathbf{X}_t$$

Note that this requires expansion to $q + 1$ states in order to have all the information for constructing the measurement equation.

Models with both AR and MA terms are fairly rare as components for other models—serially correlated shocks are usually assumed to be low order AR's, or occasionally low order MA's. When you start combining components (additively), you generate models which are equivalent to ARMA models with certain restrictions. For instance, the local level model can be transformed by differencing to:

$$\Delta y_t = \Delta x_t + \Delta v_t = w_t + \Delta v_t$$

$w_t + \Delta v_t$ is an MA(1) process, so the local level model is equivalent to y being an ARIMA(0,1,1) model. While we *could* get the same information from estimating the ARIMA model, rearranging that to get us the breakdown into the moving mean plus noise wouldn't be easy. The state-space modeling procedure generally leads to sums of independent components, which combine to produce a more complicated serial correlation pattern, rather than the ARIMA modeling approach of the single model fitting the whole pattern.

If you need to represent a full ARMA equation in state-space form, you could, in effect, combine the AR and MA. This works fine for an ARMA(1,1), which can be

written:

$$\mathbf{X}_t = \begin{bmatrix} y_t \\ \varepsilon_t \end{bmatrix} = \begin{bmatrix} \varphi_1 & \theta_1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} y_{t-1} \\ \varepsilon_{t-1} \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \end{bmatrix} \varepsilon_t$$

$$y_t = [1, 0] \mathbf{X}_t$$

While this same idea will work more generally, it can produce models with quite a few states. This is particularly true for seasonal ARMA models, where it would be quite easy to get 30-40 states with that approach. Instead, there are other ways to write the model which reduce the number of states to the smallest state size possible, which is $r = \max(p, q + 1)$. The one that is used in RATS for estimation in **BOXJENK**, and is also used in the procedure **@ARMADLM**, is due to Jones (1980). We won't get into the details of why this works, but the state representation is:

$$\mathbf{X}_t = \begin{bmatrix} y_t \\ y_{t+1|t} \\ \vdots \\ y_{t+r-1|t} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \vdots & \vdots & \ddots & \ddots \\ \varphi_r & \cdots & \varphi_2 & \varphi_1 \end{bmatrix} \begin{bmatrix} y_{t-1} \\ y_{t|t-1} \\ \vdots \\ y_{t+r-2|t-1} \end{bmatrix} + \begin{bmatrix} 1 \\ \psi_1 \\ \vdots \\ \psi_{r-1} \end{bmatrix} \varepsilon_t$$

where the notation $y_{s|t}$ means expected value (or more properly linear forecast) of y_s given information (values of y) through t , just as we've been using it. The φ are the AR coefficients (possibly padded with 0's) and the ψ are the coefficients in the moving average representation for the ARMA model. Note that the states are forecasts of the future, and when you lag one of these, you lag both the date of the observation, and the date of the information set. We'll use this type of state again when doing models with expectations in chapter 10.

2.3 Local Trend Model

The local level model works fairly well for the Nile data, which, while it drifts a bit from higher values at the beginning of the range to lower values towards the middle and end, doesn't have any obvious trend. Similarly, the drug sales series had no trend at all, and was (at least over that short span) pushed around mainly by exogenous events.

It isn't likely to work as well with a strongly trending series. Figure 2.1 shows the (log) of US GNP from the Nelson-Plosser data set, along with the recursive residuals produced from the local level model.

We don't even need to employ the diagnostics that we'll be discussing later (section 3.4) to see that there is something wrong here. All but a few of the residuals are positive, and errors are quite persistent.

However, it's also clear that a single linear trend won't fit the data well either. There are at least three distinct periods of growth, with 1933-1945 having a growth rate quite a bit higher than 1950-1970 and 1910-1929. Plus 1929-1933 is a steady downward trend for four years.

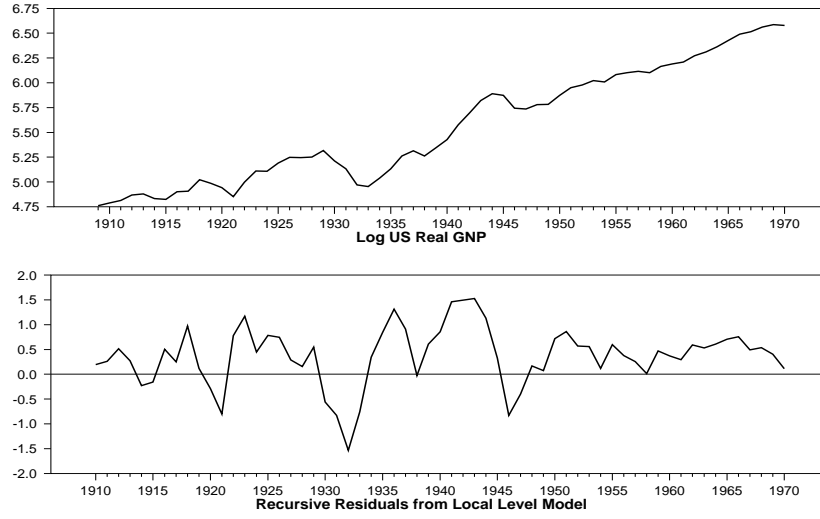


Figure 2.1: Local Level Applied to Trending Data

The standard state-space component for handling a series like this is the *local trend model*. There are several ways to write this, but the most common is something like:

$$\begin{aligned} x_t &= x_{t-1} + \tau_{t-1} + \xi_t \\ \tau_t &= \tau_{t-1} + \zeta_t \\ y_t &= x_t + \varepsilon_t \end{aligned} \tag{2.3}$$

τ_t is the local trend *rate*; x_t is the local trend itself, which is the local mean value for the observable y_t . Note the timing on how τ enters the update for x_t : it's last period's rate which is used. We need that in order to keep this in the proper form. The state-space representation is:

$$\begin{aligned} \mathbf{X}_t &= \begin{bmatrix} x_t \\ \tau_t \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_{t-1} \\ \tau_{t-1} \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \xi_t \\ \zeta_t \end{bmatrix} \\ y_t &= [1, 0] \mathbf{X}_t + \varepsilon_t \end{aligned}$$

From this, we can see that

$$\mathbf{A} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}, \mathbf{F} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \mathbf{C}' = [1, 0], \mathbf{W}_t = \begin{bmatrix} \xi_t \\ \zeta_t \end{bmatrix}, \mathbf{V}_t = \varepsilon_t$$

It's generally assumed that the three shocks are uncorrelated, so the \mathbf{SW} matrix will be diagonal.

Note that this is decomposing the single observable into *three* components. According to the model, we should be able to separate them based upon their persistence. Changes to the trend rate ζ_t have the greatest effect on the evolution of the series, since they accumulate period after period, the measurement equation error ε_t has the least (it affects just one period), and the level shock ξ_t is in between. In practice, this isn't as clear as one would like—with many

series, it's rather hard to separate the level shock from the measurement error. So the model will often be written with $\xi_t = 0$. Sometimes this is done right from the start. In other cases, it's done because the maximum likelihood estimate of σ_ξ^2 comes in negative.

The easiest way to set up the matrices for the local trend model is the procedure `@LocalDLM`. You use one of the two forms:

```
@LocalDLM(type=trend, shocks=both, a=a, f=f, c=c)
@LocalDLM(type=trend, shocks=trend, a=a, f=f, c=c)
```

The first of these includes shocks to both level and trend rate (in that order); the second includes only the shock to the trend rate, so there's only one component to W . Note that this also has option `TYPE=LEVEL` for the local level model, though that's so simple you're unlikely to use the procedure for it.

Even if we leave out the level shock, it's still possible for the local trend model to produce results that don't look "right". In order to be a "trend", a series has to be fairly stiff—you should be able to draw it with a quick sweep of the hand. But there's nothing in (2.3) to require that. If the variance of ζ_t is high (relative to the measurement error), the "trend" can move around quite a bit, possibly coming very close to tracking the data wherever it goes. Thus, on top of other restrictions, it's common to also include a restriction on the ratio between the two remaining variances. In the case of the Hodrick-Prescott filter (Hodrick & Prescott (1997)) which uses the local trend model, that ratio (measurement variance to trend rate variance) is $100 \times P^2$, where P is the number of periods per year.

The following, from Example 2.1, sets up a local trend model for the log GNP data (using the Hodrick-Prescott ratio for the variances) and graphs the recursive residuals (Figure 2.2):

```
@localdlm(type=trend, shocks=trend, a=at, f=ft, c=ct)
dlm(a=at, c=ct, f=ft, sv=sigsqeps, sw=sigsqeps/100., presample=diffuse, $
    y=logy, vhat=vhat, svhat=svhat)
set resids = %scalar(vhat)/sqrt(%scalar(svhat))
graph(hlabel="Recursive Residuals from Local Trend Model")
# resids
```

We'll discuss the `PRESAMPLE=DIFFUSE` (equivalently `EXACT`) option in Chapter 4, but note that the graph of the residuals doesn't start until 1911:1, skipping the first two periods. Because we now have *two* unit roots, it will take two observations to nail down reasonable values for the two states. This affects only Kalman filtering—Kalman smoothing is able to estimate everything back to the start of the data. Figure 2.3 shows the estimated (smoothed) trend rate (top panel) and the actual data and the smoothed trend (bottom panel). It's important to note (and note *well*), that while the local trend model is constructed assuming that the measurement error is serially uncorrelated, there is nothing

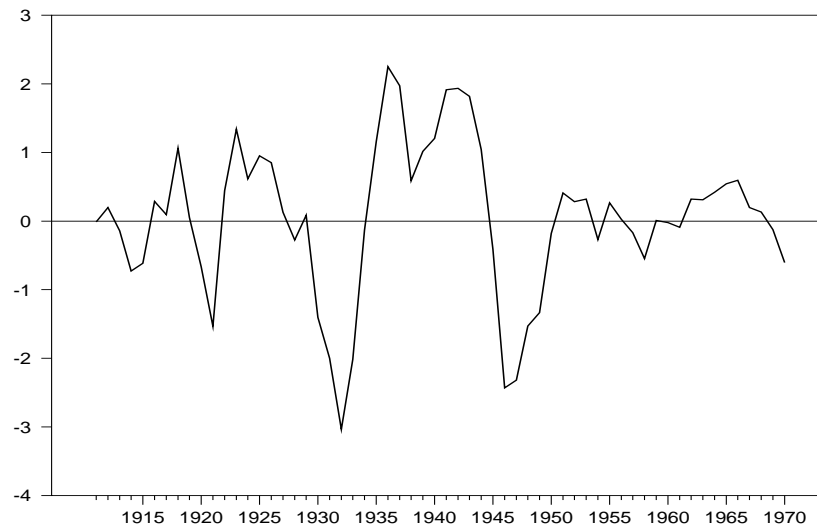


Figure 2.2: Recursive Residuals from Local Trend Model

in the calculations which will force that to happen in-sample (as is rather clear from looking at the deviations in the bottom of the graph). In this case, if we were to test the underlying assumptions, we would almost certainly reject the model on the basis of the serially correlated residuals. However, if the point is to come up with a reasonable way to separate the very long-term movements from the business cycle movements, then it appears to have done that fairly well.

```
dln(a=at,c=ct,f=ft,sv=sigsqeps,sw=sigsqeps/100.,presample=diffuse,$
    y=logy,type=smooth) / xstates

set trend = xstates(t) (1)
set rate  = xstates(t) (2)
```

2.4 Seasonals

Series with a strong seasonal component are very common in economics, but we tend to work with data that have been deseasonalized at the source. Many series (such as US GDP) aren't even published in NSA ("not seasonally adjusted") form. However, data below national aggregates (such as regional, industry or company data) might be available only in raw form.

Officially adjusted data is generally done using X12-ARIMA (available in RATS Professional). Note, however, that series adjusted by the government don't typically use standard settings for X12-ARIMA. Instead, each has its own input spec which will usually pre-select outliers and other settings so the seasonal adjustment won't change too dramatically with each added data point.

The basis for the X11 algorithm that underlies X12-ARIMA is (typically) one of

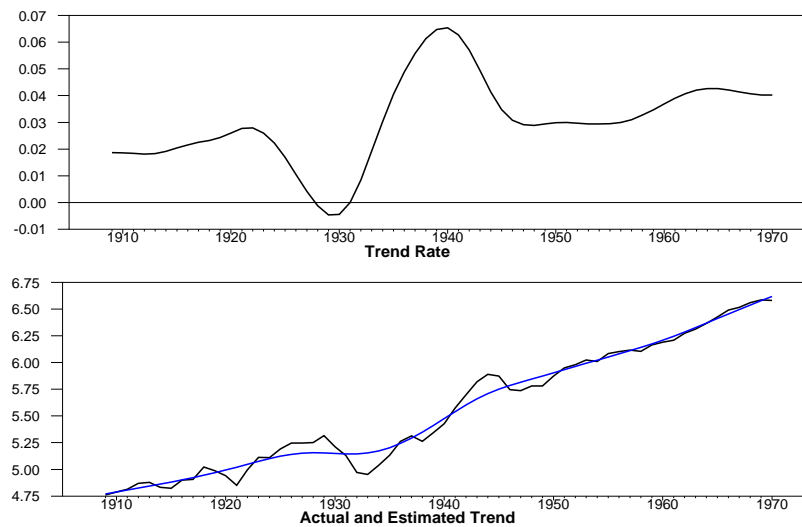


Figure 2.3: Local Trend Model. Hodrick-Prescott ratio

the following decompositions of the series y :

$$y = TC \times S \times I$$

$$\log y = \log TC + \log S + \log I$$

The first is multiplicative, the second is log additive. (Log additive is used for most series now). TC is the trend-cycle, S the seasonal and I the irregular. In X11, these are estimated using a sequence of filters, which operate across consecutive periods to estimate the trend-cycle and for specific months (quarters) across years to estimate the seasonal. It includes several iterations to refine the estimates and identify and eliminate outliers.

An alternative to this is to use a state-space model on the log additive form to isolate the three components. The local trend model is doing that for the two pieces TC and I . We just need a third component to handle seasonality.

What type of behavior do we expect out of the seasonal component S :

1. The difference between its values at a full year separation is “small”.
2. Over the course of a year, it averages either zero or close to it, so the overall level of the series is picked up by the TC .

If “small” in #1 is, in fact, zero, we’re really just using seasonal dummies. We would like to generalize that to allow for a *moving* seasonal.

In some cases, where the seasonal is linked to weather, we would also expect to see the seasonal not move a great deal from one month to the next. If the seasonal is more due to the secular calendar, that might be quite unreasonable—think about December versus January for retail sales—so we don’t want to build that into *all* models for seasonals.

In lag notation, #1 can be written loosely as

$$(1 - L^s) y_t \approx 0$$

We can factor the lag polynomial to get

$$(1 - L) (1 + L + L^2 + \dots + L^{s-1}) y_t \approx 0$$

The first factor in the polynomial isn't very interesting: if the series itself is almost constant, then rather clearly the difference a year apart is close to zero. It's the second one that relates to the seasonal. An obvious translation of this being "small" is

$$S_t + S_{t-1} + \dots + S_{t-s+1} = \eta_t$$

So our first possible state-space representation for a seasonal component is

$$\mathbf{X}_t = \begin{bmatrix} S_t \\ S_{t-1} \\ \vdots \\ S_{t-s+3} \\ S_{t-s+2} \end{bmatrix} = \begin{bmatrix} -1 & -1 & \dots & -1 & -1 \\ 1 & 0 & \dots & 0 & 0 \\ 0 & \ddots & \ddots & \vdots & \vdots \\ \vdots & \ddots & 1 & 0 & 0 \\ 0 & \dots & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} S_{t-1} \\ S_{t-2} \\ \vdots \\ S_{t-s+2} \\ S_{t-s+1} \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix} \eta_t$$

This is an *additive* or *dummy variable* seasonal model. It requires no connection between adjacent months at all and thus is most appropriate for seasonals which aren't related mainly to weather.

To model a smoother seasonal, we can further decompose the seasonal difference filter to:

$$(1 - L)(1 + L) \left(1 - 2 \cos \frac{2\pi}{S} L + L^2 \right) \left(1 - 2 \cos \frac{4\pi}{S} L + L^2 \right) \dots \left(1 - 2 \cos \frac{2(S-1)\pi}{S} L + L^2 \right)$$

The second degree terms represent the *harmonics* of the seasonal: the cycles that are an exact fraction of the seasonal. For monthly data, these are cycles of 12 months, 6 months, 4 months, 3 months and 12/5 months. (The two month cycle is covered by the two "real" terms). Any regular cycle with those lengths will, of necessity, also repeat at 12 months. If we look at one of these in isolation:

$$\left(1 - 2 \cos \frac{2\pi}{S} L + L^2 \right)$$

any sine/cosine wave of period 12 with any amplitude and phase will be zeroed out by this. The other factors will do the same to the other harmonic cycles. By combining the various harmonics, any repeating seasonal pattern can be created. The *harmonic* or *trigonometric* or *Fourier* seasonal model is to include a separate (independent) two-state model for each of the harmonics, with an extra one-state model for the period 2 harmonic. These are combined additively

to allow for complete flexibility. The cycle models are generally written in the equivalent form:

$$\begin{bmatrix} s_t \\ s_t^* \end{bmatrix} = \begin{bmatrix} \cos \lambda & \sin \lambda \\ -\sin \lambda & \cos \lambda \end{bmatrix} \begin{bmatrix} s_{t-1} \\ s_{t-1}^* \end{bmatrix} + \begin{bmatrix} \kappa_t \\ \kappa_t^* \end{bmatrix}$$

where $\lambda = \frac{2\pi j}{S}; j = 1, \dots, [(S-1)/2]$. The first term in each is the seasonal at t . This leads to a fairly complicated state representation, with a block diagonal **A** matrix and a **C** matrix which has 1's in every other slot to add up all the cycles. For that reason, it's generally best to use the procedure **@SeasonalDLM**, which will set up either type of seasonal model. You use one of the forms:

```
@SeasonalDLM(type=additive, a=a, f=f, c=c)
@SeasonalDLM(type=fourier, a=a, f=f, c=c)
```

This creates the component matrices. Note that the additive model has only one shock per period, while the Fourier model has $S-1$. This means that the **SW** matrix for the additive model is just a scalar, while it's an $(S-1) \times (S-1)$ matrix for the Fourier model.

Both of these forms, however, have $S-1$ unit roots. Because these only model the seasonal *deviations*, they need to be combined with some model (such as the local level or local trend) to give the overall level of the data. Thus we need (not surprisingly) at least S observations in order to get a full set of information about the model—one point for each period of the year.

Example 2.1 Local Level vs Local Trend Model

This demonstrates use of the local trend model (and contrasts with an inadequate choice of a local level model), applied to annual US GNP.

```

open data nelsonplosser.rat
calendar(a) 1909
data(format=rats) 1909:1 1970:1 realgnp
set logy = log(realgnp)
*
* Local level model
*
* Guess values for variances
*
filter(type=flat,extend=repeat,width=11) logy / filtered
sstats(mean) / (logy-filtered)^2>>sigsqeps
compute sigsqeta=.01*sigsqeps
*
dlm(a=1.0,c=1.0,sv=sigsqeps,sw=sigsqeta,presample=diffuse,$
    y=logy,vhat=vhat,svhat=svhat)
set resid = %scalar(vhat)/sqrt(%scalar(svhat))
spgraph(vfields=2)
    graph(hlabel="Log US Real GNP")
    # logy
    graph(hlabel="Recursive Residuals from Local Level Model")
    # resid
spgraph(done)
*
* Local trend model, using the Hodrick-Prescott ratio of variances.
*
@localdlm(type=trend,shocks=trend,a=at,f=ft,c=ct)
dlm(a=at,c=ct,f=ft,sv=sigsqeps,sw=sigsqeps/100.,presample=diffuse,$
    y=logy,vhat=vhat,svhat=svhat)
set resid = %scalar(vhat)/sqrt(%scalar(svhat))
graph(hlabel="Recursive Residuals from Local Trend Model")
# resid
*
* Redo with type=smooth to get component estimates
*
dlm(a=at,c=ct,f=ft,sv=sigsqeps,sw=sigsqeps/100.,presample=diffuse,$
    y=logy,type=smooth) / xstates
*
* Extract the local trend (first state) and the trend rate (second state)
*
set trend = xstates(t) (1)
set rate = xstates(t) (2)
*
spgraph(vfields=2,footer="Local Trend Model. Hodrick-Prescott ratio")
    graph(hlabel="Trend Rate")
    # rate
    graph(hlabel="Actual and Estimated Trend") 2
    # logy
    # trend
spgraph(done)

```

Estimating Variances

3.1 The Likelihood Function

In the local level, local trend and seasonal models, the only free parameters are the variances. To this point, we've pegged values for those. How can we *estimate* them?

The Kalman filter gives us the predictive density for \mathbf{Y}_t given $t-1$ as:

$$\mathbf{Y}_t \sim N(\mu_t + \mathbf{C}'_t \mathbf{X}_{t|t-1}, \mathbf{C}'_t \Sigma_{t|t-1} \mathbf{C}_t + \mathbf{N}_t)$$

We can use this with the standard time-series trick of factoring by sequential conditionals to get the full sample likelihood:

$$p(\mathbf{Y}_1, \dots, \mathbf{Y}_T) = p(\mathbf{Y}_1)p(\mathbf{Y}_2|\mathbf{Y}_1)p(\mathbf{Y}_3|\mathbf{Y}_1, \mathbf{Y}_2) \cdots p(\mathbf{Y}_T|\mathbf{Y}_1, \mathbf{Y}_2, \dots, \mathbf{Y}_{T-1}) \quad (3.1)$$

The variances governing the state-space model are often called *hyperparameters*, since the states are also parameters, and may, in many cases, be the actual objects of interest. Our task is to maximize the likelihood over these hyperparameters, in order to better estimate the states.

It looks as if this should be quite straightforward. A pass through the Kalman filter produces the predictive errors and variances, and we can evaluate the log likelihood by summing the log densities of the Normals. Evaluating the log likelihood given the parameters *is* just that simple. What complicates this is that there is nothing in the likelihood itself that constrains the individual variances to be non-negative. The only requirement for evaluating the log likelihood element at t is that the predictive covariance matrix:

$$\mathbf{C}'_t \Sigma_{t|t-1} \mathbf{C}_t + \mathbf{N}_t$$

be positive definite. That means that \mathbf{N}_t could be negative if the first term is positive enough and vice versa. And, unfortunately, this problem comes up all too often.

There are two main approaches for dealing with this. The first is to estimate the model unconstrained: if one variance comes in negative, zero it out and re-estimate. If that works, it should be the global maximum. If *two* variances come in negative, it's not as clear how to proceed. In that case, it's probably best to use RATS' ability to put inequality constraints on parameters.

The other method is to parameterize the variances in logs. Although that doesn't allow a true zero value, a variance which really should be zero will show up as something like -30 in logs.

It's also possible to use Bayesian methods rather than straight maximum likelihood. With a prior on the variance parameters that's constrained to be non-negative (an inverse chi-squared is the standard choice), you'll never even look at the likelihood at negative values.

There's one other numerical problem that can come up when you estimate the variances directly. Even non-zero variances can sometimes be *very* small. For instance, the variance on the trend rate shock is often a value like 10^{-8} . It can be very hard for numerical optimizers to deal with parameters that have such naturally small scales, since it's not easy to distinguish between a parameter which is naturally small and one that is small because it's really supposed to be zero. You can sometimes fix this fairly easily by just multiplying the data (*after* such things as log transformation) by 100. That increases all the component variances by a factor of 10^4 , which is usually enough to fix this problem.

One other way to deal with the problem of scale is to concentrate out one variance. As has been mentioned several times, the estimates of the means for the states are the same if all variances are multiplied by the same constant. For simplicity, let's assume that we have just one observable. Suppose that we write one of the variances (typically the measurement error variance) as λ and write all other variances as something like $\lambda \times \theta$. The sequence of predictive densities can be written as:

$$e_t \sim N(0, \lambda \sigma_t^2)$$

where e_t is the prediction error and σ_t^2 is the predictive variance computed for $\lambda = 1$. We can do this because the prediction (and thus the prediction error) is independent of λ , and the prediction error variance just scales by λ .¹ The log likelihood is thus (ignoring constants):

$$\sum_t \left(-\frac{1}{2} \log(\lambda \sigma_t^2) - \frac{1}{2} \frac{e_t^2}{\lambda \sigma_t^2} \right) = -\frac{T}{2} \log \lambda - \frac{1}{2\lambda} \sum_t \frac{e_t^2}{\sigma_t^2} - \frac{1}{2} \sum_t \log(\sigma_t^2)$$

The maximizing value for λ is

$$\hat{\lambda} = \frac{1}{T} \sum_t \frac{e_t^2}{\sigma_t^2}$$

and the concentrated log likelihood reduces to

$$-\frac{T}{2} \log \hat{\lambda} - \frac{T}{2} - \frac{1}{2} \sum_t \log(\sigma_t^2) \tag{3.2}$$

which can be optimized over the reduced parameter set θ .

¹Both e_t and σ_t^2 are functions of θ . We're suppressing that to keep the focus on λ .

What does this accomplish? The main advantage is that it takes one important part of the variance scale and allows it to be calculated exactly and directly—no numerical derivatives, no convergence checks. It also makes it more likely that you can come up with reasonable initial guess values for the other parameters, since they will be independent of scale of the data. Generally you just need to be within an order of magnitude or two on the guess values in order to get a well-behaved optimization.

If you do decide to concentrate out a variance, make sure you pick one that you think unlikely (impossible?) to be zero. If you standardized on a zero, the ratios for the other parameters won't be defined.

3.2 Estimating the Local Level Model

Example 3.1 shows four different parameter setups for estimating the two parameters in the local level model for the Nile river data. These are:

1. Both parameters included; both in direct (not logged) form
2. Both parameters included; both in log form
3. Measurement variance concentrated out; ratio in direct form
4. Measurement variance concentrated out; ratio in log form

In order to estimate parameters, you need to include a **NONLIN** instruction to indicate what parameters are to be estimated, give them initial guess values (at least in direct form, the default zero values won't work) and include an option on **DLM** to pick the optimization method to be used. With most simple models `METHOD=BFGS`, which is the main “hill-climbing” algorithm, will work fine.

The estimation can be quite sensitive to guess values, at least when you're not concentrating out one variance. In this example, we get a guess value for the measurement error variance with:

```
filter(type=centered,width=11,extend=repeat) nile / fnile
sstats(mean) / (nile-fnile)^2>>initsigsq
```

This takes a simple 11 term centered moving average as a crude estimate of the local level, and uses the mean squared difference between the data and that as the guess value. For either the local level or local trend model, this should be fairly close (within maybe 30%) for σ_ε^2 , which is all you need. The ratio of the two variances is guessed to be around .01. That turns out to be low by a factor of 10, but it's close enough to work.

The first estimates are done with:

```
compute sigsqeps=initsigsq,sigsqeta=sigsqeps*.01
nonlin sigsqeps sigsqeta
dlm(a=1.0,c=1.0,y=nile,sv=sigsqeps,sw=sigsqeta,$
presample=diffuse,method=bfgs)
```

The fourth method (which is what is used in DK) is done with:

```

nonlin psi
compute psi=log(.01)
dlm(a=1.0,c=1.0,y=nile,sv=1.0,sw=exp(psi),var=concentrate,$
    exact,method=bfgs)
disp "Sigsqeps" %variance
disp "Sigsqeta" %variance*exp(psi)

```

As this is parameterized, PSI is the log of the ratio $\sigma_\eta^2/\sigma_\varepsilon^2$. The differences here are:

1. We include the option `VARIANCE=CONCENTRATE`
2. We use `SV=1.0` to peg **SV** as the standardized variance
3. `%VARIANCE` is the maximum likelihood estimate of the variance scale factor. We need to multiply the ratio by that in order to get the estimate of the second variance.

3.3 Estimating the Local Trend Model

Before you do this, you should first ask whether you really want to. Attempts to estimate the variances in local trend models often produce results which look “wrong”. In particular, the common problem is that maximum likelihood produces a trend which isn’t stiff enough to fit our understanding of what a trend should be. In short, there’s a reason that the HP filter has pegged rather than estimated ratios.

There are potentially three variance parameters in the local trend model. We’ll start with all three, though most of the time, you’ll find (as we will) that the level variance is zero. We set up the system matrices for the model using the `@LocalDLM` procedure allowing for both shocks.

```
@localdlm(type=trend,shocks=both,a=at,f=ft,c=ct)
```

For convenience, we’ll use the `@LocalDLMin` procedure to get the guess values for the two most important variances: the irregular and the trend rate. The σ_ξ^2 is initialized at a fraction of the irregular variance.

```
@localdlminit(irreg=sigsqeps,trend=sigsqzeta) logy
compute sigsqxi=sigsqeps*.01
```

With two shocks in the state equation, the `SW` option needs a 2×2 matrix. If we (as typical) assume that the two state shocks are uncorrelated, this will be a diagonal matrix. The simplest way to input this is with the `%DIAG` function, which makes a square diagonal matrix out of a vector. Our first go at this is:

```
dlm(a=at,c=ct,f=ft,sv=sigsqeps,sw=%diag(||sigsqxi,sigsqzeta||),$
    presample=diffuse,y=logy,method=bfgs)
```

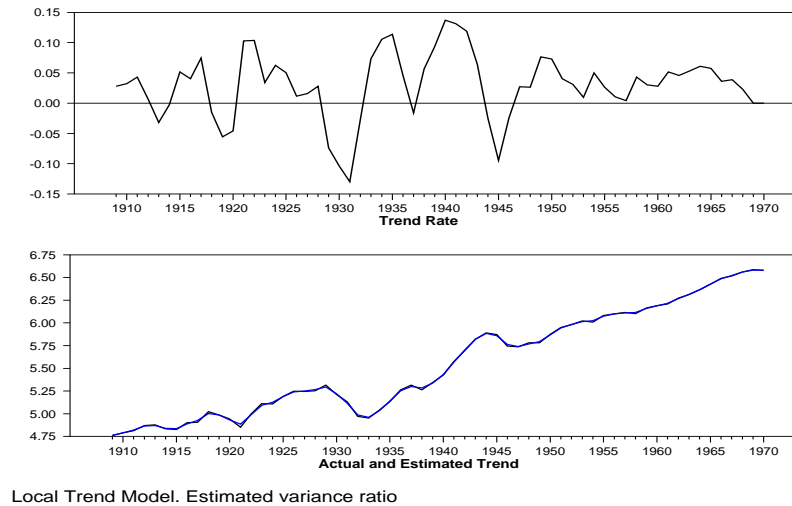


Figure 3.1: Local Trend Overfit

which gives us only one positive parameter. Interestingly, that turns out to be σ_{ξ}^2 . However, when two parameters go negative, you really don't have any clear guidance. The “obvious” fix here is to zero out the variance that usually causes problems. Thus, we do:

```
nonlin sigsqeps sigsqzeta sigsqxi=0.00
@localdlmunit(irreg=sigsqeps,trend=sigsqzeta) logy
dlm(a=at,c=ct,f=ft,sv=sigsqeps,sw=%diag(||sigsqxi,sigsqzeta||),$
presample=diffuse,y=logy,method=bfgs)
```

Setting up the parameter set that way drops `SIGSQXI` out of the estimation output.² The likelihood is down a bit, but that's to be expected, since we put a constraint on.

The “problem” comes when we re-do this with `TYPE=SMOOTH` to get the full-sample estimates of the components:

```
dlm(a=at,c=ct,f=ft,sv=sigsqeps,sw=%diag(||sigsqxi,sigsqzeta||),$
presample=diffuse,y=logy,type=smooth) / xstates
set trend = xstates(t) (1)
set rate = xstates(t) (2)
```

This produces the estimates for the trend rate and for the trend shown in Figure 3.1. Notice that the trend rate is quite variable and the smoothed estimate of the “trend” almost exactly matches the data.

As a final cautionary tale, let's go back and peg to zero the two variances which were negative in the first estimation:

²We need to redo the initial values since the previous estimation made the two remaining parameters negative.


```
nonlin sigsqeps=0.00 sigsqzeta=0.00 sigsqxi
```

Redoing the estimates with this produces a higher likelihood. In fact, this seems to be the global maximum across the permitted parameter set. And yet it's pointless as a method of estimating a trend in GNP. Without a measurement error, the smoothed value of the trend is not just similar to the data, it is *exactly* the data.

Why does this model create so many problems? Given the length of economic time series, there just isn't strong enough information to sort out information by level of persistence, particularly into *three* categories (stationary, one unit root, double unit root). This is not surprising considering that we still haven't even settled the question of whether there's even one unit root in a series like this. In practice, some sort of variance ratio pegging or at least a prior on the variance ratios would seem to be the best bet.

3.4 Diagnostics

The main diagnostics are based upon the predictive errors. Under the assumption that the model is correct, these are an independent sequence. When scaled through by the standard error of prediction, they are i.i.d standard Normal.³ Basically any test which can be applied to such a sequence can be used.

The procedure `@STAMPDiags` will do a standard set of diagnostics used by the STAMP software (co-written by Siem Jan Koopman). This includes a Ljung-Box *Q* test for serial correlation, a test for normality, and a test for heteroscedasticity. You can also use the procedure `@CUSUMTests`, which does the CUSUM and CUSUMQ tests for structural change as described in Brown et al. (1975). In example 3.3, we add the `VHAT` and `SVHAT` options to fetch the predictive residuals and their variance, convert to the recursive (or standardized predictive) residuals and run the tests:

```
dln(a=1.0,c=1.0,y=nile,sv=exp(logeps),sw=exp(logeta),$
  presample=diffuse,method=bfgs,vhat=vhat,svhat=svhat)
set resids = %scalar(vhat)/sqrt(%scalar(svhat))
@STAMPDiags resids
@CUSUMTests resids
```

The only one of these that is even slightly indicative of a problem is the CUSUMQ test, which goes slightly out of bounds at 1917. However, this seems to be due to the three largest residuals being within a small span near that point. There certainly doesn't seem to be any systematic change to the variance.

³This is describing what happens with one observable. A generalization to a vector of observables requires pre-multiplying the prediction errors by the inverse of a factor of the covariance matrix.

Example 3.1 Estimating the Local Level Model

This demonstrates various methods for estimating the component variances in the local level model. It's based upon Illustration 2.10.3 in Durbin & Koopman (2012) and described in detail in Section 3.2.

```
open data nile.dat
calendar 1871
data(format=free,org=columns,skips=1) 1871:1 1970:1 nile
*
* Guess values
*
filter(type=centered,width=11,extend=repeat) nile / fnile
sstats(mean) / (nile-fnile)^2>>initsigsq
*
*****
*
* Method one: all parameters included, in direct form
*
compute sigsqeps=initsigsq,sigsqeta=sigsqeps*.01
nonlin sigsqeps sigsqeta
*
dlm(a=1.0,c=1.0,y=nile,sv=sigsqeps,sw=sigsqeta,$
    presample=diffuse,method=bfgs)
*
*****
*
* Method two: all parameters included, log form
*
nonlin logeps logeta
compute logeps=log(initsigsq),logeta=logeps-4.0
*
dlm(a=1.0,c=1.0,y=nile,sv=exp(logeps),sw=exp(logeta),$
    presample=diffuse,method=bfgs)
disp "Sigsqeps" exp(logeps)
disp "Sigsqeta" exp(logeta)
*
*****
*
* Method three: concentrating out measurement error variance. Direct
* form. SV=1.0, SW is the ratio parameter. Include the option
* VARIANCE=CONCENTRATE.
*
compute theta=.01
nonlin theta
*
dlm(a=1.0,c=1.0,y=nile,sv=1.0,sw=theta,var=concentrate,$
    presample=diffuse,method=bfgs)
disp "Sigsqeps" %variance
disp "Sigsqeta" %variance*theta
*
*****
*
```

```

* Method four: concentrating out measurement error variance. Log
* form on other ratios. SV=1.0, SW=exp(log ratio). Include the
* option VARIANCE=CONCENTRATE.
*
nonlin psi
compute psi=log(.01)
*
dlm(a=1.0,c=1.0,y=nile,sv=1.0,sw=exp(psi),var=concentrate,$
    presample=diffuse,method=bfgs)
disp "Sigsqeps" %variance
disp "Sigsqeta" %variance*exp(psi)

```

Example 3.2 Estimating the Local Trend Model

This demonstrates estimation of the component variances in the local trend model. See Section 3.3 for details.

```

open data nelsonplosser.rat
calendar(a) 1909
data(format=rats) 1909:1 1970:1 realgnp
set logy = log(realgnp)
*
nonlin sigsqeps sigsqzeta sigsqxi
*
* Set up the DLM equations allowing for both shocks
*
@localdlm(type=trend,shocks=both,a=at,f=ft,c=ct)
*
* Get guess values for the variances for the trend rate and the
* irregular. Start with the level variance as a small fraction of
* the irregular.
*
@localdlminit(irreg=sigsqeps,trend=sigsqzeta) logy
compute sigsqxi=sigsqeps*.01
*
dlm(a=at,c=ct,f=ft,sv=sigsqeps,sw=%diag(||sigsqxi,sigsqzeta||),$
    presample=diffuse,y=logy,method=bfgs)
*
* Redo with sigsqxi pegged to 0
*
nonlin sigsqeps sigsqzeta sigsqxi=0.00
@localdlminit(irreg=sigsqeps,trend=sigsqzeta) logy
dlm(a=at,c=ct,f=ft,sv=sigsqeps,sw=%diag(||sigsqxi,sigsqzeta||),$
    presample=diffuse,y=logy,method=bfgs)
*
* Use type=smooth to get component estimates
*
dlm(a=at,c=ct,f=ft,sv=sigsqeps,sw=%diag(||sigsqxi,sigsqzeta||),$
    presample=diffuse,y=logy,type=smooth) / xstates
*
* Extract the local trend (first state) and the trend rate (second
* state)

```

```

*
set trend = xstates(t) (1)
set rate  = xstates(t) (2)
*
spgraph(vfields=2, footer="Local Trend Model. Estimated variance ratio")
  graph(hlabel="Trend Rate")
  # rate
  graph(hlabel="Actual and Estimated Trend") 2
  # logy
  # trend
spgraph(done)
*
* Redo estimation, pegging the two variances that were negative in
* the first run.
*
nonlin sigsqeps=0.00 sigsqzeta=0.00 sigsqxi
@localdlmunit(irreg=sigsqeps, trend=sigsqzeta) logy
compute sigsqxi=sigsqeps*.01
dlm(a=at, c=ct, f=ft, sv=sigsqeps, sw=%diag(||sigsqxi, sigsqzeta||), $
  presample=diffuse, y=logy, method=bfgs, type=smooth) / xstates
set trend = xstates(t) (1)
set rate  = xstates(t) (2)
graph(hlabel="Actual and Estimated Trend") 2
# logy
# trend

```

Example 3.3 Diagnostics

Demonstrates various diagnostics for state-space models. This is adapted from Illustration 2.12.3 in Durbin & Koopman (2012) and is discussed in Section 3.4.

```

open data nile.dat
calendar 1871
data(format=free, org=columns, skips=1) 1871:1 1970:1 nile
*
* Guess values
*
filter(type=centered, width=11, extend=repeat) nile / fnile
sstats(mean) / (nile-fnile)^2>>initsigsq
*
* Estimate
*
nonlin logeps logeta
compute logeps=log(initsigsq), logeta=logeps-4.0
*
dlm(a=1.0, c=1.0, y=nile, sv=exp(logeps), sw=exp(logeta), $
  presample=diffuse, method=bfgs, vhat=vhat, svhat=svhat)
*
* Get recursive residuals
*
set resids = %scalar(vhat)/sqrt(%scalar(svhat))
*

```

```
* Compute standard diagnostics
*
@STAMPDiags resids
@CUSUMTests resids
```

Initialization

We have so far generally avoided the question of what to do about the pre-sample information on the mean and variance for the states—in our notation, $\mathbf{X}_{0|0}$ and $\Sigma_{0|0}$. For small models, the method used in the West and Harrison *Kurita* example (Example 1.2) is probably as good as any—just make the mean a number in the right range, with a variance high enough to cover any likely value. That will match up with your prior information, and will let the data dominate.

That won't work well in more complicated models with more states. For instance, with a seasonal model (particularly with the Fourier form), we almost certainly don't have information at that level of detail. It would be nice to have a way to generate an appropriate prior based solely on the other inputs.

4.1 Ergodic Solution

If the state model is *not* time-invariant, that is, if one of \mathbf{A} , \mathbf{Z} , \mathbf{F} or \mathbf{SW} changes over time, there won't really be such a solution—the type of states the model generates will be different depending upon where you are. Fortunately, most models in common use *are* time-invariant. As such, we can think about solving for the “steady-state”

$$\mathbf{E}\mathbf{X} = \mathbf{A}\mathbf{E}\mathbf{X} + \mathbf{Z} \quad (4.1)$$

$$\text{var } \mathbf{X} = \mathbf{A} (\text{var } \mathbf{X}) \mathbf{A}' + \text{var } \mathbf{W} \quad (4.2)$$

assuming that the unconditional expectation and variance of \mathbf{X} is the same at each period. If this solution exists, it's known as the *ergodic* solution—these would be the mean and variance of very long realizations of the process.

These solutions will, in fact, exist if the eigenvalues of \mathbf{A} are inside the unit circle. The first equation is solved very simply for the mean, and if, as is usually the case, $\mathbf{Z} = 0$, the mean will just be zero anyway.

The variance equation, while taking a somewhat odd form, is, in fact, a linear equation in the elements of $\text{var } \mathbf{X}$. By expanding this, it can be solved by standard linear algebra techniques. The standard textbook solution for this¹ is to rearrange it into the linear system:

$$[\mathbf{I} - \mathbf{A} \otimes \mathbf{A}] \text{vec}(\Sigma_0) = \text{vec}(\Sigma_{\mathbf{W}})$$

¹See, for instance, DK, page 112.

where we're writing Σ_0 for $\text{var } \mathbf{X}$ and Σ_W for $\text{var } \mathbf{W}$. As written, this has some redundant elements since Σ_0 is symmetric. Still, with those eliminated, it requires solving a linear system with $n(n+1)/2$ components. Since solution of a system of N equations is $O(N^3)$ in arithmetic operation count, this makes this solution procedure $O(n^6)$. This starts to dominate the calculation time for even fairly modest values of n .² Instead of doing this “brute force” solution, RATS uses a more efficient technique described in Appendix E and Doan (2010), which has a solution time that is $O(n^3)$.

4.2 Diffuse Prior

In practical state-space models, it's rare for all the eigenvalues of the transition matrix to be inside the unit circle. In the simplest of the non-stationary models, the local level model, the “solution” for (4.2) is $\text{var}(\mathbf{X}) = \infty$. Suppose that we use a prior of mean 0 and variance κ where κ is “large”. The Kalman updating equation for the mean is

$$x_{1|1} = x_{1|0} + \frac{\sigma_{1|0}^2}{\sigma_{1|0}^2 + \sigma_v^2} (y_1 - x_{1|0})$$

With $\sigma_{1|0}^2 = \kappa + \sigma_w^2$, if κ is large relative to σ_w^2 and σ_v^2 , this will be approximated by

$$x_{1|1} = x_{1|0} + 1.0 (y_1 - x_{1|0})$$

or $x_{1|1} = y_1$. This shouldn't come as a surprise—with a very high prior variance, the data will dominate completely. The only data information after one data point is that one value. This calculation isn't particularly sensitive to the precise value of κ as long as it's big enough.

The variance calculation is more problematical. The calculation (in practice) is done by:

$$\sigma_{1|1}^2 = \sigma_{1|0}^2 - \frac{(\sigma_{1|0}^2)^2}{\sigma_{1|0}^2 + \sigma_v^2}$$

While we can rearrange this (algebraically) to give $\sigma_{1|1}^2 \approx \sigma_v^2$, the calculation will be done in software as the subtraction above. Subtracting two very large and almost equal numbers to produce a small number runs the danger of a complete loss of precision. With the standard 15 digit precision on a modern computer, if κ is greater than $10^{15}\sigma_v^2$, the calculation above will give zero.

In order for this to give us the desired results by using large values of κ , we need to choose a value which is large enough, but not too large. For this model, we can probably figure out a way to do that. With a more complicated model

²The most time-consuming calculation in one Kalman filter update is computing $\mathbf{A}\Sigma_{t|t}\mathbf{A}'$ which is $O(n^3)$, repeated T times. Thus, if $n^3 \gg T$, $n^6 \gg n^3T$ and the greatest time will be spent on the initial variance calculation.

with multiple variances, some of which might be quite small, we're less likely to find such a value.

Note, by the way, that one suggested strategy is to try several values for κ , and check for sensitivity.³ While better than just picking an arbitrary number, looking for sensitivity on estimates of the mean will likely overlook the more troublesome calculation of the variance. And the calculation that is *most* affected by approximation error is the state variance for Kalman *smoothing*. Those are highly suspect in any calculation done this way.⁴

This calculation is effectively identical if the model is, in fact, stationary, but a diffuse prior is being used for convenience. If, instead of the random walk, the underlying state equation is $x_t = \rho x_{t-1} + w_t$ with $|\rho| < 1$, then we still will get $x_{1|1} \approx y_1$ and $\sigma_{1|1} \approx \sigma_v^2$. This is not the same as what we would get if we used the ergodic variance, but isn't necessarily an unreasonable procedure.⁵

An alternative to approximating this was provided by Koopman (1997), which is *exact diffuse initialization*. This was later refined in Durbin & Koopman (2012). It uses the actual limits as $\kappa \rightarrow \infty$. It is implemented by writing covariance matrices in the form $\Sigma_\infty \kappa + \Sigma_*$ where Σ_∞ is the “infinite” part and Σ_* is the “finite” part. These are updated separately as needed.

This works in the following way for the local level model. The prior is mean 0. The prior covariance is $[1]\kappa + [0]$.⁶ Moving to $\sigma_{1|0}$ adds the finite σ_w^2 , producing $[1]\kappa + [\sigma_w^2]$. The predictive variance for y_1 further adds σ_v^2 , to make $[1]\kappa + [\sigma_w^2 + \sigma_v^2]$. The Kalman gain is the multiplier on the prediction error used in updating the state. That's

$$([1]\kappa + [\sigma_w^2]) ([1]\kappa + [\sigma_w^2 + \sigma_v^2])^{-1} \quad (4.3)$$

Inverses can be computed by matching terms in a power series expansion in κ^{-1} as shown in Appendix C. For computing the mean, all we need are the 1 and κ^{-1} terms, which allows us to write (4.3) as approximately:

$$([1]\kappa + [\sigma_w^2]) ([0] + [1]\kappa^{-1}) = [1] + [\sigma_w^2]\kappa^{-1}$$

where the approximation in the inverse will produce only an additional term on the order of κ^{-2} . We can *now* pass to the limit (that is, ignore all but the finite term) and get the Kalman gain as 1 (exactly), as we expected.

³For instance, Koopman et al. (1999) suggest calculating once with an standard “large” value, then recalculating with a revised value based upon the initial results.

⁴In a model with more than one diffuse state, the filtered covariance matrix after a small number of data points will still have some very large values. Those are only reduced to “finite” numbers as the result of the smoothing calculation, which uses information obtained by calculating out to the end of the data set and back, accumulating roundoff error along the way.

⁵With the ergodic variance, $x_{1|1}$ will be a weighted average of 0 and y_1 with weights equal to the ergodic variance and σ_v^2 respectively.

⁶We'll keep the different components in brackets.

Getting the filtered *variance* correct requires greater accuracy. For that, we'll need to expand the inverse to the κ^{-2} level. We need this because the updating equation is:

$$([1]\kappa + [\sigma_w^2]) - ([1]\kappa + [\sigma_w^2]) ([1]\kappa + [\sigma_w^2 + \sigma_v^2])^{-1} ([1]\kappa + [\sigma_w^2])$$

and the inverse is sandwiched between two factors, each with a κ . As a result, terms in κ^{-2} in the inverse will produce a finite value when this is multiplied out, and only κ^{-3} and above will be negligible. The second order expansion of the required inverse is

$$([0] + [1]\kappa^{-1} - [\sigma_w^2 + \sigma_v^2]\kappa^{-2})$$

The calculation of the filtered variance is most conveniently done as

$$\left(1 - ([1]\kappa + [\sigma_w^2]) ([1]\kappa + [\sigma_w^2 + \sigma_v^2])^{-1}\right) ([1]\kappa + [\sigma_w^2]) \quad (4.4)$$

With the second order expansion, the product

$$([1]\kappa + [\sigma_w^2]) ([1]\kappa + [\sigma_w^2 + \sigma_v^2])^{-1}$$

produces

$$1 - \sigma_v^2 \kappa^{-1} - O(\kappa^{-2}) \quad (4.5)$$

Plugging that into (4.4) results in $\sigma_v^2 + O(\kappa^{-1})$. Passing to the limit gives us the result we want.

Note that the calculation here that causes the problem for large, but finite, values is subtracting (4.5) from 1. If we have a loss of precision there, we won't get σ_v^2 out—we'll get zero.

While we started with an “infinite” variance, after seeing one data point, we now have $x_{1|1} = y_1$, $\sigma_{1|1} = \sigma_v^2$, which are perfectly reasonable finite values. We can Kalman filter in the standard fashion from there.

We won't go through the details of what happens when you apply this to the local trend model. That is, however, done in the Koopman paper if you're interested. What you might expect to happen is that the first data point observed resolves the level and the second resolves the trend rate. That's almost correct, but not quite. How the diffuse filter handles those early data points depends upon the “shape” of the infinite component of the covariance matrix. With the simplest matrix (the identity), the first data point actually splits the gain with 50% going to the level and 50% to the trend rate. It's not until you see the second data point that it resolves both. You might think this is a problem, but it really isn't. The predictive variance for the first two data points is infinity—the first because we don't know anything, the second because we don't know what's level and what's trend. Those two data points just get skipped in computing the likelihood function. If you compute recursive residuals, those two data points will give missing values since the standardizing variance doesn't

exist. So the odd breakdown doesn't affect the estimation or diagnostics. If you Kalman smooth, they'll get corrected to the proper values regardless.

The fact that, in the end, it doesn't matter *which* full rank matrix you use for the diffuse prior is well beyond the scope—you can see a proof in Doan (2010). That it *doesn't* is very convenient.

In general, if you start out with a diffuse matrix of rank r , the rank will reduce by the number of observables in each period. For monthly data with a local trend and seasonal model, we have 13 states. We need to see each month at least once to estimate their relative component, and one extra so we can figure out the trend rate. If, in the first year, May is missing, we won't resolve that seasonal until we actually see a value for May, so it will take a few extra data points before we can pass to the standard Kalman filter. But, in the absence of missing data, you can expect that you'll have r data points with missing values for the recursive residuals at the start.

4.3 Mixed Stationary and Non-Stationary Models

When all roots are unit roots, you can use the option `PRESAMPLE=DIFFUSE`, which requests exact diffuse treatments of the initial conditions. If all roots are inside the unit circle, the option `PRESAMPLE=ERGODIC` will do the ergodic solutions.

What happens if you have a mix of stationary and non-stationary states? For instance, you might have a model that adds stationary AR errors to a local trend. If you can block the states into the two categories, this is fairly easy. For instance, if the model is:

$$\begin{bmatrix} u_t \\ x_t \end{bmatrix} = \begin{bmatrix} \rho & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} u_{t-1} \\ x_{t-1} \end{bmatrix} + \begin{bmatrix} \epsilon_t \\ \nu_t \end{bmatrix}$$

the (partially diffuse) initial covariance matrix is:

$$\begin{bmatrix} \frac{\sigma_\epsilon^2}{1-\rho^2} & 0 \\ 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \kappa$$

This does the ergodic solution for the block of stationary states and inserts an identity submatrix for the non-stationary ones. You can input this explicitly into the **DLM** instruction using the options `SX0` for the finite part and `SH0` for the infinite part. Note that both the options need full $n \times n$ matrices.

Suppose instead, that we have the process $y_t = (1+\rho)y_{t-1} - \rho y_{t-2} + \epsilon_t$, which is an ARIMA(1,1,0) process. This has one unit root and one stationary one. While it's possible to write this in terms of $y_t - y_{t-1}$ (stationary) and y_{t-1} (non-stationary), for some applications it might be simpler to keep it in the original form. The state-space representation for that is:

$$\mathbf{X}_t = \begin{bmatrix} y_t \\ y_{t-1} \end{bmatrix} = \begin{bmatrix} 1+\rho & -\rho \\ 1 & 0 \end{bmatrix} \begin{bmatrix} y_{t-1} \\ y_{t-2} \end{bmatrix} + \begin{bmatrix} \epsilon_t \\ 0 \end{bmatrix} \quad (4.6)$$

You can also use `PRESAMPLE=ERGODIC` with this type of model. **DLM** will go through the extra step of analyzing the **A** matrix to determine the location of unit (or unstable) roots and will create the necessary transformation to isolate them. If, as in this case, there are non-stationary roots, it will produce a Generalized Ergodic initialization, which will have a partially diffuse, partially stationary variance. The description of how this is done is described (briefly) in Appendix E and in greater detail in Doan (2010).

Note that this can be a potential source of trouble if you're estimating the model and there is at least some chance that a state might be either stationary or non-stationary. For instance, in (4.6), the (unconditional) log likelihood will be discontinuous at $\rho = 1$ because a second data point gets dropped from the likelihood calculation once there are two unit roots, rather than one unit root and a very large but still stationary root. If there's a possibility of this happening, the safest way to handle it is to also include the option `CONDITION=2`. That drops the first two data points from the likelihood calculation *regardless of the number of unit roots*, and thus puts the mixed model on a comparable footing with the fully non-stationary one.⁷ You do *not* need to use the `CONDITION` option if the number of unstable roots is known—**DLM** automatically adjusts the likelihood function to take care of that.

⁷The filtering and smoothing calculations use the whole data set; `CONDITION` only affects the number of early data points at which the likelihood is ignored.

Practical Examples with a Single Observable

5.1 Basic Structural Models

Most models with a single observable use some combination of the local level or trend, possibly with one of the seasonal models. Harvey (1989) calls these *Basic Structural Models* or BSM. We’ve already seen quite a few examples of the local level and local trend model. We’ll now combine them with the seasonal model.

The data set is the famous “airline” data used by Box and Jenkins. This (Figure 5.1) is the basis for the “Airline” model, which is a multiplicative seasonal ARIMA $(0, 1, 1) \times (0, 1, 1)$ (one regular difference, one regular MA, one seasonal difference, one seasonal MA). That’s the best fitting small model for this data set, and is often used as a first guess for a fit for highly seasonal series more generally.

This has both a decided trend and a seasonal so the two obvious components are the local trend and the additive seasonal. The additive seasonal seems most appropriate since it’s the secular calendar rather than weather that governs most of the seasonality.

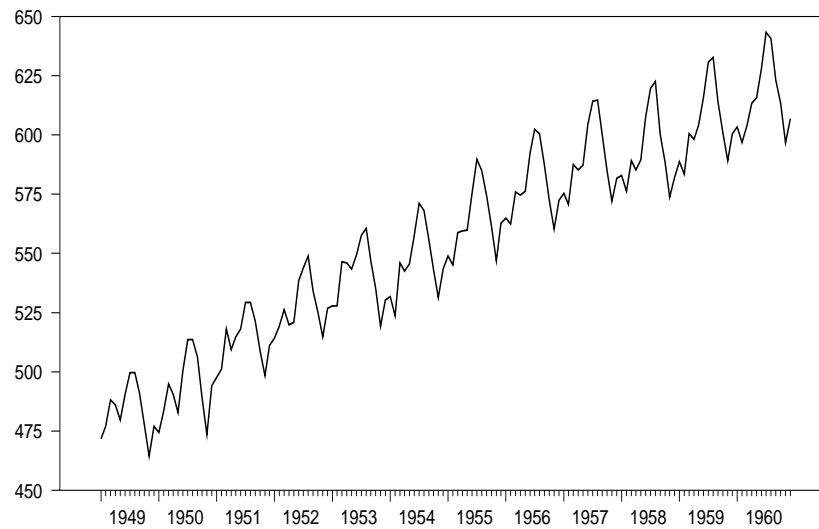
In the local trend model, we’ll allow for both shocks, although it’s likely that at least one will have to go. With both shocks included, we’ll have one observable being called upon to determine four shocks.

Set up the two component models and combine them to make the full model:

```
@localdlm(type=trend, shocks=both, a=at, c=ct, f=ft)
@seasonaldlm(type=additive, a=as, c=cs, f=fs)
compute a=at~\as, c=ct~~cs, f=ft~\fs
```

You can use `@LocalDLMin` with the `DESEASONALIZE` option to get the guess values. This removes a seasonal by regression, allowing guess values to be computed for the other parts. That corresponds to a zero variance for the seasonal.

```
nonlin sigsqeps sigsqxi sigsqzeta sigsqomega
@localdlminit(irreg=sigsqeps, trend=sigsqzeta, deseasonalize) lpass
compute sigsqxi=.01*sigsqeps, sigsqomega=0.0
```

**Figure 5.1:** (Log) Airline Passenger Data**Table 5.1:** Estimates of unconstrained BSM

DLM - Estimation by BFGS
Convergence in 28 Iterations. Final criterion was 0.0000098 <= 0.0000100
Monthly Data From 1949:01 To 1960:12
Usable Observations 144
Rank of Observables 131
Log Likelihood -373.31195

	Variable	Coeff	Std Error	T-Stat	Signif
1.	SIGSQEPS	1.20502	1.25204	0.96245	0.33583
2.	SIGSQXI	7.24771	1.90130	3.81198	0.00014
3.	SIGSQZETA	-0.00294	0.00118	-2.49550	0.01258
4.	SIGSQOMEGA	0.63562	0.43628	1.45690	0.14514

The `SW` matrix now has to cover the three state shocks. Since those are assumed to be independent, it will be done with `%DIAG`.

```
dlim(y=lpass,a=a,c=c,f=f,presample=diffuse,$
      sv=sigsqeps,sw=%diag(||sigsqxi,sigsqzeta,sigsqomega||),$
      method=bfgs) / xstates
```

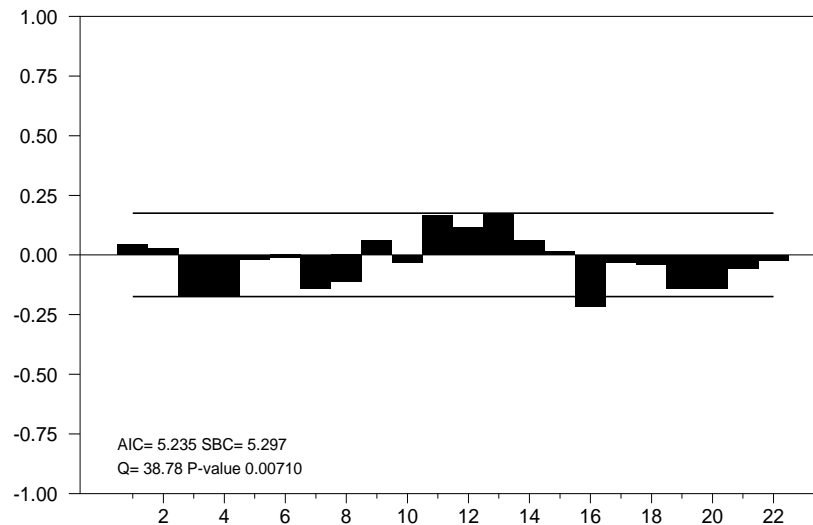
When estimated unconstrained, the variance on the trend rate (`SIGSQZETA`) comes in negative (Table 5.1).

This can be re-estimated with that pegged to zero, by altering the `NONLIN` instruction:

```
nonlin sigsqeps sigsqxi sigsqzeta=0.0 sigsqomega
dlim(y=lpass,a=a,c=c,f=f,presample=diffuse,$
      sv=sigsqeps,sw=%diag(||sigsqxi,sigsqzeta,sigsqomega||),$
      method=bfgs,vhat=vhat,svhat=svhat) / xstates
```

Table 5.2: Airline Data BSM Diagnostics

State Space Model Diagnostics		
	Statistic	Sig. Level
Q(22-2)	38.78	0.0071
Normality	0.40	0.8187
H(44)	0.84	0.5755

**Figure 5.2:** Correlations of Standardized Residuals

Diagnostics on the standardized residuals are done with:

```
set resids = %scalar(vhat)/sqrt(%scalar(svhat))
@STAMPDiags resids
@CUSUMTests resids
```

The CUSUM tests are fine. In the results from `@STAMPDiags` (Table 5.2), the one problem noted is the Q test for serial correlation. The graph of autocorrelations (Figure 5.2) shows quite a few autocorrelations which are marginally significant individually, with a pattern which tends to indicate that the seasonal component may not be adequate—11, 12, and 13 are all fairly large and of the same sign, and the large autocorrelations at 4, 7 and 8 all have seasonal “echoes” at 16, 19 and 20.

We Kalman smooth with that model to get the components. Since it was just estimated, we can just replace `METHOD=BFGS` with `METHOD=SOLVE`:

```
dlim(y=lpass,a=a,c=c,f=f,presample=diffuse,$
sv=sigsqeps,sw=%diag(||sigsqxi,sigsqzeta,sigsqomega||),$
method=solve,type=smooth) / xstates vstates
```

The local trend is captured by the first state, the seasonal by the third. To get the “deseasonalized” data, we subtract the seasonal from the actual data:

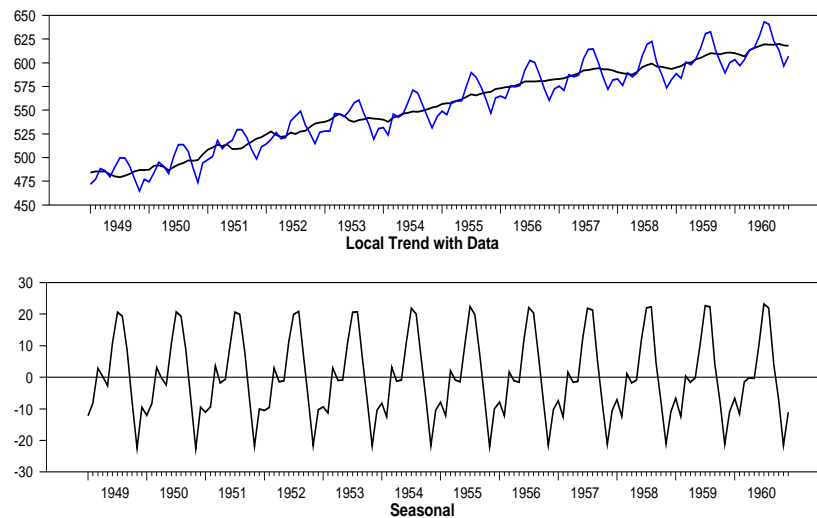


Figure 5.3: Airline Model Components

```
set trend      = xstates(t) (1)
set seasonal   = xstates(t) (3)
set deseason   = lpass-seasonal
```

In Figure 5.3, the seasonal component seems to change over the years, with a “mini-peak” in March eventually fading away, which likely explains the problem with the Q statistic. The one potential problem with these from a practical standpoint is that the “trend” seems to be a bit more uneven than we would like. It’s possible that the more flexible Fourier seasonal model would help shift some of that into the seasonal.

5.2 Trend plus Stationary Cycle

In the local trend model, the “cycle” (the gap between the series and the estimated trend) isn’t modeled explicitly. Instead, it’s just treated as the “uncorrelated” measurement error. In practice, of course, it isn’t actually uncorrelated, and we really only get reasonable results out of the model when we do something to constrain the ability of the trend to pick up short-term movements.

An alternative approach is to allow the cycle to be modeled more explicitly. The typical choice for this is a stationary AR(2) process, since it has enough flexibility to model actual (damped) sine wave cycles. The observable series is then the sum of a (non-stationary) trend model and a (stationary) cycle.

An example of this is provided in Clark (1987). The AR(2) “error” process in the local trend model needs to be incorporated into the state vector, giving the

set of equations

$$\begin{aligned}
 x_t &= x_{t-1} + \tau_{t-1} + \xi_t \\
 \tau_t &= \tau_{t-1} + \zeta_t \\
 c_t &= \varphi_1 c_{t-1} + \varphi_2 c_{t-2} + \varepsilon_t \\
 y_t &= x_t + c_t
 \end{aligned} \tag{5.1}$$

This converts into the state-space representation:

$$\begin{aligned}
 \mathbf{X}_t \equiv \begin{bmatrix} x_t \\ c_t \\ c_{t-1} \\ \tau_t \end{bmatrix} &= \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & \varphi_1 & \varphi_2 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{t-1} \\ c_{t-1} \\ c_{t-2} \\ \tau_{t-1} \end{bmatrix} + \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \xi_t \\ \varepsilon_t \\ \zeta_t \end{bmatrix} \\
 y_t &= [1, 1, 0, 0] \mathbf{X}_t
 \end{aligned} \tag{5.2}$$

This is the first model that we’ve seen that mixes stationary and non-stationary states as discussed in Section 4.3. The local trend model has two unit roots and the “cycle” (theoretically) has two stationary roots. Because most software doesn’t do the (rather complicated) specialized calculations required to handle the mixed roots correctly, the standard approach is typically to start with fully diffuse initial states (typically done with very large finite initial variances with zero means) and condition on the first few observations.¹ Because RATS *does* have the `PRESAMPLE=ERGODIC` option, you can do the correct full-information estimation. Note that the results will *not* be the same. If you feel the need to reproduce an approximate solution with diffuse priors, use the combination of `PRESAMPLE=DIFFUSE` and `CONDITION=number of conditioning observations`.

If you do `PRESAMPLE=ERGODIC`, a technical issue *can* arise with this type of model. The problem is that τ_t and c_t are distinguishable primarily because τ_t has a unit root and c_t presumably doesn’t. In practice, c_t can be fairly smooth, so its dominant root can be near one, making it hard to separate the two terms. The log likelihood has a discontinuity when c_t has a unit root as the model goes from two unit roots to three. If the data are such that taking a data point out of the log likelihood improves it,² then the estimation will not be able to move c_t into the stationary zone since that will drop it into the lower “two-root” region for the likelihood. Often, you can work around this by using `PRESAMPLE=ERGODIC` with `CONDITION=3` for a preliminary estimation of the model, which eliminates the discontinuity in the log likelihood by leaving out the third observation regardless. Once you have the model converged with the continuous likelihood (presumably to a stationary representation for c), you can

¹The conditioning needs to be on at least as many observations as states (here 4), though sometimes more than that are used.

²Whether that happens will depend upon the scale of the dependent variable—if log likelihood elements are largely negative (data scale is relatively large), then the log likelihood is higher when you reduce the number of effective observations, while if the log likelihood elements are largely positive (data scale is relatively small), the log likelihood goes down.

re-estimate without the `CONDITION` option—the root problem arises during the function evaluations when you’re still far from the optimum.

This has five free parameters, the three variances and the two AR coefficients. The **C** and **F** matrices are fixed and known. **A**, however, is fixed but depends upon the two φ coefficients. We can define that as a `FRML[RECT]`, but we’ll use an alternative which executes faster and will be useful in more complicated models—we’ll use the `START` option on **DLM** to “poke” the φ values into the proper slots in an otherwise fixed **A** matrix and then let that (now fixed) matrix be used in evaluating the log likelihood.

Example 5.2 estimates the Clark model using data on the log of US GDP from 1952:1 to 1995:3—this is based upon an example from Kim & Nelson (1999). The five parameters are declared as:

```
nonlin phi1 phi2 sigxi sigeps sigzeta
```

The component variances will be modeled in standard deviation form. The fixed elements of the **A** matrix are created with

```
dec rect a(4,4)
input a
  1 0 0 1
  0 . . 0
  0 1 0 0
  0 0 0 1
```

where we put missing values (the `.`) in the two slots that need to be filled in later. **F** and **C** are straightforward:

```
dec rect f(4,3)
input f
  1 0 0
  0 1 0
  0 0 0
  0 0 1
dec rect c(4,1)
input c
  1 1 0 0
```

Finally, we will also declare **SW** as a fixed matrix, which we will also set as part of the `START` option function:

```
dec symm sw(3,3)
```

The following is the **FUNCTION** that will finish setting up the **A** and **SW** matrices:

```
function DLMYSetup
compute a(2,2)=phi1,a(2,3)=phi2
compute sw=%diag(||sigxi^2,sigeps^2,sigzeta^2||)
end
```

The guess values are done in a relatively crude way, using a linear regression to get an estimate of the residual variance, and taking guesses for the three component variances (converted to standard deviations) from those. The AR(2) coefficients are just guessed at a relatively persistent process but still well away from a unit root. An alternative approach would be to do an HP filter, take the residuals as an estimate of the “cycle”, and do an AR(2) regression of that to get estimates of the φ and the variance of ε . See Example 5.3 which does that.

```
linreg lgdp
# lgdp{1 to 4}
*
compute sigxi=.5*sqrt(%seesq),sigeps=.5*sqrt(%seesq),$
    sigzeta=.01*sigeps
compute phi1=1.1,phi2=-.3
```

The model is then estimated with:

```
dlim(method=bfgs,start=DLMYSetup(),a=a,sw=sw,f=f,c=c,y=lgdp,$
presample=ergodic,type=smooth) 1952:1 * xstates
```

The `START` option is invoked at the beginning of each function evaluation (before `DLM` does anything else). Within a function evaluation, all the parameters stay fixed, so `A` and `SW` don’t change. That’s why `A` and `SW` aren’t declared as `FRML`’s—they depend upon free parameters but aren’t dependent upon time, and the `START` function takes care of the dependence upon parameters.

Two things to note if you compare our results with those in Kim and Nelson. First, they, in fact, used the method described above for handling the mixed roots—they conditioned on the first 20 data points (which is why the data set starts in 1947), not just 4. Second, their graphs are of the *filtered* trend and cycle estimates, while the smoothed estimates would be more typical and are what we do. The estimation results are shown in Table 5.3.

`Usable Observations` shows the number of actual values for y . `Rank of Observables` shows the number of observations actually used for computing the likelihood, which here will be two less because of the two unit roots.

The cycle coefficients show quite a persistent process (the coefficients sum to .95). Of the two parameters governing the trend, the one that often needs to get zeroed out (ξ) is the one whose variance is statistically significant, while ζ_t (the trend drift) has a non-zero variance but with a relatively high standard

Table 5.3: Estimates of Clark Model

DLM - Estimation by BFGS

Convergence in 17 Iterations. Final criterion was 0.0000000 <= 0.0000100

Quarterly Data From 1952:01 To 1995:03

Usable Observations

175

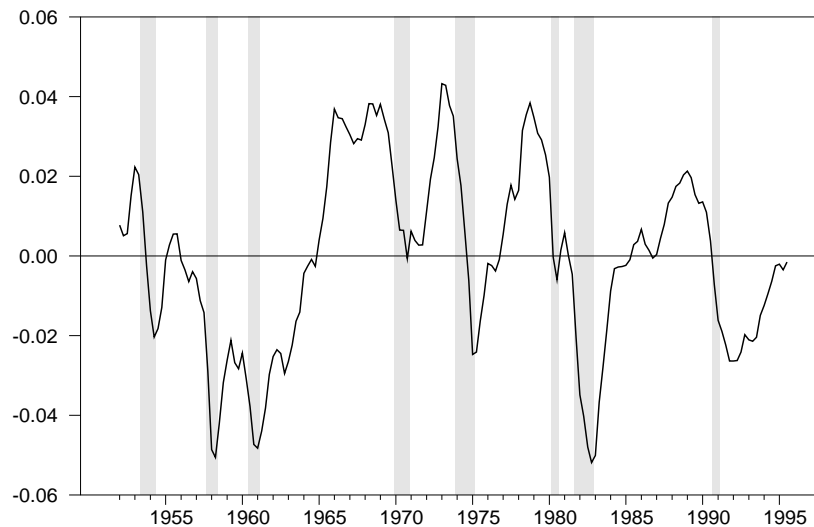
Rank of Observables

173

Log Likelihood

572.70259

	Variable	Coeff	Std Error	T-Stat	Signif
1.	PHI1	1.45563	0.14336	10.15356	0.00000
2.	PHI2	-0.50189	0.15843	-3.16798	0.00154
3.	SIGXI	0.00530	0.00183	2.88906	0.00386
4.	SIGEPS	0.00646	0.00181	3.56186	0.00037
5.	SIGZETA	0.00013	0.00017	0.76772	0.44266

**Figure 5.4:** Estimate of Cycle from Clark Model

error. The estimated cycle is shown in Figure 5.4.³ This is quite a bit smoother than the “cycle” (deviation from trend) that would be generated by the HP filter.

A “Clark” trend vs data is shown in Figure 5.5. It’s not immediately clear whether the trend estimate is “stiff” enough from looking at this because the deviation from the data is small relative to the overall range of the data. A better look at that can be seen in Figure 5.6, which graphs (over a more limited range of entries) the trend from this model vs the HP trend. As you can see there, the HP trend is much stiffer, while the Clark trend makes quite a few local “wobbles”—recall that the HP trend zeros out the variance of the short-term trend component (the ξ), while the Clark trend (with these data) effectively zeros out the variance of the *long-term* trend component (the ζ).

A similar, but more complicated, model is provided in Perron & Wada (2009). They fit several models to US GDP over the range 1947:1 to 1998:2—while sim-

³The shaded regions are the NBER recessions.

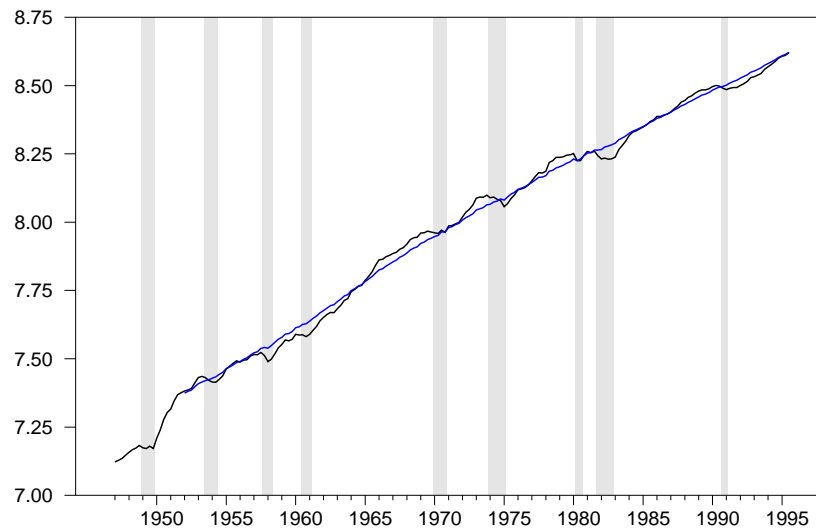


Figure 5.5: Estimate of Trend from Clark Model

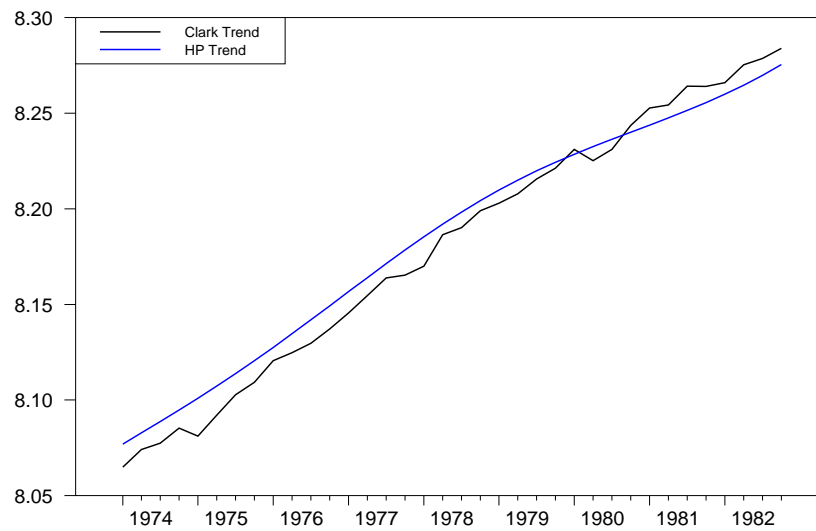


Figure 5.6: Comparison of Clark and HP Trends

ilar to the previous example, we'll use Perron and Wada's data for this. Their basic model is:

$$\begin{aligned}y_t &= x_t + c_t \\x_t &= x_{t-1} + \mu + \xi_t \\c_t &= \varphi_1 c_{t-1} + \varphi_2 c_{t-2} + \varepsilon_t\end{aligned}$$

The trend is a local trend with *fixed* growth rate rather than the variable rate allowed by Clark's model. This translates into the following state-space representation:

$$\begin{bmatrix} x_t \\ c_t \\ c_{t-1} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \varphi_1 & \varphi_2 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_{t-1} \\ c_{t-1} \\ c_{t-2} \end{bmatrix} + \begin{bmatrix} \mu \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \xi_t \\ \varepsilon_t \end{bmatrix}$$

$$y_t = \begin{bmatrix} 1 & 1 & 0 \end{bmatrix} \mathbf{X}_t$$

We can make the same use of `START` as in the previous example, but (since it's a smaller model) we'll show the alternative of setting up `A` and `SW` as `FRML`'s (the `Z` as well).

```
dec frml[rect] af
dec frml[vect] zf
dec frml[symm] swf
*
nonlin mu phi1 phi2 sigxi sigeps
```

The three component matrices that depend upon parameters are:

```
frml af = || 1.0, 0.0, 0.0|$
          0.0,phi1,phi2|$
          0.0, 1.0, 0.0||
frml zf = ||mu,0.0,0.0||
frml swf = %diag(||sigxi^2,sigeps^2||)
```

and the fixed components are:

```
compute [vect] c=||1.0,1.0,0.0||
compute [rect] f=%identity(2)~~%zeros(1,2)
```

(The `F` matrix can often be constructed using functions like this).

In Example 5.3, the following is used for getting guess values for the various parameters. This does an initial decomposition of the data into trend and cycle using an HP filter, estimating an AR(2) on the crude cycle to get guesses for the φ and variance for ξ :

Table 5.4: Perron-Wada with Fixed Rate Trend

DLM - Estimation by BFGS					
Convergence in 19 Iterations. Final criterion was 0.0000059 <= 0.0000100					
Quarterly Data From 1947:01 To 1998:02					
Usable Observations	206				
Rank of Observables	205				
Log Likelihood	-286.60535				
Variable	Coeff	Std Error	T-Stat	Signif	
1. MU	0.81191	0.05000	16.23758	0.00000	
2. PHI1	1.53030	0.10078	15.18532	0.00000	
3. PHI2	-0.60973	0.11293	-5.39899	0.00000	
4. SIGXI	0.68934	0.10362	6.65252	0.00000	
5. SIGEPS	0.61987	0.13176	4.70469	0.00000	

```

filter(type=hp) lgdp / gdp_hp
set gap_hp = lgdp - gdp_hp
linreg gap_hp
# gap_hp{1 2}
compute phi1=%beta(1),phi2=%beta(2),sigeps=sqrt(%seesq)

```

and a linear trend on the trend part to get guess values for μ , using a small scale of the variance of the errors for the variance for ε (since most of the error in this regression will be soaked up by the cycle component):

```

set trend = t
linreg gdp_hp
# constant trend
compute mu=%beta(2)
compute sigxi=sqrt(.1*%seesq)

```

The model has one unit root and two stationary roots. The authors handled this by:

1. computing the ergodic solution for the final two states (the cycle)
2. adding a large (1,1) element to allow for the unit root in state 1 and
3. dropping the first observation from the calculation of the likelihood.

This is more sophisticated than the typical handling of such models. However, again, it's better to just let **DLM** handle this with `PRESAMPLE=ERGODIC`:

```

dlm(presample=ergodic,a=af,z=zf,sw=swf,c=c,f=f,y=lgdp,$
method=bfgs,type=smooth) / xfixrate

```

The output from this is in Table 5.4. If we look at the smoothed trend estimate from this (isolated from the data so it will be clearer), the result is rather remarkable (Figure 5.7). Despite the fact that the model called for a fixed rate

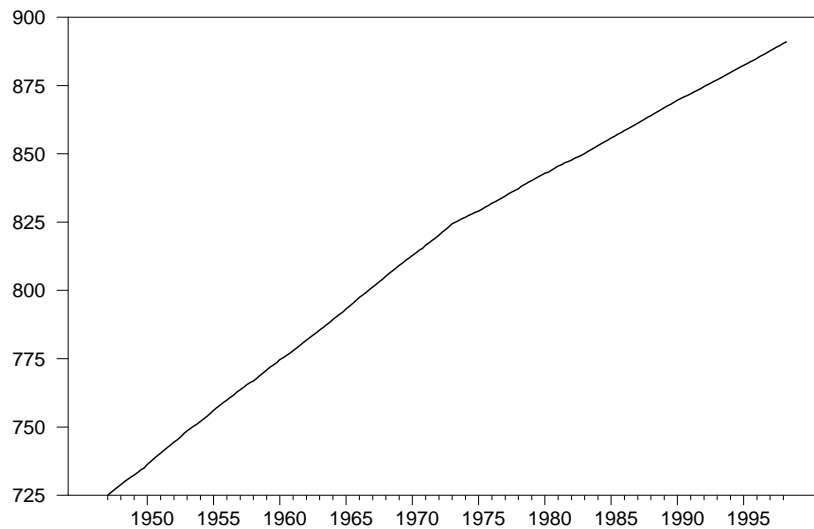


Figure 5.7: Trend from Basic Perron-Wada Model

trend over the entire data set, the smoothed results show a rather clear break in the rate in 1973:1. To make that happen, the ξ have to be mainly negative after 1973:1 and mainly positive before (which shouldn't happen if the original model were correct, and apparently it isn't).

They then move on to models which incorporate that break.⁴ That requires only the minor change of

```
set d1973 = t>1973:1
frml zf = ||mu+d*d1973,0.0,0.0||
```

and the addition of the trend shift coefficient D to the parameter set. The new output is in Table 5.5. Obviously, the D coefficient is extremely significant and the improvement in the log likelihood is substantial given that we only added the one parameter. Note that once the trend break is included, the ξ no longer is needed—in the earlier model, it was the only way to bend the trend rate down in the second half of the sample.

Perron and Wada, also, however, estimate the model allowing for a correlation between the two shocks. We have not seen this before, and you need to be *very* careful about doing this, as it often can fail to produce reasonable results. Their parameterization for this is to estimate the two standard errors and the correlation coefficient, thus the SW formula looks like:

```
frml swf = ||sigxi^2|rhone*abs(sigxi)*abs(sigeps),sigeps^2||
```

In most cases, we wouldn't recommend that way of handling this—there's nothing preventing this from estimating $|\rho| \geq 1$, which, in fact, happens in this case.

⁴The model for the trend here is *very* specific to US data and should not be expected to work in this form for other countries.

Table 5.5: Perron-Wada with Broken Trend

DLM - Estimation by BFGS

Convergence in 26 Iterations. Final criterion was 0.0000000 <= 0.0000100

Quarterly Data From 1947:01 To 1998:02

Usable Observations

206

Rank of Observables

205

Log Likelihood

-280.69682

	Variable	Coeff	Std Error	T-Stat	Signif
1.	MU	0.95140	0.02406	39.54020	0.00000
2.	D	-0.28761	0.04379	-6.56793	0.00000
3.	PHI1	1.27858	0.06539	19.55390	0.00000
4.	PHI2	-0.37255	0.06515	-5.71842	0.00000
5.	SIGXI	-0.00000	0.13919	-0.00001	0.99999
6.	SIGEPS	0.94465	0.04603	20.52136	0.00000

Table 5.6: Perron-Wada with Correlated Shocks

DLM - Estimation by BFGS with inequalities

Convergence in 24 Iterations. Final criterion was 0.0000029 <= 0.0000100

Quarterly Data From 1947:01 To 1998:02

Usable Observations	206
Rank of Observables	205
Log Likelihood	-280.5052

	Variable	Coeff	Std Error	T-Stat	Signif
1.	MU	0.9516	0.0259	36.8096	0.0000
2.	D	-0.2882	0.0455	-6.3360	0.0000
3.	PHI1	1.3283	0.1237	10.7351	0.0000
4.	PHI2	-0.4180	0.1152	-3.6294	0.0003
5.	SIGXI	0.1042	0.2007	0.5193	0.6035
6.	SIGEPS	0.8427	0.2024	4.1640	0.0000
7.	RHONE	1.0000	0.0000	0.0000	0.0000

It also starts to get rather ugly if you try to extend it beyond two shocks. With the constraint on the correlation added to the `PARMSET`,

```
nonlin mu d phil phi2 sigxi sigeps rhone rhone<=1.0
```

you get Table 5.6. If you compare the log likelihood with Table 5.5, the extra parameter hasn't given us any real improvement—all we've done is switch from the model being driven by a single shock (ε only since the variance of ξ was effectively zero), to a single shock that's part trend, part cycle. The cycle estimates (which are graphed in the program, but not shown here), with and without allowing for correlation, are almost identical.

To get a general positive semi-definite matrix, it's better to parameterize it as a packed lower triangular matrix.⁵ The following estimates the more general

⁵`SYMMETRIC` doesn't work because it doesn't force the matrix to be p.s.d.

Table 5.7: Perron-Wada with Correlated Shocks, Factor Representation

DLM - Estimation by BFGS

Convergence in 50 Iterations. Final criterion was 0.0000000 <= 0.0000100

Quarterly Data From 1947:01 To 1998:02

Usable Observations206

Rank of Observables205

Log Likelihood-280.50524

	Variable	Coeff	Std Error	T-Stat	Signif
1.	MU	0.95160	0.01391	68.42905	0.00000
2.	D	-0.28821	0.00870	-33.13242	0.00000
3.	PHI1	1.32824	0.01671	79.47858	0.00000
4.	PHI2	-0.41796	0.01243	-33.63004	0.00000
5.	SWLOWER(1,1)	-0.10418	0.00548	-19.01775	0.00000
6.	SWLOWER(2,1)	-0.84273	0.02685	-31.38815	0.00000
7.	SWLOWER(2,2)	0.00001	0.31200	0.00003	0.99998

model—the %LTOUTERXX function takes the outer product (LL') of a packed lower triangular matrix, parameterizing the matrix as its Cholesky factor. The guess values are the original guess values for the standard deviations on the diagonals, with zero on the off-diagonal.

```
dec packed swlower(2,2)
*
frml swf = %ltouterxx(swlower)
compute swlower(1,1)=sigxi0, swlower(2,1)=0.0, swlower(2,2)=sigeps0
nonlin mu d phil phi2 swlower
dlm(presample=ergodic, a=af, z=zf, sw=swf, c=c, f=f, y=lgdp, $
    method=bfgs, type=smooth)
```

If the two shocks collapse to a single one, the parameters will look like those in Table 5.7, with the (2,2) element coming in effectively zero.

5.3 Gap Models

A common type of state-space analysis is a model of a “gap” between observed and “natural” levels of a series. For instance, the following is a stylized decomposition of the unemployment rate (U):

$$U_t = N_t + G_t \quad (5.3)$$

This is often combined with a second equation (such as a Phillips curve), as in the EU’s NAWRU model (Non-Accelerating Wage Rate of Unemployment). We’ll look first at the unemployment model in isolation, and will add the second equation in Section 6.4. N is the (unobservable) NAWRU and G the unemployment gap. Coming up with a good UC model for the unemployment rate isn’t easy. Figure 5.8 shows the annual unemployment rate for the continental EU.

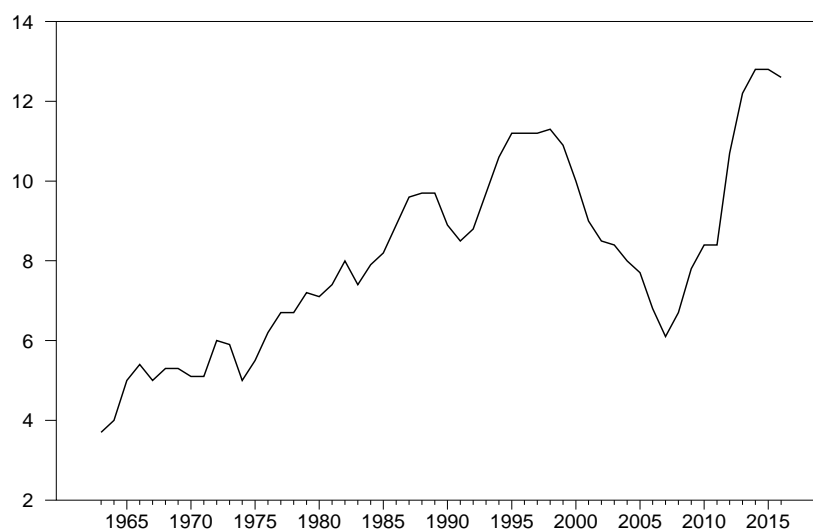


Figure 5.8: Unemployment rate

While it has a rather clear trend, the ratio of short-term movements to the trend is much, much higher than it is in the GDP data used in (for instance) the Clark model example (Figure 5.5). There is also a problem with the identification of the proper level: if N has a unit root (either one for a local level model or two for a local trend model), then you can give the G process any mean that you want without affecting the $N + G$ breakdown, since the level of a unit root process is arbitrary. The obvious choice from a statistical standpoint for the mean of G is zero, but is that reasonable from an *economic* standpoint? Are we really as likely to be below the NAWRU as above it?⁶

The standard model for G is a mean zero AR(2) model, *with roots forced to be complex*. This largely avoids the statistical problem described in Section 5.2, where the cycle could possibly add another unit root (or replace a unit root out of the trend model). This is most easily enforced by parameterizing the AR with a period (τ) and amplitude A .⁷ The AR representation will then be:

$$G_t = (2A \cos(2\pi/\tau)) G_{t-1} - A^2 G_{t-2} + a_{Gt}$$

What about a model for the NAWRU itself? Either a local level or a local trend model is a possibility. The former may have some problems dealing with the overall upward trend of the data, while the latter has the possibly unrealistic feature of a trend which is steadier than appears to be appropriate. An alternative is a “damped trend” model, which allows for a locally persistent trend rate, but where the trend rate moves back to a natural level (which could be

⁶This may be a good situation for some type of switching model to allow for different time-series behavior when above and when below NAWRU, but that’s outside the scope of this course.

⁷The process autocorrelations would be a damped sinusoid with period τ . However, note that it is possible to push this near a double-unit root with A near one and a very large value for τ , though that would probably only come up with a model for N that is far too restrictive.

zero, which we will treat as a separate model).

$$\begin{aligned} N_t &= N_{t-1} + \mu_{t-1} + a_{Pt} \\ (\mu_t - \mu_0) &= \rho(\mu_t - \mu_0) + a_{\mu t} \Rightarrow \mu_t = \rho\mu_t + (1 - \rho)\mu_0 + a_{\mu t} \end{aligned} \quad (5.4)$$

This gives us four different potential models with four different parameter sets. The program shown below takes advantage of some features added with RATS version 9.1 to make it simpler to switch out the chosen model. Note that *none* of these models is likely to work well if fully unconstrained. However, we will first (Example 5.4) use only minimal constraints to show how similarly-fitting models produce dramatically different estimates of the NAWRU.

What we will use is the `HASH` aggregator to organize both parameter sets (`PARMSET`'s) and set up functions for the state-space models. The setup functions will return the **A** matrix, and (in their arguments), the **Z**, **C**, **F** and **SW** matrices—each one of those will be used by at least one of the four NAWRU models. The `PARMSET` will list the free parameters and any standard restrictions (such as non-negativity).⁸

The free parameters across the four NAWRU models are:

```
dec real sigsqp sigsqmu
dec real rho mu0
```

Only one of the models (the damped trend) will use all four. The local level model uses just one. The code below declares the types for the two `HASH` variables. We'll use a descriptive phrase (such as "LocalLevel") to identify the element that we're defining. `DLMFUNC` is defined as a type for a function which creates system matrices for a (univariate) state-space model. The `HASH[DLMFUNC]` will be used to select the trend model, and the `GFUNC` will be used for the cycle model.

```
dec hash[parmset] nparms
newtype dlmfunc function[rect] (vect*, vect*, rect*, symm*)
dec hash[dlmfunc] nfunc
dec dlmfunc gfunc
```

The following defines the setup function, puts it into the function `HASH` and adds the `PARMSET` to that `HASH` for the simplest of the models: the local level. The function will be called as part of the `START` option (page 18), so it can use values of the free parameters (which it does here in the `SW` component).

⁸We're generally using variable names based on the EU's GAP program, which uses a general notation of $x_t = p_t + c_t$ for a trend + cycle decomposition.

```

function NLocalLevel Z C F SW
type rect NLocalLevel
type vect *Z *C
type rect *F
type symm *SW
*
compute NLocalLevel=||1.0||
compute Z=||0.0||
compute C=||1.0||
compute F=||1.0||
compute SW=||sigsgp||
end
compute nfunc("LocalLevel")=NLocalLevel
nonlin(parmset=nparms("LocalLevel")) sigsgp sigsgp>=0.0

```

The “damped drift” model is the damped trend where the mean of the trend is 0, which might be appropriate for a series (such as unemployment) which wouldn’t be *expected* to have a strong trend.

```

function NDampedDrift Z C F SW
type rect NDampedDrift
type vect *Z *C
type rect *F
type symm *SW
compute NDampedDrift=||1.0,1.0|0.0,rho||
compute Z=%zeros(2,1)
compute C=%unitv(2,1)
compute F=%identity(2)
compute SW=%diag(||sigsgp,sigsgmu||)
end
compute nfunc("DampedDrift")=NDampedDrift
nonlin(parmset=nparms("DampedDrift")) sigsgp sigsgmu rho $
    sigsgp>=0.0 sigsgmu>=0.0

```

The damped trend model with a non-zero (but unknown) target trend rate is similar, but needs a non-zero “Z” value for the trend rate equation (and needs the MU0 added to the parameter set).

```

function NDampedTrend Z C F SW
type rect NDampedTrend
type vect *Z *C
type rect *F
type symm *SW
compute NDampedTrend=||1.0,1.0|0.0,rho||
compute Z=||0.0,mu0*(1-rho)||
compute C=%unitv(2,1)
compute F=%identity(2)
compute SW=%diag(||sigsqp,sigsqmu||)
end
compute nfunc("DampedTrend")=NDampedTrend
nonlin(parmset=nparms("DampedTrend")) sigsqp sigsqmu rho mu0 $
    sigsqp>=0.0 sigsqmu>=0.0

```

Finally, the local trend model has the form that we've seen several times already:

```

function NLocalTrend Z C F SW
type rect NLocalTrend
type vect *Z *C
type rect *F
type symm *SW
compute NLocalTrend=||1.0,1.0|0.0,1.0||
compute Z=%zeros(2,1)
compute C=%unitv(2,1)
compute F=%identity(2)
compute SW=%diag(||sigsqp,sigsqmu||)
end
nonlin(parmset=nparms("LocalTrend")) sigsqp sigsqmu $
    sigsqp>=0.0 sigsqmu>=0.0
compute nfunc("LocalTrend")=NLocalTrend

```

We set up a similar function and PARMSET for the “gap” model. These don't go into the HASH since they are the same for each of the NAWRU models. Instead we set the GFUNC pointer to use this.

The parameters used are

```
dec real tauc ampc sigsqc
```

The function for setting up the state-space representation is

```

function GCycleCX Z C F SW
type rect GCycleCX
type vect *Z *C
type rect *F
type symm *SW
compute GCycleCX=||2*ampc*cos(2*pi/tauc),-ampc^2|1.0,0.0||
compute Z=%zeros(2,1)
compute C=%unitv(2,1)
compute F=%unitv(2,1)
compute SW=||sigsgc||
end
nonlin(parmset=cycleparms) sigsgc ampc tauc ampc>=0.0 ampc<=.99 $
    tauc>=0.0 tauc<=50.0 sigsgc>=0.0
*
compute gfunc=GCycleCX

```

This has some added (standard) constraints. The non-negativity on `TAUC` just picks the desired sign—since the cosine is an even function, the value is the same if the sign gets flipped. The limit of 50 is to prevent it from running off to infinity—there’s no practical difference once you’re above 50.⁹ The non-negativity constraint on `AMPC` rules out the (unlikely) rapid period-to-period oscillations associated with a negative value while the `.99` limit keeps it away from unit roots, as we want the NAWRU model to take care of those.

The following shows the function which actually gets called as part of the `START` option to “glue together” the NAWRU and gap models to form the system matrices for the complete model. The `MODEL` passed to this is the string identifying the `HASH` elements that represent the model of interest. `NFUNC(MODEL)` calls that and creates the five system matrices for the “N” model, while the `GFUNC` function pointer is called to return the gap matrices. These are combined as described on page 18. This also creates a variable called `CYCLESLOT` which shows the location in the overall state-space model for the cycle (gap) component—that can be either 2 or 3 depending upon the number of states in the NAWRU model:

⁹For numerical reasons, it would probably be better to parameterize this as the frequency rather than the period since an infinite period maps to a zero frequency.

```

declare rect adlm cdlm fdlm zdlm
declare symm swdlm
declare int cycleslot
*
* Glue together the trend and cycle models
*
```

```

function NAWRU
type string model
*
local rect adlm cdlm fdlm zdlm
local vect a f c sw y
local symm s
*
compute an=n
compute ac=g
*
compute adlm=a
compute fdlm=f
compute cdlm=c
compute zdlm=z
compute swdlm=s
*
* Figure out the cycle model
*
compute cycleslot
end
```

Given a chosen value for the model-identifying string (such as “LocalLevel”), the model could be estimated with

```

dlm(parmset=nparms(model)+cycleparms,$
start=NAWRUStart(model),presample=ergodic,condition=2,$
a=adlm,f=fdlm,z=zdlm,c=cdlm,sw=swdlm,y=lur,$
method=bfgs,itters=500,type=smooth) / xstates vstates
```

This sets the `PARMSET` equal to the “sum” of the `PARMSET` for the chosen `NAWRU` model and the `PARMSET` for the cycle model, and calls the `NAWRUStart` function just defined to create the system matrices (`ADLM`, `ZDLM`, etc.). This uses `PRESAMPLE=ERGODIC` since the models have mixed unit and stationary roots. It also uses `CONDITION=2` (page 52) so the log likelihoods from the four models will be comparable.¹⁰

Before we can estimate any of the models, we need guess values for the parameters. Different models have different parameters, but the following seems to come up with values which work reasonably well for all, at least getting the order of magnitude right:

¹⁰The local trend model has two unit roots while the other three only have one. `CONDITION=2` leaves two full observations out of the calculation of the log likelihood so all will use the same number of observations.

```

filter(type=hp) lur / lur_hp
set lur_gap = lur-lur_hp
*
set trate = lur_hp-lur_hp{1}
*
compute rho=.8
stats trate
compute mu0=%mean
compute sigsqmu=%variance*(1-rho^2)
compute sigsqp=sigsqmu*.25
*
linreg lur_gap
# lur_gap{1 2}
compute sigsqc=%sigmasq
compute tauc=6.0, ampc=.8

```

This uses an HP filter, and analyzes both the (time-varying) trend rate to get a mean (for the damped trend model) and variance (for all but the local level) for that. ρ (if it's needed) is set to a somewhat persistent value and the variance of the innovation in the trend rate (`SIGSQMU`) is backed out from the sample variance of the trend rate using that. The level variance (which is often zero in local trend models) is started at a fraction of the variance in the trend rate. `SIGSQC` is guessed at the residual variance from an AR(2) on the HP residual, and `TAUC` and `AMPC` are given reasonable values.

The following saves the guess values into the `VECTOR GUESS0` so we can reuse them without recomputing them.¹¹

```

nonlin(parmset=guesses) sigsqmu sigsqp mu0 rho sigsqc tauc ampc
compute guess0=%parmspeek(guesses)

```

We found that for some of the models, a standard (and fairly reasonable) set of guesses wasn't enough to achieve good behavior out of the estimates. The following procedure worked for all four models: first estimate the model with the "cycle" parameters fixed (at guess values which force there to be at least some cycle). This is exactly the same **DLM** instruction, but with the `PARMSET` option listing only `NPARMS (MODEL)`:

```

dlm(parmset=nparms(model), $
start=NAWRUStart(model), presample=ergodic, condition=2, $
a=adlm, f=fdlm, z=zdlm, c=cdlm, sw=swdlm, y=lur, $
method=bfgs, iters=500, type=smooth) / xstates vstates

```

That "refines" the values for the `NAWRU` parameters before estimating the full model.

¹¹The models are sufficiently different that it wouldn't be a good idea to simply assume that the estimated results from one can be fed as guesses into another.

After the model is estimated, the following pulls out the (smoothed) estimates of the NAWRU and the gap (which uses the value of `CYCLESLOT` to get the correct component), and graphs the data against the NAWRU estimate on one graph, and shows the gap on another:

```
set nawru = xstates(t)(1)
set gap   = xstates(t)(cycleslot)
*
spgraph(header="Decomposition using model "+model,vfields=2)
  graph(key=below,klabels=||"Data","NAWRU"||) 2
  # lur
  # nawru
  graph(key=below,klabels=||"Gap"||)
  # gap
spgraph(done)
```

The control loop to do all four models takes the following form:

```
dofor model = "LocalLevel" "DampedDrift" "DampedTrend" "LocalTrend"
  compute %parmspoke(guesses,guess0)
  ... estimate model
  ... do graphs
end dofor model
```

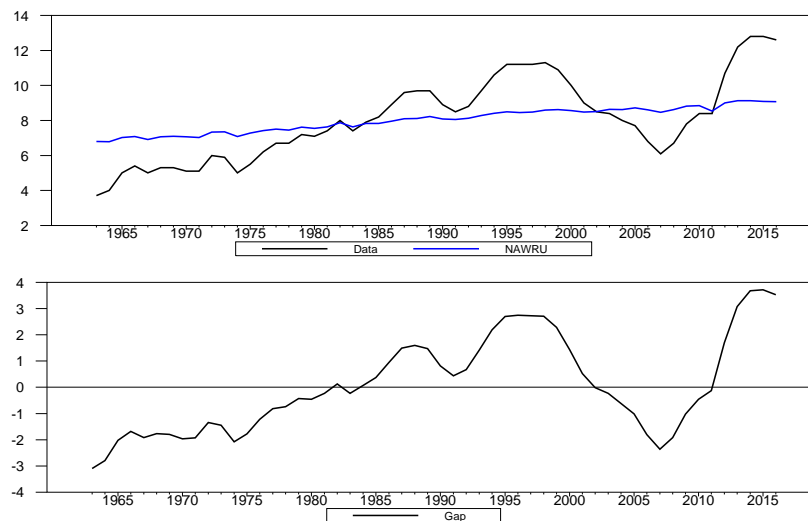
Of course, you don't have to estimate all four models, if you have one that you've already picked. Running through all four is being done here for illustration.

The maximum likelihood estimates for the local level model are shown in Table 5.8 and Figure 5.9. Needless to say, this is probably *not* what one would be looking for, as the smoothed estimate of the NAWRU is almost dead flat through the entire sample. In sharp contrast are the damped drift (Table 5.9 and Figure 5.10) and damped trend estimates (Table 5.10 and Figure 5.11), which both have so much flexibility that they allow the NAWRU to almost exactly match the observed data. Note, however, that despite the damped models giving almost the exact opposite behavior for the NAWRU estimates compared with the local level, the log likelihoods are quite similar.¹² In fact, they're close enough that the AIC criteria for the local level and the more heavily parameterized damped trend are almost identical if you only count six parameters for the latter (since `SIGSQP` ends up being zero). The maximum likelihood estimates for the local trend model end up with both the trend variances coming in zero, so the smoothed estimate is for a linear trend throughout the range (Table 5.11 and Figure 5.12).

¹²The log likelihoods for the four models are directly comparable because we used the `CONDITION` option. Without that, the Rank of Observables would be 53 rather than 52 for the first three models and the log likelihoods would be slightly smaller (since the log likelihood elements are generally negative) compared with the local trend.

Table 5.8: Unrestricted Local Level Model

DLM - Estimation by BFGS with inequalities					
Convergence in 35 Iterations. Final criterion was 0.0000006 <= 0.0000100					
Annual Data From 1965:01 To 2016:01					
Usable Observations		52			
Rank of Observables		52			
Log Likelihood		-46.6182			
	Variable	Coeff	Std Error	T-Stat	Signif
1.	SIGSQP	0.0801	0.1128	0.7094	0.4781
2.	SIGSQC	0.2406	0.1407	1.7102	0.0872
3.	AMPC	0.7897	0.0664	11.8981	0.0000
4.	TAUC	50.0000	0.0000	0.0000	0.0000

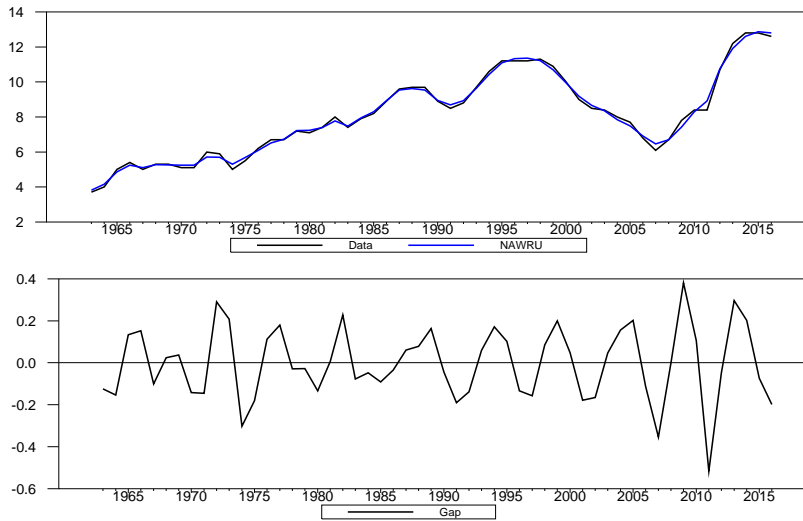
**Figure 5.9:** NAWRU with Local Level

It's not hard to believe that generally unrestricted estimates of these models might fail to work well since we've already seen what can happen when you do unconstrained maximum likelihood on UC models. A question then becomes whether it's possible to "fix" the model without going to a much more elaborate model type. The prior belief would be that the estimated NAWRU's are too stiff for the local level and local trend models and too variable for the damped trend models. In Example 5.5, we'll look at changes we can make to the damped drift model to give what would probably be seen as superior results.

What's wrong with Figure 5.10? The trend rate is clearly moving far too quickly and the gap isn't big enough. Either an upper bound on the movement of the trend rate or a lower bound on the variance of the gap (or both) would improve that. Another possibility is the Hodrick-Prescott approach of somehow fixing a ratio between the two, but both that, and even the simple limits on the variances are complicated by the fact that both the trend rate and the gap are modeled as autoregressions (an AR(1) for the trend rate and an AR(2) for the

Table 5.9: Unrestricted Damped Drift

DLM - Estimation by BFGS with inequalities					
Convergence in 19 Iterations. Final criterion was 0.0000025 <= 0.0000100					
Annual Data From 1965:01 To 2016:01					
Usable Observations	52				
Rank of Observables	52				
Log Likelihood	-45.0907				
	Variable	Coeff	Std Error	T-Stat	Signif
1.	SIGSQP	-0.0000	0.0000	-0.2619	0.7934
2.	SIGSQMU	0.2130	0.1061	2.0073	0.0447
3.	RHO	0.6108	0.1762	3.4664	0.0005
4.	SIGSQC	0.0397	0.0421	0.9441	0.3451
5.	AMPC	0.7833	0.1648	4.7524	0.0000
6.	TAUC	4.7935	0.7973	6.0119	0.0000

**Figure 5.10:** NAWRU with Damped Drift

gap). The variance on each component is thus a complicated function of both the innovation variance and the parameters governing the autoregression.¹³ The variance of the μ (trend rate) process is

$$\text{var}(\mu_t) = \sigma_\mu^2 / (1 - \rho^2) \quad (5.5)$$

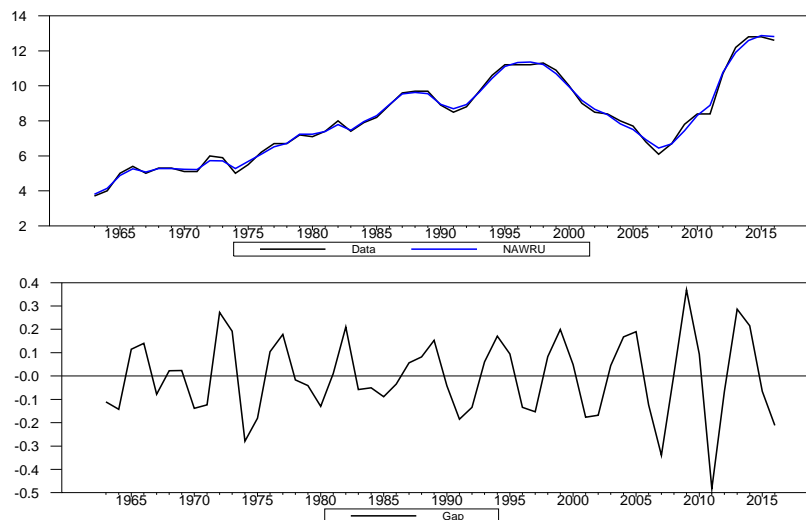
while the variance of the gap process is

$$\text{var}(G_t) = \frac{\sigma_c^2}{(1 - A^2) \left(1 + A^2 - \frac{4A^2 \cos^2(2\pi/\tau)}{1 + A^2} \right)} \quad (5.6)$$

¹³Note also that the variance in the trend rate equation in the local level model is the variance in the increment of a random walk, while it's the innovation variance in an AR(1) in the damped drift model, which have quite different effects on the process.

Table 5.10: Unrestricted Damped Trend Model

DLM - Estimation by BFGS with inequalities					
Convergence in 22 Iterations. Final criterion was 0.0000013 <= 0.0000100					
Annual Data From 1965:01 To 2016:01					
Usable Observations	52				
Rank of Observables	52				
Log Likelihood	-44.5484				
	Variable	Coeff	Std Error	T-Stat	Signif
1.	SIGSQP	-0.0000	0.0000	0.0000	0.0000
2.	SIGSQMU	0.2205	0.1195	1.8445	0.0651
3.	RHO	0.5561	0.1966	2.8278	0.0047
4.	MU0	0.1618	0.1520	1.0642	0.2872
5.	SIGSQC	0.0358	0.0459	0.7793	0.4358
6.	AMPC	0.8002	0.1860	4.3018	0.0000
7.	TAUC	4.8411	0.7837	6.1773	0.0000

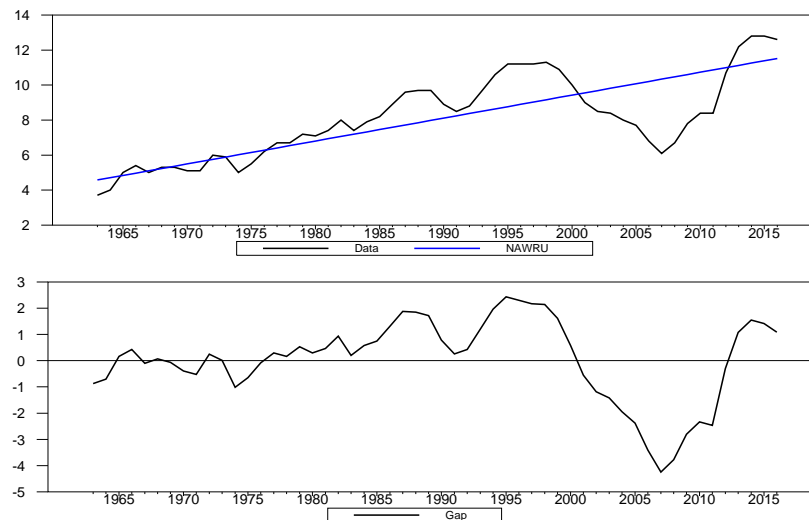
**Figure 5.11:** NAWRU with Damped Trend

(The general formula for the variance of an AR(2) process can be found in Hamilton (1994)—this substitutes in for the parameterization that we are using). The approach made in the GAP program is to combine upper limits on σ_μ^2 with lower limits on σ_c^2 . Because those have only an indirect effect on the process (because of where they fit in (5.5) and (5.6), it might require a fair bit of experimentation to get “reasonable” results. Since (5.5) is simpler, and since it’s probably safe to say that the overly variable trend rate is the main problem, it probably makes more sense to work on limiting $\text{var}(\mu_t)$, which is easy to do with inequality constraints in the parameter set.

Note that the basic constraints that we imposed in Example 5.4 are independent of the scale of the data. The only constraint that might be at all sensitive to the input data is the restriction on τ , which probably would need to be larger

Table 5.11: Unrestricted Local Trend

DLM - Estimation by BFGS with inequalities					
Convergence in 39 Iterations. Final criterion was 0.0000062 <= 0.0000100					
Annual Data From 1965:01 To 2016:01					
Usable Observations	52				
Rank of Observables	52				
Log Likelihood	-46.2845				
	Variable	Coeff	Std Error	T-Stat	Signif
1.	SIGSQP	-0.0000	0.0000	0.0000	0.0000
2.	SIGSQMU	-0.0000	0.0000	0.0000	0.0000
3.	SIGSQC	0.3011	0.0579	5.1992	0.0000
4.	AMPC	0.7028	0.0547	12.8519	0.0000
5.	TAUC	50.0000	0.0000	0.0000	0.0000

**Figure 5.12:** NAWRU with Local Trend

if this were monthly (where 50 is around 4 years) rather than annual (50 is 50 years). However, any *non-zero* constraints (upper or lower bounds) on any of the variances will need to adapt to the data and the situation. In this case, the “too large” variance for μ is .339. For comparison, we’ll add a restriction that the variance of μ can be no larger than .20. This is done in Example 5.5.

First off, this takes the overall setup for the trend models from Example 5.4 and puts them into a separate source file, so it can be used easily in other applications (like this one). It uses the same code for computing guess values. However, since we’re only looking at the damped drift model specifically, it doesn’t need the loop over the four model types. Instead, we do

Table 5.12: Damped Drift Model with Variance Restriction

DLM - Estimation by BFGS with inequalities

Convergence in 38 Iterations. Final criterion was 0.0000024 <= 0.0000100

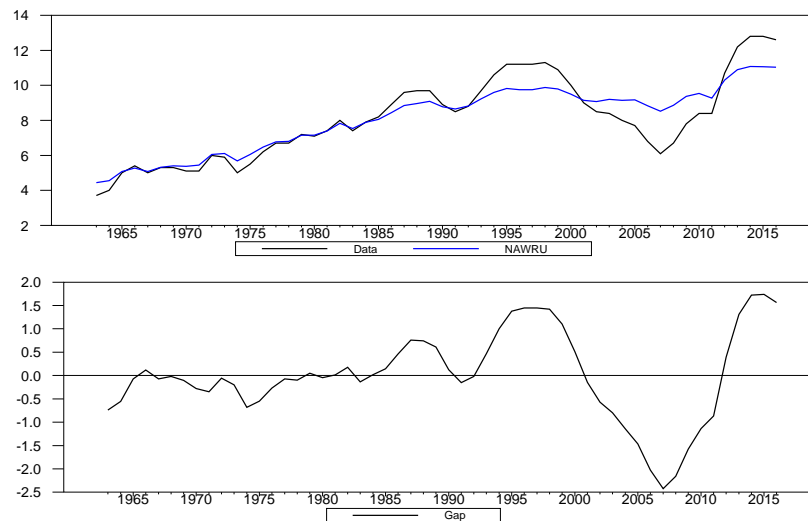
Annual Data From 1965:01 To 2016:01

Usable Observations 52

Rank of Observables 52

Log Likelihood -45.6114

	Variable	Coeff	Std Error	T-Stat	Signif
1.	SIGSQP	-0.0000	0.0000	-4.3556	0.0000
2.	SIGSQMU	0.1724	0.0605	2.8484	0.0044
3.	RHO	0.3717	0.4071	0.9131	0.3612
4.	SIGSQC	0.1466	0.1235	1.1870	0.2352
5.	AMPC	0.7903	0.2496	3.1656	0.0015
6.	TAUC	24.5532	30.4823	0.8055	0.4205

**Figure 5.13:** NAWRU with Damped Drift with Variance Restrictions

```

compute model="DampedDrift"
*
nonlin(parmset=limits) sigsqmu/(1-rho^2) <= .20
*
compute %parmspoke(guesses, guess0)

dln(parmset=nparms(model)+limits, $
  start=NAWRUStart(model), presample=ergodic, condition=2, $
  a=adlm, f=fdlm, z=zdlm, c=cdlm, sw=swdlm, y=lur, $
  method=bfgs, iters=500, type=smooth) / xstates vstates

```

```

*
dln(parmset=
start=NAWRU
a=adlm, f=f
method=bfgs

```

Note that the two **DLM** instructions are almost identical to the ones used in the

earlier example except that they add `+LIMITS` to the `PARMSET` options. The results are seen in Table 5.12 and Figure 5.13. You can verify that the variance restriction is binding. The log likelihood of -45.6114 is only slightly worse than the unrestricted value of -45.0907 (Table 5.9), and we would easily accept the (one) restriction based upon a likelihood ratio test.¹⁴ Note that despite the fact that the log likelihood is only slightly different, the NAWRU-gap decomposition is completely different from the one in Figure 5.10 and it also is quite a bit different from the overly stiff results (for the NAWRU) from the local level and local trend models. If you experiment a bit, you'll see that a variance restriction of .25 or more will give largely the same decomposition as the unrestricted damped drift model, and a variance restriction of .10 will be similar to the unrestricted local trend model. The range of roughly .15 to .20 is the "sweet spot" for what appears to be a "proper" decomposition of the data.

Note that it is *very* difficult to get a similar decomposition using direct restrictions on the more basic parameters like σ_μ^2 and σ_c^2 , even in combination, without effectively forcing a specific value for one or the other. The log likelihood is quite flat in σ_μ^2 for values of $\text{var}(\mu_t)$ in that range of .15 to .20 until ρ gets close to 1 (when it turns into a local trend model).

¹⁴This test would have standard asymptotics since it isn't a boundary restriction.

5.4 Time-Varying Coefficients in a Linear Model

A linear model with time-varying coefficients is another example of a state-space model—one that can be analyzed in several different ways within RATS. The most common form for this is

$$y_t = x_t \beta_t + v_t \quad (5.7)$$

$$\beta_t = \beta_{t-1} + w_t \quad (5.8)$$

where the coefficients β_t evolve according to a random walk. The β_t forms the state vector, (5.7) is the measurement equation (with the regressors x_t forming the time-varying C matrices), and (5.8) is the state equation.

This is general enough to handle quite a few possibilities. For instance, one limit case is where $w_t \equiv 0$ (which would be implemented with **DLM** by simply leaving out the **SW** option). That gives a “fixed coefficients” model. In a set of examples, we’ll look at a model from Kim & Nelson (1999). This is a linear “money demand” function with money growth as a function of several variables.

5.4.1 Fixed Coefficients

In Example 5.6, full-sample fixed-coefficients estimates (output shown in Table 5.13) are done with

```
linreg mlgr 1962:1 *
# constant dintlag inflag surplag mllag
```

Some of the coefficients make sense, some not so much. If interest rates have been going up (**DINTLAG**, which is the lagged change of the 3 month T-Bill rate, is positive), money growth is negative. If fiscal policy has been contractionary,¹⁵ money growth tends to be contractionary as well. Lagged inflation is the least significant of the factors, but what money growth there is tends to accomodate inflation. Finally money growth has some inertia, but a .28 autoregressive coefficient isn’t *that* high.

Sequential (Kalman filtered) estimates can be done simply using the **RLS** (Recursive Least Squares) instruction. This has quite a few options, with **COHIST** being used here to save a **VECT[SERIES]** with sequential coefficient estimates.

```
rls(cohist=co_rls) mlgr 1962:1 *
# constant dintlag inflag surplag mllag
```

The main use of **RLS** is to test for the appropriateness of a fixed coefficient (and fixed variance) model. While the coefficient estimates change from period to period as data are added to the sample, the changes should be somewhat

¹⁵**SURPLAG** is the lag of the “full employment surplus” which is the budget surplus/deficit adjusted to the equivalent level if the economy were at full employment. A positive number indicates relatively tight fiscal policy.

Table 5.13: Money Growth: Full Sample Estimates

Linear Regression - Estimation by Least Squares				
Dependent Variable M1GR				
Quarterly Data From 1962:01 To 1985:04				
Usable Observations	96			
Degrees of Freedom	91			
Centered R ²	0.48606			
R-Bar ²	0.46347			
Uncentered R ²	0.87196			
Mean of Dependent Variable	1.50641			
Std Error of Dependent Variable	0.87226			
Standard Error of Estimate	0.63892			
Sum of Squared Residuals	37.14741			
Regression F(4,91)	21.51583			
Significance Level of F	0.00000			
Log Likelihood	-90.64430			
Durbin-Watson Statistic	1.96529			
Variable	Coeff	Std Error	T-Stat	Signif
1. Constant	0.64152	0.17581	3.64899	0.00044
2. DINTLAG	-0.47362	0.06774	-6.99220	0.00000
3. INFLAG	0.13155	0.07668	1.71550	0.08966
4. SURPLAG	-0.64827	0.21202	-3.05757	0.00293
5. M1LAG	0.27590	0.08466	3.25878	0.00157

limited if the fixed coefficient model is appropriate. The changes will likely be quite large near the start of the data (and, in fact, `COHIST` doesn't even save coefficient estimates until there's enough data to get a full rank estimate), but they should settle down reasonably quickly.

A slightly more complicated setup which can be used for sequential estimates for both fixed and time-varying coefficients uses the **SYSTEM** definitions, **ESTIMATE** and **KALMAN** instructions. For the single equation model, this can be done with

```
equation(lastreg) mdeq
*
system mdeq
end(system)
*
estimate 1962:1 1962:1+4
do time=1962:1+5,1985:4
    kalman(cohistory=co_kalman)
end do time
```

You need to be quite explicit with the sample ranges here. This first estimates the equation using the first five data points (starting at 1962:1), which is the smallest sample size that yields a full rank estimate.¹⁶ The Kalman filter is

¹⁶**RLS**, by default, starts with only the minimal sample size, though that can be overridden

then run over the remainder of the sample.¹⁷ **KALMAN** also has a **COHISTORY** option for saving the sequential estimates. However, the great advantage of this (compared to **RLS** or the use of **DLM** that we will look at next) is that you can do additional computations, such as forecasts with “rolling” samples, by inserting them after the **KALMAN** instruction. For that use, you would typically use a *much* larger than minimal sample on the **ESTIMATE**.

Finally, we see how the fixed coefficients estimates can be done with **DLM**. There’s only one variance in the model (for the equation error), so there is no need to do non-linear estimation—one trip through the data with the Kalman filter will allow the variance to be calculated directly. Thus, we can use **METHOD=SOLVE** rather than **METHOD=BFGS**. The concentrated variance is handled, as before, with the combination of **SV=1.0** and **VARIANCE=CONCENTRATE**. The simplest way to handle the time-varying **C** matrices is to use **%EQNXVECTOR** (page 16) applied to the equation formed from the earlier **LINREG** (or **RLS**). Finally, the pre-sample for this is fully diffuse since all states follow a random walk, so the option used is **PRESAMPLE=DIFFUSE**.

```
dlim(y=mlgr, c=%eqnxvector(mdeq,t), sv=1.0, method=solve, $
      variance=concentrate, presample=diffuse) 1962:1 1985:4 $
      xstates vstates
```

We should note here why the estimates are all starting in 1962:1. In the book, Kim and Nelson start the filter in 1959:3 but ignore the first ten periods when computing the likelihood, which would mean starting the likelihood in 1962:1. However, starting the filter at 1959:3 and starting the calculation of the likelihood at 1962:1 is not really comparable to starting the sample in 1962:1 in the **LINREG**, since **DLM** is still Kalman filtering through the first ten data points while the **LINREG** would be ignoring those data. In fact, what *is* comparable, and what we do here, is starting the sample at the same point (here 1962:1) and using **PRESAMPLE=DIFFUSE**. Because there are five (=number of regressors) unit roots in the state vector, the first five sets of coefficients aren’t well-estimated (the **RLS** instruction above just skips those), so those aren’t included in the graphs. The dynamic estimates of the coefficients for the **CONSTANT** can be graphed using:

```
compute bstart=(1962:1)+5
*
set b0 bstart * = xstates(t)(1)
graph(footer="Kalman filtered regression coefficient b0")
# b0
```

with the **CONDITION** option. Because **ESTIMATE** is generally used for full-sample estimation of the model rather than for initializing a Kalman filter, its default is the full usable sample.

¹⁷Note that this uses the fact that you can “overindex” the quarters in a **yyyy:qq** expression, so 1962:1+5 is the same as 1963:2.

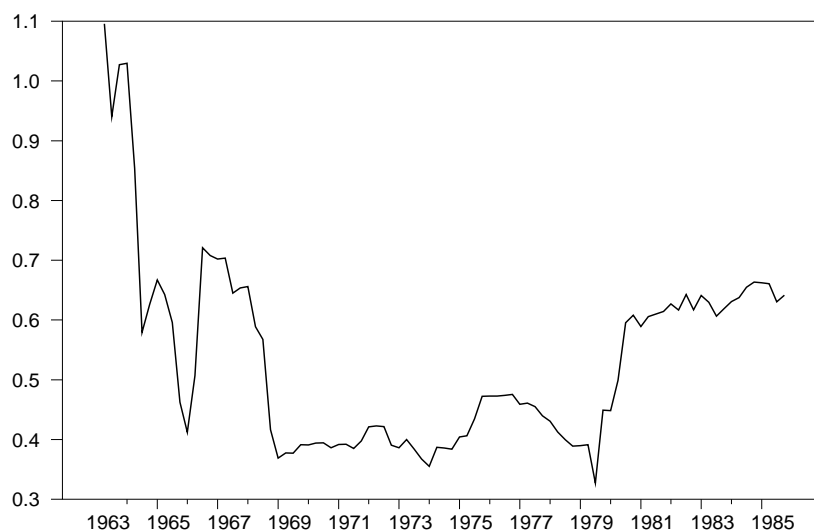


Figure 5.14: Kalman filtered estimates for intercept

producing Figure 5.14, which skips over the first five entries where the estimates have no precision. These will match exactly the `CO_RLS(1)` and `CO_KALMAN(1)` series generated using the other instructions, and the final value will match exactly the estimate from the **LINREG**. The one thing that *won't* match (as the instructions are written) is the log likelihood—**LINREG** and **RLS** give -90.6443 and **DLM** gives -98.2784. The difference is that **LINREG** computes the likelihood conditional on β , while **DLM** computes the unconditional likelihood—it marginalizes out the coefficients at each stage. If you add the option `FREE=5` to the **DLM**, the log likelihood will be computed to reverse out the lack of conditioning, and will match up with the **LINREG** calculation.¹⁸

5.4.2 Time-varying coefficients

As you can see from Figure 5.14, it's possible for a “fixed coefficients” model to produce quite a bit of variation in sequential estimates.¹⁹ However, Figure 5.14 shows a rather dramatic change in the 1979-1980 period when the Federal Reserve was changing policy towards fighting inflation—a pattern which also appears in the `B1` (`DINTLAG` coefficient) and `B4` (`M1LAG` coefficient) graphs.²⁰ A time-varying coefficients model allows the money growth equation to “evolve”

¹⁸This makes no practical difference here, but can in other situations where you are using the state-space formulation, where, for convenience, part (rather than all) of the state-vector are time-invariant coefficients. `FREE` indicates the number of freely estimated fixed coefficients. These need to be the final coefficients in the state vector, so you have to carefully arrange the regression.

¹⁹The Kalman *smoothed* estimates, on the other hand, will be dead flat at the end-of-sample values.

²⁰`B2`, which is the coefficient on the lagged inflation rate, also shows quite a bit of movement at the time, but because the scale of the graph is dominated by the early entries, it isn't as obvious as with the others.

to reflect changes in policy. An alternative would be a more complete structural break, dummied out a period seen as reflecting a change in regime. However, the latter would generally require a fairly substantial part of the sample to be *after* the regime change in order to get reasonable estimates, since its estimates would be computed independently of the earlier period. The TVC model uses that earlier information to generate a “prior” for the coefficients in the new regime, basically saying that the growth function may be different, but will have at least some major similarities.

To implement a TVC model, we need to give a non-zero matrix for the variance of w_t . There are many different ways to handle this, depending upon the situation. The one thing that is almost *never* done is to parameterize this with a single fully non-zero covariance matrix. First of all, the size of that matrix gets rather unwieldy, rather quickly, as the size of the model increases. With a five variable regression, it has 15 free elements; with ten regressors, it’s up to 55. If the variances are to be estimated, the typical method is to just estimate the diagonal, that is, to assume the increments are independent across coefficients. It’s possible to mix fixed and variable coefficients by zeroing out particular rows and columns in the SW matrix. Note that if any “slope” coefficient is varying, you probably want the intercept to vary as well, as the intercept has to pick up the effect of a changing mean in the explanatory function.

If you select (peg) values for (the diagonals of) SW , it’s very common to choose variances which are way too high to make sense. This is perhaps because of intuition which might go into the choice of the variance in the related “random coefficients” model

$$\beta_t = \beta_0 + w_t$$

There, the variance of w_t is indicating the overall spread in the random coefficients—the variance of $\beta_t - \beta_s$ is $2 \times \text{var}(w)$ regardless of choice of t and s . However, in the random walk model, the variance of $\beta_t - \beta_s$ is $|t - s| \text{var}(w)$, so accumulates linearly with the gap between entries, becoming potentially quite large in a long data set. If you’re trying to back out a value from some prior belief, you need to have some idea of what type of spread is reasonable *over a particular time period* and divide by the number of entries in that amount of time. For instance, with weekly data, if a range of .4 top to bottom would seem reasonable over the course of a year, then convert to .1 as the standard deviation (which gives .4 as ± 2 standard deviations), then $.01 = .1^2$ as the variance, then finally $.01/52 \approx .0002$ as the variance of the increment. If you skip the /52 step, you will get estimated coefficients which jump all over the place.

If, instead, you are estimating the drift variances as “hyperparameters”, you may face one of two (precisely opposite) “problems”:

1. A variance might estimate zero (or even negative if not constrained).
2. A variance might be unreasonably high.

We can look at this more carefully with the likelihood from a model with just a single regressor:

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

This gives us

$$\begin{aligned}y_t &= x_t\beta_{t-1} + x_tw_t + v_t \\ y_{t|t-1} &\sim N(x_t\beta_{t-1}, \sigma_w^2x_t^2 + \sigma_v^2)\end{aligned}$$

This first thing to note is that, as long as $\sigma_w^2x_t^2 + \sigma_v^2$ is positive, the conditional density is well-defined, even if one or the other of σ_w^2 or (less likely) σ_v^2 is negative. We've seen this before with UC models, where a component may not be necessary. Of course, if an unconstrained likelihood is maximized with $\sigma_w^2 < 0$, there is no way to interpret that, so we end up with the (properly constrained) optimization at $\sigma_w^2 = 0$. And there is nothing wrong with that—after all, if the true model *isn't* time-varying, that's the correct value. And even if a very modest amount of time-variation is appropriate, the nature of the likelihood is such that many samples would give a likelihood optimum with a negative value of σ_w^2 and thus (when constrained) a zero. This is the so-called “pile-up” effect—zero seems to come up more than one would expect. Note also that conventional tests can't be used to test for σ_w^2 since it's on the boundary rather than the interior.

While it's not unreasonable to get a zero variance, an estimated variance which is too high to be realistic is more problematic. The log likelihood has two components: the log variance term is hurt as the variance goes up, while the standardized squared residual improves. Now, the improvement in the latter tends to be focused on hard-to-fit points (if a one-step residual is zero, it doesn't matter what the variance is), while the cost in the former is spread across all points. From this, one thing which is clear is that if you apply a TVC model to a relatively short data set, you should expect that you are likely to get estimates which show time-variation whether it's there or not. Even in a somewhat larger data set, outliers near the end of the range will often be interpreted as time variation—this is because a sizable shift in coefficients towards the end to pick up the outlier won't affect predictions at as many data points as a similar shift earlier in the data. Of course, you could also have a high level of coefficient drift if the model is somehow misspecified, either having a sharp (rather than very gradual) change or simply missing some important explanatory variable. At any rate, don't blindly accept the results.

Example 5.7 uses **DLM** to estimate the model with time-varying parameters with the drift variances freely estimated. As described above, the coefficient drift terms are assumed to be independent, so we have six overall variances to estimate: the equation variance, and one drift variance for each of the five coefficients. These will all be estimated in standard deviation form.

We first estimate the model by least squares. This provides some information for guess values, and also allows us to set up the remainder of the program to adjust to changes in the explanatory variables:

```
linreg mlgr 1962:1 *
# constant dintlag inflag surplag mllag

equation(lastreg) mdeq
dec vect sigmav(%nreg)
dec real sigmae
```

The guess values are scales of the corresponding variances in the least squares regression. The drift variances are a *small* multiple of the regression standard errors.

```
compute sigmae=.5*sqrt(%seesq)
compute sigmav=.01*%stderrs
```

The **DLM** instruction is similar to the one used with the fixed coefficients, but needs options for **SW** and **SV** (both of which have their formulas written straight on the instruction), and **METHOD=BFGS**, since this has to be estimated.

```
nonlin sigmae sigmav
dln(y=mlgr, c=%eqnxvector(mdeq,t), sw=%diag(sigmav.^2), sv=sigmae^2, $
presample=diffuse, method=bfgs) 1962:1 1985:4 xstates vstates
```

The estimates of the standard deviations are in Table 5.14.²¹ The one that seems completely out-of-line is **SIGMAV(3)**, which is for lagged inflation. This is several times the standard error estimated in the **LINREG**, and remember, this is *per quarter*. Not surprisingly, the graph of the coefficient estimates (Figure 5.15) shows wild variation, particularly in the period after 1979.

It's almost impossible to come up with any policy description out of that. If we were really interested in pursuing this further, it would probably make sense to come up with a different proxy for inflation than the quarterly change in the CPI. By contrast, the other “slope” coefficients either stay relatively constant through the 1980's, or have a fairly rapid change over a two-year period and then hold steady.

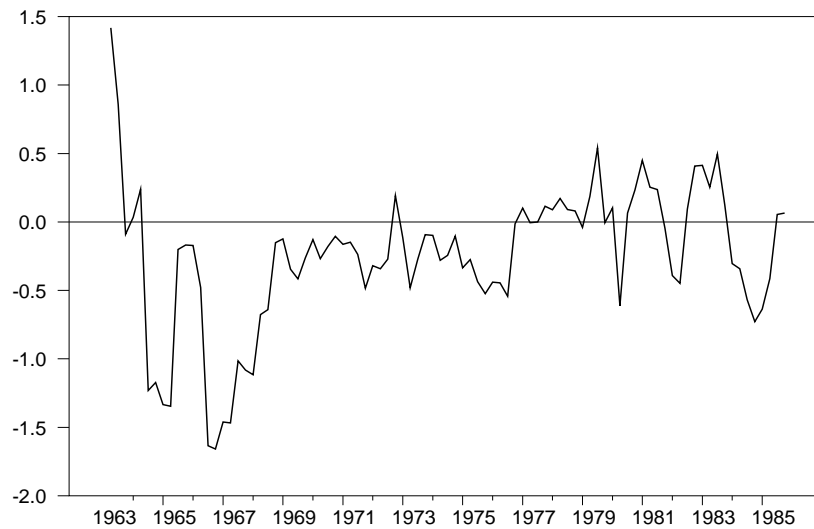
Note that, to this point, we have looked at the “filtered” estimates of the coefficients. You can do smoothed estimates by adding **TYPE=SMOOTHED** to the **DLM** instruction.

²¹Again, note that the log likelihood isn't comparable to that from the **LINREG** because it's unconditional.

Table 5.14: Money Growth: Estimation of TVC Standard Deviations

DLM - Estimation by BFGS
 Convergence in 50 Iterations. Final criterion was 0.0000007 \leq 0.0000100
 Quarterly Data From 1962:01 To 1985:04
 Usable Observations 96
 Rank of Observables 91
 Log Likelihood -92.33153

	Variable	Coeff	Std Error	T-Stat	Signif
1.	SIGMAE	0.38969	0.05054	7.71124	0.00000
2.	SIGMAV(1)	0.05588	0.08346	0.66961	0.50311
3.	SIGMAV(2)	0.03360	0.05219	0.64373	0.51975
4.	SIGMAV(3)	0.26293	0.04771	5.51103	0.00000
5.	SIGMAV(4)	0.00390	0.00107	3.66164	0.00025
6.	SIGMAV(5)	-0.02964	0.02288	-1.29533	0.19521

**Figure 5.15:** Time-varying estimate of inflation coefficient

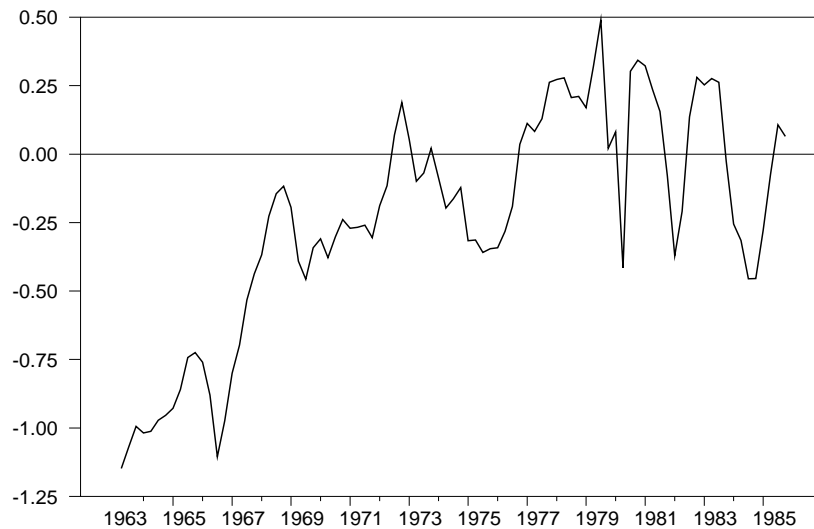


Figure 5.16: Smoothed Time-varying estimate of inflation coefficient

```
dln(y=mlgr, c=%eqnxvector(mdeq,t), sw=%diag(sigmax.^2), sv=sigmae^2, $
    presample=diffuse, method=solve, type=smooth) 1962:1 1985:4 $
    xstates vstates
set b2 bstart * = xstates(t) (3)
graph(footer="Smoothed Estimates of Inflation Coefficient")
# b2
```

Whether filtered or smoothed estimates (Figure 5.16 for the inflation coefficient) are most appropriate will depend upon the situation. Not surprisingly, the two are quite different near the start of the sample where the filtered estimates are based upon very little data. The most telling difference later in the sample is that the smoothed estimates turn positive in 1977, several years before both “spike” in 1979. If we are trying to make some inference about policy from these (which likely isn’t a good idea), the smoothed estimates are probably *not* what we want, since they would always seem to anticipate future actions.

We will look at this model again using Gibbs sampling methods in Section 8.2.2.

5.4.3 TVC with empirical drift variance

An alternative approach to time-varying coefficients which usually avoids the problem with individual coefficients getting too variable is shown in Example 5.8. This uses as variance for the drift a scale of the least squares estimate of the covariance matrix of the coefficients, thus reducing the number of free parameters to just two: the equation variance and the scaling value. In this example, we use the full-sample estimate of the linear regression, but it’s also common to use an early (“training”) subsample. This method of handling time-variation is particularly common in the VAR literature, where freely estimated

Table 5.15: Money Growth: Estimation with Scaled Covariance Matrix

DLM - Estimation by BFGS					
Convergence in 31 Iterations. Final criterion was 0.0000000 <= 0.0000100					
Quarterly Data From 1962:01 To 1985:04					
Usable Observations		96			
Rank of Observables		91			
Log Likelihood		-96.28635			
<hr/>					
	Variable	Coeff	Std Error	T-Stat	Signif
1.	SIGMAE	0.57405	0.00418	137.24366	0.00000
2.	SWSCALE	0.46065	0.00243	189.76219	0.00000

variances on individual coefficients would be both infeasible (due to the size of the parameter space) and also almost certain to produce poor results due to “over-hyperparameterization”—allowing too much freedom in the specification.

Again, start with the linear regression. Save the covariance matrix. If you do ROBUSTERRORS or any other HAC covariance calculation, don’t multiply by %SEESQ.

```
linreg mlgr 1962:1 *
# constant dintlag inflag surplag mllag
*
equation(lastreg) mdeq
compute swbase=%seesq*%xx
```

Again, the guess value for the equation standard deviation comes from scaling down the least squares result. The guess on the scale factor is small, but non-zero.

```
dec real swscale
dec real sigmae
*
compute sigmae=.5*sqrt(%seesq)
compute swscale=.01
```

The DLM instruction is the same as before except for the change to the SW option. The estimates are in Table 5.15.

```
nonlin sigmae swscale
*
dlm(y=mlgr,c=%eqnxvector(mdeq,t),sw=swscale*swbase,sv=sigmae^2,$
presample=diffuse,method=bfgs) 1962:1 1985:4 xstates vstates
```

Note that SWSCALE is not particularly *small*—for all coefficients other than inflation, it produces drift variance which is higher than the previous model. By contrast, for inflation, it is *much* lower which results in a “calmer” coefficient (Figure 5.17).

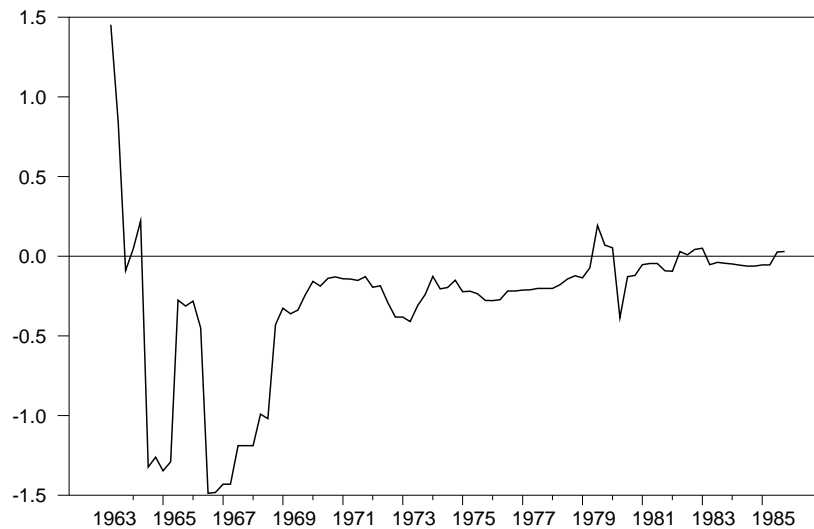


Figure 5.17: TVC estimate of inflation coefficient with scaled covariance matrix

Example 5.1 Airline Data

This fits several models to the Box-Jenkins airline data, as covered in Section 5.1.

```
cal(m) 1949
open data airline.prn
data(format=prn,org=columns,compact=sum) 1949:1 1960:12 pass
set lpass = 100.0*log(pass)
*
graph(footer="Box-Jenkins airline passenger data")
# lpass
*
* (a) Deterministic trend and seasonal
*
seasonal seasons
set trend = t
linreg lpass
# constant trend seasons{0 to -10}
set olsresids = %resids
*
* (b) "Airline" ARIMA model
*
boxjenk(diffs=1,sdifs=1,sma=1,ma=1,maxl) lpass
set arimaresids = %resids
*
* Components for BSM
*
@localdlm(type=trend,shocks=both,a=at,c=ct,f=ft)
@seasonaldlm(type=additive,a=as,c=cs,f=fs)
compute a=at~\as,c=ct~~cs,f=ft~\fs
*
* (c) BSM unconstrained.
*
```

```

nonlin sigsqeps sigsqxi sigsqzeta sigsqomega $
    sigsqxi>=0.0 sigsqzeta>=0.0
@localdlmunit(irreg=sigsqeps,trend=sigsqzeta,deseasonalize) lpass
compute sigsqxi=.01*sigsqeps,sigsqomega=0.0
*
dlm(y=lpass,a=a,c=c,f=f,presample=diffuse,$
    sv=sigsqeps,sw=%diag(||sigsqxi,sigsqzeta,sigsqomega||),$
    method=bfgs) / xstates
*
* (d) BSM constrained
*
nonlin sigsqeps sigsqxi sigsqzeta=0.0 sigsqomega
dlm(y=lpass,a=a,c=c,f=f,presample=diffuse,$
    sv=sigsqeps,sw=%diag(||sigsqxi,sigsqzeta,sigsqomega||),$
    method=bfgs,vhat=vhat,svhat=svhat) / xstates
*
* Do diagnostics
*
set resid = %scalar(vhat)/sqrt(%scalar(svhat))
@STAMPDiags resid
@CUSUMTests resid
*
dlm(y=lpass,a=a,c=c,f=f,presample=diffuse,$
    sv=sigsqeps,sw=%diag(||sigsqxi,sigsqzeta,sigsqomega||),$
    method=solve,type=smooth) / xstates vstates
*
* Extract the trend and the seasonal
*
set trend      = xstates(t)(1)
set seasonal   = xstates(t)(3)
set deseason   = lpass-seasonal
*
graph(footer="Local Trend with Data") 2
# trend
# lpass
graph(footer="Deseasonalized Data") 2
# deseason
# lpass
graph(footer="Seasonal")
# seasonal
*
set bsmresids = %scalar(vhat)
*
* Compare the (unscaled) residuals from the three models. These should
* be comparable. However, the ARIMA model is defined only from 1950:2 on
* (because of lags), and the residuals for the state space model are not
* reliable until the same point (while resolving the unit roots) so we
* only graph those.
*
graph(footer="Comparison of Residuals",key=below,$
    klables=||"State Space","ARIMA","Regression"||) 3
# bsmresids 1950:2 *
# arimaresids 1950:2 *
# olsresids 1950:2 *

```

Example 5.2 Clark Model

This analyzes data using the model from Clark (1987). This is the first model described in Section 5.2.

```

open data clark_bivariate.xls
calendar(q) 1947:1
data(format=xls,org=columns) 1947:01 1995:03 gdpq lhur
*
set lgdp = log(gdpq)
*
nonlin phi1 phi2 sigxi sigeps sigzeta
*
dec rect a(4,4)
input a
  1 0 0 1
  0 . . 0
  0 1 0 0
  0 0 0 1
dec rect f(4,3)
input f
  1 0 0
  0 1 0
  0 0 0
  0 0 1
dec rect c(4,1)
input c
  1 1 0 0
*
dec symm sw(3,3)
*
* Function for doing the final setup on each function evaluation.
*
function DLMYSetup
compute a(2,2)=phi1,a(2,3)=phi2
compute sw=%diag(||sigxi^2,sigeps^2,sigzeta^2||)
end
*
* Use a linear regression to get some ballpark estimates for the
* standard deviations.
*
linreg lgdp
# lgdp{1 to 4}
*
compute sigxi=.5*sqrt(%seesq),sigeps=.5*sqrt(%seesq),$
  sigzeta=.01*sigeps
compute phi1=1.1,phi2=-.3
*
dlm(method=bfgs,start=DLMYSetup(),a=a,sw=sw,f=f,c=c,y=lgdp,$
  presample=ergodic,type=smooth) 1952:1 * xstates
*
* Extract the trend and cycle components
*
```

```

set trend 1952:1 * = xstates(t) (1)
set cycle 1952:1 * = xstates(t) (2)
*
@nbercycles(down=recessions)
graph(shading=recessions,footer=$
  "Real GDP and its trend component\\Clark UC Model") 2
# lgdp
# trend
graph(shading=recessions,footer=$
  "Cyclical component of real GDP\\Clark UC Model")
# cycle
*
* Compare Clark model trend with HP trend
*
filter(type=hp) lgdp 1952:1 * hptrend
graph(footer="Comparison of trend estimates",$
  key=upleft,klables=||"Clark Trend","HP Trend"||) 2
# trend 1974:1 1982:4
# hptrend 1974:1 1982:4

```

Example 5.3 Trend plus Stationary Cycle Model

This fits several unobserved components models with trend plus stationary cycle from Perron & Wada (2009). The discussion of this starts on page 62.

```

open data lgdp.txt
calendar(q) 1947
data(format=free,org=columns) 1947:01 1998:02 lgdp
*
* Everything is run on 100*log data
*
set lgdp = 100.0*lgdp
set d1973 = t>1973:1
*
@NBERCycles(down=recession)
*
* UC-0 decomposition with AR(2) cycle, no break in trend rate.
* Uncorrelated component variances.
*
* Components which depend upon free parameters
*
dec frml[rect] af
dec frml[vect] zf
dec frml[symm] swf
*
nonlin mu phi1 phi2 sigxi sigeps
*
frml af = || 1.0, 0.0, 0.0|$
           0.0,phi1,phi2|$
           0.0, 1.0, 0.0||
frml zf = ||mu,0.0,0.0||
frml swf = %diag(||sigxi^2,sigeps^2||)

```

```

*
* Fixed components
*
compute [vect] c=||1.0,1.0,0.0||
compute [rect] f=%identity(2)~~%zeros(1,2)
*
* Get initial guess values. This is probably overkill. It extracts
* an HP trend, and runs an AR(2) on the "gap" to get an estimate of
* the two cycle coefficients and the standard error. It then
* estimates the trend rate by a linear regression of the HP trend
* on 1 and t. The standard error in that is likely to be fairly
* high relative to the standard error in the shock to the trend
* process (since the latter is a shock to a unit root process and
* so gets accumulated), so it gets scaled down.
*
filter(type=hp) lgdp / gdp_hp
set gap_hp = lgdp - gdp_hp
linreg gap_hp
# gap_hp{1 2}
compute phi1=%beta(1),phi2=%beta(2),sigeps=sqrt(%seesq)
*
set trend = t
linreg gdp_hp
# constant trend
compute mu=%beta(2)
compute sigxi=sqrt(.1*%seesq)
*
* Save these for later
*
compute sigxi0=sigxi,sigeps0=sigeps
*
dlim(presample=ergodic,a=af,z=zf,sw=swf,c=c,f=f,y=lgdp,$
      method=bfgs,type=smooth) / xfixrate
*
set fixtrend = xfixrate(t)(1)
graph(footer="Trend from Fixed Trend Rate")
# fixtrend
*
* Same with broken trend rate
*
nonlin mu d phi1 phi2 sigxi sigeps
*
frml zf = ||mu+d*d1973,0.0,0.0||
*
dlim(presample=ergodic,a=af,z=zf,sw=swf,c=c,f=f,y=lgdp,$
      method=bfgs,type=smooth) / xbreakrate
*
set breakcycle = xbreakrate(t)(2)
set breaktrend = xbreakrate(t)(1)
*
graph(footer="Comparison of Trends",key=upleft,$
      klabels=||"Fixed Rate Trend","Broken Rate Trend"||) 2
# fixtrend
# breaktrend

```

```

*
* Unrestricted UC model, allowing for correlation between the
* shocks. The maximum occurs at rho=1 (possible since we have only
* one observable, so having just a single effective shock is OK).
*
compute rhone=0.0,sigxi=.1
nonlin mu d phil phi2 sigxi sigeps rhone rhone<=1.0
*
frml swf = ||sigxi^2|rhone*abs(sigxi)*abs(sigeps),sigeps^2||
dlim(presample=ergodic,a=af,z=zf,sw=swf,c=c,f=f,y=lgdp,$
      method=bfgs,type=smooth) / xsur
*
set surtrend = xsur(t) (1)
set surcycle = xsur(t) (2)
*
graph(footer="Comparison of cycles",key=upleft,$
      klabels=||"With correlation","Without correlation"||) 2
# surcycle
# breakcycle
*
* Alternative parameterization for covariance matrix of shocks.
*
dec packed swlower(2,2)
*
frml swf = %ltouterxx(swlower)
*
* Guess values are the original sigxi and sigeps with 0 correlation.
*
compute swlower(1,1)=sigxi0,swlower(2,1)=0.0,swlower(2,2)=sigeps0
nonlin mu d phil phi2 swlower
dlim(presample=ergodic,a=af,z=zf,sw=swf,c=c,f=f,y=lgdp,$
      method=bfgs,type=smooth)

```

Example 5.4 Gap Model with Unrestricted Estimates

This demonstrates (generally) unrestricted estimation of the (univariate) state-space model for unemployment from the European Commission's GAP model. This is the first model described in Section 5.3.

```

open data "gap44_data.xls"
calendar(a) 1963:1
data(format=xls,org=columns) 1963:01 2016:01 year lur dws ddtot
*
* Parameters used in the NAWRU models
*
dec real sigsqp sigsqmu
dec real rho mu0
*
dec hash[parmset] nparms
newtype dlimfunc function[rect] (vect*,vect*,rect*,symm*)
dec hash[dlimfunc] nfunc
dec dlimfunc gfunc

```

```

*****
*
* Create functions and parameter sets for the different NAWRU models
*
function NLocalLevel Z C F SW
type rect NLocalLevel
type vect *Z *C
type rect *F
type symm *SW
*
compute NLocalLevel=||1.0||
compute Z=||0.0||
compute C=||1.0||
compute F=||1.0||
compute SW=||sigsqp||
end
compute nfunc("LocalLevel")=NLocalLevel
nonlin(parmset=nparms("LocalLevel")) sigsqp sigsqp>=0.0
****
function NDampedDrift Z C F SW
type rect NDampedDrift
type vect *Z *C
type rect *F
type symm *SW
compute NDampedDrift=||1.0,1.0|0.0,rho||
compute Z=%zeros(2,1)
compute C=%unitv(2,1)
compute F=%identity(2)
compute SW=%diag(||sigsqp,sigsqmu||)
end
compute nfunc("DampedDrift")=NDampedDrift
nonlin(parmset=nparms("DampedDrift")) sigsqp sigsqmu rho $
    sigsqp>=0.0 sigsqmu>=0.0
****
function NDampedTrend Z C F SW
type rect NDampedTrend
type vect *Z *C
type rect *F
type symm *SW
compute NDampedTrend=||1.0,1.0|0.0,rho||
compute Z=||0.0,mu0*(1-rho)||
compute C=%unitv(2,1)
compute F=%identity(2)
compute SW=%diag(||sigsqp,sigsqmu||)
end
compute nfunc("DampedTrend")=NDampedTrend
nonlin(parmset=nparms("DampedTrend")) sigsqp sigsqmu rho mu0 $
    sigsqp>=0.0 sigsqmu>=0.0
****
function NLocalTrend Z C F SW
type rect NLocalTrend
type vect *Z *C
type rect *F
type symm *SW

```



```

compute NLocalTrend=||1.0,1.0|0.0,1.0||
compute Z=%zeros(2,1)
compute C=%unitv(2,1)
compute F=%identity(2)
compute SW=%diag(||sigsqp,sigsqmu||)
end
nonlin(parmset=nparms("LocalTrend")) sigsqp sigsqmu $
  sigsqp>=0.0 sigsqmu>=0.0
compute nfunc("LocalTrend")=NLocalTrend
****
*****
*
* Parameters used in the "gap" model
*
dec real tauc ampc sigsqc
*
* Gap model with complex roots
*
function GCycleCX Z C F SW
type rect GCycleCX
type vect *Z *C
type rect *F
type symm *SW
compute GCycleCX=||2*ampc*cos(2*pi/tauc),-ampc^2|1.0,0.0||
compute Z=%zeros(2,1)
compute C=%unitv(2,1)
compute F=%unitv(2,1)
compute SW=||sigsqc||
end
nonlin(parmset=cycleparms) sigsqc ampc tauc ampc>=0.0 ampc<=.99 $
  tauc>=0.0 tauc<=50.0 sigsqc>=0.0
*
* Set the cycle function to GCycleCX
*
compute gfunc=GCycleCX
*****
declare rect adlm cdml fdml zdml
declare symm swdlm
declare int cycleslot
*
* Glue together the trend and cycle models
*
function NAWRUStart model
type string model
*
local rect an ac fn fc
local vect cn zn cc zc
local symm swn swc
*
compute an=nfunc(model)(zn,cn,fn,swn)
compute ac=gfunc(zc,cc,fc,swc)
*
compute adlm=an~\ac
compute fdml=fn~\fc

```

```

compute cdlm=cn~~cc
compute zdlm=zn~~zc
compute swdlm=swn~\swc
*
* Figure out where the cycle is
*
compute cycleslot=%rows(an)+1
end
*****
*
* Starting values. (We don't need all of these for most models, but
* common values should work OK anyway).
*
filter(type=hp) lur / lur_hp
set lur_gap = lur-lur_hp
*
set trate = lur_hp-lur_hp{1}
*
* Use an AR(1) with a .8 coefficient as the standard for generating
* guess values. (That should at least get the order of magnitude OK for
* all models).
*
compute rho=.8
stats trate
compute mu0=%mean
compute sigsqmu=%variance*(1-rho^2)
compute sigsqp=sigsqmu*.25
*
linreg lur_gap
# lur_gap{1 2}
compute sigsqc=%sigmasq
compute tauc=6.0,ampc=.8
*
* Save the guess values into a VECTOR associated with PARMSET GUESSES so
* we can easily recall them.
*
nonlin(parmset=guesSES) sigsqmu sigsqp mu0 rho sigsqc tauc ampc
compute guess0=%parmspeek(guesSES)
**
dec string model
*
do for model = "LocalLevel" "DampedDrift" "DampedTrend" "LocalTrend"
  *
  * Copy back in the guess values
  *
  compute %parmspoke(guesSES, guess0)
  *
  * Estimate the model with the cycle parameters left out
  *
  dlm(parmset=nparms(model), $
    start=NAWRUstart(model), presample=ergodic, condition=2, $
    a=adlm, f=fdlm, z=zdlm, c=cdlm, sw=swdlm, y=lur, $
    method=bfgs, iters=500, type=smooth) / xstates vstates
  *

```

```

* Estimate the full model
*
* If any of the variances went (slightly) negative, put them back on
* the boundary.
*
compute sigsqp=%max(sigsqp,0.0)
compute sigsqmu=%max(sigsqmu,0.0)
*
dlim(parmset=nparms(model)+cycleparms,$
      start=NAWRUStart(model),presample=ergodic,condition=2,$
      a=adlm,f=fdlm,z=zdlm,c=cdlm,sw=swdlm,y=lur,$
      method=bfgs,itters=500,type=smooth) / xstates vstates
*
set nawru = xstates(t)(1)
set gap    = xstates(t)(cycleslot)
*
spgraph(header="Decomposition using model "+model,vfields=2)
graph(key=below,klabels=||"Data","NAWRU"||) 2
# lur
# nawru
graph(key=below,klabels=||"Gap"||)
# gap
spgraph(done)
end dofor model

```

Example 5.5 Gap Model with Restrictions

Estimation of a NAWRU gap model with restrictions on variances. This is the second model in Section 5.3.

```

open data "gap44_data.xls"
calendar(a) 1963:1
data(format=xls,org=columns) 1963:01 2016:01 year lur dws ddtot
*
* Pull in the functions for the various forms
*
source nawrumodel.src
*
* Parameters used in the "gap" model
*
dec real tauc ampc sigsqc
*
* Gap model
*
function GCycleCX Z C F SW
type rect GCycleCX
type vect *Z *C
type rect *F
type symm *SW
compute GCycleCX=||2*ampc*cos(2*%pi/tauc),-ampc^2||1.0,0.0||
compute Z=%zeros(2,1)
compute C=%unitv(2,1)

```

```

compute F=%unitv(2,1)
compute SW=||sigsgc||
end
compute gfunc=GCycleCX
nonlin(parmset=cycleparms) sigsgc ampc tauc ampc>=0.0 ampc<=.99 $
    tauc>=0.0 tauc<=50.0 sigsgc>=0.0
*****
*
* Starting values. (We don't need all of these for most models, but
* common values should work OK anyway).
*
filter(type=hp) lur / lur_hp
set lur_gap = lur-lur_hp
*
set trate = lur_hp-lur_hp{1}
*
* Use an AR(1) with a .8 coefficient as the standard for generating
* guess values. (That should at least get the order of magnitude OK
* for all models).
*
compute rho=.8
stats trate
compute mu0=%mean
compute sigsgmu=%variance*(1-rho^2)
compute sigsgp=sigsgmu*.25
*
linreg lur_gap
# lur_gap{1 2}
compute sigsgc=%sigmasq
compute tauc=6.0, ampc=.8
*
* Save the guess values into a VECTOR associated with PARMSET
* GUESSES so we can easily recall them.
*
nonlin(parmset=guesses) sigsgmu sigsgp mu0 rho sigsgc tauc ampc
compute guess0=%parmspeek(guesses)
**
compute model="DampedDrift"
*
nonlin(parmset=limits) sigsgmu/(1-rho^2) <=.20
*
compute %parmspoke(guesses, guess0)
*
* Estimate the model with the cycle parameters left out
*
dlm(parmset=nparms(model)+limits, $
    start=NAWRUStart(model), presample=ergodic, condition=2, $
    a=adlm, f=fdlm, z=zdlm, c=cdlm, sw=swdlm, y=lur, $
    method=bfgs, iters=500, type=smooth) / xstates vstates
*
* Estimate the full model
*
dlm(parmset=nparms(model)+cycleparms+limits, $
    start=NAWRUStart(model), presample=ergodic, condition=2, $

```

```

a=adlm, f=fdlm, z=zdlm, c=cdlm, sw=swdlm, y=lur, $
method=bfgs, iters=500, type=smooth) / xstates vstates
*
set nawru = xstates(t)(1)
set gap = xstates(t)(cycleslot)
*
spgraph(header="Decomposition using model "+model+" restricted", vfields=2)
graph(key=below, klabels=| | "Data", "NAWRU" | |) 2
# lur
# nawru
graph(key=below, klabels=| | "Gap" | |)
# gap
spgraph(done)

```

Example 5.6 Linear regression with fixed coefficients

This shows several ways to compute sequential estimates of a linear regression with fixed coefficients. See Section 5.4.1 for discussion.

```

open data tvp.xls
cal(q) 1959:3
data(format=xls, org=columns) 1959:03 1985:04 mlgr dintlag inflag $
surplag mllag
*
* Least squares estimates of the money demand function. This is done
* using the sample from 1962:1 on.
*
linreg mlgr 1962:1 *
# constant dintlag inflag surplag mllag
*
* Recursive least squares estimates (equivalent to "time-varying"
* parameters with the time-variation shut down).
*
rls(cohist=co_rls) mlgr 1962:1 *
# constant dintlag inflag surplag mllag
*
equation(lastreg) mdeq
*
system mdeq
end(system)
*
estimate 1962:1 1962:1+4
do time=1962:1+5, 1985:4
kalman(cohistory=co_kalman)
end do time
*
* Since there is only one variance, we can use SV=1.0 and
* VARIANCE=CONCENTRATE with METHOD=SOLVE to handle it---there's no
* need for iterative estimation.
*
dlm(y=mlgr, c=%eqnxvector(mdeq, t), sv=1.0, method=solve, $
variance=concentrate, presample=diffuse) 1962:1 1985:4 $

```

```

      xstates vstates
compute bstart=(1962:1)+5
*
set b0 bstart * = xstates(t) (1)
graph(footer="Kalman filtered regression coefficient b0")
# b0
set b1 bstart * = xstates(t) (2)
graph(footer="Kalman filtered regression coefficient b1")
# b1
set b2 bstart * = xstates(t) (3)
graph(footer="Kalman filtered regression coefficient b2")
# b2 bstart *
set b3 bstart * = xstates(t) (4)
graph(footer="Kalman filtered regression coefficient b3")
# b3
set b4 bstart * = xstates(t) (5)
graph(footer="Kalman filtered regression coefficient b4")
# b4

```

Example 5.7 Linear regression with TVC, diagonal drift variance

This estimates a linear regression using time-varying coefficients, with independent drifts with estimated variances. This is the first example from Section 5.4.2.

```

open data tvp.xls
cal(q) 1959:3
data(format=xls,org=columns) 1959:03 1985:04 m1gr dintlag inflag $
      surplag mllag
*
* Least squares estimates of the money demand function. This is done
* using the sample from 1962:1 on.
*
linreg m1gr 1962:1 *
# constant dintlag inflag surplag mllag
*
equation(lastreg) mdeq
dec vect sigmav(%nreg)
dec real sigmae
*
* Get guess values based upon the results of a linear regression.
* All standard deviations are scales of the corresponding variances
* in the least squares regression. (The drift ones a much smaller
* multiple).
*
compute sigmae=.5*sqrt(%seesq)
compute sigmav=.01*%stderrs
*
nonlin sigmae sigmav
*

```

```

dlm(y=mlgr,c=%eqnxvector(mdeq,t),sw=%diag(sigmav.^2),sv=sigmae^2,$
    presample=diffuse,method=bfgs) 1962:1 1985:4 xstates vstates
compute bstart=(1962:1)+5
*
set b0 bstart * = xstates(t)(1)
graph(footer="Time-varying regression coefficient b0")
# b0
set b1 bstart * = xstates(t)(2)
graph(footer="Time-varying regression coefficient b1")
# b1
set b2 bstart * = xstates(t)(3)
graph(footer="Time-varying regression coefficient b2")
# b2
set b3 bstart * = xstates(t)(4)
graph(footer="Time-varying regression coefficient b3")
# b3
set b4 bstart * = xstates(t)(5)
graph(footer="Time-varying regression coefficient b4")
# b4
*
* Smoothed estimates
*
dlm(y=mlgr,c=%eqnxvector(mdeq,t),sw=%diag(sigmav.^2),sv=sigmae^2,$
    presample=diffuse,method=solve,type=smooth) 1962:1 1985:4 $
    xstates vstates
set b2 bstart * = xstates(t)(3)
graph(footer="Smoothed Estimates of Inflation Coefficient")
# b2

```

Example 5.8 Linear regression with TVC, empirical drift variance

This demonstrates a linear regression with time-varying coefficients using an empirically determined drift variance.

```

open data tvp.xls
cal(q) 1959:3
data(format=xls,org=columns) 1959:03 1985:04 mlgr dintlag inflag $
    surplag mllag
*
* Least squares estimates of the money demand function
*
linreg mlgr 1962:1 *
# constant dintlag inflag surplag mllag
*
equation(lastreg) mdeq
*
* This uses a scale of the full sample covariance matrix as the
* drift variance.
*
compute swbase=%seesq*%xx
*

```

```

dec real swscale
dec real sigmae
*
* Get guess value from the results of a linear regression. Start
* with a small scale on the covariance matrix.
*
compute sigmae=.5*sqrt(%seesq)
compute swscale=.01
*
nonlin sigmae swscale
*
dlim(y=mlgr, c=%eqnxvector(mdeq, t), sw=swscale*swbase, sv=sigmae^2, $
      presample=diffuse, method=bfgs) 1962:1 1985:4 xstates vstates
compute bstart=(1962:1)+5
*
set b0 bstart * = xstates(t) (1)
graph(footer="Time-varying regression coefficient b0")
# b0
set b1 bstart * = xstates(t) (2)
graph(footer="Time-varying regression coefficient b1")
# b1
set b2 bstart * = xstates(t) (3)
graph(footer="Time-varying regression coefficient b2")
# b2
set b3 bstart * = xstates(t) (4)
graph(footer="Time-varying regression coefficient b3")
# b3
set b4 bstart * = xstates(t) (5)
graph(footer="Time-varying regression coefficient b4")
# b4

```


Practical Examples with Multiple Observables

When we have multiple observables, we need to decide how closely the dynamic models for the variables will be linked. Will we have a “seemingly unrelated” model, where each has its own set of states, or will we have at least some common ones? Either way, these models tend to be *much* more complicated than models with single observables.

6.1 Indicator Models

In a series of papers, Stock and Watson have proposed state-space models for extracting business cycle indicators, applying them to larger and larger groups of data series. The simplest of these is from Stock & Watson (1991). The data are four “coincident” indicators: industrial production, employment, sales and personal income. It’s assumed that from these, it’s possible to extract a single index C_t (we’ll use their notation, then translate that into inputs for **DLM** later). With any model like this, there are four properties that must be determined:

1. What are the time series properties of the index?
2. How will the index load onto the observables?
3. What will be the time series properties of the model errors: the part of each series not explained by the index?
4. What will be the contemporaneous correlation properties of the model errors?

Partly to avoid some more technical issues that would arise from unit roots, the model is being done on growth rates, so the index is a common growth rate, which will be represented as ΔC_t .¹ This is assumed to follow a (stationary) AR(2) process (a common choice in these types of models as it allows for “cycles”). Each observable is assumed to have a linear loading on the index, giving us:

$$y_t = \beta + \gamma \Delta C_t + u_t \quad (6.1)$$

where y_t , β and γ are all 4-vectors, as is u_t . The u_t components are assumed to follow independent AR(2) processes which are also independent of ΔC_t . Thus

¹This will be accumulated up at the end to form the indicator index.

(6.1) isn't yet in the form of a measurement equation, since we need to create states for handling the serial correlation in the errors.

In all, this model will need ten states: two for the dynamics of ΔC_t and two for each of the four error processes. There will be five shocks. The state model is, under these assumptions, just the gluing together of five separate models, which means diagonal concatenation. Since the A matrix depends upon quite a few free parameters, the simplest way to handle this is to set up most of the matrix (which is mainly zeros and ones), then use a function to fix the elements which depend upon the free parameters. The C matrix (for **DLM**) will have the (transposed) form:

$$\begin{bmatrix} \gamma_1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \gamma_2 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ \gamma_3 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ \gamma_4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \quad (6.2)$$

The first column of this has the loadings from the index to the four observables, while the trailing eight columns have the loadings from the four AR(2) error processes to their variable. Again, most of this can be set once, but we'll need to use the same setup function to fill in the part of this that depends upon the parameters at the start of each function evaluation.

One final issue here is that, if the γ parameters are freely estimated, then there is a lack of identification of the scale of the ΔC_t process. We don't face this with a single observable, because the loading is naturally fixed at one. We could also fix one of the loadings at one, or otherwise normalize those, but it's simpler to normalize by making the variance of the shock to ΔC_t equal to one.

Example 6.1 is written to allow simple changes to the number of variables and the number of lags per variable. In

```
compute nvar=4
dec vect beta(nvar) gamma(nvar)
dec vect sigma(nvar)
dec vect[int] lags(nvar) apos(nvar)
```

BETA are the means, GAMMA are the loadings from the cycle, SIGMA are the variable-specific shock variances, LAGS has the number of lags, and APOS the position of the variable-specific component in the state vector (with two lags on everything, APOS will have 3, 5, 7 and 9).

This initializes the BETA's to the means and SIGMA's to half the variance of each of the indicators.

```

compute gamma=%const(1.0)
stats(noprint) ipgrow
compute lags(1)=2,beta(1)=%mean,sigma(1)=%variance*.5
stats(noprint) incgrow
compute lags(2)=2,beta(2)=%mean,sigma(2)=%variance*.5
stats(noprint) salesgrow
compute lags(3)=2,beta(3)=%mean,sigma(3)=%variance*.5
stats(noprint) empgrow
compute lags(4)=2,beta(4)=%mean,sigma(4)=%variance*.5

```

These will have the AR coefficients on the variable-specific components:

```

dec vect[vect] d(nvar)
ewise d(i)=%zeros(lags(i),1)

```

The following function creates the augmented state-space matrix for a general $AR(p)$:

$$\begin{bmatrix} \varphi_1 & \varphi_2 & \dots & \varphi_{p-1} & \varphi_p \\ 1 & 0 & \dots & 0 & 0 \\ 0 & \ddots & \ddots & \vdots & \vdots \\ \vdots & \ddots & 1 & 0 & 0 \\ 0 & \dots & 0 & 1 & 0 \end{bmatrix}$$

from the VECTOR PHI:

```

function %dlmar phi
type rect %dlmar
type vect phi
*
local int i j
*
dim %dlmar(%size(phi),%size(phi))
ewise %dlmar(i,j)=%if(i==1,phi(j),i==j+1)
end

```

The fixed parts of **A**, the (entirely fixed) **F** matrix and the fixed parts of **C** are created starting with:

```

dec rect a f c
compute a=%dlmar(%zeros(nc,1))
compute f=%unitv(nc,1)
dim c(0,0)

```

For **A** and **F**, these are for the cycle component. (For **C**, that will be added later). This adds in the variable-specific components:

```

do i=1,nvar
  compute apos(i)=%rows(a)+1
  compute a=a~\%dlmar(%zeros(lags(i),1))
  compute f=f~\%unitv(lags(i),1)
  compute c=c~\%unitv(lags(i),1)
end do i

```

The `APOS` vector keeps track of the position in the state vector of the component for the variable i error process. At each point, that will be one slot beyond the number that have already been added.

The `C` matrix (the transpose of (6.2)) is built up by a vertical concatenation of the loadings on the cycle to the loadings just constructed for the variable-specific components:

```
compute c=%zeros(nc,nvar)~~c
```

The first (now) row of this will have the loadings; the other `NC-1` rows just added will all be zeros as they are the lags of the cycle state.

The function to put the parameters in the correct spots is:

```

function %%DLMSSetup
compute %psubmat(a,1,1,tr(phi))
do i=1,nvar
  compute %psubmat(a,apos(i),apos(i),tr(d(i)))
end do i
compute %psubmat(c,1,1,tr(gamma))
compute sw=%diag(sigmac~~sigma)
end %%DLMSSetup

```

The `TR(...)` functions are needed because `VECTORS` are considered to be vertical and we need those to be filling in across the row. The placement of the variable-specific parameters is why we created the `APOS` vector earlier. `SIGMAC` is being normalized to 1.0 and won't be in the parameter set, but we'll write it that way for clarity.

The observables are the growth rates minus the β . In the paper, the β are fixed at the sample mean, so the interpretation of the model is that a positive value of the index would be associated with above average growth. The most flexible way to handle this is to use the growth rates (only) as the observables, and use the option `MU=BETA` on `DLM` to handle the means. We'll leave `BETA` out of the parameter set to achieve the same effect as using de-meanned data.

```

dec frml[vect] yf
frml yf = ||ipgrow,incgrow,salesgrow,empgrow||

```

Estimation is fairly straightforward. The `START` option (page 18) is called at each function evaluation, to finish creating the `A`, `C` and `SW` matrices. We take

Table 6.1: Estimates of Stock-Watson Indicator Model

DLM - Estimation by BFGS					
Convergence in 16 Iterations. Final criterion was 0.0000074 <= 0.0000100					
LOW ITERATION COUNT ON BFGS MAY LEAD TO POOR ESTIMATES FOR STANDARD ERRORS					
Monthly Data From 1959:02 To 1987:12					
Usable Observations		347			
Rank of Observables		1388			
Log Likelihood		-1166.4302			
	Variable	Coeff	Std Error	T-Stat	Signif
1.	GAMMA(1)	0.6267	0.0422	14.8548	0.0000
2.	GAMMA(2)	0.2259	0.0211	10.7184	0.0000
3.	GAMMA(3)	0.4557	0.0385	11.8238	0.0000
4.	GAMMA(4)	0.2279	0.0179	12.7477	0.0000
5.	D(1)(1)	-0.0214	0.1191	-0.1798	0.8573
6.	D(1)(2)	-0.0787	0.1069	-0.7367	0.4613
7.	D(2)(1)	-0.1123	0.0597	-1.8806	0.0600
8.	D(2)(2)	0.1542	0.0613	2.5177	0.0118
9.	D(3)(1)	-0.4158	0.0577	-7.2102	0.0000
10.	D(3)(2)	-0.2252	0.0564	-3.9957	0.0001
11.	D(4)(1)	-0.4570	0.0587	-7.7846	0.0000
12.	D(4)(2)	-0.1362	0.0583	-2.3354	0.0195
13.	SIGMA(1)	0.1811	0.0331	5.4645	0.0000
14.	SIGMA(2)	0.1379	0.0123	11.2184	0.0000
15.	SIGMA(3)	0.6755	0.0590	11.4490	0.0000
16.	SIGMA(4)	0.1223	0.0107	11.4726	0.0000
17.	PHI(1)	0.4873	0.0808	6.0328	0.0000
18.	PHI(2)	0.1541	0.0786	1.9595	0.0501

care of initial conditions for the filter with `PRESAMPLE=ERGODIC`. Note that there is no `SV` option, since the measurement errors have been pulled into the states. The output is in Table 6.1. Note that there are 347 observations and 1388 (= 4×347 observables). If there were unit roots, the number of observables would be reduced, but this is a fully stationary model.

```
dln(start=%DLMSetup(), y=yf, sw=sw, c=c, a=a, f=f, mu=beta, $
    presample=ergodic, type=filter, $
    pmethod=simplex, pitters=5, method=bfgs) 1959:2 * xstates
```

The indicator series is the first component of the state vector, which, as the model is estimated, is the difference of the intended indicator, so we extract that and accumulate it. As this is constructed, it somewhat arbitrarily fixes the pre-sample value of the indicator at zero. The graph of the filtered version is Figure 6.1.

```
set indicator 1959:2 * = xstates(t)(1)
acc indicator
graph(footer="Coincident Indicator, Filtered Version")
# indicator
```

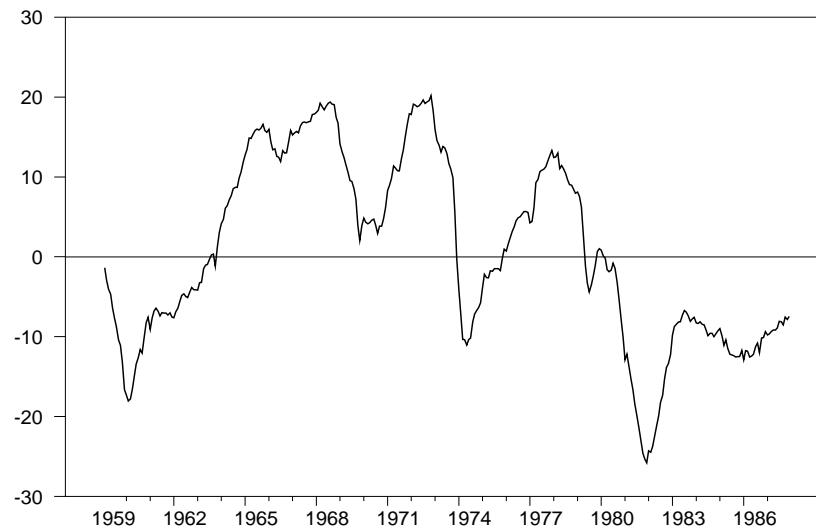


Figure 6.1: Coincident Indicator, Filtered Version

6.2 Multivariate H-P Filter

A similar type of state-space model is the generalization of the Hodrick-Prescott filter to more than one series. The complication here is that, by modeling the data, and not growth rates, we have to deal with issues of non-stationarity models. Another complication comes from trying to maintain the general behavior of the HP filter. This section is based (loosely) on Koziicki (1999), though, in the end, she estimated the common trend by averaging the input series rather than doing a joint likelihood analysis as we are.

The HP filter is based upon a local trend model, so we'll start with that, generating the trend x_t . For two variables, the stylized model will be something like:

$$y_{1t} = \alpha_1 + \gamma_1 x_t + noise_{1t}$$

$$y_{2t} = \alpha_2 + \gamma_2 x_t + noise_{2t}$$

Now the HP filter uses a fixed ratio of 1600 (for quarterly data) between the measurement error variance and the trend shock variance. In order to get that same type of "stiffness" in the trend, we'll have to do something similar here. Because we want that behavior, we can't just estimate the variances of the noises. Aside from the standard problem with these models that estimating the variances tends to produce a trend which is too variable to be of much use, estimating the variances would tend to make the (relatively) more variable series more important for identifying the trend, as a greater improvement to the likelihood would be made by trying to improve its fit.

By pegging the noise variances as a diagonal matrix, we can control which series are more and less important by adjusting the values. An identity matrix treats all the same. To favor one series over another, you would lower its relative variance.

This still isn't quite enough to control the overall "stiffness" of the trend since the variation in the single trend is being used across multiple series. A final normalization is that $\gamma' \Sigma_v^{-1} \gamma = 1$, basically a control on the length of the loadings, relative to the pegged values for the noise variances.

There's one other tricky aspect to this, which comes up because of the non-stationary x_t process. The model has 3 "intercepts": one in each equation plus the unknown level of the trend process. With only two observables, we again have a redundancy. The easiest way to deal with this is to constrain the intercepts in the measurement equations to sum to zero.

The state equation here is just a single standard local trend model—the action is all in the C matrix. In this case, we just define a `FRML[RECT]`:

```
dec frml[rect] cf
frml cf = tr(gamma)~~%zeros(1,n)
```

The `N` is used in the example program to allow more than two variables. For just two variables, you can constrain the intercepts to sum to one by making them take the form $\theta[1, -1]$, where θ is the (scalar) parameter to be estimated. The code below is a generalization of this to higher dimensions. The `ONEPERP` matrix spans the space of all vectors which are orthogonal to the vector of ones.

Note that the parameter set includes the constraint described above on the size of the γ . Because this is an implicit constraint, you have to use the `==` notation, which causes the optimizer to use constrained optimization with Lagrange multipliers. In all earlier examples, we've been able to use simple constraints like `SIGSQOMEGA=0.0`, which are done by direct substitution.

```
dec vect gamma(n)
dec vect theta(n-1)
*
nonlin gamma theta %qform(inv(sv),gamma)==1.0
compute gamma=%const(1.0),theta=%const(0.0)
*
compute oneperp=%perp(%ones(1,n))
```

The model is estimated with

```
dln(a=a,f=f,mu=oneperp*theta,sw=sw,sv=sv,c=cf,y=yf,type=smooth,$
presample=diffuse,var=concentrate,method=bfgs) / xstates
```

`VAR=CONCENTRATE` is used because we're using relative variances (`SV` is just the identity matrix and `SW` is `1.0/1600`). Note that this goes straight to smoothed state estimates, since that's what you typically want with HP filtering.

There's still a bit of work unwinding the information from the model. The HP trend itself (first state), in isolation, probably isn't very interesting, so we need

to pull out the loadings for a series and apply it to the state vector to get the trend that applies to each series. The loadings for the first series are in the first column of the `CF` matrix, so we extract that (using `%XCOL`) and dot it with the state vector. We also need to adjust this for the intercept, which is a function of the underlying θ parameters. This requires taking a row out of the `ONEPERP` array and dotting that with `THETA`:

```
set canf = %dot(%xcol(cf,1),xstates(t))+%dot(%xrow(oneperp,1),theta)
set usaf = %dot(%xcol(cf,2),xstates(t))+%dot(%xrow(oneperp,2),theta)
```

Those are the estimated trend series. The *cycles* can be obtained as the residual from the data:

```
set cancycle = canrgdp-canf
set usacycle = usargdp-usaf
```

We can do a comparison of the bivariate HP cycles with the univariate estimates using

```
filter(type=hp) canrgdp / canhp
set canhpcycle = canrgdp-canhp
graph(footer="Canadian Cycle Comparison",$
      key=loleft,klables=||"Bivariate","Univariate"||) 2
# cancycle
# canhpcycle
```

and similarly for the US. The Canadian graph is shown in Figure 6.2. Note that the levels (of either cycle) aren't really determined by the model itself—that was one of the issues raised above. When you Kalman smooth this type of model, the smoothed values adjust to zero out the mean of the gap.² If you go back to the original data, you'll find that Canadian GDP growth is similar to that in the US during (roughly) the first and last thirds of the sample, but higher during the middle period, which results in the behavior of the bivariate gap seen here. The US graph shows generally the opposite behavior.

6.3 Dynamic (Multiple) Factor Models

The previous examples can be interpreted as a form of factor model, as there is a single unobservable process which is used to explain common behavior of two or more series. Even with just a single factor, there were issues achieving identification: in Example 6.1, this was handled by pegging the variance of the

²The (univariate) HP filter minimizes over the sequence $\{x_t\}$ the objective function:

$$\sum_{t=1}^T (y_t - x_t)^2 + \lambda(x_t - 2x_{t-1} + x_{t-2})^2$$

If the mean of $y_t - x_t$ were *not* zero, you could increase every value of x_t by that mean, which would reduce the first term but have no effect on the second (since the coefficients sum to zero). The same type of thing happens in the bivariate case.

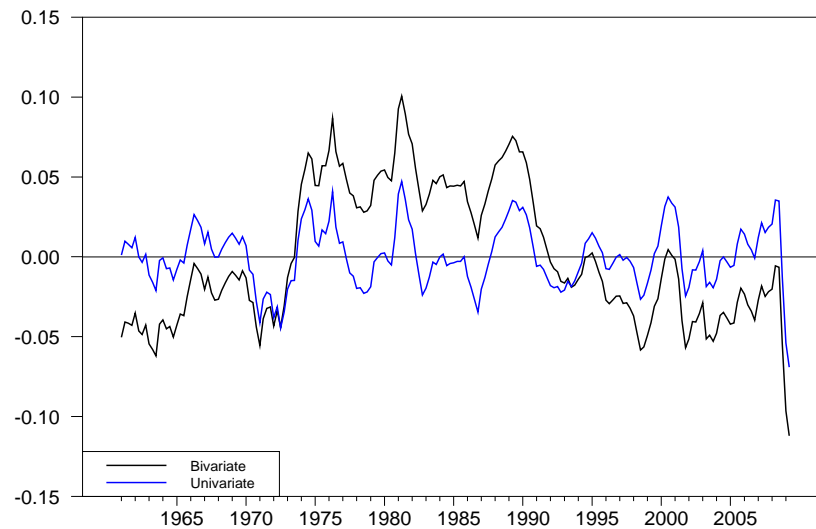


Figure 6.2: Canadian Cycle Comparison

shock in the cycle to 1, and in Example 6.2, there was both a scale peg *and* a level peg. If we want to extend this type of model to allow for multiple factors, these types of issues will get substantially more complicated. Suppose we take a very general measurement equation:

$$Y_t = \Lambda F_t + U_t \quad (6.3)$$

where Y is an m -vector of observables and F is a p vector of unobservable factors. At some point, we need to determine how we will be able to tell the factors apart. There are two obvious ways:

1. They can have different loading patterns in the Λ matrix. For instance, one factor might have zero loadings on a subset of variables.
2. They can have different dynamics. For instance, one could have a unit root and another could be stationary. Note that this can be hard to enforce—stationary processes can have a dominant root that approaches one and thus creates an identification problem with a forced unit root factor.

It's *possible* to estimate factors without so many identifying restrictions using “two-step” procedures which use principal components to isolate the factors. This makes no assumptions about the dynamics of the factors and instead works with just the observation equation (6.3). However, there are several problems with principal components:

1. They are dependent upon the relative scales of the data series, which is why they are often run on a correlation matrix.
2. On their own, they really aren't designed to handle dynamics.

3. Because principal components are associated with dominant eigenvalues of the cross product of data, you can be in a situation where minor changes in the cross product (due to bootstrapping or simulation) can create a situation where principal components “switch places”.

Example 6.3 and Example 6.4 apply two different techniques to a dynamic factor model applied to a well-known set of bond yields. The first of these is taken directly from Durbin & Koopman (2012), Section 8.6—this is simplest of the models in Diebold et al. (2006) (DRA for short). The model is

$$\begin{aligned} \mathbf{y}_t &= \Lambda \mathbf{F}_t + \varepsilon_t \\ \mathbf{F}_t &= \Phi \mathbf{F}_{t-1} + \eta_t \end{aligned}$$

where \mathbf{y}_t is a vector of bond yields, \mathbf{F}_t are (unobservable) factors and the loadings Λ are functions of the maturity of the bond and a small number of parameters to be estimated—in this case,

$$\Lambda_{i1} = 1, \Lambda_{i2} = \frac{1 - \exp(-\lambda\tau_i)}{\lambda\tau_i}, \Lambda_{i3} = \Lambda_{i2} - \exp(-\lambda\tau_i) \quad (6.4)$$

where τ_i is the maturity of bond i . In this model, the first factor will be the “level” component of the yield curve (shifts all yields up or down by a fixed amount), the second the “slope” and the third the “curvature”.

The data set has 17 sets of bond yields, monthly data from 1972:1 to 2000:12³

```
open data bonds.xls
calendar(m) 1972:1
data(format=xls,org=columns) 1972:01 2000:12 m3 m6 m9 m12 m15 $
m18 m21 m24 m30 m36 m48 m60 m72 m84 m96 m108 m120
```

The series are named “m” + the time to maturity in months. We’ll find it convenient in models like this to code the dependent variables into the explanatory variables of an EQUATION:

```
equation yvars *
# m3 to m120
```

An immediate advantage is that we can vary the set of variables easily and automatically adjust the program for its size using %EQNxxx functions:

```
compute n=%eqnsize(yvars)
```

Since we need the time to maturity to compute the lambda’s, we can strip those out of the names using:

```
dec vect tau(n)
ewise tau(i)=%value(%mid(%eqnreglabels(yvars)(i),2,0))
```

³Durbin and Koopman incorrectly show the data starting in 1985.

(%MID(string,p,0) returns the substring from position p to the end).

These dimension the various matrices needed for the state-space model (NF is the number of factors):

```
compute nf=3
*
dec real lam
dec rect a(nf,nf)
dec symm sw(nf,nf)
dec vect swdiag(nf)
dec rect lambda(nf,n)
dec vect svdiag(n)
dec symm sv
```

A is the Φ from the formulas, and is freely estimated. The ε_t and η_t are assumed to be independent of each other, with, in this case, the ε_t (with relatively small dimension) allowed to be correlated among themselves while the higher dimensioned η_t are uncorrelated internally.⁴ The free parameters are thus the elements of A, SW, the diagonal elements of SV (which will be modeled in log form) and the scalar LAM, which is the free value in the loadings.

Because Λ , which is the C matrix for DLM, depends upon the value of the parameter LAM, it has to be computed on each function evaluation. Since it doesn't depend upon the data (the τ_i is fixed for a given series), it can be evaluated once as part of the START option. The following function fills in the elements:

```
function FLoadSetup
type rect FLoadSetup
dim FLoadSetup(3,n)
do i=1,n
  compute t1=tau(i)*lam
  compute FLoadSetup(1,i)=1
  compute FLoadSetup(2,i)=(1-exp(-t1))/t1
  compute FLoadSetup(3,i)=FLoadSetup(2,i)-exp(-t1)
end do i
end FLoadSetup
```

This is the actual function used in the START option—it computes Λ and also converts the log parameters of the diagonals of SV into a usable matrix:

```
function DRASetup3
compute lambda=FLoadSetup()
compute sv=%diag(%exp(svdiag))
end
```

The following sets the guess values. The A is guessed as quite persistent, which certainly would be expected as bond yields themselves are persistent. It's not as

⁴DRA also consider slightly simpler models where the ε_t are also independent of each other.

clear how to guess values for \mathbf{SW} , since \mathbf{A} and \mathbf{SW} interact to give the variance of the factors themselves—this gives each factor a variance of roughly 5. ($.5/(1 - .95^2)$). \mathbf{SVDIAG} are the logs of the variances of the variable-specific errors, which are assumed to be relatively small (assuming that the yield curve will explain most of the variation). A guess for \mathbf{LAM} would be known from the literature on such models.

```
compute a = .95*%identity(nf)
compute sw = .50*%identity(nf)
compute svdiag = %fill(n, 1, -5.0)
compute lam = .1
```

This estimates the free parameters and computes Kalman smoothed state estimates. The `DRASetup3` function generates the \mathbf{LAMBDA} and \mathbf{SV} matrices, and the `%EQNXVECTOR` (page 16) function is used to pull out the observed data from the \mathbf{YVARS} equation:

```
nonlin a sw svdiag lam
dlm(startup=%(DRASetup3()), $
    a=a, sw=sw, sv=sv, c=lambda, y=%eqnxvector(yvars, t), $
    presample=ergodic, method=bfgs, iters=400, type=smooth) / factors
```

The following extracts the three factors and also a proxy for each factor based upon the data:

```
set betal = factors(t) (1)
set beta2 = factors(t) (2)
set beta3 = factors(t) (3)
*
set level = m120
set slope = m3-m120
set curve = -m120+2*m24-m3
```

and those are graphed creating Figure 6.3.

One thing to note is that the largest eigenvalue of the estimated Φ is just below 1. Some of the more complicated models in DRA add in observable macroeconomic factors, which push the eigenvalues even closer to (and perhaps all the way to) unity. That can cause numerical problems with a discontinuous likelihood. If you look at some of the other examples in the complete DRA replication, you'll note some added steps to avoid problems with mixed unit and stationary roots.

Example 6.4 by contrast, uses principal components to estimate the factors. While there are different ways to use the PC calculation, we'll use it directly on the measurement equation:

$$\mathbf{y}_t = \mathbf{\Lambda} \mathbf{F}_t + \varepsilon_t$$

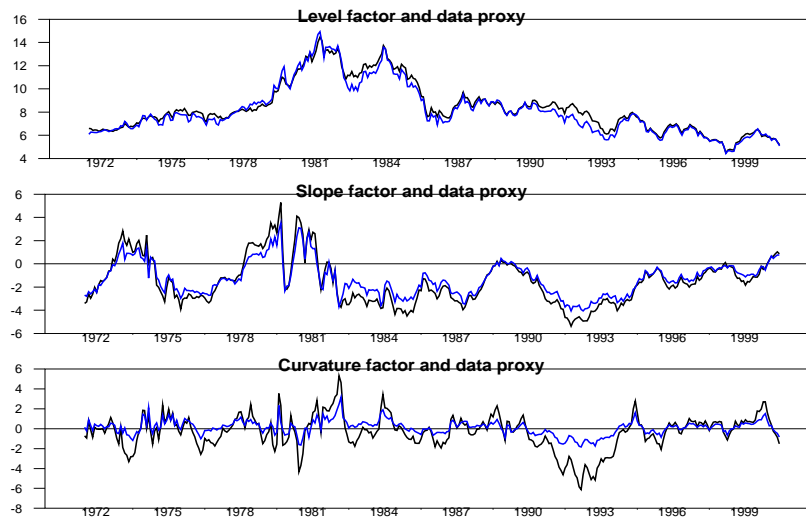


Figure 6.3: Smoothed Estimates of Yield Curve Factors

The \mathbf{F}_t are chosen to minimize across choices of Λ and \mathbf{F}_t

$$\sum_t \sum_i (y_{it} - \Lambda_i \mathbf{F}_t)^2 \quad (6.5)$$

subject to the normalization $\Lambda' \Lambda = \mathbf{I}$. Note that this (unlike the very specific loadings in Example 6.3) isn't enough to identify individual factors, since you can switch the order without violating the normalization restriction. In fact, given any Λ , and any orthogonal $p \times p$ matrix \mathbf{Q} (that is, rotation), $(\Lambda \mathbf{Q})(\mathbf{Q}' \mathbf{F}_t)$ produces an alternative set of factors which minimizes (6.5) subject to the normalization. The standard solution is to take as Λ the (transpose) of the eigenvectors corresponding to the p largest eigenvalues of the cross product matrix of the data $\mathbf{Y}'\mathbf{Y}$. The factors themselves are constructed as

$$\mathbf{F}_t = \Lambda' \mathbf{Y}_t$$

As we said earlier, common practice is to standardize the Y variables (each to mean zero, variance one, equivalently to use a correlation matrix). Because (6.5) treats all variables equally, it could be heavily influenced by scale—take a set of variables and multiply just one by (say) 1000, and the dominant factor will be almost exactly equal to the variable with the outsized scale. Note, however, that this isn't required if the variables naturally have a similar scale and level (as in this case), and, in fact, standardizing is sometimes exactly the *wrong* thing to do. For instance, in Bai (2004), the result for choosing the number of factors relies upon variances increasing with T at a predictable rate; however, standardizing makes them 1 regardless of the number of data points. As is the case with the bond yields, Bai's example is for a set of series (price indices) which are similar by construction.

In RATS, principal components can be estimated using either the `@PRINCOMP` or `@PRINFAC` procedures. `@PRINCOMP` is simpler and is the best choice when

you really just need the factors. (@PRINFACTORS allows you to more carefully study different rotations).

As above, we'll use 3 factors. We'll assume that in (6.3), the elements of U are independent AR(2) processes. The following creates the 3 principal factors (controlled by the NCOMPS option) for the explanatory variables in the YVARS equation. The 3 factors are copied into the VECT[SERIES] called FACTORS. There are separate CENTER and CORR options (which by default aren't used) for subtracting the mean (CENTER) or converting to correlations (CORR) if those are appropriate. Here, we aren't using them.

```
compute nc=3
compute nlags=2
*
@princomp(equation=yvars,ncomps=nc) / factors
```

The factors will now be treated as known in the state-space model, so what's unknown are the loadings, and the parameters governing the error processes:

```
dec vect[vect] phis(n)
dec vect[vect] gammas(n)
dec vect swvar(n)
dec symm sw(n,n)
```

The PHIS are the AR coefficients for each series, the GAMMAS are the loadings, and the SWVAR are the variable-specific variances.

This uses **SWEEP** to regress the dependent variables on the factors to get guess values for the loadings and also the residuals that will be used to initialize the other parameters.

```
sweep(series=resids,coeffs=loadings)
# %rlfromtable(%eqntable(yvars))
# factors
```

This copies out the loadings into the GAMMA VECTOR's, and runs autoregressions on the residuals to get guesses for the PHI's and the variances.

```
do i=1,n
  compute gammas(i)=%xcol(loadings,i)
  linreg(noprint) resids(i)
  # resids(i){1 to nlags}
  compute phis(i)=%beta
  ewise swvar(i)=%seesq
end do i
```

This sets up the fixed elements of the **A** matrix, and the **C** and **F** matrices (which are fully fixed). This is all based on the assumption that the number

of lags in the variable-specific AR processes are the same (at `NLAGS`) for all variables. With a bit of effort (with the use of a positioning `VECTOR[INT]` as in Example 6.1), this could be generalized to allow differing lag lengths.

```
dec rect a(nlags*n,nlags*n)
ewise a(i,j)=(i==j+1).and.%clock(i,nlags)<>1
*
dec rect f(nlags*n,n)
ewise f(i,j)=(i==(j-1)*nlags+1)
*
dec rect c(nlags*n,n)
ewise c(i,j)=(i==(j-1)*nlags+1)
```

This next function is used to “poke” the AR coefficients into the proper locations and convert the variances to a covariance matrix for the `SW`:

```
function SetupPCDLM
do i=1,n
  *
  * This is the first position of the states for the
  * error process for variable i
  *
  compute ipos=(i-1)*nlags+1
  compute %psubmat(a,ipos,ipos,tr(phis(i)))
end do i
compute sw=%diag(swvar)
end
```

and this is a (time-varying) function which computes the `MU` option using the factors (which are assumed to be known) and the loadings (which are being estimated):

```
function PCmuF t
type vect PCmuF
type integer t
*
dim PCmuF(n)
ewise PCmuF(i)=%dot(gammas(i),%xt(factors,t))
end PCmuF
```

and this does the estimation. Note that this takes a while as there are a large number of parameters (51 in the loadings alone):

```
nonlin phis gammas swvar
dlm(y=%eqnxvector(yvars,t),start=SetupPCDLM(),a=a,c=c,sw=sw,$
    f=f,mu=PCmuf(t),presample=ergodic,pmethod=simplex,piters=5,$
    method=bfgs) / xstates
```

The `START` option calls the `SetupPCDLM` function, which sets the **A** and **SW** matrices using the current parameters. The `MU` option depends both upon data and parameters, so it uses the `PCmuf` function, calling it with the current value of **T**.

This does a graph (Figure 6.4) of the factors (which we could have done earlier, since they don't depend upon the state-space model):

```
graph(footer="Factors",key=loleft,klabels=||"F1","F2","F3"||) 3
# factors(1)
# factors(2)
# factors(3)
```

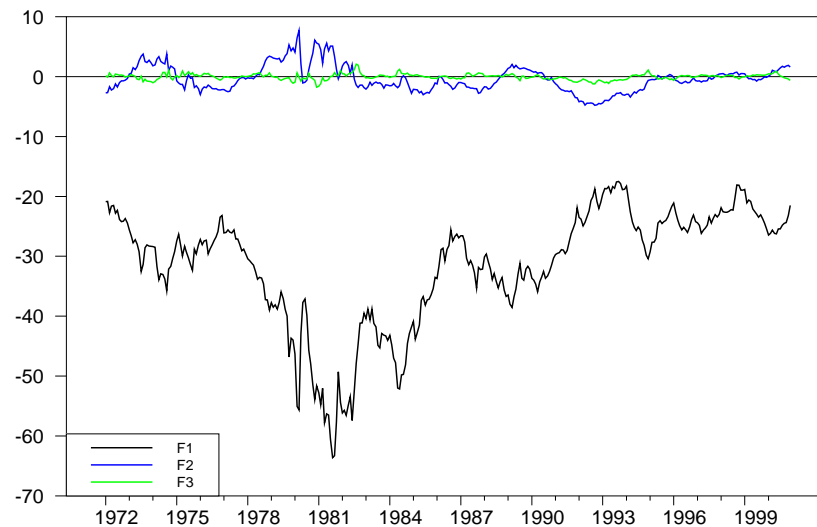


Figure 6.4: Factors from Principal Components

Note that, because the standard for principal components is to normalize the loadings (to the identity matrix inner product), the scales of the factors themselves show how much of the overall variation is explained by each. Also, note that the signs aren't determined, so here the first factor is coming in negative (and the corresponding loading will be negative as well). As a mechanical procedure, `@PRINCOMP` won't always know what signs you might expect. If you have a strong prior as to what the sign of either a factor or loading should be, you can check that early on and flip the sign of the factor right away.

This generates a series from the maturity in months, and extracts the loadings for the three factors, then does a scatter graph of the loadings vs maturity (Figure 6.5):


```

set maturity 1 n = %value(%mid(%eqnreglabels(yvars)(t),2,-1))
set load1 1 n = gammas(t)(1)
set load2 1 n = gammas(t)(2)
set load3 1 n = gammas(t)(3)

scatter(footer="Loadings vs Maturity",style=line,$
        key=below,klabels=||"F1","F2","F3"||) 3
# maturity load1
# maturity load2
# maturity load3

```

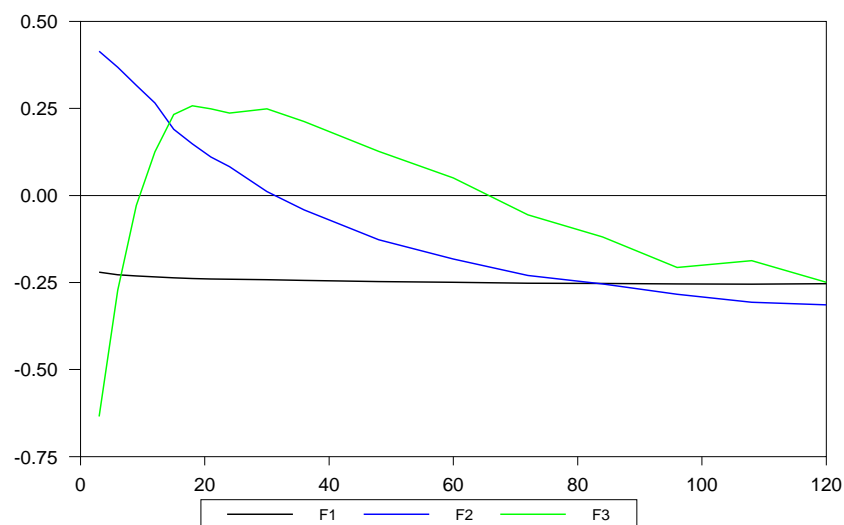


Figure 6.5: Loadings vs Maturity from Principal Components

Despite coming at this from a completely different direction, this comes out with (in effect) the level, slope and curvature factors.

One thing to note is because of the assumptions which leave the equations unconnected statistically (since the factors are treated as exogenous in the state-space model), the results can be obtained (theoretically exactly) by using a sequence of **BOXJENK** AR(2) models with the **FACTORS** as extra regressors. This does that for the **M3** series:

```

boxjenk(gls,ar=2) m3
# factors

```

Even doing 17 of these would be quite a bit faster than doing the joint estimate. However, this depends rather strongly on there being no statistical connection among the equations.

6.4 Gap Models

There have been a number of multivariate state-space models proposed to analyze the “gap” between actual and potential values, combining an unobserved components model with a Phillips curve. The hope is that the Phillips curve equation will help improve the estimate of the latent gap that would come from a simple univariate model (as in Section 5.3). For instance, the following combines the model from that section with a second one on the wage rate:

$$U_t = N_t + G_t \quad (6.6)$$

$$\Delta W_t = \mu_W - \beta G_t + \gamma' Z_t + a_{Wt} \quad (6.7)$$

where U is the unemployment rate, W is the (log) wage rate (both observable), Z are observable regressors, while N is the (unobservable) NAWRU and G the unemployment gap. In this, μ_W is the intercept and a_{Wt} is the error term. If G were known, (6.7) could be estimated by a linear regression. If we posit a UC model for N , we can estimate that and get G as the residual, and plug that into (6.7), but a combined model would give us (one assumes) more efficient estimates both of the gap itself, and of the coefficients in the Phillips curve.

Example 6.5 is based upon an example of the European Commission’s GAP model. The unemployment model is the local trend (or RW(2)) for the NAWRU with an AR(2) for the gap/cycle. The Phillips’ curve will use both current and one lag on G .

(6.7) will have undoubtedly have serially correlated errors as written. The GAP program allows for two different methods for handling this:

1. ARMAX adds the lagged dependent variable to (6.7) as one of the explanatory variables and assumes a_{Wt} is serially uncorrelated.
2. REGARIMA makes a_{Wt} an autoregressive error process.

The REGARIMA form is what is used by the RATS **BOXJENK** instruction. With it, the explanatory part of (6.7) forms the mean of ΔW_t so β measures immediate shifts to the level of ΔW_t due to G . With ARMAX, the effect of G_t will be more gradual. In this example, we’ll use ARMAX. However, to allow for an easier switch to the other form, we’ll treat a_{Wt} as a state rather than making it a measurement error (thus, all the shocks will be included in the state vector).

First off, we pull in the source file with all the forms for the NAWRU discussed in Section 5.3 and pick the “LocalTrend” option. This also chooses an AR(2) model for the cycle (gap):

```
source nawrumodel.src
compute NAWRUModel="LocalTrend"
compute ar_cycle=2
```

This sets up an **EQUATION** with the observables for **Y**:

```
equation yeqn *
# lur dws
```

Following the procedure from GAP, this pads out the lagged value of `DWS` with zeroes, and creates another `EQUATION` to represent the “exogenous” variables `Z` (with the intercept merged into this, since there’s no real difference between it and the other regressors):

```
set padx = %if(t==1,0.0,dws{1})
equation zeqn *
# constant padx
```

This sets the number of lags of the “cycle” (unemployment gap) that are included in the Phillips curve. 0 is always included, so 1 means that you have both 0 and 1.

```
compute pc_cyclelags=1      ;* from 0 to this value
```

This takes the observables (from the `Y`’s and `Z`’s) and figures out the maximum range. Everything else in the model is latent and can be generated over whatever range is needed.

```
inquire(reglist) rstart rend
# %rlfromtable(%eqntable(yeqn)) %rlfromtable(%eqntable(zeqn))
```

The guess values for the various components of the unemployment model are calculated exactly as they were in Example 5.4. Using the preliminary estimate of the gap from those guess calculations, we get starting values for the Phillips curve using:

```
linreg dws
# lur_gap{0 to pc_cyclelags} %rlfromtable(%eqntable(zeqn))
```

and this pulls the subvectors out of that and creates the `PARMSET` for the Phillips curve regression parameters:

```
dec vect beta gamma
nonlin(parmset=pcregparms) beta gamma
*
compute beta    = %xsubvec(%beta,1,pc_cyclelags+1)
compute gamma   = %if(%eqnsize(zeqn)>0,$
                    %xsubvec(%beta,pc_cyclelags+2,%nreg),%zeros(0,0))
```

and the next sets up and initializes the `PARMSET` for the Phillips curve error process. `ALPHA` is for the (unused here) `REGARIMA` autoregressive parameters:

```
dec vect alpha(0)
compute sigsqPi = %seesq
nonlin(parmset=pcerrparms) sigsqPi alpha
```

We're now ready to start putting together the state-space model. There are three blocks to this:

1. The NAWRU model
2. The cycle (gap) model
3. The Phillips curve error process

If we slightly abuse notation to write \mathbf{N} and \mathbf{G} to represent blocks required to represent a component, we have the state equation being:

$$\begin{bmatrix} \mathbf{N}_t \\ \mathbf{G}_t \\ a_{Wt} \end{bmatrix} = \begin{bmatrix} \mathbf{A}_N & 0 & 0 \\ 0 & \mathbf{A}_G & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{N}_{t-1} \\ \mathbf{G}_{t-1} \\ a_{W,t-1} \end{bmatrix} + \begin{bmatrix} \mathbf{F}_N & 0 & 0 \\ 0 & \mathbf{F}_G & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{a}_{Nt} \\ \mathbf{a}_{Gt} \\ a_{Wt} \end{bmatrix}$$

with measurement equation

$$\begin{bmatrix} u_t \\ dw_t \end{bmatrix} = \begin{bmatrix} 0 \\ \gamma' Z_t \end{bmatrix} + \begin{bmatrix} \mathbf{C}_N & \mathbf{C}_G & 0 \\ 0 & \beta & 1 \end{bmatrix} \begin{bmatrix} \mathbf{N}_t \\ \mathbf{G}_t \\ a_{Wt} \end{bmatrix}$$

The only possible adjustment that might need to be made to this is if the number of lags of G needed for β is greater than the number that are naturally needed to represent the G component. In this example, that's not an issue (β is dimension 2 and G is represented as an AR(2)), but we'll write this to allow for different values. (If you want the REGARIMA form for the Phillips curve serial correlation, you need to make the obvious adjustment to the final block of states.)

This uses the **NAWRUSize** function from the `nawrumodel.src` file to get the number of states required by the chosen trend model:

```
compute n_trend=NAWRUSize(NAWRUModel)
```

and this figures out how many states we need to include in the block for the cycle: it's the maximum of the number in its AR and the number (plus one, since the current value is included) that the Phillips curve wants:

```
compute pos_cycle=n_trend+1
compute n_cycle=%imax(ar_cycle,pc_cyclelags+1)
```

This figures out the number of state components needed for the dynamics in the Phillips Curve. If (as is here) we use the ARMAX form and include the autoregressive terms as part of the regressor list, we just use the one state with no dynamics; if we use the REGARIMA form we need the number of desired lags.

```
compute pos_pcstate=pos_cycle+n_cycle
compute n_pcstate=%if(pc_regarima,pc_arlags,1)
```

Unlike the univariate models, we'll do the cycle using just straight AR coefficients, rather than enforcing complex roots. That will make it simpler to adjust the number of lags to 1 or 3 if needed. This sets up the parameters and copies in appropriate guesses:

```
dec real sigsqc
dec vect phi(ar_cycle)
compute phi=%zeros(ar_cycle,1)
begin
  if ar_cycle>=2
    compute phi(1)=2*ampc*cos(2*%pi/tauc),phi(2)=-ampc^2
  else
    if ar_cycle==1
      compute phi(1)=ampc
    end
  *
  nonlin(parmset=cycleparms) sigsqc phi sigsqc>=0.0
```

The cycle function is similar to before except for sizes of the matrices (which adapt to the number of states required) and the first row of the **A** matrix, which here is the **PHI** vector, possibly with added zeros if the number of states needed for the Phillips curve is greater than `ar_cycle`:

```
function GCycle Z C F SW
type rect GCycle
type vect *Z *C
type rect *F
type symm *SW
compute GCycle=%dlmar(phi~~%zeros(n_cycle-ar_cycle,1))
compute Z=%zeros(n_cycle,1)
compute C=%unitv(n_cycle,1)
compute F=%unitv(n_cycle,1)
compute SW=||sigsqc||
end
compute gfunc=GCycle
```

This is the function for doing the components in the Phillips Curve error model:

```

function PCErrModel Z C F SW
type rect PCErrModel
type vect *Z
type vect *C
type rect *F
type symm *SW
*
compute SW=||sigsqPi||
*
if pc_regarima {
    compute PCErrModel=%dlmar(alpha)
    compute Z=%zeros(pc_arlags,1)
    compute F=%unitv(pc_arlags,1)
    compute C=%unitv(pc_arlags,1)
}
else {
    compute PCErrModel=||0.0||
    compute Z=||0.0||
    compute F=||1.0||
    compute C=||1.0||
}
end

```

In this case (we're not doing `pc_regarima`), we do the “else” part, which sets up a trivial state-space.

This is the “start” function for **DLM**. This uses **NAWRUStart** to set up the blocks for the trend and cycle (which are put into `xDLM` matrices), gets the components for the Phillips Curve error process and combines them using the standard methods for unrelated state space models. The only direct connection is putting the `BETA`'s into the proper location for the Phillips Curve measurement equation.

```

function DLMSetup
compute NAWRUStart (NAWRUModel)
*
* And this gets the PC error components
*
local rect ae fe
local vect ze ce
local symm swe
compute ae=PCErrModel (ze, ce, fe, swe)
*
* Combine the trend/cycle with the PC error components
*
compute adlm=adlm~\ae
compute fdm=fdm~\fe
compute cdlm=cdlm~\ce
compute zdlm=zdlm~~ze
compute swdlm=swdlm~\swe
*
* And poke the loadings on the cycle at the proper location
*
compute %psubmat (cdlm, pos_cycle, 2, beta)
end

```

This defines the observables themselves:

```

dec frml [vector] yf
frml yf = %eqnxvector (yeqn, t)

```

and this defines the MU FRML for the part of the Phillips curve regression that depends upon observables:

```

dec frml [vect] muf
frml muf = || 0.0, %eqnvalue (zeqn, t, gamma) ||

```

If the model is estimated without any restrictions (other than the non-negativity on the variances), we end up with the same behavior as we had for the Local Trend NAWRU model in Example 5.4: the two variances in the trend model end up at zero, which means that the smoothed estimates of the trend are just a linear time trend. Instead, this includes a set of restrictions used in a GAP program example which mainly have the effect of forcing a certain modest level of change in the trend rate (governed by SIGSQMU). If you want to try this without those restrictions, leave the +RESTRICTIONS off the PARMSET option on the DLM. The output is in Table 6.2.

Table 6.2: Bivariate Gap Model with Restrictions

DLM - Estimation by BFGS with inequalities					
Convergence in 16 Iterations. Final criterion was 0.0000023 <= 0.0000100					
Annual Data From 1965:01 To 2016:01					
Usable Observations		52			
Rank of Observables		104			
Log Likelihood		90.0496			
	Variable	Coeff	Std Error	T-Stat	Signif
1.	SIGSQP	0.0648	0.0638	1.0163	0.3095
2.	SIGSQMU	0.0500	0.0000	0.0000	0.0000
3.	SIGSQC	0.1750	0.0000	0.0000	0.0000
4.	PHI(1)	1.3548	0.3026	4.4774	0.0000
5.	PHI(2)	-0.4872	0.2026	-2.4054	0.0162
6.	SIGSQPI	0.0003	0.0001	4.6861	0.0000
7.	BETA(1)	-0.0069	0.0042	-1.6455	0.0999
8.	BETA(2)	0.0050	0.0041	1.2148	0.2244
9.	GAMMA(1)	-0.0001	0.0024	-0.0544	0.9566
10.	GAMMA(2)	0.2797	0.1396	2.0031	0.0452

```

nonlin(parmset=restrictions) sigsqmu>=.05 sigsqmu<=.08 $
  sigs qc<=.175 sigsqp<=.10
compute allparms=nparms (NAWRUModel)+cycleparms+pcerrparms+pcregparms
dmlm(start=DLMSetup(),parmset=allparms+restrictions,y=yf,mu=muf,$
  a=adlm,c=cdlm,f=fdlm,z=zdlm,sw=swdlm,$
  presample=ergodic,condition=2,$
  method=bfgs,itors=500,type=smooth) rstart rend xstates

```

Restrictions on SIGSQMU (trend rate variance) and SIGSQC (cycle innovation variance) are both binding: for SIGSQMU, at its lower bound and for SIGSQC, at its upper bound. You need to be careful about the t -statistics on parameters that hit binding constraints—in this case, the SIGSQMU shows a zero standard error (which it's supposed to), but gets a garbage t -stat due to numerical errors in detecting the situation. A parameter on a binding constraint isn't testable based upon t 's, so you should ignore those when reporting your results.

The cycle coefficients in the Phillips curve equation are probably not what one would hope—BETA(1), which is on the current gap, has the correct sign, but isn't really significant, and BETA(2) has the wrong sign. A joint test (which you can easily set up using the *Statistics-Regression Tests* wizard) comes in insignificant:

```

test(zeros,title="Joint test of Gap in Phillips Curve")
# 7 8

```

Joint test of Gap in Phillips Curve		
Chi-Squared(2)=	2.727090 or F(2,*)=	1.36354 with Significance Level 0.25575256

The following pulls out the (smoothed) NAWRU and cycle estimates and graphs them (Figure 6.6):

```
set nawru = xstates(t) (1)
set gap    = xstates(t) (pos_cycle)
spgraph(vfields=2)
graph(header="Unemployment and NAWRU") 2
# lur
# nawru
graph(header="Unemployment Cycle")
# gap
spgraph(done)
```

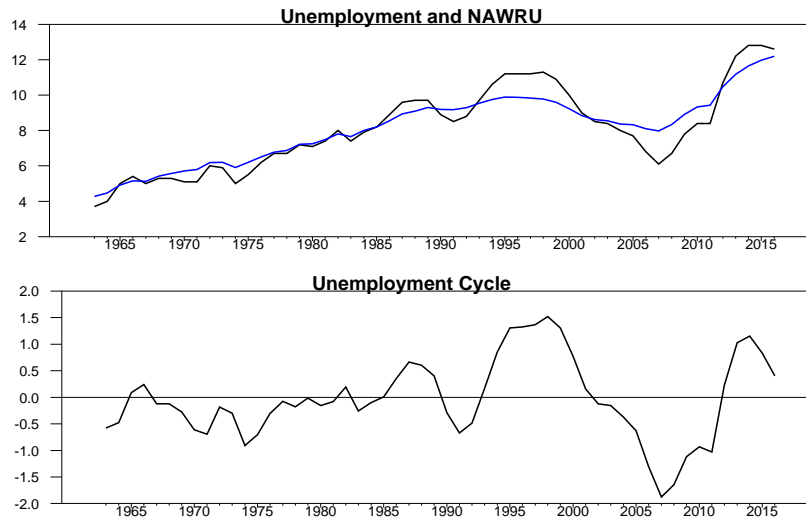


Figure 6.6: Unemployment Decomposition from Bivariate Gap Model

Given the behavior of the cycle, it's not a surprise that the coefficients on the AR create complex roots, even though we didn't enforce that—you can almost read off the 5-6 year (20-24 quarter) period from the graph. You can use **SUMMARIZE** to calculate estimates of those and (delta-method) approximated standard errors:

```
summarize(title="ML cycle amplitude",parmset=allparms) sqrt(-phi(2))
summarize(title="ML cycle tau",parmset=allparms) $
  2*pi/(%acos(phi(1)/(2*sqrt(-phi(2)))))
```

Example 6.1 Stock-Watson Indicator Model

This is the Stock-Watson (single) business cycle indicator model, discussed in Section 6.1.

```

open data sw1991.rat
calendar(m) 1958
data(format=rats) 1958:1 1987:12 ip gmyxp8 mt82 lpnag lpmhuadj
*
set emp      = lpmhuadj
set ipgrow   = 100.0*log(ip/ip{1})
set incgrow  = 100.0*log(gmyxp8/gmyxp8{1})
set salesgrow = 100.0*log(mt82/mt82{1})
set empgrow  = 100.0*log(emp/emp{1})
*
* Parameters for index
* NC is the autoregression lag for it. The innovation variance is pegged
* to 1.0.
*
compute nc=2
dec vect phi
compute phi=%zeros(nc,1)
dec real sigmac
compute sigmac=1.0
*
* Parameters for the series. sigma is initialized based upon the sample
* means. This allows for differing lags in the error terms for different
* equations. (In this example, they're all 2).
*
compute nvar=4
dec vect beta(nvar) gamma(nvar)
dec vect sigma(nvar)
dec vect[int] lags(nvar) apos(nvar)
*
compute gamma=%const(1.0)
stats(noprint) ipgrow
compute lags(1)=2,beta(1)=%mean,sigma(1)=%variance*.5
stats(noprint) incgrow
compute lags(2)=2,beta(2)=%mean,sigma(2)=%variance*.5
stats(noprint) salesgrow
compute lags(3)=2,beta(3)=%mean,sigma(3)=%variance*.5
stats(noprint) empgrow
compute lags(4)=2,beta(4)=%mean,sigma(4)=%variance*.5
*
* These are the AR coefficients on the variable-specific components.
*
dec vect[vect] d(nvar)
ewise d(i)=%zeros(lags(i),1)
*
* This takes a p-vector representing the AR coefficients into an p x p
* matrix for an autoregressive component in a state-space representation.
*
function %dlmar phi

```

```

type rect %dlmar
type vect phi
*
local int i j
*
dim %dlmar(%size(phi),%size(phi))
ewise %dlmar(i,j)=%if(i==1,phi(j),i==j+1)
end
*
* Construct fixed parts of the A, F and C matrices. The variable
* elements (those that depend upon the free parameters) will be "poked"
* in later.
*
dec rect a f c
*
* Start with the component for the cycle.
*
compute a=%dlmar(%zeros(nc,1))
compute f=%unitv(nc,1)
dim c(0,0)
*
* Add in the variable-specific components
*
do i=1,nvar
  compute apos(i)=%rows(a)+1
  compute a=a~\%dlmar(%zeros(lags(i),1))
  compute f=f~\%unitv(lags(i),1)
  compute c=c~\%unitv(lags(i),1)
end do i
compute c=%zeros(nc,nvar)~~c
*
dec symm sw(nvar+1,nvar+1)
*
function %%DLMSSetup
compute %psubmat(a,1,1,tr(phi))
do i=1,nvar
  compute %psubmat(a,apos(i),apos(i),tr(d(i)))
end do i
compute %psubmat(c,1,1,tr(gamma))
compute sw=%diag(sigmac~~sigma)
end %%DLMSSetup
*
* The observables are the growth rates. The MU option is used to
* subtract off their means. The paper uses the sample means rather than
* estimating them, so we leave beta out of the parameter set.
*
dec frml[vect] yf
frml yf = ||ipgrow,incgrow,salesgrow,empgrow||
*
* Estimate the parameters by maximum likelihood.
*
nonlin gamma d sigma phi
*
dlm(start=%%DLMSSetup(),y=yf,sw=sw,c=c,a=a,f=f,mu=beta,$

```

```

    presample=ergodic,type=filter,$
    pmethod=simplex,piters=5,method=bfgs) 1959:2 * xstates
*
* The indicator series is the first component of the state vector. The
* accumulated value of this would give the estimate of the actual
* coincident indicator. As mentioned in the paper, both the filtered
* version and the smoothed version of this have their own values. The
* filtered would give the current state of the business cycle, while the
* smoothed value would give a retrospective look.
*
* As this is constructed, it somewhat arbitrarily fixes the pre-sample
* value of the indicator at zero.
*
set indicator 1959:2 * = xstates(t)(1)
acc indicator
graph(footer="Coincident Indicator, Filtered Version")
# indicator
*
dlim(start=%DLMSetup(),y=yf,sw=sw,c=c,a=a,f=f,presample=ergodic,$
      type=smooth) 1959:2 * xstates
set indicator 1959:2 * = xstates(t)(1)
acc indicator
graph(footer="Coincident Indicator, Smoothed Version")
# indicator

```

Example 6.2 Bivariate H-P Filter

This estimates a bivariate form of Hodrick-Prescott filter (using a common trend element). This is discussed in Section 6.2.

```

open data oecdgdp.rat
calendar(q) 1961
data(format=rats) 1961:1 2009:2 cannaexcp01sts usanaexcp01sts
*
set canrgdp = log(cannaexcp01sts)
set usargdp = log(usanaexcp01sts)
*
compute n=2
*
dec frml[vect] yf
frml yf = ||canrgdp,usargdp||
*
* Standard HP variance ratio for quarterly data.
*
compute lambda=1600.0
*
* Standard local trend model for the common growth component
*
@LocalDLM(type=trend,shock=trend,a=a,f=f)
compute sw=1.0/lambda
*
compute [symm] sv=%identity(n)

```

```

*
* Each series is modeled as  $y(i) = a(i) + \gamma(i) * \text{growth} + v(i)$ 
* Because there are  $n+1$  "intercepts" in the complete model (1 for each
* data series + 1 in the growth component), the series intercepts are
* constrained to sum to zero. (This is accomplished by using the %perp
* of a vector of 1's premultiplying an  $n-1$  vector of free coefficients).
* The "sv" matrix in the DLM gives the weighting scheme applied to the
* errors in the observables. If this is the identity matrix, the series
* get equal weights.
*
dec vect gamma(n)
dec vect theta(n-1)
*
* Because the HP filter is only able to estimate relative variances, the
* gammas need one scale normalization to prevent an indeterminacy.
*
nonlin gamma theta %qform(inv(sv),gamma)==1.0
compute gamma=%const(1.0),theta=%const(0.0)
*
compute oneperp=%perp(%ones(1,n))
*
dec frml[rect] cf
frml cf = tr(gamma)~~%zeros(1,n)
*
dlm(a=a,f=f,mu=oneperp*theta,sw=sw,sv=sv,c=cf,y=yf,presample=diffuse,$
    var=concentrate,type=smooth,method=bfgs) / xstates
*
set canf = %dot(%xcol(cf,1),xstates(t))+%dot(%xrow(oneperp,1),theta)
set usaf = %dot(%xcol(cf,2),xstates(t))+%dot(%xrow(oneperp,2),theta)
*
set cancycle = canrgdp-canf
set usacycle = usargdp-usaf
*
filter(type=hp) canrgdp / canhp
set canhpcycle = canrgdp-canhp
graph(footer="Canadian Cycle Comparison",$
    key=lleft,klables=||"Bivariate","Univariate"||) 2
# cancycle
# canhpcycle
*
filter(type=hp) usargdp / usahp
set usahpcycle = usargdp-usahp
graph(footer="USA Cycle Comparison",$
    key=lleft,klables=||"Bivariate","Univariate"||) 2
# usacycle
# usahpcycle

```

Example 6.3 Dynamic Factor Model with Theoretical Loadings

This estimates a 3-factor dynamic factor model on bond rates using theoretical loadings. This is the first model covered in Section 6.3.

```
open data bonds.xls
calendar(m) 1972:1
data(format=xls,org=columns) 1972:01 2000:12 m3 m6 m9 m12 m15 m18 $
    m21 m24 m30 m36 m48 m60 m72 m84 m96 m108 m120
*
equation yvars *
# m3 to m120
*
compute n=%eqnsize(yvars)
*
* tau is the maturity, which is stripped out of the names of the series.
* %mid(string,2,0) takes the characters in <<string>> from 2 to the end.
*
dec vect tau(n)
ewise tau(i)=%value(%mid(%eqnreglabels(yvars)(i),2,0))
*
* Three factor model
*
compute nf=3
*
dec real lam
dec rect a(nf,nf)
dec symm sw(nf,nf)
dec vect swdiag(nf)
dec rect lambda(nf,n)
dec vect svdiag(n)
dec symm sv
*****
*
* Function for generating the "C" (loadings) matrix for DLM.
*
function FLoadSetup
type rect FLoadSetup
dim FLoadSetup(3,n)
do i=1,n
    compute t1=tau(i)*lam
    compute FLoadSetup(1,i)=1
    compute FLoadSetup(2,i)=(1-exp(-t1))/t1
    compute FLoadSetup(3,i)=FLoadSetup(2,i)-exp(-t1)
end do i
end FLoadSetup
*****
*
* Calculations fixed across time, but dependent upon parameters
*
function DRASetup3
compute lambda=FLoadSetup()
```

```

compute sv=%diag(%exp(svdiag))
end
*****
* Guess values
*
compute a =.95*%identity(nf)
compute sw=.50*%identity(nf)
compute svdiag=%fill(n,1,-5.0)
compute lam=.1
*
* Full Q matrix
*
nonlin a sw svdiag lam
dlm(startup=%(DRASetup3()),$,
    a=a,sw=sw,sv=sv,c=lambda,y=%eqnxvector(yvars,t),$,
    presample=ergodic,method=bfgs,itors=400,type=smooth) / factors
*
set beta1 = factors(t) (1)
set beta2 = factors(t) (2)
set beta3 = factors(t) (3)
*
set level = m120
set slope = m3-m120
set curve = -m120+2*m24-m3
spgraph(vfields=3,footer="Figure 8.12 Smoothed Estimates of Factors")
graph(header="Level factor and data proxy") 2
# beta1
# level
graph(header="Slope factor and data proxy") 2
# beta2
# slope
graph(header="Curvature factor and data proxy") 2
# beta3
# curve
spgraph(done)

```

Example 6.4 Dynamic Factor Model, Principal Components

This estimates a 3-factor dynamic factor model on bond rates using principal components. This is the second model covered in Section 6.3.

```

open data bonds.xls
calendar(m) 1972:1
data(format=xls,org=columns) 1972:01 2000:12 m3 m6 m9 m12 m15 m18 $
    m21 m24 m30 m36 m48 m60 m72 m84 m96 m108 m120
*
equation yvars *
# m3 to m120
*
compute n=%eqnsize(yvars)
*
* Simple principal components with idiosyncratic AR errors.

```

```

*
compute nc=3
compute nlags=2
*
@princomp(equation=yvars, ncomps=nc) / factors
*
dec vect[vect] phis(n)
dec vect[vect] gammas(n)
dec vect swvar(n)
dec symm sw(n,n)
*
sweep(series=resids, coeffs=loadings)
# %rlfromtable(%eqntable(yvars))
# factors
do i=1,n
    compute gammas(i)=%xcol(loadings,i)
    linreg(noprint) resids(i)
    # resids(i){1 to nlags}
    compute phis(i)=%beta
    ewise swvar(i)=%seesq
end do i
*
dec rect a(nlags*n, nlags*n)
ewise a(i,j)=(i==j+1).and.%clock(i, nlags) <> 1
*
dec rect f(nlags*n, n)
ewise f(i,j)=(i==(j-1)*nlags+1)
*
dec rect c(nlags*n, n)
ewise c(i,j)=(i==(j-1)*nlags+1)
*
function SetupPCDLM
do i=1,n
    *
    * This is the first position of the states for the
    * error process for variable i
    *
    compute ipos=(i-1)*nlags+1
    compute %psubmat(a, ipos, ipos, tr(phis(i)))
end do i
compute sw=%diag(swvar)
end
*
function PCmuF t
type vect PCmuF
type integer t
*
dim PCmuF(n)
ewise PCmuF(i)=%dot(gammas(i), %xt(factors,t))
end PCmuF
*
nonlin phis gammas swvar
dlm(y=%eqnxvector(yvars,t), start=SetupPCDLM(), a=a, c=c, sw=sw, $
    f=f, mu=PCmuF(t), presample=ergodic, pmethod=simplex, peters=5, $

```



```

    method=bfgs) / xstates
*
* Graph of factors
*
graph(footer="Factors",key=loleft,klabels=||"F1","F2","F3"||) 3
# factors(1)
# factors(2)
# factors(3)
*
set maturity 1 n = %value(%mid(%eqnreglabels(yvars)(t),2,-1))
set load1 1 n = gammas(t)(1)
set load2 1 n = gammas(t)(2)
set load3 1 n = gammas(t)(3)
*
* Graph of loadings vs maturity
*
scatter(footer="Loadings vs Maturity",style=line,$
    key=below,klabels=||"F1","F2","F3"||) 3
# maturity load1
# maturity load2
# maturity load3
*
* Note that because of the assumptions which leave the equations
* unconnected statistically (since the factors are treated as exogenous
* in the DLM), the results can be obtained (theoretically exactly) by
* using a sequence of BOXJENK AR(2) models with the FACTORS as extra
* regressors.
*
boxjenk(gls,ar=2) m3
# factors

```

Example 6.5 Bivariate Gap model

This estimates a bivariate “gap” model, including a decomposition of the unemployment rate combined with a Phillips curve. This is from Section 6.4.

```

open data "gap44_data.xls"
calendar(a) 1963:1
data(format=xls,org=columns) 1963:01 2016:01 year lur dws ddtot
*
* Pull in the NAWRU models and choose the local trend with an AR(2) cycle:
*
source nawrumodel.src
*
compute NAWRUModel="LocalTrend"
compute ar_cycle=2
*
* Observables from measurement equation
*
equation yeqn *
# lur dws
*

```

```

* Exogenous regressors (if any) in Phillips curve. This does an "ARMA-X"
* handling of the Phillips Curve autoregression (that is, the lag is
* treated as a regressor). An alternative and more complicated method is
* "REG-ARIMA", which uses a separate AR noise term instead.
*
compute pc_arlags=0
compute pc_regarima=0
*
set padx = %if(t==1,0.0,dws{1})
equation zeqn *
# constant padx
*
compute pc_cyclelags=1      ;* from 0 to this value
*
inquire(reglist) rstart rend
# %rlfromtable(%eqntable(yeqn)) %rlfromtable(%eqntable(zeqn))
*
* Starting values for unemployment model. (This covers parameters used
* by all the different possible NAWRU models).
*
filter(type=hp) lur / lur_hp
set lur_gap = lur-lur_hp
*
set trate = lur_hp-lur_hp{1}
*
compute rho=.8
stats trate
compute mu0=%mean
compute sigsqmu=%variance*(1-rho^2)
compute sigsqp=sigsqmu*.25
*
linreg lur_gap
# lur_gap{1 2}
compute sigsqc=%sigmasq
compute tauc=6.0,ampc=.8
*
* Starting values for Phillips curve (taking the HP estimate of the gap
* as a regressor).
*
linreg dws
# lur_gap{0 to pc_cyclelags} %rlfromtable(%eqntable(zeqn))
*
* Copy guess values into the regression parameter vectors and set up
* their parameter set.
*
dec vect beta gamma
nonlin(parmset=pcregparms) beta gamma
*
compute beta    = %xsubvec(%beta,1,pc_cyclelags+1)
compute gamma   = %if(%eqnsize(zeqn)>0,$
                    %xsubvec(%beta,pc_cyclelags+2,%nreg),%zeros(0,0))
*
* Set up the parameters for the PC error process. (alpha would be for
* the lag coefficients in the REGARIMA form, which here is empty)

```

```

*
compute pc_arlags=%if(pc_regarima,pc_arlags,0)
dec vect alpha(pc_arlags)
compute sigsqPi = %seesq
nonlin(parmset=pcerrparms) sigsqPi alpha
*
* System matrices
*
function %dlmar phi
type rect %dlmar
type vect phi
*
local int i j
*
dim %dlmar(%size(phi),%size(phi))
ewise %dlmar(i,j)=%if(i==1,phi(j),i==j+1)
end
*
compute n_trend=NAWRUSize(NAWRUModel)
*
* Number of state components needed for the cycle. If the PC needs a
* lot, we may need more than are actually needed to represent the cycle
* itself.
*
compute pos_cycle=n_trend+1
compute n_cycle=%imax(ar_cycle,pc_cyclelags+1)
*
* Number of state components needed for the dynamics in the PC. For
* simplicity, we include the idiosyncratic shock in the state vector,
* even if there are no dynamics.
*
compute pos_pcstate=pos_cycle+n_cycle
compute n_pcstate=%if(pc_regarima,pc_arlags,1)
*
* Function for doing the components in the cycle model. This does direct
* parameterization of the AR coefficients.
*
dec real sigsqc
dec vect phi(ar_cycle)
compute phi=%zeros(ar_cycle,1)
begin
  if ar_cycle>=2
    compute phi(1)=2*ampc*cos(2*%pi/tauc),phi(2)=-ampc^2
  else
    if ar_cycle==1
      compute phi(1)=ampc
    end
  end
end
*
nonlin(parmset=cycleparms) sigsqc phi sigsqc>=0.0
*
function GCycle Z C F SW
type rect GCycle
type vect *Z *C
type rect *F

```

```

type symm *SW
compute GCycle=%dlmar(phi~~%zeros(n_cycle-ar_cycle,1))
compute Z=%zeros(n_cycle,1)
compute C=%unitv(n_cycle,1)
compute F=%unitv(n_cycle,1)
compute SW=||sigsqc||
end
compute gfunc=GCycle
*
* Function for doing the components in the error model
*
function PCErrModel Z C F SW
type rect PCErrModel
type vect *Z
type vect *C
type rect *F
type symm *SW
*
compute SW=||sigsqPi||
*
if pc_regarima {
    compute PCErrModel=%dlmar(alpha)
    compute Z=%zeros(pc_arlags,1)
    compute F=%unitv(pc_arlags,1)
    compute C=%unitv(pc_arlags,1)
}
else {
    compute PCErrModel=||0.0||
    compute Z=||0.0||
    compute F=||1.0||
    compute C=||1.0||
}
end
*
* Patch over the elements in the state matrices with functions of the
* parameters.
*
function DLMSSetup
*
* This takes care of the combined trend and cycle. (Creating the "xDLM"
* matrices).
*
compute NAWRUSstart(NAWRUModel)
*
* And this gets the PC error components
*
local rect ae fe
local vect ze ce
local symm swe
compute ae=PCErrModel(ze,ce,fe,swe)
*
* Combine the trend/cycle with the PC error components
*
compute adlm=adlm~\ae

```

```

compute fd1m=fd1m~\fe
compute cd1m=cd1m~\ce
compute zd1m=zd1m~~ze
compute swd1m=swd1m~\swe
*
* And poke the loadings on the cycle at the proper location
*
compute %psubmat(cd1m,pos_cycle,2,beta)
end
*
* Defines the observables
*
dec frml[vector] yf
frml yf = %eqnvector(yeqn,t)
*
* And the regression values on observables
*
dec frml[vect] muf
frml muf = ||0.0,%eqnvalue(zeqn,t,gamma)||
*
* Kalman filtering with ML. These use presample=ergodic with
* condition=2 which should give the same results as the deJong method.
* (Without CONDITION=2, the results will differ slightly because
* Koopman's method can make use of the rank one information in the 2nd
* entry prediction, while deJong's doesn't).
*
nonlin(parmset=restrictions) sigsqmu>=.05 sigsqmu<=.08 $
    sigsqc<=.175 sigsqp<=.10
compute allparms=nparms(NAWRUModel)+cycleparms+pcerrparms+pcregparms
dlm(start=DLMSetup(),parmset=allparms+restrictions,y=yf,mu=muf,$
    a=ad1m,c=cd1m,f=fd1m,z=zd1m,sw=swd1m,$
    presample=ergodic,condition=2,$
    method=bfgs,itors=500,type=smooth) rstart rend xstates
*
test(zeros,title="Joint test of Gap in Phillips Curve")
# 7 8
*
set nawru = xstates(t)(1)
set gap = xstates(t)(pos_cycle)
spgraph(vfields=2)
    graph(header="Unemployment and NAWRU") 2
    # lur
    # nawru
    graph(header="Unemployment Cycle")
    # gap
spgraph(done)
*
summarize(title="ML cycle amplitude",parmset=allparms) sqrt(-phi(2))
summarize(title="ML cycle tau",parmset=allparms) $
    2*pi/(%acos(phi(1)/(2*sqrt(-phi(2))))))

```

Interpolation and Distribution

One important (and non-trivial) use of state-space techniques for single series is interpolation or distribution of data. These techniques are often known generically as “interpolation”, but, as pointed out by Chow and Lin, there are actually two quite distinct applications: *interpolation*, which is applied (primarily) to stocks, and *distribution*, which is applied to flows. By far the more commonly used of these is distribution (which is also the more difficult), in which the constructed series is designed to sum to the observed series over the course of the interval. We’ll concentrate on it in this Chapter.

It’s assumed that we may have some useful information from other data at the target frequency. The following set of options takes in almost all known techniques for this. The connection between the high-frequency information and the interpolated series is represented by one of the models:

$$I_t = H_t + u_t \quad (7.1)$$

$$I_t = H_t u_t \quad (7.2)$$

$$\log I_t = H_t + u_t \quad (7.3)$$

The high-frequency information from related series takes one of the forms:

$$H_t = O_t \beta, O_t \text{ observable} \quad (7.4)$$

$$H_t \text{ observable} \quad (7.5)$$

$$H_t = 0 \quad (7.6)$$

and the time series model for the deviation between the high-frequency and the interpolated data is one of:

$$u_t = \rho u_{t-1} + v_t \quad (7.7)$$

$$u_t = u_{t-1} + v_t \quad (7.8)$$

$$u_t = (1 + \rho)u_{t-1} - \rho u_{t-2} + v_t \quad (7.9)$$

While it’s possible to allow for a more complicated time series structure for the noise, in practice, there’s no need for that; the requirement that the data hit known values at a regular interval will dominate any short-term dynamics.

The observation equation connecting the interpolated data with the observed

data is one of:

$$L_t = I_t, t = q, 2q, \dots \quad (7.10)$$

$$L_t = \sum_{s=0}^{q-1} I_{t-s}, t = q, 2q, \dots \quad (7.11)$$

where q is the ratio between the desired higher frequency and the observed lower frequency: $3 = 12/4$ for quarterly to monthly. The first of these is the formal definition of interpolation, while the second is distribution.

7.1 Linear Model

The combination of (7.1) with (7.4) and (7.7) forms the procedure from Chow & Lin (1971). While there are many examples of the use of this, it seems likely that this is misspecified. If (as is typical), I_t is an $I(1)$ process, then it has to be co-integrated with some member of O_t in order for the noise to be stationary. This is probably overstating that relationship. Replacing (7.7) with (7.8) is the method from Fernandez (1981), while using (7.9) is from Litterman (1983).

While including a unit root in the disturbance process is likely to be much closer to a realistic description of a relationship than the stationary disturbance in Chow-Lin,¹ the linear relationship between a variable like GDP and its related series would again seem misspecified—such variables are almost always modeled in logarithms. The choice of the linear relationship (7.1) is more a matter of computational convenience when combined with the adding-up constraint (7.11). After showing how to handle the strictly linear models with state-space techniques, we'll move on to the more realistic log-linear model.

A couple of things to note. First, all the combinations above can be done using the **@DISAGGREGATE** procedure, so you don't have to do any programming if you just want to employ these methods in their standard forms. Second, if you read any of the papers cited above, you'll note that none of them (explicitly) use state-space techniques. There are many published papers which, in effect, re-derive Kalman smoothing calculations for specific models (or just invert a full $T \times T$ matrix when Kalman smoothing could do the calculation more efficiently).

In translating the linear models into state-space form, the states will be u_t and its lags. How many lags do we need? The evolution of the states requires none for (7.7) and (7.8) and one for (7.9).² Where we will need extra lags is in the (final form of) the measurement equation (7.11). We'll need q total for that, so we'll have to expand the state vector with lags to give us $u_t, u_{t-1}, \dots, u_{t-q+1}$.

If we substitute (7.1) into (7.11), we get a measurement equation of:

$$L_t = (O_t + \dots + O_{t-q+1})\beta + [1, \dots, 1] [u_t, \dots, u_{t-q+1}]'$$

¹Litterman shows that it produces more accurate values for a number of examples.

²Remember that the first lag will be in the lag term in the state equation.

Note that we only have an observation on this every q periods. $O_t + \dots + O_{t-q+1}$ is known, so given β , we can subtract those terms off the observable to get:

$$L_t - (O_t + \dots + O_{t-q+1})\beta = [1, \dots, 1] [u_t, \dots, u_{t-q+1}]' \quad (7.12)$$

The combination of (7.12) and the appropriate state representation for the u_t process will form a state-space model for the data. We can estimate the β by maximum likelihood and use Kalman smoothing to get the distributed series I_t . Kalman smoothing gives us smoothed estimates of u_t —we just need to add that to the computed values of H_t .

One thing to note is that the noise model (7.7) is stationary, (7.8) has one unit root and (7.9) has one unit root and one stationary root. You can use the option `PRESAMPLE=ERGODIC` to handle the initial mean and variance for any of these. If you look at the `@DISAGGREGATE` procedure, it uses a `G` matrix which is a set of difference operations for the two unit root choices. Either method is fine. The stationary noise model has a zero mean, so the O_t variables should include a constant. The unit root noise processes can seek their own level, so O_t should *not* include a constant: it will be redundant.

7.2 Log-Linear Model

Let's now look at how we can handle the more realistic log-linear relationship. (7.11) would now need to be written:

$$L_t = \exp(\log I_t) + \dots + \exp(\log I_{t-q+1})$$

since our state-space model is able to generate $\log I_t$. The state equation is linear in $\log I_t$, but the measurement equation is non-linear. What we will describe now is the simplest case of the *Extended Kalman Filter*, which generalizes the Kalman filter to allow for non-linearities. This treats the non-linearities by repeated linearizations. Each linearized version can be solved using the standard Kalman filter.

About what do we linearize? $\log I_t$ is a function of the O_t variables and the unobservable states u_t . We can use the values for that from a previous linearization pass (for iteration 1, taking $u_t = 0$). For convenience, write $i_t = \log I_t$, and \tilde{i}_t as the Kalman smoothed estimate of i_t from the previous iteration. Then the linearized version is:

$$\begin{aligned} L_t &\approx \exp(\tilde{i}_t) + \dots + \exp(\tilde{i}_{t-q+1}) \\ &\quad + \exp(\tilde{i}_t)(i_t - \tilde{i}_t) + \dots + \exp(\tilde{i}_{t-q+1})(i_{t-q+1} - \tilde{i}_{t-q+1}) \\ &= \exp(\tilde{i}_t) (1 - \tilde{i}_t) + \dots + \exp(\tilde{i}_{t-q+1}) (1 - \tilde{i}_{t-q+1}) \\ &\quad + \exp(\tilde{i}_t)i_t + \dots + \exp(\tilde{i}_{t-q+1})i_{t-q+1} \end{aligned} \quad (7.13)$$

The first line terms in (7.13) are fixed for a given iteration and can be subtracted from L_t . The related regressors are a bit more complicated now, since,

when we substitute in the definition of i_t , we get for the second set of terms:

$$(\exp(\tilde{i}_t)O_t + \dots + \exp(\tilde{i}_{t-q+1})O_{t-q+1})\beta + \exp(\tilde{i}_t)u_t + \dots + \exp(\tilde{i}_{t-q+1})u_{t-q+1}$$

The combined “regressor” to determine β now changes from iteration to iteration, as do the weights on the states.

There’s one minor problem with how this will operate. Iterating on the above gives us a solution for the states given β . But β needs to be estimated as well. One possibility is to update the expansion point with each iteration on β , that is, do just one linearization per iteration. However, there’s a simpler way to handle this. Given the rest of the structure of the state-space model, the model is a linear equation in β , which can be estimated directly by a form of GLS. In fact, the papers cited all show the form of the GLS estimator. With **DLM**, the most convenient way to do this is to add the coefficients to the states, gluing together a model with the noise model states plus

$$\beta_t = \beta_{t-1} + 0$$

The loadings on those “states” will be

$$(\exp(\tilde{i}_t)O_t + \dots + \exp(\tilde{i}_{t-q+1})O_{t-q+1})$$

When you do this, the coefficients should be the last block of states, and you should include the option `FREE=number of regressors`. That adjusts the log likelihood so it will give the value conditional on β , which is the form you would use in doing this by GLS.

7.3 Proportional Denton Method

As an example, we’ll look a distribution method which uses just a single related series. This is the proportional Denton method. You can find a description of this in Bloem et al. (2001), chapter 6. Statistics bureaus often have different reads on the same series at different recording frequencies. The general assumption is that the coarser the collection interval, the more accurate the data. For instance, in the U.S., some of the information which goes into GDP calculations comes from tax returns, which are only available well into the next year, while other information is available monthly or quarterly. The proportional Denton method assumes that (for instance), the annual data are accurate. The distributed quarterly data are to sum to the annual value, but should (roughly) show the movements apparent in the observations of the less accurate observed quarterly data.

The proportional Denton method is described on page 87 of the IMF manual as the solution to:³

$$\min \sum_{t=2}^T \left(\frac{I_t}{H_t} - \frac{I_{t-1}}{H_{t-1}} \right)^2$$

³Variable names have been changed to match our discussion

over $\{I_t\}$ subject to

$$L_t = I_t + \dots + I_{t-q+1}; t = q, 2q, \dots$$

You'll notice again that this is not written in state-space form. However, if we define u_t by $I_t = H_t u_t$, then, since H_t are known, the problem can be rewritten as:

$$\min \sum_{t=2}^T (u_t - u_{t-1})^2 \quad (7.14)$$

over $\{u_t\}$ subject to

$$L_t = [H_t, \dots, H_{t-q+1}][u_t, \dots, u_{t-q+1}]'; t = q, 2q, \dots \quad (7.15)$$

The solution to this turns out to be the same as Kalman smoothing on the model with state equation (augmented by lags):

$$u_t = u_{t-1} + w_t$$

and measurement equation (7.15). The sum in (7.14) is just the sum of squared w_t . Since those are assumed to be i.i.d. Normal, that, in effect, gives the unconditional density for the w_t . Kalman smoothing gives us the *conditional* means of the states and the disturbances subject to the observations.

Example 7.1 does an annual to quarterly distribution so $q = 4$. This means the A and F matrices can be set up by:

```
dec rect a(4,4)
input a
  1 0 0 0
  1 0 0 0
  0 1 0 0
  0 0 1 0
compute f=%unitv(4,1)
```

As we can see from (7.15), the C matrix here depends upon the data. We can set this up with

```
dec frml[vec] cf
frml cf = ||quarter{0}, quarter{1}, quarter{2}, quarter{3}||
```

The only thing not “known” is the variance of w_t , but since we’re interested only in the means, we don’t even need to know that or estimate it. We can just make it 1.0 and be done. The distributed values are obtained here by multiplying the observed quarterly data by the Kalman smoothed estimates of u_t :

```
dlim(type=smoothed, a=a, c=cf, y=annual, f=f, sw=1.0, $
presample=ergodic) 1998:1 2000:4 xstates
set distrib = %scalar(xstates)*quarter
```

Example 7.1 Proportional Denton method

Note that this is also part of the @DISAGGREGATE procedure.

```

cal(q) 1998
all 2000:4
data(unit=input) / quarter annual
  98.2 100.8 102.2 100.8 99.0 101.6 102.7 101.5 100.5 103.0 103.5 101.5
  . . . 4000 . . . 4161.4 . . . .
*
* This setup is specific to quarterly from annual. The series "quarter"
* is the quarterly version of the data, and "annual" is the annual one.
* For this to work correctly, your annual series needs to be read into
* the quarterly frame with data only for the final quarter of each year.
* If you use one of the "smarter" data formats which does an automatic
* expansion, you'll need to patch the annual series. You can do this by
*
* set annual = %if(%period(t)==4,annual,%na)
*
dec rect a(4,4)
input a
  1 0 0 0
  1 0 0 0
  0 1 0 0
  0 0 1 0
compute f=%unitv(4,1)
dec frml[vec] cf
*
* The states are defined to be the ratio between the benchmark estimates
* of the quarterly data and the observed quarterly data. The measurement
* equation says that the benchmark estimates sum to the annual number
* with no error.
*
* To adapt to monthly from quarterly, just drop the 3rd lag from the
* definition of cf.
*
frml cf = ||quarter{0},quarter{1},quarter{2},quarter{3}||
*
dlm(type=smoothed,a=a,c=cf,y=annual,f=f,sw=1.0,presample=ergodic) $
  1998:1 2000:4 xstates
*
* The distributed data is obtained by multiplying the quarterly
* observables by the Kalman smoothed states.
*
set distrib = %scalar(xstates)*quarter
print / distrib quarter annual

```

Simulation Methods

This chapter looks at various methods of analyzing state-space models which rely upon simulations. Typically, these are used for Gibbs sampling of underlying parameters of the model.

8.1 Conditional Simulation

It's easy to generate an *unconditional* draw for the states because of the sequential independence. Given \mathbf{X}_{t-1} , we draw \mathbf{W}_t and compute $\mathbf{X}_t = \mathbf{A}_t\mathbf{X}_{t-1} + \mathbf{Z}_t + \mathbf{F}_t\mathbf{W}_t$. We repeat this as necessary. This is done using **DLM** with the option `TYPE=SIMULATE` (rather than `TYPE=FILTER` or `TYPE=SMOOTH`). That, however, isn't very interesting since the generated states will have no relationship to the observed data. The one real use of this is for out-of-sample simulations, allowing for analysis of more complicated functions of forecasts than just the mean and variance.

Of greater interest is a draw for the states *conditional on the data*. The Kalman smoother gives us the mean and covariance matrix for each state individually, but that isn't enough to allow us to do draws since, conditional on the data, the states are highly correlated. Conditional simulation is even more complicated than Kalman smoothing, but can be done with a couple of Kalman smoothing passes, one on simulated data. Conditional simulation is chosen in RATS with the option `TYPE=CSIMULATE` (conditional simulation).

The main use of conditional simulation is in Bayesian techniques such as Markov Chain Monte Carlo. In fact, it is sometimes known as *Carter-Kohn*, after the algorithm described in Carter & Kohn (1994) as a step in a Gibbs sampler. Conditional on data and other parameters, it gives a draw for the states—using those, the next step would be to produce a draw for the other parameters conditional on the states. However, it also has value in other situations, if we're interested in *any* information about the states beyond their mean and variance.

As a simple example, we'll work with the Nile data again (Example 8.1). What we'll do is slightly different from illustration 2.6.1 in DK. Since we now know how to handle the initial guess values, we'll use the `PRESAMPLE=DIFFUSE` option. However, we can't do that (from the start of data) with straight simulation since there isn't a finite variance from which to draw. So for that, we will still use a large finite value. The smoothed and simulated states are generated by:

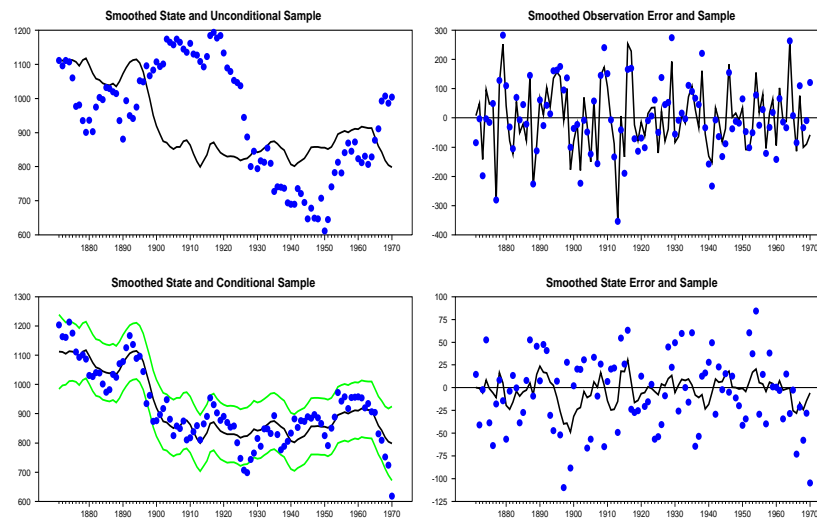


Figure 8.1: Simulated Data

```
d1m(a=1.0,c=1.0,presample=diffuse,sv=15099.0,sw=1469.1,$
    type=smooth,y=nile,vhat=vhat,what=what) / xstates vstates
d1m(a=1.0,c=1.0,sx0=1.e+7,sv=15099.0,sw=1469.1,$
    type=simulate,yhat=nilehat) / xstates_s
d1m(a=1.0,c=1.0,presample=diffuse,sv=15099.0,sw=1469.1,$
    type=csimulate,y=nile,vhat=vdraws,what=wdraws) / xstates_c
```

In the top left pane of Figure 8.1, you can see that the unconditionally simulated series (shown with dots) shows no real relationship to the smoothed estimate other than right at the start, and that's only because we shifted the series to start at the same location. Compare that with the conditionally simulated data in the bottom left. There, you'll notice that, while the upper and lower bounds for the smoothed data are roughly parallel, a "typical" conditionally simulated series doesn't *itself* parallel those. In other words, the mean and bounds aren't showing a typical "shape", with the only uncertainty being the level. Instead, the conditionally simulated series can meander quite a bit (largely) inside those limits. They still tend to be fairly smooth, but nowhere near as smooth as one would think from looking at just the smoothed series and the bounds.

8.2 Gibbs Sampling

State-space models can be very conveniently analyzed using Gibbs sampling techniques (Appendix F). This is done by treating the unobservable components of the model (the states, the observation errors and the state shocks) as additional parameters and using conditional simulation to "sample" them as a block given the more standard parameters of the model. Once that is done, those unobservables are now "data", which often allows inference on standard

parameters using well-known methods.¹

8.2.1 Local Level Model

In the local level model

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

the two free parameters are the two variances, σ_η^2 and σ_ε^2 . Conditional on those two (and the observed data y), the x , η and ε can be jointly sampled (assuming independent Normal shocks) using conditional simulation. With η and ε treated as *known*, we would have simply

$$\begin{aligned}\varepsilon_t &\sim N(0, \sigma_\varepsilon^2) \\ \eta_t &\sim N(0, \sigma_\eta^2)\end{aligned}$$

to do inference on σ_ε^2 and σ_η^2 , which is the well-understood problem of estimating variances given sample data. Once those are sampled, a sweep is complete and we return to the conditional simulation of the state-space model.

This is done in Example 8.2. While we have shown several different ways to parameterize variances (log, standard deviations and variances directly), for Gibbs sampling they are always done directly in variance form, since the sampling procedures will enforce non-negativity. For inference on σ_ε^2 , conditional on the full set of ε_t ,

$$f(\varepsilon | \sigma_\varepsilon^2) \propto (\sigma_\varepsilon^2)^{-T/2} \exp\left(-\frac{\sum \varepsilon_t^2}{2\sigma_\varepsilon^2}\right) \quad (8.1)$$

If we look at this as a density for σ_ε^2 , it's a scaled inverse chi-squared (Appendix A.7). The standard “non-informative” (Jeffrey’s) prior for this is

$$f(\sigma_\varepsilon^2) \propto (\sigma_\varepsilon^2)^{-1/2} \quad (8.2)$$

which is a inverse chi-squared with 0 degrees of freedom.² Using Bayes’ rule to combine (8.1) and (8.2) into a density for σ_ε^2 , we get that it’s a scaled inverse chi-squared with T degrees of freedom. We can draw a value for that taking $\sum \varepsilon_t^2$ divided by a draw for a chi-squared with T degrees of freedom, which gives draws roughly centered on $\sum \varepsilon_t^2 / T$.

Because we are assuming that η and ε are independent, we do the analogous calculation to sample σ_η^2 .

If we were doing inference on the error variance from a regression model, this would probably work just fine. In state-space models, we have to be a bit more

¹If you want to learn more about Gibbs Sampling and related techniques, we would recommend that you get the *Bayesian Econometrics* e-course.

²Yes, zero. It’s improper.

careful. As we've seen, with more shocks than observables it's possible for the likelihood maximizer to be at a zero value for one or more variances. (By contrast, a regression model has just the one variance). Suppose that we happen to sample a value of σ_η^2 that's near zero. When we then do the conditional simulation of the state-space model given that, basically by construction the next set of values for η will have to be near zero, hence their sum of squares will be near zero, and the next draw for σ_η^2 will probably be even *closer* to zero, etc. In the terminology of Markov Chains, the zero variance is an *absorbing state* (page 288). To prevent this, it's necessary to have at least a weakly informative prior. If the prior is a scaled inverse chi-squared with ν degrees of freedom and scale τ^2 , that's like adding ν artificial data points with sum of squares $\nu\tau^2$. Because the sum of squares is a fixed non-zero value, it prevents the numerator of the draw from collapsing to zero.

Now an informative prior for a variance needs to have *some* idea of what the proper scale (τ^2) is. Ideally, you want that to adapt to the data using some type of preliminary analysis—too often, we've seen someone copy values from another person's sampler that were correct for the original data, but incorrect for the new data. Suppose that you take a value of τ^2 that was chosen for a set of data that had been multiplied by 100 and apply it instead to unscaled data. The sum of squares in the unscaled data will be on the order of .0001 that of the scaled data. Thus, a prior intended to be weakly informative actually is likely to dominate the data. In Example 8.2, to center the priors we will use as the variances the guess values we used for maximum likelihood estimation. Both use just one degree of freedom, which should be enough if the goal is to prevent the sampler from collapsing to an absorbing state.

```
filter(type=centered,width=11,extend=repeat) nile / fnile
sstats(mean) / (nile-fnile)^2>>initsigsq
*
compute sigsqeps=initsigsq,sigsqeta=sigsqeps*.01
*
* Prior for eps
*
compute s2eps=sigsqeps,nueps=1.0
*
* Prior for eta
*
compute s2eta=sigsqeta,nueta=1.0
```

For Gibbs sampling, you need a “burn-in” set of draws during which you expect (hope?) that the chain will converge away from the guess values. How many will be needed will depend upon quite a few things—the shape of the likelihood, the design of the sampler, the reasonableness of the guess values. How many draws you want or need depends upon the accuracy that you desire versus the amount of time you want to allow. What level of accuracy you can get will depend upon both the number of draws (in general, quadrupling the draws

doubles the accuracy) and on the “forgetfulness” of the chain. The first of these is easy to control, the latter may not be—it will depend upon whether there’s any good alternative to what you’ve done. The amount of time to process draws tends to be linear in the total number of draws. The amount of time for processing the *output* of the chain can often go up much quicker than that—if it involves sorting or calculation of sample percentiles, it’s quite possible to take longer to convert draws into usable output than it did to draw them in the first place.

```
compute nburn =1000
compute ndraws=25000
```

In this case, we’re doing 1000 burn-in draws and 25000 saved ones. 1000 should be enough burn-ins for this chain since

- it has just two blocks
- all the sampling is done from the actual conditionals (that is, it’s not Metropolis-Hastings)
- the conditional simulation lets us sample the whole batch of shock “parameters” together

In fact, even 100 may be enough,³ but you’ll probably never be able to publish a paper with 100 burn-in to 25000 saved draws. In general, there’s no harm to additional burn-in’s other than the time it takes, which here is minimal.

One very important piece of advice: *never* start with a “production” number of draws. Start with 100 of each, or even 10 of each, and make sure the sampler is doing what you want. (Use **PRINT** options, **DISPLAY** instructions, etc.) We’ve had users waste literally weeks trying to interpret results from a chain that didn’t work. Once it appears to be producing a reasonable sequence of draws, take the “debugging” instructions and options out and increase the number of draws.

You also have to decide what information that you would like to keep from the sampler. What we’ll do here is to save all the draws for the two variances. We won’t do anything with the states themselves—in this case, there isn’t much that they can tell us that we couldn’t get from Kalman smoothing.

Here, we’ll save these draws into two separate series, which are initialized with:

```
set sigepsd 1 ndraws = 0.0
set sigetad 1 ndraws = 0.0
```

The general loop structure is similar for almost any Gibbs sampling:⁴

³If you start with the boundary value of SIGSQETA=0, so only the weak prior is allowing σ_η^2 to move, by the time to get 100 draws in, the effect of the starting value has dissipated.

⁴Version 9.2 added the *Help—Pseudo-code Generator—Monte Carlo/Gibbs Loop* operation which creates this code block for you.


```

infobox(action=define,progress,lower=-nburn,upper=ndraws) $
  "Gibbs Sampling"
do draw=-nburn,ndraws
  ...do the draws
  infobox(current=draw)
  if draw>0 {
    ...save what you want
  }
end do draw
infobox(action=remove)

```

The **INFOBOX** lets you know about the progress of the sampler (with a guess as to time to completion, so you can stop it and adjust the draws if it will take too long). This isn't necessary when you start with the recommendation of maybe 100 draws, but it's probably not a bad idea to get in the habit of putting it in right at the start. Note that this loop actually discards `NBURN+1` draws (since 0 is also skipped), but it's easier to write it this way.

The “do the draws...” part consists of the conditional simulation, which includes the **WHAT** and **VHAT** options to fetch the simulated state shocks (**WHAT**) and measurement equation errors (**VHAT**). Each of these produces a **SERIES [VECT]** (since each can have more than one component), so we'll need to use **ETAHAT (T) (1)** and **EPSHAT (T) (1)** to get entry **T** of the two errors.

```

dlm(a=1.0,c=1.0,y=nile,sv=sigsqeps,sw=sigsqeta,presample=diffuse,$
  type=csimulate,what=etahat,vhat=epshat)

```

The draw for σ_η^2 is done with:

```

sstats / etahat(t)(1)^2>>sumsqeta
compute sigsqeta=(sumsqeta+nueta*s2eta)/%ranchisqr(%nobs+nueta)

```

using **SSTATS** to sum up the squared values of **ETAHAT**, then drawing from the posterior scaled inverse chi-squared. A similar calculation is done for σ_ε^2 :

```

sstats / epshat(t)(1)^2>>sumsqeps
compute sigsqeps=(sumsqeps+nueps*s2eps)/%ranchisqr(%nobs+nueps)

```

The “save what you want...” is

```

compute sigepsd(draw)=sigsqeps
compute sigetad(draw)=sigsqeta

```

Outside the loop, we compute empirical densities of the draws for the variances with ⁵

⁵These were both done initially with the *(Kernel) Density Estimation* wizard, with some minor adjustments to improve the appearance.

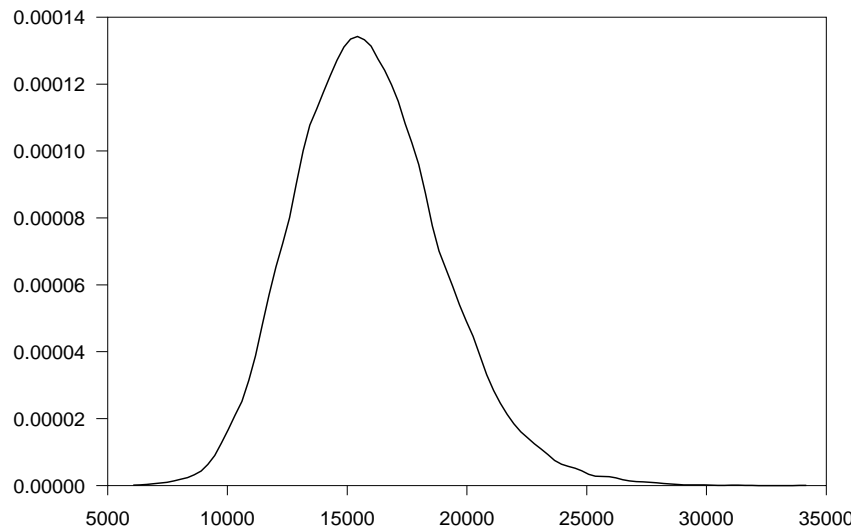


Figure 8.2: MCMC Density of σ_{ε}^2

```
density(smoothing=1.5,grid=automatic) sigepsd / xeps feps
scatter(style=line,vmin=0.0,footer="MCMC Density of SIGEPSSQ",
# xeps feps
```

and

```
density(smoothing=1.5,grid=automatic) sigetad / xeta feta
scatter(style=line,vmin=0.0,footer="MCMC Density of SIGETASQ",$
  smpl=xeta<=6000.0)
# xeta feta
```

The density graph for σ_{ε}^2 looks fairly routine (Figure 8.2). The maximum likelihood estimate is 15098 with a standard error of 3092, and this matches up quite nicely with a Normal distribution with those parameters.

The density for σ_{η}^2 is a whole different matter (Figure 8.3). The maximum likelihood estimate is 1469 with a standard error of 1260. The marginal mode, however, is more like 750.⁶ The shape is because the likelihood is *extremely* flat in σ_{η}^2 —if the asymptotic distribution is correct, the log likelihood is only lower by .5 way down at 209 (one standard error below the likelihood-maximizing 1469.) The log likelihood actually drops a *bit* quicker than that, but the variances in the 600 to 700 range *do* fit almost as well as 1469. So even though we have a very weak prior, the data evidence is even weaker. Even the Jeffrey's prior (which, by the way, can be done with a ν value of 0) would penalize 1500 vs 750 by .5 log 2 or .347 and the log likelihood loss in moving to 750 is less than that. Thus the posterior is shifted well towards zero.

⁶The mean, however, is still somewhere in the 1400 range because of the very long right tail of the distribution.

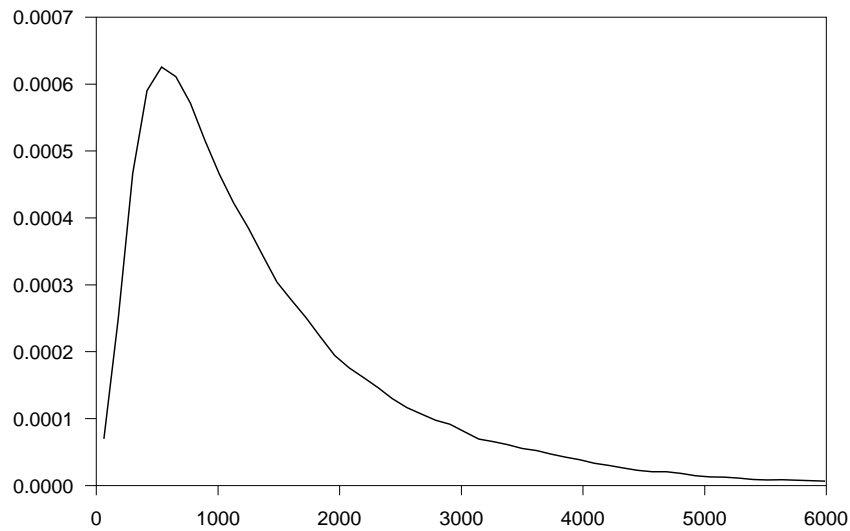


Figure 8.3: MCMC Density of σ_{η}^2

8.2.2 Time-varying Coefficients

It’s possible to analyze the time-varying coefficients model from Section 5.4 using Gibbs sampling. However, it’s important to note that Gibbs sampling (with the standard blocking) is inherently a “smoothing” operation, not a filter, so Gibbs sampling will be primarily useful for inference on the underlying “hyperparameters” and end-of-sample coefficients—see page 87.

The Gibbs Sampling equivalent of Example 5.7 would have independent scaled inverse chi-squared priors for each of the drift variances. As discussed on page 151, these need to be at least mildly informative to prevent a zero absorbing state. An interesting question then is what to use as the scales of those priors. As we’ve seen before with TVC models, the fact that the coefficients are sensitive to the scales of both the dependent and explanatory variables and to the overall set of variables included makes it very unlikely that there will be a reasonable way to come up with fully *a priori* scale factors. Instead, these are typically created from the output from OLS regressions—we did that in Example 5.7 for *guess values*. But guess values don’t change the likelihood in the earlier example, while the variance scales *do* effect the posterior in the Gibbs sampler, so we need to pay attention to how the prior interacts with the data.

In Example 8.3, we use the full sample estimates to get scaling values for the priors. This is the same “money growth” model as in Example 5.7, and, again, we’ll focus on the behavior of the coefficient on lagged inflation.

```

linreg mlgr 1962:1 *
# constant dintlag inflag surplag mllag
*
compute sstart=%regstart(), send=%regend()
compute nbeta=%nreg
*
dec vect swprior(nbeta) swdraw(nbeta)
dec real svprior svdraw

```

This is designed to be easily adapted to other regressions, with `NBETA` being made equal to the number of regressors. `SSTART` and `SEND` are the full sample range.

We have (in this case) six hyperparameters, each of which needs a prior and a current “draw value”. We group these into the (scalar) equation variance and the `VECTOR` of drift variances. The prior that we’ll use on the equation variance has a one degree of freedom and a scale of half the variance from the fixed coefficient model. This is unlikely to have much effect at all on the estimates, since that’s probably a reasonable guess for the scale (and the degrees of freedom are small, so it won’t shift results much anyway).

```

compute svprior=.5*%seesq
compute svdraw =svprior
compute svnu    =1.0

```

The drift variances are a whole different matter. Even with the help of a fixed coefficients regression, there are no obvious choices for the prior. What you might choose may depend quite a bit on what the goal of the analysis is. If you merely want to allow for a modest amount of drift, then a prior with a smallish scale factor and a fairly high degrees of freedom value will probably work to that end. If you are interested in examining how much “break” there seems to be in the specification, you would want to use a much lower degrees of freedom so the prior won’t overwhelm the data evidence. We’ll adopt the second approach. The scale is .01 times the full sample variance estimate for each coefficient, with one degree of freedom.⁷

```

compute swprior=.01*%stderrs.^2
compute swdraw =swprior
compute swnu    =1.0

```

With roughly 100 data points, the .01 multiplier means that the prior would allow for roughly one standard error of drift over the course of the sample. That’s not very much. If we were planning on using a prior with a much higher degrees of freedom, we might want to rethink that. With a low degrees of

⁷There is no specific reason for using a common scaling of the regression variance, or a common degrees of freedom for each component of the drift variance, except that it’s clearly simpler to set up, and avoids “overhyperparameterizing”.

freedom count, we are “suggesting” that relatively modest drift is more likely than substantial drift, but are allowing for both.

This sets the burn-in and kept draws. Again, note that you shouldn’t *start* with a production number like this—make sure the sampler is working before going to production quantities (though this number takes only about a minute).

```
compute nburn=10000
compute ndraws=25000
```

These are for keeping track of the sum and sum of squared coefficients for computing the mean and error bands for the time-varying coefficients:

```
dec vect[series] sumbeta(nbeta) sumsqbeta(nbeta)
clear(zeros) sumbeta sumsqbeta
```

and this is for keeping track of the draws for the variance hyperparameters:

```
dec series[vect] sigdraws
nonlin(parmset=sigsqs) swdraw svdraw
gset sigdraws 1 ndraws = %parmspeek(sigsqs)
```

This will save all the sampled values for those, which will allow us to do diagnostics on the Markov Chain, and will also allow us to look at density functions—since these are variance parameters (and thus strictly positive and probably having a highly non-normal distribution), saving only the first two moments (as we are doing with the regression coefficients) won’t provide enough descriptive information. The `PARMSET` won’t actually be used directly in estimation; instead, it is a simple way to organize the set of parameters that we want to save.

We have the standard Gibbs draw loop, with pseudo-code:

```
infobox(action=define,progress,lower=-nburn,upper=ndraws) $
  "Gibbs Sampling"
do draw=-nburn,ndraws
  ... draw parameters ...
  infobox(current=draw)
  if draw>0
    ... save information ...
  end do draw
infobox(action=remove)
```

The first step inside for drawing the parameters is to do conditional simulation. This draws a set of time-varying coefficients (the series of states), the coefficient drift shocks (the `WHAT`) and equation shocks (the `VHAT`) given the model and the current values for the variance parameters:

```
d1m(y=m1gr,c=%eqnxvector(mdeq,t),sw=%diag(swdraw),sv=svdraw,$
presample=diffuse,type=csimulate,what=what,vhat=vhat) $
sstart send xstates
```

As before, we draw the equation variance given $VHAT$'s:

```
sstats sstart send vhat(t)(1)^2>>rssv
compute svdraw=(rssv+svnu*svprior)/%ranchisqr(%nobs+svnu)
```

The drift variances can be drawn in a loop (thus making it easy to change the size of the model) with:

```
do i=1,nbeta
  sstats sstart send what(t)(i)^2>>rssw
  compute swdraw(i)=(rssw+swnu*swprior(i))/%ranchisqr(%nobs+swnu)
end do i
```

When it's time to save the information, we add the components of the simulated states and their squares to make the running totals for computing the first and second moments:

```
do i=1,nbeta
  set sumbeta(i) sstart send = sumbeta(i)+xstates(t)(i)
  set sumsqbeta(i) sstart send = sumsqbeta(i)+xstates(t)(i)^2
end do i
```

and save the current set of sampled variances using `%PARMSPEEK`:

```
compute sigdraws(draw)=%parmspeek(sigsqs)
```

After the loop is done, we compute the mean and standard deviations of the time-varying coefficients, and use them to make upper and lower (one standard deviation) bounds:

```
dec vect[series] mean(nbeta) upper(nbeta) lower(nbeta)
do i=1,nbeta
  set sumbeta(i) = sumbeta(i)/ndraws
  set sumsqbeta(i) = sqrt(sumsqbeta(i)/ndraws-sumbeta(i)^2)
  set mean(i) = sumbeta(i)
  set upper(i) = sumbeta(i)+sumsqbeta(i)
  set lower(i) = sumbeta(i)-sumsqbeta(i)
end do i
```

In this case, we want to focus on the inflation coefficient, so we graph that (Figure 8.4) with:

```
graph(footer="Inflation Coefficient") 3
# mean(3) sstart send
# upper(3) sstart send 2
# lower(3) sstart send 2
```

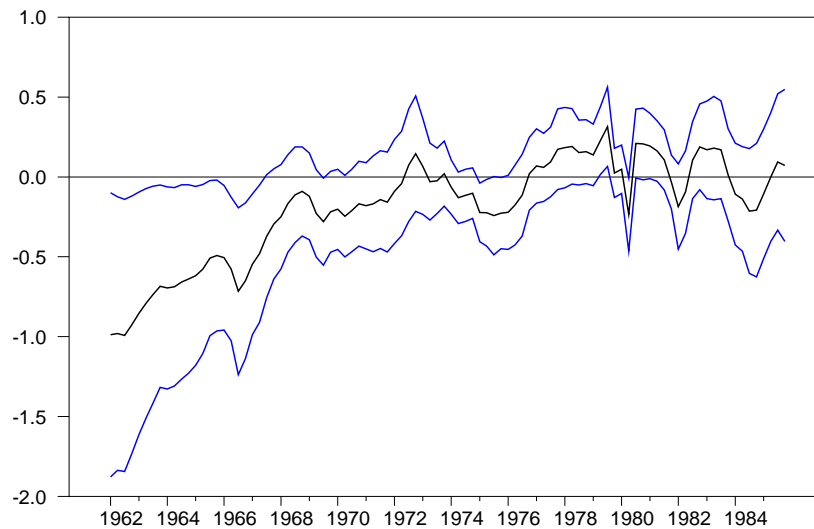


Figure 8.4: Inflation Coefficient with Independent Drift Variances

Not surprisingly, this is similar to what we see when we did the smoothed estimates with the maximum likelihood estimates (Figure 5.16). We can look more closely (Figure 8.5) at the density for the drift variance on that coefficient with

```
set tvinfl 1 ndraws = sigdraws(t) (3)
density(smoothing=1.5,grid=auto,maxgrid=400) tvinfl / xinfl finfl
scatter(style=line,vmin=0.0,$
  footer="Posterior Density of Inflation Drift Variance")
# xinfl finfl
```

The first line in this pulls the “sample” for the third drift variance out of the saved history of the coefficients.⁸ The remainder is a lightly edited version of what is produced with the *(Kernel) Density Estimation* wizard.

Note that this is multi-modal, with a broad mode roughly consistent with the maximum likelihood estimates (Table 5.14),⁹ and a spike near the prior. What are we to make of this? Given how “loose” the prior was, it’s a bit hard to imagine why it would do anything more than just shift the mode over a bit towards zero. However, the point estimates show a great deal more certainty about the large drift variance for this coefficient than is actually warranted. A standard deviation of .05, which is almost 4.5 estimated standard deviations from the point estimate and thus supposedly well out in the tails, would generate a likelihood ratio statistic of only 3.14, which isn’t significant at even the

⁸The PARMSET that was “peeked” to fill this in was structured with the drift variances first and the equation variance last. So if we wanted to do the same with the equation variance, we would use SIGDRAWS(T) (6), or, more generally, SIGDRAWS(T) (NBETA+1).

⁹Table 5.14 shows estimates of standard deviations while Figure 8.5 is variances. Converting the point estimates to variances would give a coefficient of .069 with a standard error of roughly .025.

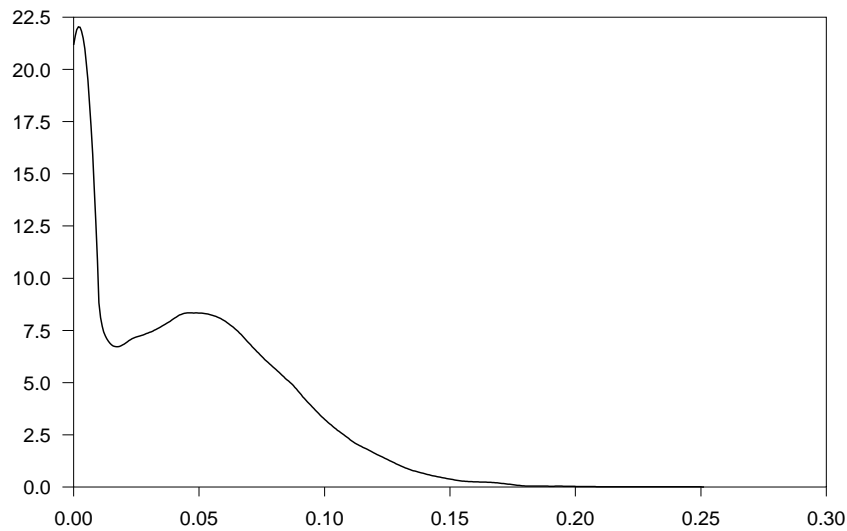


Figure 8.5: Posterior Density of Inflation Drift Variance

.05 level.¹⁰ Thus the data evidence isn't strong enough to overwhelm the prior. Even a modestly tighter prior will dampen down the high variation in the coefficients. If we graph the draws for the inflation drift variance (Figure 8.6), you can see what a (near) absorbing state looks like, with the variance hitting near zero around draw 5000 for over 2500 consecutive draws and having a few other shorter spots where it sticks near zero. This is *not* how a well-behaved Gibbs sampler would look—if we wanted to do a serious analysis of this model, we would need to use a more complicated sampling scheme that would make it possible to move more easily between the two modes.

TVC with empirical drift variance

We can also apply the same ideas from Example 5.8 of using the empirical covariance matrix from a subsample of the regression as the basis for the prior for the drifts. This is commonly used in larger models such as VAR's, where there are simply too many coefficients to handle with independently chosen drift variances.

The prior now will be an inverse Wishart (Appendix A.11). Note that with an inverse Wishart, the prior is only proper if the degrees of freedom is greater than $n - 1$ (where n is the size, which here is the number of regressors).

Example 8.4 uses the idea of a “training sample” at the start of the data set to create information to be used in forming the prior:

¹⁰You can compute this by pegging the value of `SIGMAV(3)` in Example 5.7.

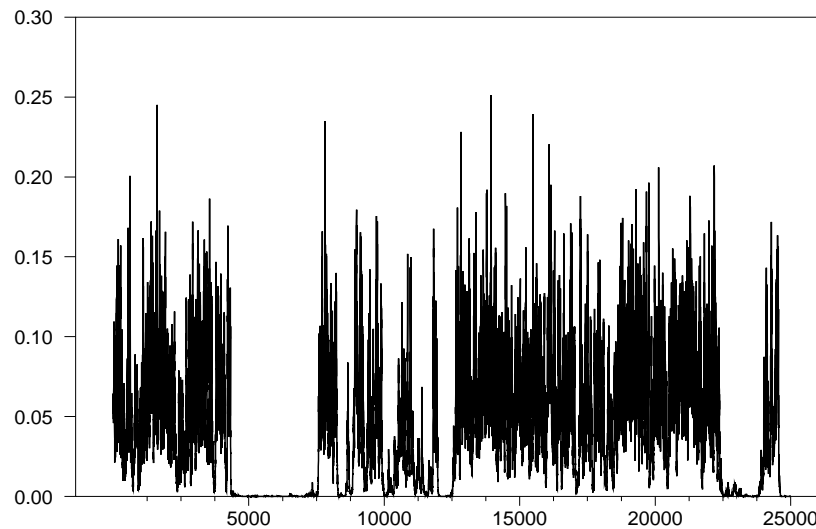


Figure 8.6: Draws for Inflation Drift Variance

```
linreg mlgr 1962:1 1971:4
# constant dintlag inflag surplag mllag
*
* Start and end of the "working" sample
*
compute sstart=%regend()+1,send=1985:4
compute nbeta=%nreg
equation(lastreg) mdeq
```

The working sample (marked by `SSTART` and `SEND`) starts the entry after the 10 year (40 quarter) training sample and continues to the end of the sample. The pre-sample mean for the coefficients is the training sample estimate and the pre-sample covariance is 4 times the training sample estimate of the covariance matrix:

```
compute covmat=%seesq*%xx
*
* Pre-sample mean/covariance matrix for coefficients
*
compute x0 =%beta
compute sx0=4.0*covmat
```

This sets up the prior for the coefficient drift. This uses a scaled version of the training sample covariance matrix as the center of the distribution. Note that, by convention, the inverse Wishart is typically parameterized using the centering matrix times the degrees of freedom. However, we write this more naturally as the centering matrix itself and multiply by the degrees of freedom later. The `KQ` value (squared) determines the level of scaling—smaller gives a lower expected drift. The .1 value for `KQ` allows for only a relatively modest amount of drift—over 100 entries, the expected drift is on the order of the

training sample covariance. The degrees of freedom (`SWNU`) are here set to 10 which is fairly loose (anything 5 or below is improper).

```
compute kq=.1
compute swprior=kq^2*covmat
compute swnu=10.
compute swdraw=swprior
```

This is the prior for the equation variance (which, as usual, is a scaled inverse chi-squared):

```
compute svprior=%seesq
compute svnu=10.
compute svdraw=svprior
```

While in this case we also used the same number of degrees of freedom for both the priors, there's no reason to do that in general. While it's *possible* for excessive variation in the coefficients to push the equation errors down towards zero, that's unlikely, so a relatively loose prior on this (fewer degrees of freedom than this) won't change the results much.

This sets up the various arrays for saving information. This is set to keep track of a great deal, probably more than you would likely use in practice. The obvious ones are `SUMBETA` and `SUMSQBETA` for keeping track of the sums and sums of squares of the individual coefficients over time so we can generate means and variances of them. Each coefficient will have one element in the `VECT[SERIES]`. The `SWDRAWS` and the `SVDRAWS` are keeping track of the entire set of draws of the variance of the coefficients (just the variances, not the off-diagonal elements) and equation variances.

```
dec vect[series] sumbeta(nbeta) sumsqbeta(nbeta)
clear(zeros) sumbeta sumsqbeta
dec vect[series] swdraws(nbeta)
do i=1,nbeta
  set swdraws(i) 1 ndraws = 0.0
end do i
set svdraws 1 ndraws = 0.0
```

We have the usual Gibbs sampling loop. Inside it, we first draw the time-varying coefficients given the current variances:

```
dlim(y=mlgr,c=%eqnxvector(mdeq,t),sw=swdraw,sv=svdraw,$
  x0=x0,sx0=sx0,type=csimulate,what=what,vhat=vhat) $
  sstart send xstates
```

The coefficients themselves are in the `XSTATES`, the draws for the equation errors are in `VHAT` and the coefficient shocks are in `WHAT`.

The draw for the equation variance is as before:

```

sstats sstart send vhat (t) (1)^2>>rssv
compute svdraw=(rssv+svnu*svprior)/%ranchisqr(%nobs+svnu)

```

The draw for the drift variance is, if you look at the details, basically the same thing applied to an $n \times n$ symmetric matrix. The “sums of squares” is accumulated into the matrix `RSSW`. This includes the contribution of the prior, which is the degrees of freedom times our prior belief as to the centering constant. The draw is then generated from an inverse Wishart.

```

compute rssw=swnu*swprior
do t=sstart, send
  compute rssw=rssw+%outerxx(what(t))
end do t
*
compute swdraw=%ranwisharti(%decomp(inv(rssw)), %nobs+swnu)

```

The bookkeeping on positive values of `DRAW` is done as:

```

if (draw>0) {
  do i=1,nbeta
    set sumbeta(i) sstart send = sumbeta(i)+xstates(t)(i)
    set sumsqbeta(i) sstart send = sumsqbeta(i)+xstates(t)(i)^2
  end do i
  compute %pt(swdraws, draw, %xdiag(swdraw))
  compute svdraws(draw)=svdraw
}

```

which accumulates the coefficients and their squares (in the `DO I` loop), and saves draws for the drift variances (diagonal of `SWDRAW`) and for the equation variance.

8.2.3 Models with Regressions/Autoregressions

In the previous models, the only underlying parameters were variances, which could be handled using a relatively straightforward sampling scheme. Models where some of the parameters are regression coefficients or autoregressive (or ARMA) parameters can be quite a bit more complicated.

The simplest case is where a measurement equation looks like a standard regression, but where some of the regressors are state variables. For instance, if, as in Section 6.4,

$$dw_t = \mu_w + \beta_0 C_t + \beta_1 C_{t-1} + \gamma_1 Z_t + \varepsilon_{wt} \quad (8.3)$$

where C is a state variable and Z is a known exogenous variable; given C , this is just a standard linear regression with (one assumes) Normal errors and the regression parameters (μ_w , β_0 , β_1 and γ_1) can be drawn using standard methods (assuming you’re using a Normal prior). The C ’s themselves will be drawn by

simulating the complete model given all the underlying parameters. The same idea would be applied to a VAR if it has the standard VAR form.

Autoregressions (or ARMA models) or autoregressive error processes are substantially more complicated. First off, a “flat” prior is no longer likely to be reasonable. Second, the likelihood function for an AR or ARMA process doesn’t admit a simple sampling scheme like a linear regression.

We’ll base Example 8.5 on an example from Brockwell & Davis (2002). This estimates a regression of the level of Lake Huron on a time trend, with AR(2) errors:

$$\begin{aligned} L_t &= \alpha + \beta t + Z_t \\ Z_t &= \rho_1 Z_{t-1} + \rho_2 Z_{t-2} + \varepsilon_t \end{aligned}$$

Maximum likelihood point estimates can be computed using **BOXJENK** with the GLS option:

```
boxjenk(ar=2,reg,maxl) lake
# constant trend
```

If we write this in state-space form, we have

$$\begin{aligned} L_t &= \{\alpha + \beta t\} + [1]Z_t \\ \begin{bmatrix} Z_t \\ Z_{t-1} \end{bmatrix} &= \begin{bmatrix} \rho_1 & \rho_2 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} Z_{t-1} \\ Z_{t-2} \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} \varepsilon_t \end{aligned}$$

The regression term in braces will be handled using a MU option. The **C** matrix is $[1, 0]$. The only error is in the state equation, so **SV** will be 0 (and thus won’t be needed).

We can put this into state-space form starting with:

```
linreg(define=regeqn) lake / u
# constant trend
frml(lastreg,parmset=regparms,vector=b) mufrml
```

which sets up the **FRML** for the regression part, creating the **VECTOR** for the parameters (called **B**) and creating the **PARMSET** **REGPARMS** to keep track of that.

This runs an AR(2) on the residuals to initialize the parameters for the error process, and sets up a second **PARMSET** for the AR coefficients and the equation variance.

```
dec vect rho(2)
nonlin(parmset=arparms) rho sigsq
linreg u
# u{1 2}
compute rho=%beta,sigsq=%sigmasq
```

Table 8.1: Lake Level Model Estimates

DLM - Estimation by BFGS

Convergence in 10 Iterations. Final criterion was 0.0000044 <= 0.0000100

Annual Data From 1875:01 To 1972:01

Usable Observations 98

Rank of Observables 98

Log Likelihood -101.1983

	Variable	Coeff	Std Error	T-Stat	Signif
1.	RHO(1)	1.0048	0.0960	10.4670	0.0000
2.	RHO(2)	-0.2913	0.0989	-2.9466	0.0032
3.	SIGSQ	0.4566	0.0655	6.9715	0.0000
4.	B(1)=Constant	10.0915	0.4632	21.7870	0.0000
5.	B(2)=TREND	-0.0216	0.0081	-2.6616	0.0078

and this sets up the `FRML[RECT]` for the `A` matrix (which needs a `FRML` since it depends upon the unknown AR coefficients), the fixed `F` and `C` matrices. `NZ` is made equal to the number of states in the noise model and `NZPARMS` is the number of free parameters in the noise model, which are both 2 here, but might not always be the same size. The model is then estimated (output in Table 8.1):

```
dec frml[rect] af
frml af = ||rho(1),rho(2)||
          1.0 , 0.0 ||
compute f=%unitv(2,1)
compute c=%unitv(2,1)
compute nz=2
compute nzparms=2
dlm(y=lake,a=af,mu=mufrml,f=f,c=c,sw=sigsq,$
    parmset=arparms+regparms,presample=ergodic,method=bfgs)
```

Needless to say, if all you need are the point estimates, use **BOXJENK** instead.

To do Gibbs sampling, we need priors for the regression coefficients, for the AR parameters and for the variance. The priors for those three blocks would typically be done independently, but as we'll see, treating the AR parameters and the variance separately may not be the best idea.

The prior for the regression coefficients will be Gaussian with zero means for both. The precision on the intercept is zero (that is, we use a “flat” prior), while we use a standard deviation of .25 (precision 16) for the trend.

```
compute [vector] bbprior=||0.0,0.0||
compute [symm]   bhprior=%diag(||0.0,1.0/.25^2||)
```

The prior on the trend would depend upon the units of measurement of the lake level. Here, it's feet on a gauge, so we aren't guessing whether the trend is up or down, but we think it unlikely it's as much as .5 feet (15 cm) up or down

per year. If it were measured in centimeters, we would multiply the standard deviation by 30 and the precision would be $1.0/7.5^2$.

The prior for the AR coefficients isn't as simple. An AR(1) process isn't so difficult—the coefficient has to be in $(-1,1)$, probably is positive in this and most situations, and often (particularly in economic data) is above .5. A beta prior (Appendix A.3) can take care of all of those at once; for instance, a beta with parameters 1.5 and .5 will have mean .75 and a standard deviation of .25, and so will be a vague “positive, more likely large than small”. An AR(2) is harder because it could have either two real roots or two complex roots and those different types of processes have rather different behavior and parameters. If you want to restrict yourself to strictly real roots, you can put priors on the two roots; if you want to restrict yourself to strictly complex roots, you can put priors on the cycle and amplitude of the complex root pair. But either of those restricts the set of processes that can be generated.

In this case, we'll use a Gaussian prior on the AR(2) centered on $Z_t = .8Z_{t-1} + 0Z_{t-2}$ with a precision matrix of

$$\begin{bmatrix} 40 & -20 \\ -20 & 20 \end{bmatrix}$$

truncated to the stationary sets of parameters. This has a standard deviation of roughly .15 on the first lag, a bit larger (roughly .22) on the second, and the two are negatively correlated (which would be the expected behavior if the series is relatively smooth). This allows for either real or complex roots, though two *large* real roots would be nearly impossible because of the zero mean on the second lag. A relatively standard prior with truncation to the stationary zone is a common way of handling autoregressions and VAR's. This has the great advantage that you don't have to describe analytically the stationary zone—all you have to do is draw by standard methods using the standard prior but reject the draw and try again if the generated parameters are non-stationary.

The full sample likelihood for an AR process doesn't have a simple form, so we will use Metropolis within Gibbs to take care of it. It's *possible* to do the conditional likelihood (on the first two observations), which turns the likelihood into that of a standard Gaussian regression and that's relatively standard practice with higher-dimensional VAR's which could easily have 50-100 parameters. Here, we'll show how to handle draws from the likelihood generated by the state-space model for the entire data set. We'll do Independence Chain, which should work relatively well on a two parameter block using the typical choice of (a scale-up of) the asymptotic Normal distribution from the maximum likelihood estimates as the proposal. The following will pull the estimated coefficients and their covariance matrix out of the maximum likelihood estimates for the full model—because we arranged the parameter set to do the `ARPARMS` first, they're just the first block in each matrix. We use the `NZPARMS` value defined earlier to get the correct number of parameters out of this. After a

bit of experimentation, we decided on a scale factor of 1.5 on the factor of the covariance matrix—we'll explain this later.

```
compute [vector] rbprop=%xsubvec(%beta,1,nzparms)
compute [rect]   fbprop=1.5*%decomp(%xsubmat(%xx,1,nzparms,1,nzparms))
```

We'll use an inverse chi-squared prior with one degree of freedom for the variance. Note that this is the variance of the innovation in the AR(2) process so our information about it may be rather weak—even if we had fairly good information about the variance of the Z process, the variance on the innovation will depend strongly on the persistence of the AR coefficients.

```
compute s2eps=0.5^2
compute nueps=1.0
```

We're setting this for 1000 burn-in and 10000 saved draws. We will again emphasize the usual advice to start with a smaller number on a new project.

```
compute nburn=1000
compute ndraw=10000
```

We'll save the full sets of regression and full sets of AR coefficients, in separate batches:

```
dec series[vect] regdraws
dec series[vect] ardraws
*
gset regdraws 1 ndraw = %parmspeek(regparms)
gset ardraws 1 ndraw = %parmspeek(arparms)
```

We're now ready to start a Gibbs sweep. One additional item is that, because we're doing Metropolis draws, we'll keep track of the number of acceptances (in the `ACCEPT` variable):

```
infobox(action=define,progress,lower=-nburn,upper=ndraw) $
  "Gibbs sampling"
compute accept=0
do draw=-nburn,ndraw
```

Given the AR process, the regression coefficients have a Gaussian GLS likelihood. Because the GLS coefficients change from sweep to sweep, this requires generating the filtered cross-product matrix each time through. The most straightforward way to do this is to embed the regression in a “time-varying” coefficients model with no time-variation:

$$y_t = X_t \beta_t + Z_t$$

$$\beta_t = \beta_{t-1} + 0$$

Z can be represented by any univariate state-space model (a UC, AR, ARMA, even ARIMA models with unit roots), and values for β and its covariance matrix at the end of the sample will be the full information GLS estimates and their covariance matrix. This appends to the state-space model for the error process an identity for the \mathbf{A} matrix, an extra row of zeros for \mathbf{F} and the X_t vector for \mathbf{C} . Since everything is known (except the β , which are being treated as a state), this can be done with one pass through the filter using `METHOD=SOLVE`:

```
dlim(y=lake,a=af~\%identity(nreg),f=f~~%zeros(nreg,1),sw=sigsq,$
     c=c~~%eqnxvector(regeqn,t),presample=ergodic,method=solve) / $
     xstates vstates
```

This pulls the mean and precision (inverse of the covariance matrix) for the GLS estimates. Note that the states are arranged with the Z states first, so this pulls out the second block:

```
compute bdata=%xsubvec(xstates(%regend()),nz+1,nz+nreg)
compute hdata=inv(%xsubmat($
                    vstates(%regend()),nz+1,nz+nreg,nz+1,nz+nreg))
```

Combine that with the prior mean and precision to draw a set of coefficients:

```
compute b=%ranmvpost(hdata,bdata,bhprior,bbprior)
```

These are put into the `VECTOR B` which is used as the coefficient vector for the `MUFRML` formula, so that's automatically re-defined. We use that to get the current noise series:

```
set z = lake-mufrml
```

We now use `DLM` to evaluate the log likelihood at the current settings for the parameters. We have to do this since β and σ^2 have changed since the previous time we drew ρ . To that, we add the log of the prior evaluated at the current value of `RHO`—that gives us the log posterior of ρ conditional on the other parameters.

```
dlim(y=z,a=af,f=f,c=c,sw=sigsq,presample=ergodic,method=solve)
compute logplast=%logl+%logdensity(inv(rhprior),rho-rbprior)
```

We now draw a proposal from the distribution chosen above for the ρ vector and reject any that produce a root that's out of bounds. Given how the `%POLYxxx` functions work, we check the *smallest* root of the polynomial and reject if it's smaller than 1. We save the log kernel of the draw into `LOGQTEST`.


```

    compute rholast=rho
:redraw
    compute [vector] rho=rbprop+%ranmvnormal(fbprop)
    if (%polysmallestroot(1.0~-1.0*rho) < 1.0)
        goto redraw
    compute logqtest=%ranlogkernel()

```

We now compute the log posterior with the candidate ρ vector:

```

dlim(y=z,a=af,f=f,c=c,sw=sigsq,presample=ergodic,method=solve)
compute logptest=%logl+%logdensity(inv(rhprior),rho-rbprior)

```

This computes the probability of acceptance for the Metropolis test using the two log posteriors (last and test) and the two log kernels of the draws:

```

compute alpha=exp(logptest-logplast+logqlast-logqtest)

```

This next does the Metropolis test and, if the proposal gets accepted, saves the log kernel into LOGQLAST and increments the ACCEPT value. If the proposal is rejected, it restores the saved value for the RHO vector:

```

if alpha>1.0.or.%ranflip(alpha) {
    compute logqlast=logqtest
    compute accept =accept+1
}
else
    compute rho=rholast

```

We then draw the innovation variance using conditional simulation of the model as we've seen quite a few times now:

```

dlim(y=z,a=af,f=f,c=c,sw=sigsq,presample=ergodic,$
    type=csimulate,what=what,swhat=swhat)
*
sstats / what(t)(1)^2>>sumsqeps
compute sigsq=(sumsqeps+nueps*s2eps)/%ranchisqr(%nobs+nueps)

```

This completes a sweep. The next updates the progress bar, showing the percentage of accepted draws. With Independence Chain Metropolis, you would like that to be *high*—a low value suggests that you need to make at least some change to the proposal distribution, usually to scale up the covariance matrix. In this example, we get around 30% acceptances, which is reasonable. This could probably be improved fairly easily by changing the mean to some average of the original point estimates and the prior mean, which we would do if this were a more serious project. Note that the acceptance rate without the 1.5 scaling on the factor is around 10%, which really *isn't* good for this type of sampler.

```
infobox(current=draw) %strval(100.0*accept/(draw+nburn+1), "##.#")
```

Finally, this saves the information when the `DRAW` count is positive:

```
if draw>0 {
  compute regdraws(draw)=%parmspeek(regparms)
  compute ardraws(draw)=%parmspeek(arparms)
}
```

Since this example is mainly for illustration (and the results end up being largely consistent with the information from the **BOXJENK** estimates), we'll use this to demonstrate different ways to display information from this type of sampler. The simplest is to just pull out a series of draws and do statistics:

```
set trenddraws 1 ndraw = regdraws(t) (2)
stats trenddraws
```

We omit the output from this, but not too surprisingly (given how weak the prior on the trend rate was), it's quite similar to the original estimates.

The following does a graph (Figure 8.7 of the estimated density for the ρ_1 parameter along with its prior:

```
density(smoothing=2.0,grid=automatic,maxgrid=100) rho1draws / $
  xrho1 frho1
set frho1pr 1 100 = exp(%logdensity($
  1.0/rhprior(1,1),xrho1-rbprior(1)))
scatter(footer="Density Estimates for rho1",style=line,vmin=0.0,$
  key=below,klabls=||"Posterior","Prior"||) 2
# xrho1 frho1
# xrho1 frho1pr
```

(The 100 on the **SET** instruction for `FRHO1PR` matches the number of grid points in the **DENSITY** instruction since it's computing the density at the values of the `XRHO1` series). This is clearly a blend of the data (point estimate 1.005) and prior (mean .8).

Finally, this does a bivariate Gaussian kernel estimate of the bivariate density for the pair of AR coefficients and does a contour graph (Figure 8.8). Note that this takes a noticeable amount of time—as we mentioned earlier, sometimes the “post-processing” can take quite a while with a large number of draws.

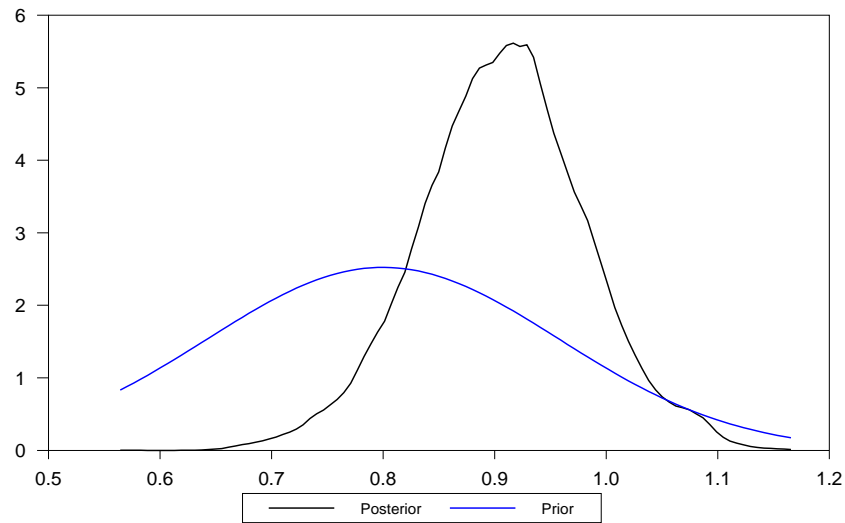


Figure 8.7: Density of ρ_1

```

compute h=.10
dec vect r1grid(101) r2grid(101)
dec rect fgrid(101,101)
compute r1grid=%seqrangle(0.70,1.10,101)
compute r2grid=%seqrangle(-0.4,0.05,101)
compute fgrid=%zeros(101,101)
do draw=1,ndraw
  ewise fgrid(i,j)=fgrid(i,j)+exp(-.5/h^2*($
    (rho1draws(draw)-r1grid(i))^2+(rho2draws(draw)-r2grid(j))^2))
end do draw
gcontour(x=r1grid,y=r2grid,f=fgrid,$
  footer="Density of Draws for AR coefficients")

```

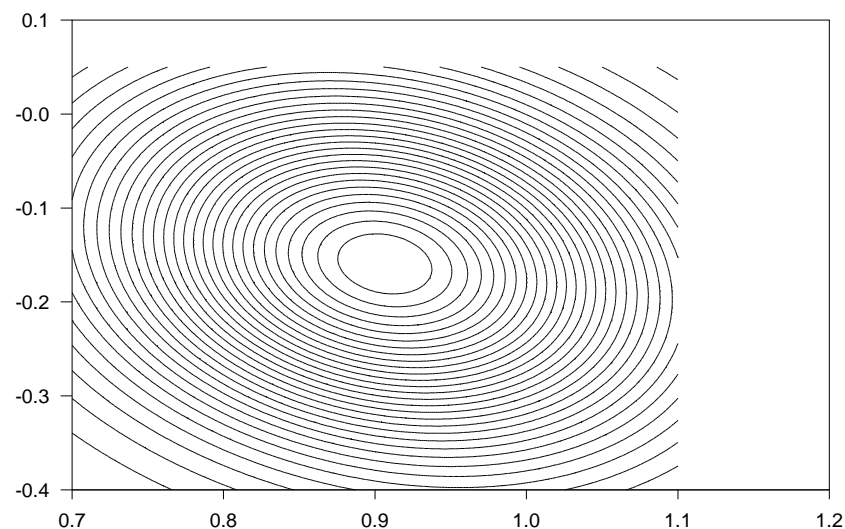


Figure 8.8: Contour Graph of AR coefficients

Example 8.1 Simulations

This demonstrates both unconditional and conditional simulations in the local level model. It's covered at the beginning of Section 8.1.

```

open data Nile.dat
calendar 1871
data(format=free,org=columns,skips=1) 1871:1 1970:1 Nile
*
* Smoothed
*
dlm(a=1.0,c=1.0,presample=diffuse,sv=15099.0,sw=1469.1,$
    type=smooth,y=Nile,vhat=vhat,what=what) / xstates vstates
*
* Unconditionally simulated
*
dlm(a=1.0,c=1.0,sx0=1.e+7,sv=15099.0,sw=1469.1,$
    type=simulate,yhat=Nilehat) / xstates_s
*
* Conditionally simulated
*
dlm(a=1.0,c=1.0,presample=diffuse,sv=15099.0,sw=1469.1,$
    type=csimulate,y=Nile,vhat=vdraws,what=wdraws) / xstates_c
*
* Because the initial state for the unconditional sample is random with
* a huge variance, it wouldn't work to put the direct draw for the
* states on the same graph as the smoothed states. Instead, we peg the
* sampled states to start at the same value as the smoothed states (by
* subtracting off the initial difference).
*
set smoothed = %scalar(xstates)
set simulate = %scalar(xstates_s)-%scalar(xstates_s(1871:1))+$
               smoothed(1871:1)
set csimulate = %scalar(xstates_c)
set upper     = smoothed+2.0*sqrt(%scalar(vstates))
set lower     = smoothed-2.0*sqrt(%scalar(vstates))
*
spgraph(vfields=2,hfields=2)
graph(overlay=dots,ovsame,header=$
      "Smoothed State and Unconditional Sample") 2
# smoothed
# simulate
graph(overlay=dots,ovsame,header=$
      "Smoothed State and Conditional Sample") 4
# smoothed
# upper      / 3
# lower      / 3
# csimulate  / 2
*
set smoothedobsv = %scalar(vhat)
set sampleobsv   = %scalar(vdraws)
graph(overlay=dots,ovsame,header=$
      "Smoothed Observation Error and Sample") 2

```

```

# smoothedobsv
# sampleobsv
*
set smoothedstatew = %scalar(what)
set samplestatew   = %scalar(wdraws)
graph(overlay=dots,ovsame,header=$
      "Smoothed State Error and Sample") 2
# smoothedstatew
# samplestatew
spgraph(done)

```

Example 8.2 Gibbs Sampling with Local Level Model

This does Gibbs Sampling for the variances in a local level model. This is covered in Section 8.2.1.

```

open data nile.dat
calendar 1871
data(format=free,org=columns,skips=1) 1871:1 1970:1 nile
*
* Guess values
*
filter(type=centered,width=11,extend=repeat) nile / fnile
sstats(mean) / (nile-fnile)^2>>initsigsq
*
compute sigsqeps=initsigsq,sigsqeta=sigsqeps*.01
*
* Prior for eps
*
compute s2eps=sigsqeps,nueps=1.0
*
* Prior for eta
*
compute s2eta=sigsqeta,nueta=1.0
*
compute nburn =1000
compute ndraws=25000
*
set sigepsd 1 ndraws = 0.0
set sigetad 1 ndraws = 0.0
*
infobox(action=define,progress,lower=-nburn,upper=ndraws) $
"Gibbs Sampling"
do draw=-nburn,ndraws
*
* Conditionally simulate the data. In this model, we don't need
* to save the states since all the information is in the sample
* WHAT and VHAT.
*
dlm(a=1.0,c=1.0,y=nile,sv=sigsqeps,sw=sigsqeta,presample=diffuse,$
    type=csimulate,what=etahat,vhat=epshat)
*

```

```

* Simulate sigsqeta using the sample information about W
*
sstats / etahat(t)(1)^2>>sumsqeta
compute sigsqeta=(sumsqeta+nueta*s2eta)/%ranchisqr(%nobs+nueta)
*
* Simulate sigsqeps using the sample information about V
*
sstats / epshat(t)(1)^2>>sumsqeps
compute sigsqeps=(sumsqeps+nueps*s2eps)/%ranchisqr(%nobs+nueps)
*
* Save the variances if we're past the burn-in
*
infobox(current=draw)
if draw>0 {
    compute sigepsd(draw)=sigsqeps
    compute sigetad(draw)=sigsqeta
}
end do draw
infobox(action=remove)
*
density(smoothing=1.5,grid=automatic) sigepsd / xeps feps
scatter(style=line,vmin=0.0,footer="MCMC Density of SIGEPSSQ")
# xeps feps
*
density(smoothing=1.5,grid=automatic) sigetad / xeta feta
scatter(style=line,vmin=0.0,footer="MCMC Density of SIGETASQ",$
    smpl=xeta<=6000.0)
# xeta feta

```

Example 8.3 TVC Model, Gibbs Sampling with Independent Drift Variances

This demonstrates Gibbs sampling with independent drift variances from Section 8.2.2.

```

open data tvp.xls
cal(q) 1959:3
data(format=xls,org=columns) 1959:03 1985:04 mlgr dintlag $
    inflag surplag mllag
*
* Least squares estimates of the money demand function
*
linreg mlgr 1962:1 *
# constant dintlag inflag surplag mllag
*
* Start and end of the "working" sample
*
compute sstart=%regstart(), send=%regend()
compute nbeta=%nreg
*
dec vect swprior(nbeta) swdraw(nbeta)
dec real svprior svdraw

```

```

*
compute svprior=.5*%seesq
compute svdraw =svprior
compute svnu    =1.0
*
compute swprior=.01*%stderrs.^2
compute swdraw =swprior
compute swnu    =1.0
*
equation(lastreg) mdeq
*
compute nburn=10000
compute ndraws=25000
*
* For keeping track of the sum and sum of squared coefficients for
* computing the mean and error bands.
*
dec vect[series] sumbeta(nbeta) sumsqbeta(nbeta)
clear(zeros) sumbeta sumsqbeta
*
* For keeping track of the draws for the variance hyperparameters.
*
dec series[vect] sigdraws
nonlin(parmset=sigsqs) swdraw svdraw
gset sigdraws 1 ndraws = %parmspeek(sigsqs)
*
infobox(action=define,progress,lower=-nburn,upper=ndraws) $
    "Gibbs Sampling"
do draw=-nburn,ndraws
    *
    * Draw time-varying coefficients using conditional simulation
    *
    dlm(y=mlgr,c=%eqnxvector(mdeq,t),sw=%diag(swdraw),sv=svdraw,$
        presample=diffuse,type=csimulate,what=what,vhat=vhat) $
        sstart send xstates
    *
    * Draw equation variance given vhat's
    *
    sstats sstart send vhat(t)(1)^2>>rssv
    compute svdraw=(rssv+svnu*svprior)/%ranchisqr(%nobs+svnu)
    *
    * Draw drift variances given what's
    *
    do i=1,nbeta
        sstats sstart send what(t)(i)^2>>rssw
        compute swdraw(i)=(rssw+swnu*swprior(i))/%ranchisqr(%nobs+swnu)
    end do i
    infobox(current=draw)
    *
    * For positive draws, save information
    *
    if draw>0 {
        do i=1,nbeta
            set sumbeta(i)    sstart send = sumbeta(i)+xstates(t)(i)

```

```

        set sumsqbeta(i) sstart send = sumsqbeta(i)+xstates(t)(i)^2
      end do i
      compute sigdraws(draw)=%parmspeek(sigsqs)
    }
  end do draw
  infobox(action=remove)
  *
  * Compute entry-by-entry means, and upper and lower (one standard
  * deviation) bounds on the coefficients.
  *
  dec vect[series] mean(nbeta) upper(nbeta) lower(nbeta)
  do i=1,nbeta
    set sumbeta(i) = sumbeta(i)/ndraws
    set sumsqbeta(i) = sqrt(sumsqbeta(i)/ndraws-sumbeta(i)^2)
    set mean(i) = sumbeta(i)
    set upper(i) = sumbeta(i)+sumsqbeta(i)
    set lower(i) = sumbeta(i)-sumsqbeta(i)
  end do i
  *
  * Graph the time-varying estimate of the inflation coefficient. Again,
  * note that this is a smoothed estimate.
  *
  graph(footer="Inflation Coefficient") 3
  # mean(3) sstart send
  # upper(3) sstart send 2
  # lower(3) sstart send 2
  *
  set tvinfl 1 ndraws = sigdraws(t)(3)
  density(smoothing=1.5,grid=auto,maxgrid=400) tvinfl / xinfl finfl
  scatter(style=line,vmin=0.0,$
    footer="Posterior Density of Inflation Drift Variance")
  # xinfl finfl
  *
  graph(nodates,footer="Draws for Inflation Drift Variance")
  # tvinfl

```

Example 8.4 TVC Model, Gibbs Sampling with Correlated Empirically-Based Drift

This demonstrates Gibbs sampling with correlated empirically-based drift. This is covered in Section 8.2.2.

```

open data tvp.xls
cal(q) 1959:3
data(format=xls,org=columns) 1959:03 1985:04 mlgr dintlag $
  inflag surplag mllag
  *
  * Least squares estimates of the money demand function over
  * "training" sample.
  *
  linreg mlgr 1962:1 1971:4
  # constant dintlag inflag surplag mllag

```



```

*
* Start and end of the "working" sample
*
compute sstart=%regend()+1, send=1985:4
compute nbeta=%nreg
equation(lastreg) mdeq
*
compute covmat=%seesq*%xx
*
* Pre-sample mean/covariance matrix for coefficients
*
compute x0 =%beta
compute sx0=4.0*covmat
*
* Prior for the drift variance. We start the draws at the prior
* "mean".
*
compute kq=.1
compute swprior=kq^2*covmat
compute swnu=10.
compute swdraw=swprior
*
* This is for the sum of the outer products of the sampled W's
*
dec symm rssw(nbeta,nbeta)
*
* Prior for the equation variance
*
compute svprior=%seesq
compute svnu=10.
compute svdraw=svprior
*
compute nburn=10000
compute ndraws=25000
*
dec vect[series] sumbeta(nbeta) sumsqbeta(nbeta)
clear(zeros) sumbeta sumsqbeta
dec vect[series] swdraws(nbeta)
do i=1,nbeta
    set swdraws(i) 1 ndraws = 0.0
end do i
set svdraws 1 ndraws = 0.0
*
infobox(action=define,progress,lower=-nburn,upper=ndraws) $
    "Gibbs Sampling"
do draw=-nburn,ndraws
    *
    * Draw time-varying coefficients using conditional simulation
    *
    dlm(y=mlgr,c=%eqnxvector(mdeq,t),sw=swdraw,sv=svdraw,$
        x0=x0,sx0=sx0,type=csimulate,what=what,vhat=vhat) $
        sstart send xstates
    *
    * Draw equation variance given vhat's

```

```

*
sstats sstart send vhat(t)(1)^2>>rssv
compute svdraw=(rssv+svnu*svprior)/%ranchisqr(%nobs+svnu)
*
* Draw drift variance given what's
*
compute rssw=swnu*swprior
do t=sstart, send
  compute rssw=rssw+%outerxx(what(t))
end do t
compute swdraw=%ranwisharti(%decomp(inv(rssw)),%nobs+swnu)
infobox(current=draw)
if (draw>0) {
  do i=1,nbeta
    set sumbeta(i) sstart send = sumbeta(i)+xstates(t)(i)
    set sumsqbeta(i) sstart send = sumsqbeta(i)+xstates(t)(i)^2
  end do i
  compute %pt(swdraws,draw,%xdiag(swdraw))
  compute svdraws(draw)=svdraw
}
end do draw
infobox(action=remove)
*
dec vect[series] mean(nbeta) upper(nbeta) lower(nbeta)
do i=1,nbeta
  set sumbeta(i) = sumbeta(i)/ndraws
  set sumsqbeta(i) = sqrt(sumsqbeta(i)/ndraws-sumbeta(i)^2)
  set mean(i) = sumbeta(i)
  set upper(i) = sumbeta(i)+sumsqbeta(i)
  set lower(i) = sumbeta(i)-sumsqbeta(i)
end do i
graph 3
# mean(3) sstart send
# upper(3) sstart send 2
# lower(3) sstart send 2
*
set tvinfl 1 ndraws = swdraws(3)
density(smoothing=2.0,grid=automatic,maxgrid=400) tvinfl / xinfl finfl
scatter(style=line,vmin=0.0,$
  footer="Posterior Density of Inflation Drift Variance")
# xinfl finfl
density(smoothing=2.0,grid=automatic,maxgrid=400) svdraws / xsv fsv
scatter(style=line,vmin=0.0,$
  footer="Posterior Density of Equation Variance")
# xsv fsv

```

Example 8.5 Gibbs Sampling with Linear Regression with AR errors

This demonstrates Gibbs Sampling for a linear regression with autoregressive (AR(2)) errors. This is from Section 8.2.3.

```
open data lake.dat
cal 1875
data(format=free,org=columns) 1875:1 1972:1 lake
*
* Estimate the time trend model and get the residuals
*
set trend = t
*
* Estimate a linear regression with an AR(2) noise term by
* (full-sample) maximum likelihood using BOXJENK.
*
boxjenk(ar=2,glsl) lake
# constant trend
*
* Same thing with DLM
*
* Do the linear regression to set up the regression parameters
*
linreg(define=regeqn,robust,lwindow=newey,lags=5) lake / u
# constant trend
frml(lastreg,paramset=regparms,vector=b) mufrml
compute nreg=%nreg
*
* Do an AR(2) on the residuals to set up the noise process
* parameters.
*
dec vect rho(2)
nonlin(paramset=arparms) rho sigsq
linreg u
# u{1 2}
compute rho=%beta,sigsq=%sigmasq
*
dec frml[rect] af
frml af = ||rho(1),rho(2)||$
          1.0 , 0.0 ||
compute f=%unitv(2,1)
compute c=%unitv(2,1)
compute nz=2
compute nzparms=2
*
dml(y=lake,a=af,mu=mufrml,f=f,c=c,sw=sigsq,$
    paramset=arparms+regparms,presample=ergodic,method=bfgs)
*
* Gibbs sampling
*
* Gaussian prior on regression coefficients. (Input as mean and
* precision). The prior for the intercept is diffuse (zero
```

```

* precision). The prior for the trend is 0 mean with a standard
* deviation of .25 (precision=16).
*
compute [vector] bbprior=||0.0,0.0||
compute [symm]   bhprior=%diag(||0.0,1.0/.25^2||)
*
* Gaussian prior on AR coefficients. There's no perfect way to
* specify a prior on an AR(2), since you can have either real or
* complex roots.
*
compute [vector] rbprior=||0.8,0.0||
compute [symm]   rhprior=20.0*||2.0|-1.0,1.0||
compute logqlast=0.0
*
* Pull the proposal density from the results of the DLM instruction.
* This is the mean and a scale of the Cholesky factor of the
* covariance matrix.
*
compute [vector] rbprop=%xsubvec(%beta,1,nzparms)
compute [rect]   fbprop=1.5*%decomp(%xsubmat(%xx,1,nzparms,1,nzparms))
*
* Inverse chi-squared prior on the variance
*
compute s2eps=0.5^2
compute nueps=1.0
*
compute nburn=1000
compute ndraw=10000
*
dec series[vect] regdraws
dec series[vect] ardraws
*
gset regdraws 1 ndraw = %parmspeek(regparms)
gset ardraws  1 ndraw = %parmspeek(arparms)
*
infobox(action=define,progress,lower=-nburn,upper=ndraw) $
  "Gibbs sampling"
compute accept=0
*
do draw=-nburn,ndraw
  *
  * Do DLM with augmented model to get the GLS estimates (given the
  * rho's and sigma).
  *
  dlm(y=lake,a=af~\%identity(nreg),f=f~~%zeros(nreg,1),sw=sigsq,$
    c=c~~%eqnxvector(regeqn,t),presample=ergodic,method=solve) / $
    xstates vstates
  *
  * Extract the mean and precision from the augmented model
  *
  compute bdata=%xsubvec(xstates(%regend()),nz+1,nz+nreg)
  compute hdata=inv(%xsubmat($
    vstates(%regend()),nz+1,nz+nreg,nz+1,nz+nreg))
  *

```

```

* Draw the beta coefficients
*
compute b=%ranmvpost(hdata,bdata,bhprior,bbprior)
*
* There's nothing to simulate since the errors are exactly
* computable given the coefficients.
*
set z = lake-mufrml
*
* Compute the log likelihood at the current beta, rho and sigsq
* and add the log prior on the rho's to get the log posterior of
* the rho's conditional on beta and sigsq.
*
dlim(y=z,a=af,f=f,c=c,sw=sigsq,presample=ergodic,method=solve)
compute logplast=%logl+%logdensity(inv(rhprior),rho-rbprior)
*
* Draw a proposal (save the current rho vector in case we need
* to keep it).
*
compute rholast=rho
:redraw
compute [vector] rho=rbprop+%ranmvnormal(fbprop)
if (%polysmallestroot(1.0~-1.0*rho) < 1.0)
    goto redraw
compute logqtest=%ranlogkernel()
*
* Compute the log posterior with the proposed rho's
*
dlim(y=z,a=af,f=f,c=c,sw=sigsq,presample=ergodic,method=solve)
compute logptest=%logl+%logdensity(inv(rhprior),rho-rbprior)
*
* Do the Metropolis test
*
compute alpha=exp(logptest-logplast+logqlast-logqtest)
*
if alpha>1.0.or.%ranflip(alpha) {
    compute logqlast=logqtest
    compute accept =accept+1
}
else
    compute rho=rholast
*
* Draw the variance
*
dlim(y=z,a=af,f=f,c=c,sw=sigsq,presample=ergodic,$
    type=csimulate,what=what,swhat=swhat)
*
sstats / what(t)(1)^2>>sumsqeps
compute sigsq=(sumsqeps+nueps*s2eps)/%ranchisqr(%nobs+nueps)
infobox(current=draw) %strval(100.0*accept/(draw+nburn+1),"###.##")
if draw>0 {
    compute regdraws(draw)=%parmspeek(regparms)
    compute ardraws(draw)=%parmspeek(arparms)
}

```

```

end do draw
infobox(action=remove)
*
set trenddraws 1 ndraw = regdraws(t) (2)
stats trenddraws
*
set rho1draws 1 ndraw = ardraws(t) (1)
set rho2draws 1 ndraw = ardraws(t) (2)
*
density(smoothing=2.0,grid=automatic,maxgrid=100) rho1draws / $
    xrho1 frho1
set frho1pr 1 100 = exp(%logdensity($
    1.0/rhprior(1,1),xrho1-rbprior(1)))
scatter(footer="Density Estimates for rho1",style=line,vmin=0.0,$
    key=below,klabels=| "Posterior","Prior" |) 2
# xrho1 frho1
# xrho1 frho1pr
*
compute h=.10
dec vect r1grid(101) r2grid(101)
dec rect fgrid(101,101)
compute r1grid=%seqrang(0.70,1.10,101)
compute r2grid=%seqrang(-0.4,0.05,101)
compute fgrid=%zeros(101,101)
do draw=1,ndraw
    ewise fgrid(i,j)=fgrid(i,j)+exp(-.5/h^2*($
        (rho1draws(draw)-r1grid(i))^2+(rho2draws(draw)-r2grid(j))^2))
end do draw
gcontour(x=r1grid,y=r2grid,f=fgrid,$
    footer="Density of Draws for AR coefficients")

```

Non-Normal Errors and Non-linear Models

The (standard) Kalman filter relies upon three key assumptions:

1. The state equation is linear in the states
2. The measurement equation is linear in the states
3. The errors are Gaussian and independent across time

If any of these assumptions is violated, the Kalman filter described in Section 2.1 will fail in some way to be an exact calculation. At an minimum (for instance, if the only failure is Gaussianity), the likelihood won't be correct though the estimates will still be minimum variance. Any non-linearity will require either a different form of filtering, such as particle filtering (Section 9.4) which is a simulation technique, or use of the Kalman filter through linearization (Section 9.3) to approximate the dynamical system. We already saw an example of the latter in Section 7.2. The use of the standard Kalman filter on a linearized system is known as the Extended Kalman Filter (or EKF) or Non-Linear Kalman Filter. These tend to be relatively straightforward, as they just require linearization around some expansion point.

There are various techniques which can be applied to approximate solutions for non-normal errors. For some types of models (such as with discrete data), there is an updating system which is similar to the Kalman filter, but isn't close enough to be doable with **DLM**. A couple of examples of that are Brockwell and Davis, example 8.8.7 (RATS example file `itsf2p306.rpf`) and Durbin and Koopman, example 14.6 (RATS example file `durk2p324.rpf`). We'll concentrate on models that are mainly handled with **DLM**.

9.1 Stochastic volatility model

The stochastic volatility model takes the form

$$y_t = \sqrt{h_t} \varepsilon_t \tag{9.1}$$

$$\log h_t = \alpha + \beta \log h_{t-1} + \eta_t \tag{9.2}$$

$$\begin{bmatrix} \varepsilon_t \\ \eta_t \end{bmatrix} \sim N \left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & \sigma^2 \end{bmatrix} \right) \tag{9.3}$$

While outwardly similar to a GARCH model, there are some technical problems created because the unobservable variance has its own independent shock. (In the ARCH/GARCH models, the variance can be computed recursively from the data). There are a number of methods of estimating the free parameters of this, most of which use (rather complicated) simulations. A simpler technique is to approximate it with a standard state-space model.

If we square and take logs of both sides of (9.1), we get

$$\log y_t^2 = \log h_t + \log \varepsilon_t^2$$

$\log y_t^2$ is a perfectly good observable (aside from not being defined when $y_t = 0$). Combined with (9.2), we have a state-space model with $\log h_t$ as the only state. However, the measurement error in this ($\log \varepsilon_t^2$) 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

$$\log y_t^2 + 1.27 = \log h_t + \tilde{\varepsilon}_t; \tilde{\varepsilon}_t \sim N(0, \pi^2/2)$$

Kalman filtering and smoothing with the combination of this and (9.2) 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 I.

Example 9.1 is taken from the *first* edition of DK. The one difference between that example and the discussion above (other than renaming of a couple of parameters) is that the variance of $\tilde{\varepsilon}_t$ is treated as a free parameter rather than being pegged to the $\pi^2/2$ implied by the model.

As noted earlier, the one problem with $\log y_t^2$ as an observable is that it's possible for it to be undefined if any of the data points are zero (quite possible with daily return data). The standard fix for this is to replace any zeros with a small positive number.

```
compute meanx2=%digamma(0.5)-log(0.5)
set ysq = log(xrate^2)-meanx2
*
nonlin phi sw sv kappa
compute phi=.9,sw=.011,sv=1.0,kappa=0.0
*
dlm(y=ysq,a=phi,c=1.0,z=kappa,sw=sw,sv=sv,$
    presample=ergodic,vhat=vhat,svhat=svhat,$
    method=bfgs,reject=abs(phi)>=1.0) 1 945 states
```

¹ $\psi(x)$ is the *digamma* function, computable in RATS using %DIGAMMA(x).

Here `PHI` is what we called β and `KAPPA` is α . The `REJECT` option is included to protect against trying to evaluate the function at impermissible values. Note that this model is sometimes estimated with a random walk for $\log h_t$. To do that, get rid of `KAPPA`, take `PHI` out of the parameter set and make it 1.0, and remove the `REJECT` option.

Note that we have a much more extensive example of this in the replication files for Harvey et al. (1994).

9.2 *t* Distributed Errors

The Normal has a density which is too thin-tailed for some data sets. Now, if you have a few very serious outliers, one way to handle it is to use a time-varying variance for the measurement equation. For instance, if you think that the observations in 1980 might have triple the variance of the rest of the sample, using an option like

```
d1m(...,sv=%if(t>=1980:1.and.t<=1980:12,3*sigsqeps,sigsqeps),...
```

will allow for that. (You could also easily have two completely separate variances on different segments of the estimation range). For an outlier which seems just off the map, you can use the `SMPL` option to knock it out of the likelihood—that’s the equivalent of giving it an infinite measurement error variance.

If, however, you have a scattering of large errors not linked in any obvious way, you might be better off modeling the errors as t with a fairly small number of degrees of freedom. One useful way to look at t distributed errors is that they are Normal with “free-form” heteroscedasticity—the variances for the Normal are independent “draws” with an inverse chi-squared distribution. Most of the draws for the variance will be near the center of the distribution, but you’ll get occasional ones which are quite large.

The variance of $\sigma^2\xi$ where ξ is a t with ν degrees of freedom is $\sigma^2\nu/(\nu - 2)$. This parameterizes the t based upon the variance of the underlying Normal (σ^2) and the degrees of freedom. In practice, it’s often better to parameterize it with the variance of the distribution itself (that is, with $\lambda = \sigma^2\nu/(\nu - 2)$). So long as $\nu > 2$, the two are equivalent. The advantage of the second is numerical stability—the variance of the distribution is what we can more easily identify from the data. In order to match that as we vary ν , we would have to change σ^2 quite dramatically, particularly as we get ν below 10.

The Kalman filter can’t be applied directly with errors being t distributed—the sum of a Normal and a t isn’t a standard distribution. If you approximate it with a Normal, as we did in the stochastic volatility case, you don’t get any advantage from using the t , since all you’re using are the first two moments, and ignoring the kurtosis. An alternative is to put the approximation a bit

deeper into the calculation. If we parameterize the error distribution as being a t with variance σ_ε^2 and degrees ν , then the density function (ignoring factors that don't include ε_t) is:

$$(\sigma_\varepsilon^2 (\nu - 2) + \varepsilon_t^2)^{-\frac{\nu+1}{2}}$$

The log of this is

$$-\frac{\nu+1}{2} \log (\sigma_\varepsilon^2 (\nu - 2) + \varepsilon_t^2) \quad (9.4)$$

The log density for $\varepsilon_t \sim N(0, \sigma_t^2)$ is (again ignoring terms without ε_t^2)

$$-\frac{1}{2} \frac{\varepsilon_t^2}{\sigma_t^2} \quad (9.5)$$

where σ_t^2 is the time-varying variance for period t . If we take a first order Taylor series of (9.4) with respect to ε_t^2 (evaluated at a previous value of $\tilde{\varepsilon}_t^2$), we get the approximation:

$$-\frac{1}{2} \frac{\nu+1}{\sigma_\varepsilon^2 (\nu - 2) + \tilde{\varepsilon}_t^2} \varepsilon_t^2 \quad (9.6)$$

By inspection, the combination of (9.6) and (9.5) gives us

$$\sigma_t^2 = \frac{\sigma_\varepsilon^2 (\nu - 2) + \tilde{\varepsilon}_t^2}{\nu + 1}$$

An instructive way to write this is:

$$\sigma_t^2 = \frac{\sigma_\varepsilon^2 \times (\nu - 2) + (\tilde{\varepsilon}_t^2/3) \times 3}{\nu + 1}$$

This is a weighted average of σ_ε^2 and $\tilde{\varepsilon}_t^2/3$. Based upon this, the “outliers” are points where $\tilde{\varepsilon}_t^2 > 3\sigma_\varepsilon^2$, since those will be the ones where we make $\sigma_t^2 > \sigma_\varepsilon^2$. We can iterate over this to convergence. This is quite similar to iterated weighted least squares to robustify a linear regression.

While this works quite nicely for analyzing the states given the parameters, the one problem is that it doesn't give us an accurate value for the log likelihood itself. If we don't know ν or the variances, we can't estimate them by maximum likelihood using the value of %LOGL produced by this. In Example 9.2, they're instead estimated by the method of moments, which isn't as sensitive to specification.

The iteration process starts with the standard estimates assuming Normal errors and a large guess value for ν . An iteration on this is done using the following:

```
set fiddle = ((nu-2)+eps^2/veps)/(nu+1)
dlm(start=(sw=%diag(||exp(2*phil),exp(2*phis)||)),a=a,c=c,f=f,$
     y=gas,sv=fiddle,sw=sw,presample=diffuse,var=conc,method=bfgs,$
     vhat=vhat,type=smooth,print=lastone) / xstates vstates
set eps = vhat(t)(1)
```

FIDDLE is the adjustment series for the variances. This is concentrating out σ_ε^2 , which is why it's computed with normalized values of EPS. The relative variances for the seasonal and trend are parameterized in log square root form. The new value of EPS is the Kalman smoothed estimate for the error. The updated value of ν is computed using the sample kurtosis from the error process:

```
stats(noprint) eps
compute nu=4.0+6.0/%kurtosis
compute veps=%variance
```

This is repeated until convergence (of the values of NU).

9.3 Non-linearities in the Equations

A non-linearity in the *measurement* equation is particularly easy to handle through linearization. If

$$\mathbf{y}_t = \psi(\mathbf{X}_t) + \mathbf{v}_t \quad (9.7)$$

$$\mathbf{y}_t \approx \psi(\tilde{\mathbf{X}}_t) + \psi'(\tilde{\mathbf{X}}_t) (\mathbf{X}_t - \tilde{\mathbf{X}}_t) + \mathbf{v}_t = \left(\psi(\tilde{\mathbf{X}}_t) - \psi'(\tilde{\mathbf{X}}_t) \tilde{\mathbf{X}}_t \right) + \psi'(\tilde{\mathbf{X}}_t) \mathbf{X}_t + \mathbf{v}_t \quad (9.8)$$

where $\tilde{\mathbf{X}}_t$ is an expansion point for time t for the linearization for \mathbf{X}_t . This produces a linear measurement equation where $\left(\psi(\tilde{\mathbf{X}}_t) - \psi'(\tilde{\mathbf{X}}_t) \tilde{\mathbf{X}}_t \right)$ is a constant (given the expansion point) and will be the μ_t in the measurement and $\psi'(\tilde{\mathbf{X}}_t)$ is \mathbf{C}' . In practice, much of ψ is often linear, which reduces the need for linearization to just a term or two.

One can also linearize a non-linear *state* equation (which can also be combined with a non-linear measurement equation in a single model in the obvious way). If

$$\mathbf{X}_t = \phi(\mathbf{X}_{t-1}) + \mathbf{F}\mathbf{W}_t \quad (9.9)$$

the linearization is

$$\begin{aligned} \mathbf{X}_t &\approx \phi(\tilde{\mathbf{X}}_{t-1}) + \phi'(\tilde{\mathbf{X}}_{t-1}) (\mathbf{X}_{t-1} - \tilde{\mathbf{X}}_{t-1}) + \mathbf{F}\mathbf{W}_t \\ &= \left(\phi(\tilde{\mathbf{X}}_{t-1}) - \phi'(\tilde{\mathbf{X}}_{t-1}) \tilde{\mathbf{X}}_{t-1} \right) + \phi'(\tilde{\mathbf{X}}_{t-1}) \mathbf{X}_{t-1} + \mathbf{F}\mathbf{W}_t \end{aligned}$$

$\left(\phi(\tilde{\mathbf{X}}_{t-1}) - \phi'(\tilde{\mathbf{X}}_{t-1}) \tilde{\mathbf{X}}_{t-1} \right)$ is the \mathbf{Z}_t in the linearized state equation and $\phi'(\tilde{\mathbf{X}}_{t-1})$ is the \mathbf{A}_t . The linearized measurement equation generally works fairly well—the linearized *state* equation takes us a bit farther away from where we might expect that the approximate Kalman filter will be successful. Much of this is due to how it affects the start of the recursion. First, there is no “ergodic” solution for the linearized state, since the matrices in the state equation are time-varying. Also, it depends upon the expansion in the *lag* of \mathbf{X} , which doesn't exist pre-sample. In addition, the approximation error tends to accumulate quicker when it applies directly to the period-to-period evolution of the states.

One important technical question is how to choose \tilde{X}_t . In the engineering literature, it is very common to choose $\tilde{X}_t = X_{t|t-1}$ that is, the “best guess” that would be available prior to observing Y_t , and if you have a “real-time” system that you’re employing, that would be the obvious choice. If you are interested, however, in smoothed estimates of the states rather than just the current filtered ones, that’s not necessarily the best way to handle it, since the predicted estimates at the start of the sample can be quite poor. An alternative if you want good estimates across the full sample is to use the smoothed estimates ($\tilde{X}_t = X_{t|T}$), which is what is done in the **@DISAGGREGATE** procedure to handle the log-linear model (Section 7.2). The difficulty there is that the smoothed estimates change every time the expansion points change, even if the model itself is treated as known, which requires either iterating to convergence (as was done with the t -error handling in Section 9.2) or doing a single smooth at reasonable guess values and sticking with it.

Example **9.3** show the Extended Kalman Filter in practice—it is based upon Matheson & Stavrev (2013). This does a relatively standard “NAIRU-Phillips Curve” model (similar to Section 6.4) but allows for time-varying coefficients in the measurement equations—the multiplicative interaction between those coefficients and the unobservable states create the non-linearity. The paper also imposes inequality restrictions on the states (the time-varying coefficients) themselves, which we’ll take up in Section 9.5.

This starts with a Phillips curve equation, which has (observable) inflation (π_t) equal to expected inflation (π_t^e), plus terms in the unemployment gap (difference between observable unemployment u_t and the unobservable NAIRU (u_t^*)), and an observable import price push (π_t^m). The coefficients on those last two terms are allowed to vary as random walks. The resulting equation is

$$\pi_t = \pi_t^e - \kappa_t(u_t - u_t^*) + \gamma_t \hat{\pi}_t^m + \varepsilon_t^\pi \quad (9.10)$$

(Unobservable) expected inflation is derived from two observables using a time-varying average:

$$\pi_t^e = \theta_t \bar{\pi}_t + (1 - \theta_t) \pi_{t-1}^{(4)} \quad (9.11)$$

$\bar{\pi}_t$ is a long-term inflation expectation and $\pi_{t-1}^{(4)}$ is year-over-year inflation (lagged). Note that this equation has no error term and can just be substituted into (9.10). That can be rearranged to make

$$\pi_t = \pi_{t-1}^{(4)} + \theta_t(\bar{\pi}_{t-1} - \pi_{t-1}^{(4)}) - \kappa_t(u_t - u_t^*) + \gamma_t \hat{\pi}_t^m + \varepsilon_t^\pi \quad (9.12)$$

θ_t is also assumed to follow a random walk.

The state equation for the NAIRU gap is assumed to be the first order autoregression:

$$u_t - u_t^* = \rho(u_{t-1} - u_{t-1}^*) + \varepsilon_t^{(u-u^*)} \quad (9.13)$$

while the NAIRU itself evolves as a random walk:

$$u_t^* = u_{t-1}^* + \varepsilon_t^{u^*}$$

π_t , u_t , $\hat{\pi}_t^m$, $\bar{\pi}_t$ and $\pi_{t-1}^{(4)}$ are observable. The states are u_t^* and the time-varying coefficients κ_t , γ_t and θ_t . One can simplify the writing of the model by adding $u_t - u_t^*$ to the state vector as well, and then including the identity

$$u_t \equiv u_t^* + (u_t - u_t^*) \quad (9.14)$$

The alternative is to rearrange (9.13) to

$$u_t = \rho u_{t-1} + u_t^* - \rho u_{t-1}^* + \varepsilon_{t-1}^{u-u^*} \quad (9.15)$$

(which requires adding u_{t-1}^* to the state vector since both current and lagged values of the state are needed for this equation) and also rearrange (9.12) to separate the observable u_t from the state u_t^* . Needless to say, treating u_t^* and $u_t - u_t^*$ as separate states is quite a bit simpler.

With two separate states for unemployment, the measurement equations are (9.12) and (9.14). The state equations themselves are linear. The non-linearity comes in (9.12), which has the multiplicative interaction term between states κ_t and $(u_t - u_t^*)$. If we call the latter G_t for short, the linearization of that would have:

$$\kappa_t G_t \approx \tilde{\kappa}_t \tilde{G}_t + \tilde{\kappa}_t (G_t - \tilde{G}_t) + \tilde{G}_t (\kappa_t - \tilde{\kappa}_t) = \left(-\tilde{\kappa}_t \tilde{G}_t \right) + \tilde{\kappa}_t G_t + \tilde{G}_t \kappa_t \quad (9.16)$$

The first term in the final expression is a constant (as far as the state-space model is concerned) and thus goes into the MU. The other two give the **C** matrix values for the G and κ states respectively. The linearized measurement equation can be written:

$$\begin{bmatrix} \pi_t \\ u_t \end{bmatrix} = \begin{bmatrix} \pi_{t-1}^{(4)} + \tilde{\kappa}_t \tilde{G}_t \\ 0 \end{bmatrix} + \begin{bmatrix} -\tilde{\kappa}_t & 0 & -\tilde{G}_t & \bar{\pi}_{t-1} - \pi_{t-1}^{(4)} & \pi_t^m \\ 1 & 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} G_t \\ u_t^* \\ \kappa_t \\ \theta_t \\ \gamma_t \end{bmatrix} + \begin{bmatrix} \varepsilon_t^\pi \\ 0 \end{bmatrix}$$

If we estimate this model with freely estimated drift variances, it is likely to produce nonsensical results. First off, the NAIRU decomposition will have the same problem that we saw in Section 5.3—there just is too little difference in log likelihood across a wide range of different decompositions. To handle that, the authors impose one of two different variance ratios between the variances in the two components of the gap model, which they call “stable” and “flexible” (referring to the behavior of the NAIRU component).

The authors also put limits on the drift variances for the Phillips curve parameters. While this isn't at all unreasonable (we saw in Section 5.4 that those can come in too high to be sensible), the method used seems a bit arbitrary. They run 40 period rolling regressions and estimate the variance from the differences in the parameter estimates in adjacent windows. However, the regression run with the window from 1 to 40 and the regression from 2 to 41 have 39 observations in common so aren't really going to tell you much about possible period-to-period fluctuations. To be more specific, consider applying this

method to the local level model:

$$y_t = x_t + v_t$$

$$x_t = x_{t-1} + w_t$$

If we take the mean over 2 through 41 minus the mean over 1 through 40, it's just

$$\frac{y_{41} - y_1}{40} = \frac{x_{41} - x_1}{40} + \frac{v_{41} - v_1}{40} = \frac{w_2 + \dots + w_{41}}{40} + \frac{v_{41} - v_1}{40}$$

The variance of this is

$$\frac{\sigma_w^2}{40} + \frac{2\sigma_v^2}{1600}$$

which is likely to be a (serious) underestimate of information about σ_w^2 (unless the data are very noisy, that is σ_v^2 is large in comparison, in which case the whole idea is suspect). Note also that 40 is completely arbitrary—a wider window would give a smaller limit, a narrower one would give a larger one. While the idea is not well-motivated, the example will follow the paper's lead.

The data set is a reproduction of the original with an extension to 2016. This does the initial transformations. Note that we will lose five initial data points: PISTAR needs four (three lags plus one more for creating the inflation rate in the first place), and we use a lag of PISTAR in the model.

```
set u      = unrates
set pi     = 400.0*log(cpi/cpi{1})
set pim    = 400.0*log(mdef/mdef{1})-pi
set pibar  = ptrcpi
set pistar = .25*(pi+pi{1}+pi{2}+pi{3})
```

To initialize this, we use an HP filter to generate a preliminary U^* and the gap derived from that.

```
filter(type=hp) u / ustar_0
set ugap_0 = u-ustar_0
```

This sets up the rolling regressions. They're done by non-linear least squares though they could be done (with some rearrangement of variables) using OLS—it's just easier to set this up as a FRML.

```
nonlin(parmset=rollparms) kappa_r theta_r gamma_r
frml rollpi pi = pistar{1}+theta_r*(pibar-pistar{1})-$
      kappa_r*ugap_0+gamma_r*pim
```

This sets the span for the rolling regressions, the overall sample start and end (SSTART and SEND) and the start for the rolling regressions (RSTART):

```

compute span    =40
compute sstart  =1961:02
compute rstart  =sstart+span-1
compute send    =2016:04

```

This does the full sample regression

```

compute kappa_r=0.1,gamma_r=0.1,theta_r=0.5
nlls (parmset=rollparms,frml=rollpi) pi sstart send
compute sv=%sigmasq

```

and this does the rolling sample regressions and saves the series of estimated coefficients.

```

clear(zeros) kappa_re theta_re gamma_re
do time=rstart,send
  compute kappa_r=0.1,gamma_r=0.1,theta_r=0.5
  nlls (parmset=rollparms,frml=rollpi,noprint) pi time-(span-1) time
  compute kappa_re(time)=kappa_r
  compute theta_re(time)=theta_r
  compute gamma_re(time)=gamma_r
end do time

```

This extends the coefficient series back to the start of the sample by repeating the initial window for the early part of the data set:

```

set kappa_0 = %if(t<=rstart,kappa_re(rstart),kappa_re)
set theta_0 = %if(t<=rstart,theta_re(rstart),theta_re)
set gamma_0 = %if(t<=rstart,gamma_re(rstart),gamma_re)

```

This next computes the average squared change of the coefficients (over the part of the sample which had enough data to do the regressions):

```

sstats(mean) rstart+1 send (kappa_0-kappa_0{1})^2>>sigsqup_kappa $
                             (theta_0-theta_0{1})^2>>sigsqup_theta $
                             (gamma_0-gamma_0{1})^2>>sigsqup_gamma

```

This gets guess values for the AR coefficient in the gap model, and its variance:

```

linreg ugap_0
# ugap_0{1}
compute rho=%beta(1)
compute swugap=%sigmasq

```

This now sets up the linearized MU and C formulas. Note that MU has one term due to linearization, and the PISTAR1 that's part of the original (9.12). Note also that because κG enters the Phillips curve with a negative sign, the members of MU and C due to the expansion of that have the opposite signs from (9.16).

```

dec frml[vect] muf
dec frml[rect] cf
dec frml[symm] svf
dec frml[symm] swf
*
frml muf = ||+kappa_0*ugap_0+pistar{1},0.0||

*
frml cf = ||-kappa_0      ,1.0|$
           0.0           ,1.0|$
           -ugap_0       ,0.0|$
           (pibar-pistar{1}),0.0|$
           pim            ,0.0||

```

and this sets up the variance FRML's and the parameter set for estimating the free parameters.

```

dec vector swtvc(4)
*
frml svf = %diag(||sv,0.0||)
frml swf = %diag(swugap~~swtvc)
*
nonlin(parmset=base) rho sv swugap swtvc

```

These are the two variance ratio pegs for the components of the NAIRU decomposition. The first is the “stable” one and the second the “flexible”. SWUGAP is the variance in the gap autoregression, SWTVC(1) is the drift variance in the NAIRU, so the first allows relatively less variability in the NAIRU.

```

nonlin(parmset=stable)    swugap=15.0*swtvc(1)
nonlin(parmset=flexible) swugap=5.0*swtvc(1)

```

It's important to note that SWUGAP is the variance in an AR process and SWTVC(1) is the variance of the increment in a random walk, so the two aren't strictly comparable—how SWUGAP affects the variance of the gap process depends upon the value of ρ as well, so a smaller value of ρ (with these data, it comes in at a high value of .95) might require higher multiples to achieve a desired amount of movement. As we did in Section 5.3, an alternative approach would be to peg the ratio of $\text{SWUGAP} / (1 - \rho^2)$ to $\text{SWTVC}(1)$, that is,

```
SWUGAP=something*SWTVC(1)*(1-RHO^2)
```

which would adapt to different values of ρ . (The “something” would have to be very different from the 15 and 5 used here).

The **A** matrix is just an identity matrix other than $A_{1,1}$. %%DLMSetsup will reset that to the current test value of ρ :


```

dec rect a(5,5)
compute a=%na~\%identity(4)
*
function %%DLMSetup
compute a(1,1)=rho
end

```

This sets up the limits on the parameters. All the drift variances are set to be non-negative, and the three time-varying parameter variances are limited as described above. In addition, ρ is limited to go no higher than .95—if it gets much higher, you start to see confusion between the gap and the NAIRU itself. (It's possible that doing the variance ratio limit with the $(1 - \rho^2)$ factor included might eliminate the need for that).

```

nonlin(parmset=limits) swtvc(1)>=0.0 rho<=.95 $
                    swtvc(2)>=0.0 swtvc(2)<=sigsqup_kappa $
                    swtvc(3)>=0.0 swtvc(3)<=sigsqup_theta $
                    swtvc(4)>=0.0 swtvc(4)<=sigsqup_gamma

```

This sets guess values for the random walk variances. The last three are based upon the limits already computed and the first (for the NAIRU) allows for drift of about ± 2 over 10 quarters.

```

compute swtvc(1)=.1
compute swtvc(2)=.5*sigsqup_kappa
compute swtvc(3)=.5*sigsqup_theta
compute swtvc(4)=.5*sigsqup_gamma

```

This does a single Kalman smooth at the guess values to get the expansion points for the linearization that will be used in the later analysis:

```

dlm(start=%%DLMSetup(), a=a, c=cf, y=||pi,u||, mu=muf, sv=svf, sw=swf, $
    presample=ergodic, type=smooth) sstart send xstates vstates
set ugap_0 = xstates(t)(1)
set kappa_0 = xstates(t)(3)

```

The paper does graphs with quite a few different variations on the estimates. To help organize this, we use a set of `HASH` aggregators. The final results will be saved in `HASH`'es layered three deep: the outermost is the choice between stable and flexible variance restrictions (shortened to keys of "stab" and "flex"), the middle one is filtered vs smoothed estimates ("kf" and "sm") and the innermost between the estimate, and lower and upper (one standard deviation) bounds ("est", "lo", "hi"):

```

dec hash[parmset] pegs
compute pegs("stab")=stable
compute pegs("flex")=flexible
*
dec hash[hash[hash[series]]] h_ustar h_kappa h_theta h_gamma
dec hash[hash[series]] h_pie
dec hash[series[vect]] h_xstates
dec hash[series[symm]] h_vstates

```

This does estimation of the parameters (combined with filtered estimates of the states) and smoothed estimates of the states for each of the two pegs on the variance ratios. It changes the limits on those by using PEGS(ROOT), where PEGS is a HASH[PARMSET] just defined which chooses between the STABLE and FLEXIBLE PARMSETS.

```

dec string root ktype
do for root = "stab" "flex"
  dlm(start=%%DLMSsetup(), a=a, c=cf, y=||pi, u||, mu=muf, sv=svf, sw=swf, $
    type=filter, presample=ergodic, $
    parmset=base+pegs(root)+limitsmethod=bfgs, condition=10) $
    sstart send h_xstates("kf") h_vstates("kf")
*
  dlm(start=%%DLMSsetup(), a=a, c=cf, y=||pi, u||, mu=muf, sv=svf, sw=swf, $
    type=smooth, presample=ergodic) $
    sstart send h_xstates("sm") h_vstates("sm")

```

Note that this uses the CONDITION option. The 10 is somewhat arbitrary, but you probably want to do at least five, as the ability of the model to predict those early data points is extremely weak.

The estimated states and covariance matrices are saved into HASHes of their own so the organization of the output can be done inside a DOFOR loop over the HASH keys ("kf" and "sm"). We omit most of this, since it's just the same set of lines applies to different elements out of the state vector:

```

do for ktype = "kf" "sm"
  set h_ustar(root) (ktype) ("est") = h_xstates(ktype) (t) (2)
  set h_ustar(root) (ktype) ("hi")   = h_xstates(ktype) (t) (2) + $
                                         sqrt(h_vstates(ktype) (t) (2,2))
  set h_ustar(root) (ktype) ("lo")   = h_xstates(ktype) (t) (2) - $
                                         sqrt(h_vstates(ktype) (t) (2,2))
  ...
  set h_pie(root) (ktype) = h_theta(root) (ktype) ("est") *pibar+$
                           (1-h_theta(root) (ktype) ("est")) *pistar{1}
end do ktype

```

The calculation of the inflation expectation is the one that has a different pattern. That takes the weighted average of two observables (PIBAR and PISTAR1) using the estimated time-varying thetas.

The output from the two models is shown in Tables 9.1 and 9.2. Note that in both models, *all* of the upper bounds (on ρ and the three drift variances) have been hit. If we do the standard “random walk” analysis on the θ coefficients for instance, over 10 years (40 quarters), the variance of that is $40 \times .0027$ or roughly .1, for a standard deviation of a bit over .3. Although we haven’t constrained it directly, it’s hard to imagine θ being outside $[0,1]$ (since it forms the weights on a weighted average), so a standard deviation of .3 over 40 quarters would pretty much let it roam over the full sensible range. Thus, the upper bound may still be a bit too high for a practical model.

The unemployment rate decomposition and the model’s inflation predictions are shown in Figure 9.1. The blue lines are from the stable model and the red lines from the flexible. (In the unemployment rate there are estimates with one standard deviation bounds). The inflation expectations are almost identical, which is why you can see almost no blue, as it gets overwritten by the red. The NAIRU’s are *not* identical as the flexible model tends to (as designed) move more with the peaks and troughs. As the log likelihood is quite a bit higher with the flexible model, it’s probably a good guess that without putting the limit on the variance ratio, a fully unconstrained maximum would have the NAIRU much more closely matching the unemployment rate itself (which certainly wouldn’t be the intent of the model). As we’ve seen before, these types of models depend on rather strong restrictions to produce “reasonable” results.

The time-varying coefficients themselves are shown in Figure 9.2. Not surprisingly, the filtered (left column) estimates are a bit wild at the start. As mentioned earlier, even though the drift variances on these had an upper bound (which was binding on the estimates), they still may be too high—if you compare (for instance) the filtered and smoothed estimates of θ in the 2008-9 recession, the filtered estimates seem to overreact, as there seems to be no permanent effect judging from the smoothed estimates. The fact that even the smoothed estimates for κ and γ go negative at parts of the sample presumably led the authors to put restrictions on the states, which is quite a bit more complicated technically than they describe. See Section 9.5.

9.4 Particle filtering

Particle filtering is a method for simulating non-linear dynamic models using a set of “particles” which track possible paths that the system could take. The linear Kalman filter operates by taking a Normal distribution of the states at time $t - 1$ and using the fact that the model predicts a joint Normal distribution at t for the states X_t and the observables y_t , generates a “post-observation” Normal distribution for X_t . With a Normal distribution, the mean and covariance matrix fully describe the distribution. By contrast, particle filtering generates a cloud of particles at $t - 1$ which are thought to fairly represent the states at $t - 1$ given the data through $t - 1$ and does inference about X_t by simulating its

Table 9.1: Stable, Unconstrained States

DLM - Estimation by BFGS with inequalities					
Convergence in 22 Iterations. Final criterion was 0.0000008 <= 0.0000100					
Quarterly Data From 1963:04 To 2016:04					
Usable Observations		213			
Rank of Observables		426			
Log Likelihood		-463.5915			
	Variable	Coeff	Std Error	T-Stat	Signif
1.	RHO	0.9500	0.0000	0.0000	0.0000
2.	SV	1.5824	0.1788	8.8504	0.0000
3.	SWUGAP	0.1137	0.2481	0.4581	0.6469
4.	SWTVC(1)	0.0076	0.0007	10.1832	0.0000
5.	SWTVC(2)	0.0090	0.0000	0.0000	0.0000
6.	SWTVC(3)	0.0027	0.0000	0.0000	0.0000
7.	SWTVC(4)	0.0002	0.0000	0.0000	0.0000

Table 9.2: Flexible, Unconstrained States

DLM - Estimation by BFGS with inequalities					
Convergence in 16 Iterations. Final criterion was 0.0000086 <= 0.0000100					
Quarterly Data From 1963:04 To 2016:04					
Usable Observations		213			
Rank of Observables		426			
Log Likelihood		-449.9448			
	Variable	Coeff	Std Error	T-Stat	Signif
1.	RHO	0.9500	0.0000	0.0000	0.0000
2.	SV	1.4342	0.1613	8.8937	0.0000
3.	SWUGAP	0.0957	0.0103	9.2881	0.0000
4.	SWTVC(1)	0.0191	0.0020	9.4393	0.0000
5.	SWTVC(2)	0.0090	0.0000	0.0000	0.0000
6.	SWTVC(3)	0.0027	0.0000	0.0000	0.0000
7.	SWTVC(4)	0.0002	0.0000	0.0000	0.0000

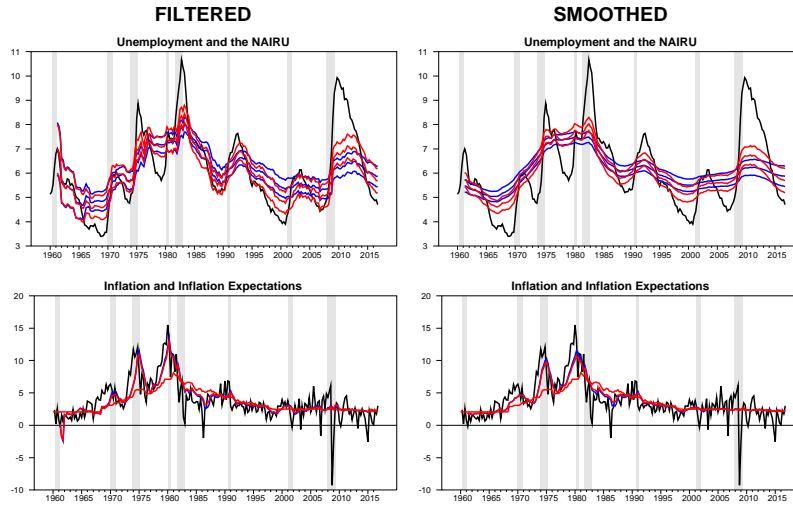


Figure 9.1: Unemployment and Inflation

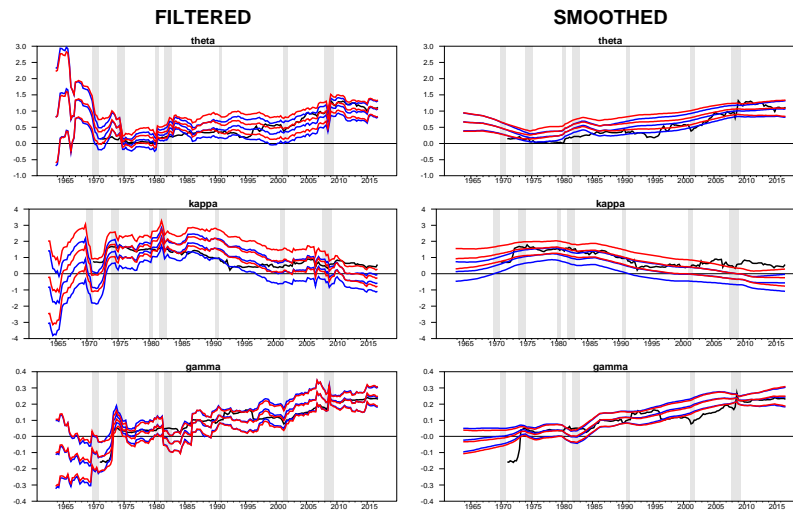


Figure 9.2: Unemployment and Inflation

distribution given the particles at $t-1$, the model, and the data. This generates a new series of particles at t , and the process is moved forward.

Particle filtering uses the method of *importance sampling* to simulate the states—see Appendix G for a general discussion. This requires that you be able to compute the densities for the transition function (given the value of X_{t-1}), and for the measurement equation (given the value of X_t). The “given the” is key: for the Gaussian linear model, we know the *complete* distribution, and can map complete distributions to complete distributions. With a non-linear model, it’s unlikely that we will ever have a known distribution at any point, beyond whatever assumption we make for the pre-sample. Instead of a known distribution, we have a large set of particle values with probability weights. If X_{t-1} is (treated as) known, then (for instance), in (9.9) (assuming the residuals are Gaussian),

$$\mathbf{X}_t \sim N(\phi(\mathbf{X}_{t-1}), \mathbf{F}\Sigma_W\mathbf{F}') \quad (9.17)$$

and we could easily handle t distributed errors as well. Similarly (9.7) will typically have a readily computable density for y_t given X_t .

Particle filtering is a form of importance sampling where the x ’s in the descriptions in the Appendix are actually sequences. If we use the notation $X_{1:t}$ to represent the values of X over the period 1 through t ,² then if $h(X_{1:t})$ is a (measurable) function of interest (typically involving just X_t), f is the actual joint density function of the sequence and g is the importance function for it (which we’ll determine later), then

$$\begin{aligned} E\{h(X_{1:t})|y_{1:t}\} &= \int h(X_{1:t})f(X_{1:t}|y_{1:t})dX_{1:t} \\ &= \int \left(h(X_{1:t}) \frac{f(X_{1:t}|y_{1:t})}{g(X_{1:t}|y_{1:t})} \right) g(X_{1:t}|y_{1:t})dX_{1:t} \end{aligned} \quad (9.18)$$

We can rewrite the density sequentially as

$$\begin{aligned} f(X_{1:t}|y_{1:t}) &= f(X_{1:t-1}|y_{1:t-1})f(X_t, y_t|X_{1:t-1}, y_{1:t-1})/f(y_t) \\ &= f(X_{1:t-1}|y_{1:t-1})f(y_t|X_{1:t}, y_{1:t-1})f(X_t|X_{1:t-1}, y_{1:t-1})/f(y_t) \end{aligned} \quad (9.19)$$

It’s the second and third factors that (9.7) and (9.9) provide. We can rewrite the importance density as

$$g(X_{1:t}|y_{1:t}) = g(X_t|X_{1:t-1}, y_{1:t})g(X_{1:t-1}|y_{1:t}) \quad (9.20)$$

Now the importance function is under *our* control. If we choose to follow a sequential process and only choose X_t itself at time t without revising the $X_{1:t-1}$, then the last factor in (9.20) is identical to the same thing conditioned just on $y_{1:t-1}$. If we define

$$w_t \equiv f(X_{1:t}|y_{1:t})/g(X_{1:t}|y_{1:t}) \quad (9.21)$$

²This is the notation used in Durbin & Koopman (2012).

then (9.19) and (9.20) can be combined into

$$w_t = w_{t-1} \times (1/f(y_t)) f(y_t|X_{1:t}, y_{1:t-1}) f(X_t|X_{1:t-1}, y_{1:t-1}) / g(X_t|X_{1:t-1}, y_{1:t}) \quad (9.22)$$

As mentioned in the Appendix, typically the true density functions for f and g involve some either ugly or possibly uncomputable integrating constants. The $f(y_t)$ in the recursion falls into that category—we have no way to compute it, but it’s a constant as far as x is concerned, so it can be ignored when we do the weighted sums.

Example 9.2 is taken from Chapter 12 of Durbin & Koopman (2012). This is just a particle filter implementation of the Kalman filtering for the Nile River data (Example 1.1)—since we can compute the actual distributions, it will give us a way to see how well the particle filter fares. The fact that the model is linear really isn’t *that* important, since the only part on the calculation that depends upon the nonlinear equations is a straightforward computation of (log) densities. To review, the model is

$$y_t = \alpha_t + \varepsilon_t$$

$$\alpha_t = \alpha_{t-1} + \eta_t$$

and we will be doing inference (filtering) on the local levels α_t . There is only one state in this case; however, the program is written to keep track of a state VECTOR. That will make it simpler to adapt to more complicated situations.

The first method will be called a bootstrap filter without resampling.³ We will take the parameters of the model as given:⁴

```
compute sigmaeta=sqrt(1469.1)
compute sigmax0 =sqrt(1.e+7)
compute sigmaepssq=15099.0
```

SIGMAETA and SIGMAEPSSQ are (roughly) the maximum likelihood estimates, while SIGMAX0 is a “large” value for the pre-sample variance.

This sets up the bookkeeping for the (10000) particles:

```
compute nstate=1
compute npart=10000
*
dec vect[vect] alphanew(npart)
dec series[vect[vect]] alpha
ewise alphanew(i)=%zeros(nstate,1)
gset alpha = alphanew
```

ALPHA makes up the particles. In the VECT[VECT] construction, the outer dimension is to keep track of the 10000 particles, and the inner is for the members of the state “vector” (which here just has the one element).

³As we’ll see, this is *not* a good procedure.

⁴There is starting to develop a literature on using particle filters to evaluate the likelihood function, but that requires a much more precise set of calculations.

As mentioned in the Appendix, the calculation of the weights can be complicated by over/underflow problems. To avoid this, we'll update the *log* weights instead. This sets that up:

```
dec series[vect] logw
dec vect wt(npart)
gset logw      = %zeros(npart,1)
```

These are apparently all initialized to a probability of 1 (log is 0), but we only need relative weights, and the weights on all particles are equal pre-sample.

Note that we don't *need* to save the full histories of either of these (the `SERIES[...]` constructions), as we can (just as in the Kalman filter) summarize all information through $t - 1$ just in the particles at $t - 1$ and their corresponding weights.

The following sets up the “bookkeeping” for the estimates of the first and second moments of the states (again, written to handle the case of a full vector rather than a scalar):

```
dec series[vect] alphahat
dec series[symm] alphahatxx
*
gset alphahat    = %zeros(nstate,1)
gset alphahatxx = %zeros(nstate,nstate)
```

The following is to keep track of the “effective sample size”, which is an estimate of how well the particles are working—we don't want to see particles drifting off into the tails of the distribution.

```
set ess      = 0.0
```

Unlike Gibbs sampling, this has simulations where we loop over time on the *outside*. The “pseudo-code” for this is:

```
do time=1871:1,1970:1
  ...update particles at time given time-1...
end do time
```

The particle update starts by simulating a new value for each particle using the probability distribution in the transition function. Here, for the first period, that involves merely initializing the particles by picking randomly from $N(0, \text{big variance})$. For subsequent periods, the new distribution is the old value plus a random increment.


```

do part=1,npart
  if time==1871:1 {
    compute alpha(time) (part)=sigmax0*%ranmat(1,1)
    compute laggedlogw=0.0
  }
  else {
    compute alpha(time) (part)=alpha(time-1) (part)+%ran(sigmaeta)
    compute laggedlogw=logw(time-1) (part)
  }
  compute logw(time) (part)=laggedlogw+$
    %logdensity(sigmaepssq,nile(time)-alpha(time) (part) (1))
end do part

```

Now, as we've designed this, we're drawing directly from the conditional density for X_t given the past (which would typically be the case), so

$$g(X_t|X_{1:t-1}, y_{1:t}) = f(X_t|X_{1:t-1}, y_{1:t-1}) \quad (9.23)$$

and those terms cancel in the weight update (9.22), so the only factors in the new weight will be the previous weight, the (unobservable constant) $f(y_t)$, and $f(y_t|X_{1:t}, y_{1:t-1})$, which is just the Normal density. Since we're accumulating these in logs, we just need to add the lagged log weight⁵ to the log density of the observable to get the new (relative) log weight on the particle.

The following does an “overflow-safe” calculation of the actual weights on the particles (at period `time`). First, `logw` itself is transformed by subtracting from each element the largest value that it takes, so after each period, `logw` will range from 0 down. Those values are exp'ed to give the relative weights (which can be no larger than 1), and then converted to probabilities by dividing by the sum of the relative weights.

```

compute maxlogw=%maxvalue(logw(time))
compute logw(time)=logw(time)-maxlogw
compute wt=%exp(logw(time))
compute wt=wt/%sum(wt)

```

We need the true weights at this point to compute the estimate (weighted mean) of the first and second moments. (Note that these are all initialized to zero before the start of the outer loop).

```

do part=1,npart
  compute alphahat(time)=alphahat(time)+$
    wt(part)*alpha(time) (part)
  compute alphahatxx(time)=alphahatxx(time)+$
    wt(part)*%outerxx(alpha(time) (part))
end do part

```

⁵The variable `LAGGEDLOGW` is used for that because there is no “lag” for the first period.

The next line computes the “effective sample size”. Because

$$\text{var} \left(\sum_i w_i \varepsilon_i \right) = \sigma_\varepsilon^2 \sum_i w_i^2$$

and the same calculation with equally weighted terms would have variance σ_ε^2/N , the reciprocal of the sum of squared weights is the (equally weighted) sample size that would match the variance of the sample average. If, for instance, one weight is very near one (thus all others quite small), the effective sample size would be just a bit over 1.

```
compute ess(time)=1.0/%normsqr(wt)
```

That completes the loop. Outside of it, this computes the sample mean estimates of the states (into AHATPF) and their variance (into PHAT), with upper and lower 90% bounds.⁶

```
set ahatpf = alphahat(t)(1)
set phat   = alphahatxx(t)(1,1)-ahatpf(t)^2
set lowerpf = ahatpf+sqrt(phat)*%invnormal(.05)
set upperpf = ahatpf+sqrt(phat)*%invnormal(.95)
```

These are graphed (Figure 9.3) with comparisons to the same model analyzed with the Kalman filter itself.

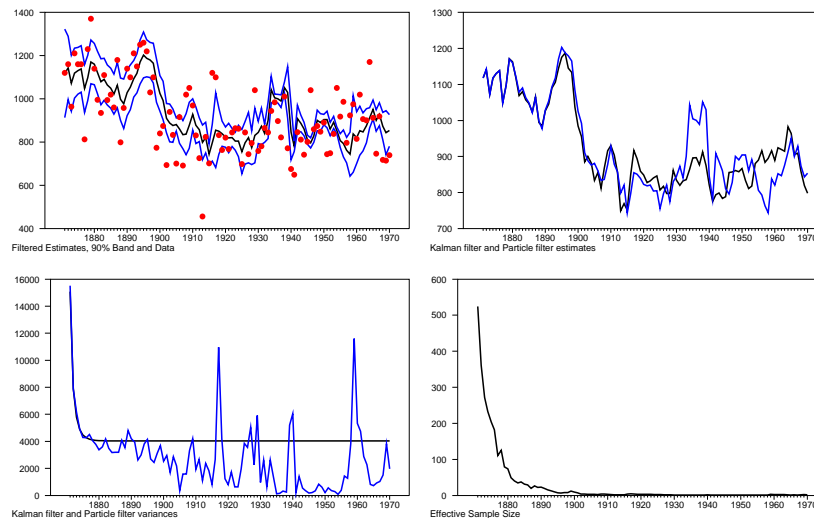


Figure 9.3: Bootstrap Particle Filter without Resampling

Needless to say, the results are rather disappointing. The estimates are fairly reasonable for about half the sample and then fall apart. This is hardly a surprise when you look at the effective sample size, which starts at around

⁶Note that the correction factors in (G.2) were already incorporated when we normalized the WT vector before doing the calculation.

500 (out of 10000) and drops fairly rapidly. What happens? The initial values are a wild set of random numbers which have very little to do with the data. Most particles start at very wrong values, and the process (which chooses to update each based upon its past value) gets very little chance for any particle that starts in the tail to ever get back to the actual data range. On the other hand, note that the estimates do quite a good job of tracking until the effective sample size deteriorates to below about 50, which suggests that, with a better sampling procedure, this should work.

The simplest fix is to “resample” the particles at each time period. We’re running into problems with this simple calculation because each particle is attempting to track the behavior of the system over time. That might be important if we were trying to get *smoothed* estimates, but not if we’re only filtering. At the end of period t , we have a collection of probability weighted values for X_t . Instead of passing them through in that form to $t+1$, we can draw a new set of (equally weighted) values by drawing with replacement using those probability weights. This breaks the connection between elements of $X_{1:t-1}$ and $X_{1:t}$, but again, as long as we’re only interested in filtering, that’s not a problem. The code loop is the same as before, except that at the end of a trip through the data at an entry, we use the following to resample the values:

```
do i=1,npart
  compute redraw=%ranbranch(wt)
  compute alphanew(i)=alpha(time)(redraw)
end do i
```

`%RANBRANCH(WT)` draws a position in the vector of particles using the probability weights given by `WT`. Because we need *all* the original values of X_t until we’re done with the resampling, the sampled values are put first into `ALPHANEW` (a `VECT[VECT]`). Once we’re done with all the sampling, `ALPHA(TIME)` is replaced with the resampled values, and the weights are reset to be equal. (Any equal number is fine, so we use log weights of zero).

```
compute alpha(time)=alphanew
compute logw(time)=%zeros(npart,1)
```

This greatly improves the results (Figure 9.4). The initial effective sample size is still roughly 500 since that’s still a blind draw. But after observing y_1 and resampling, the sample size pops up to over 5000 and generally stays above that.

Particle filtering is still a fairly young topic and improvements are being made to it fairly regularly (in many disciplines—such as engineering). Durbin & Koopman (2012) includes some recent suggestions for improving on the methods used in Example 9.4.

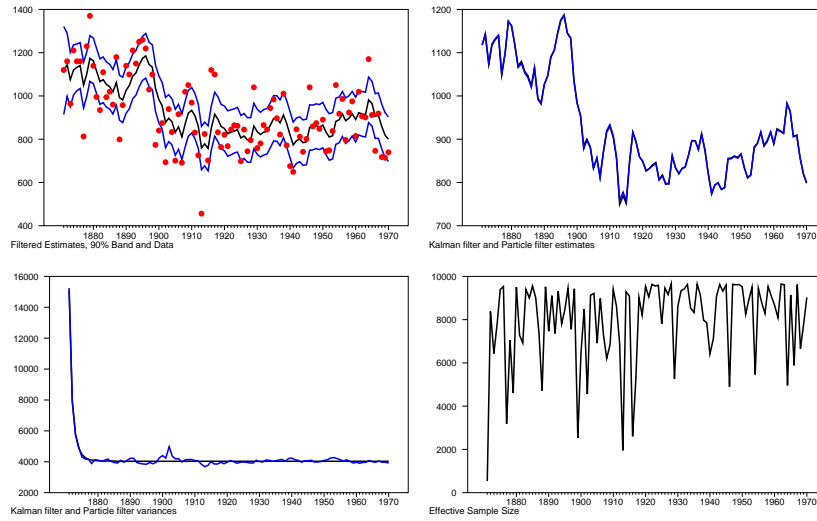


Figure 9.4: Bootstrap Particle Filter with Resampling

9.5 Restrictions on the States

Equality restrictions on the states are straightforward, and, in fact, we’ve already used them without a second thought. For instance, the gap model has a measurement equation with $U_t \equiv N_t + G_t$, which will force the states underlying N and G to be less than full rank. This works because an equality restriction on a Normal distribution still produces a Normal distribution, just with reduced rank, and there is never a requirement that the states themselves be full rank—just the predictive density.

Inequality restrictions are a very different situation and there are no simple solutions for them. Simon (2010) offers a survey of various techniques of approximate solution for state-space models with inequality constraints. (The paper starts with equality constraints). To see why this is a problem, consider the case of a local level model

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

but add in the constraint that $x_t \leq 1$. If we look at filtering inference for x_1 , the pre-sample information will give us x_1 as diffuse, combined with $y_1 \sim N(x_1, \sigma_\varepsilon^2)$. For Bayesian inference, the latter can be reversed to $x_1 \sim N(y_1, \sigma_\varepsilon^2)$. If it weren’t for the constraint, this would be the filtered distribution for x_1 . *With* the constraint, x_1 will be the same Normal, but truncated above at 1. If we try to roll this forward to x_2 , we no longer have a convenient distribution: the predictive density of x_2 is the sum of the truncated Normal and an independent untruncated Normal, and that has no closed form. If we just have the one truncation at each point, it would be possible to compute the mean and variance of the truncated distribution, and roll *that* forward to Kalman filter as the best *linear* projection (rather than maximum likelihood). This is what Simon calls

the PDF method. However, if we have *multiple* constraints per time period, there are no closed formulas for the adjusted means and covariance matrices.⁷ One possibility (which is discussed in the Simon survey article) is to use particle filtering (Section 9.4). However, that requires that the state-space model actually be fully known, which is probably a more reasonable assumption in the engineering literature than in economics where the parameters typically have to be estimated.

An alternative is, at each stage, to compute the standard unconstrained solution of mean $\mathbf{x}_{t|t}$ and covariance matrix $\Sigma_{t|t}$ and then compute the \mathbf{x} which minimizes:

$$(\mathbf{x} - \mathbf{x}_{t|t})' (\Sigma_{t|t})^{-1} (\mathbf{x} - \mathbf{x}_{t|t}) \quad (9.24)$$

subject to the constraints. That's a standard quadratic programming problem which Simon calls *Estimate Projection*.⁸ In the simple case above, that would mean taking 1 if $y_1 > 1$ and y_1 otherwise. With a single constraint, this is always easy, since the constraint is either binding (so you take the constrained value) or not (so you take the $\mathbf{x}_{t|t}$). With multiple constraints at a given t , it's not as easy, since it's necessary to check for different combinations of active constraints to see which combination yields the best value for (9.24).

That will give a new $\hat{\mathbf{x}}_{t|t}$ estimate of the filtered state that satisfies the constraints. The next question is what to do to roll this forward to the next period. There are two choices that seem to suggest themselves for handling the covariance matrix:

1. Use the standard calculation, ignoring whether the constraints are binding or not.
2. Use the reduced rank calculation, conditioning on the binding constraints (if any).

For the simple case that we've been examining, the two would be the same unless the constraint is binding at a given point, when the second would use 0. This will obviously give a different idea about the uncertainty of the filtered state, but in practice probably won't give *too* much of a difference in the estimates of *future* states.

Another possibility is to use a non-linear Kalman filter (Section 9.3) using a functional form for x_t which has the desired range. With the upper bound at 1, for instance, we could use $x_t \approx 1 - \exp(-\chi_t)$, or a system of

$$\begin{aligned} y_t &= 1 - \exp(-\chi_t) + v_t \\ \chi_t &= \chi_{t-1} + w_t \end{aligned}$$

⁷This is closely related to the problem of estimating multinomial probit models, but harder because there might be some doubly truncated states, and the multinomial probit doesn't require solving a whole sequence of optimization problems.

⁸ $\Sigma_{t|t}$ isn't necessarily invertible, which is why the generalized inverse is used.

x_t can never be exactly one, but can get arbitrarily close (and note that the mean could never be exactly one anyway if we could do exact calculations). The one tweak that is needed is that the variances in the evolution of χ_t and that of x_t will be different. Now, with the bounds on x is not really possible for η in

$$x_t = x_{t-1} + \eta_t$$

to be mean zero with constant variance when x_{t-1} is near the boundary, so even *that* has to be an approximation. However, if we linearize the random walk in x around a common expansion point, we get

$$\begin{aligned} 1 - \exp(-\tilde{\chi}_t) + \exp(-\tilde{\chi}_t)(\chi_t - \tilde{\chi}_t) &\approx 1 - \exp(-\tilde{\chi}_t) + \exp(-\tilde{\chi}_t)(\chi_{t-1} - \tilde{\chi}_t) + \eta_t \\ \Rightarrow \exp(-\tilde{\chi}_t)(\chi_t - \chi_{t-1}) &\approx \eta_t \end{aligned}$$

Thus, if the original model had $\text{var}(\eta_t) = \sigma_\eta^2$, the variance in the random walk for the χ_t index will be the time-varying

$$\exp(+2\tilde{\chi}_t)\sigma_\eta^2 \quad (9.25)$$

In practice, $\tilde{\chi}_t$ has to be truncated so this doesn't get too large: placing an upper bound of 3 (in this case), seems to work fine. As in this case, a (translated) exponential can be used for a single bound on a state, while a logistic (possibly with scaling and translation) can be used to handle upper and lower bounds—the logistic would need upper and lower bounds on the expansion points for the same reason.

Example 9.5 demonstrates various techniques applied to the local level model with the state variable truncated above at 1. The data is actually generated using an untruncated model (with a particular random number seed, which causes it to spend a considerable amount of time above 1), so we can see how the various methods deal with the boundary. Note that several of the methods require some features added to Version 9.2 to allow on-the-fly adjustments to the state estimates and covariance matrices. All of these treat all the component variances as known (at the values used in simulating the data).

The first method is to use estimate projection with the standard filtered covariance matrix passed through. So the `FPOST` function fixes the mean if it's out-of-bounds but does nothing to the covariance matrix.

```
function WithStdSigma xt vt
type vector *xt
type symm    *vt
*
if xt(1)>1.0
    compute xt(1)=1.0
end
*
dlim(fpost=WithStdSigma,y=y,c=1.0,sw=sigsqw,sv=sigsqv,$
    presample=diffuse,type=filter) / xstates_std vstates_std
set x_std = xstates_std(t)(1)
```

The same thing, but with the *constrained* covariance matrix is done with

```
function WithConSigma xt vt
type vector *xt
type symm    *vt
*
if xt(1)>1.0
    compute xt(1)=1.0, vt(1,1)=0.0
end
*
dlm(fpost=WithConSigma,y=y,c=1.0,sw=sigsqw,sv=sigsqv,$
    presample=diffuse,type=filter) / xstates_con vstates_con
set x_con = xstates_con(t)(1)
```

If the constrained mean is taken, its variance is zeroed out.

This calculates the truncated mean and variance. Note that these assume the conditional distribution is Normal—the truncated mean and variance calculations *do* depend upon the distribution.

```
function WithTruncation xt vt
type vector *xt
type symm    *vt
*
local real beta sigma mills
compute sigma=sqrt(vt(1,1))
compute beta=(1.0-xt(1))/sigma
compute mills=%mills(beta)
compute xt(1)-=sigma*mills
compute vt(1,1)=vt(1,1)*(1-beta*mills-mills^2)
end
*
dlm(fpost=WithTruncation,y=y,c=1.0,sw=sigsqw,sv=sigsqv,$
    presample=diffuse,type=filter) / xstates_trunc vstates_trunc
set x_trunc = xstates_trunc(t)(1)
```

This uses the exponential approximation with the predicted state as the expansion point. Note that the expansion point series (\tilde{X}_T) is truncated at 3 as described above.

```

set xtilde = 0.0
*
function EKFPredicted xt
type vector *xt
*
compute xtilde(t)=%min(xt(1),3.0)
end
*
frml muf = 1-exp(-xtilde)-xtilde*exp(-xtilde)
frml cf  = exp(-xtilde)
frml swf = exp(+2.0*xtilde)*sigsqw
*
dln(fpre=EKFPredicted,y=y,a=1.0,mu=muf,c=cf,sw=swf,sv=sigsqv,$
    presample=diffuse) / xstates_exp0 vstates_exp0
set x_exp0 = 1-exp(-xstates_exp0(t)(1))

```

This is the same thing, but uses a smoothed estimate for the expansion points. (The MUF, CF and SWF are all the same as the ones above).

```

dln(fpre=EKFPredicted,y=y,a=1.0,mu=muf,c=cf,sw=swf,sv=sigsqv,$
    presample=diffuse,type=smooth) / xstates_sm
set xtilde = %min(xstates_sm(t)(1),3.0)
dln(y=y,a=1.0,mu=muf,c=cf,sw=swf,sv=sigsqv,$
    presample=diffuse) / xstates_exp1 vstates_exp1
set x_exp1 = 1-exp(-xstates_exp1(t)(1))

```

Finally, this does a particle filter with resampling, using a “blind” draw for $x_{t|t-1}$, that is, a draw not taking the value of y into account. This simplifies the filter quite a bit ($g = f$), but works well here only because of the close connection between x and y .


```

compute npart=10000
dec vect xpart(npart) wt(npart) xresample(npart)
ewise xpart(i)=%uniform(-1.0,1.0)
set x_pf 1 400 = 0.0
do time=1,400
  do i=1,npart
    compute xtest=%rantruncate(xpart(i),sqrt(sigsqw),%na,1.0)
    compute ptest=%logdensity(sigsqv,y(time)-xtest)
    compute wt(i)=ptest
    compute xpart(i)=xtest
  end do i
  compute maxlogw=%maxvalue(wt)
  ewise wt(i)=exp(wt(i)-maxlogw)
  compute wt=wt/%sum(wt)
  compute x_pf(time)=%dot(wt,xpart)
  ewise xresample(i)=xpart(%ranbranch(wt))
  compute xpart=xresample
end do time

```

This gives us six sets of estimates—the program graphs three pairs, using the particle filter as the “gold standard”. One thing to note is that all six give almost identical results whenever the state is well away from the boundary, so the differences are only in the handling of the situation *near* the boundary. The first two (which take the projected state) are very similar, with just some very slight differences where the first (Figure 9.5, which uses the unconstrained covariance matrix) moves a bit farther away from the boundary than the alternative that zeros out the filtered variance. The two exponential methods are almost identical, Figure 9.6 showing the one with the smoothed expansion point and are very similar to the projection results as well. Perhaps not surprisingly, Figure 9.7, which uses the mean and variance of the truncated distribution, produces an almost identical result to the particle filter. (The black graph for the particle filter is almost completely overwritten by the blue for the truncated estimates). Unfortunately, if you have more than one state that’s being constrained, there is no simple formula for computing these.

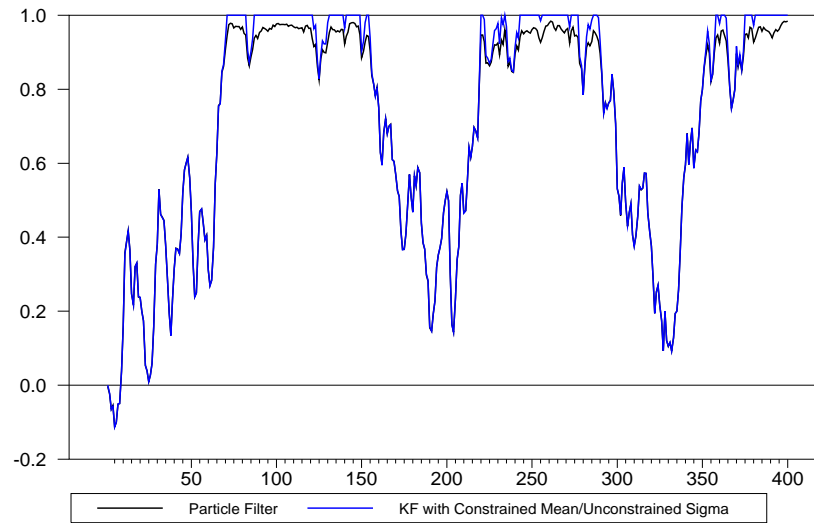


Figure 9.5: Using Constrained Mean/Unconstrained Variance

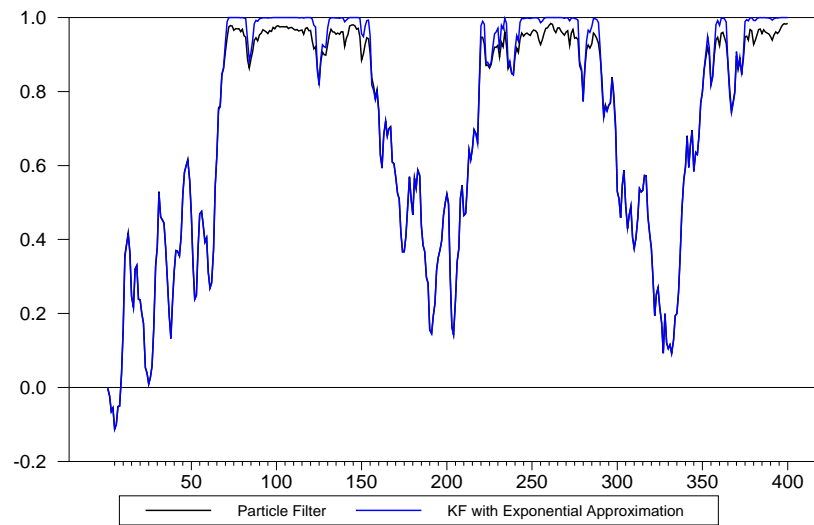


Figure 9.6: Using Exponential Approximation

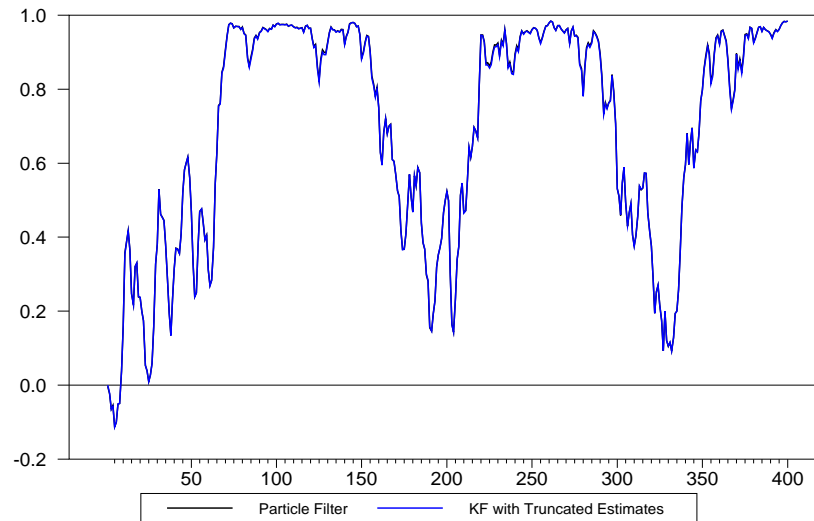


Figure 9.7: Using Truncated Distribution Mean/Variance

Example 9.1 Stochastic Volatility Model

This is the example from Section 9.1. It's adapted from an example in Durbin and Koopman's first edition.

```
open data sv.dat
data(format=free,skip=1) 1 945 xrate
*
* Note - Harvey, Ruiz and Shephard took the (sample) mean out of the
* return series before squaring.
*
compute meanx2=%digamma(0.5)-log(0.5)
set ysq = log(xrate^2)-meanx2
*
nonlin phi sw sv kappa
compute phi=.9,sw=.011,sv=1.0,kappa=0.0
*
* Estimate model and compute diagnostics on the recursive residuals
*
dln(y=ysq,a=phi,c=1.0,z=kappa,sw=sw,sv=sv,$
    presample=ergodic,vhat=vhat,svhat=svhat,$
    method=bfgs,reject=abs(phi)>=1.0) 1 945 states
*
set recresid = %scalar(vhat)/sqrt(%scalar(svhat))
@STAMPDiags(ncorrs=29) recresid
*
* Recalculate smoothed values
*
dln(y=ysq,a=phi,c=1.0,z=kappa,sw=sw,sv=sv,$
    presample=ergodic,type=smooth) 1 945 states
set stderrs = exp(.5*%scalar(states))
graph(footer="Fig 9.12 Estimated Volatility for the Pound Series")
# stderrs
```

```

*
* Empirical estimate, using exponential smoothing on squared xrate.
*
set xratesqr = xrate^2
esmooth(alpha=.02,smoothed=xratevol) xratesqr 1 945
set xratevol = sqrt(xratesqr)
graph(key=upleft,klabels=|"Empirical Estimate","Model Estimate"|) 2
# xratevol
# stderrs

```

Example 9.2 Fat-Tailed Errors

This is from DK, their example 14.3. This is the example from Section 9.2.

```

open data gas.dat
cal(q) 1960
data(format=free,skip=1) 1 107 gas
*
* Set up the structural model with additive seasonal and local trend
* with no level shock.
*
@SeasonalDLM(type=additive,a=as,c=cs,f=fs)
@LocalDLM(type=trend,a=al,c=cl,f=fl)
compute c=cl~~cs
compute a=al~~as
compute f=fl~~fs
*
* Estimate with Gaussian errors. This concentrates out the variance in
* the measurement equation, and estimates the variance in the trend
* rate and seasonal disturbances in log relative standard error form.
*
nonlin phil phis
compute phil=-2.0,phis=-4.0
dln(start=(sw=%diag(||exp(2*phil),exp(2*phis)||)),a=a,c=c,f=f,$
    y=gas,sv=1.0,sw=sw,exact,var=conc,method=bfgs,$
    vhat=vhat,type=smooth) / xstates vstates
*
* Pull out seasonal and irregular
*
set seasonal = xstates(t) (3)
set irreg    = vhat(t) (1)
spgraph(vfields=2,$
    footer="Fig 14.4 Analyses of gas data based upon Gaussian model")
graph(header="Gaussian conditional seasonal")
# seasonal
graph(header="Gaussian conditional irregular")
# irreg
spgraph(done)
*
compute nu=50.0
compute veps=%variance

```

```

set eps = 0.0
compute lastone=0
do iters=1,20
  compute lastnu=nu
  set fiddle = ((nu-2)+eps**2/veps)/(nu+1)
  dlm(start=(sw=%diag(||exp(2*phil),exp(2*phis)||)),a=a,c=c,f=f,$
    y=gas,sv=fiddle,sw=sw,exact,var=conc,method=bfgs,$
    vhat=vhat,type=smooth,print=lastone) / xstates vstates
  if lastone
    break
  set eps = vhat(t)(1)
  stats(noprint) eps
  *
  * Use method of moments to estimate nu and sigma epsilon
  *
  compute nu=4.0+6.0/%kurtosis
  compute veps=%variance
  compute lastone=fix((abs(nu-lastnu)<.01))
end do iters
*
set seasonal = xstates(t)(3)
set irreg    = vhat(t)(1)
spgraph(vfields=2,$
  footer="Fig 14.5 Analyses of gas data based upon t-model")
graph(header="t conditional seasonal")
# seasonal
graph(header="t conditional irregular")
# irreg
spgraph(done)

```

Example 9.3 Non-Linear Kalman Filter

This is based upon Matheson & Stavrev (2013) with a reconstructed (and extended) data set, estimating a state-space model with a non-linear measurement equation. This does *not* impose constraints on the states (time-varying coefficients). It's discussed in Section 9.3.

```
*
* Get styles to do thin and fat colored lines for the estimates and
* bounds.
*
open styles mscolors.txt
grparm(import=styles)
*
cal(q) 1960
*
* This is a reconstructed data set extended to 2016:4. (Note:
* PTRPCE is the long-term expected PCE inflation which is not used).
*
open data msexteneded.rat
data(format=rats) 1960:01 2016:04 ptrcpi ptrpce cpi unrate mdef
*
set u      = unrate
set pi     = 400.0*log(cpi/cpi{1})
set pim    = 400.0*log(mdef/mdef{1})-pi
set pibar  = ptrcpi
set pistar = .25*(pi+pi{1}+pi{2}+pi{3})
*
@nbercycles(down=nber)
*
* States are u-u* (ugap), u*, kappa, theta, gamma in that order.
* The observables are pi and u.
*
* These are the expansion points for the measurement equation for
* pi.
*
dec series kappa_0 ugap_0
dec series theta_0 gamma_0
*
* Initialize the linearization with an HP filtered unemployment for ustar
*
filter(type=hp) u / ustar_0
*
set ugap_0 = u-ustar_0
*
* Do rolling regressions
*
nonlin(parmset=rollparms) kappa_r theta_r gamma_r
frml rollpi pi = pistar{1}+theta_r*(pibar-pistar{1})-$
      kappa_r*ugap_0+gamma_r*pim
*
* Span for rolling regressions, first entry available for
```

```

* estimation (we lose four data points in computing pistar, then
* use a lag of pistar as well), first end point for which rolling
* regressions can be done and last entry available for estimation.
*
compute span    =40
compute sstart  =1961:02
compute rstart  =sstart+span-1
compute send    =2016:04
*
* This does the full sample regression
*
compute kappa_r=0.1,gamma_r=0.1,theta_r=0.5
nlls(parmset=rollparms,frml=rollpi) pi sstart send
compute sv=%sigmasq
*
* This does rolling sample regressions and saves the series of
* estimated coefficients.
*
clear(zeros) kappa_re theta_re gamma_re
do time=rstart,send
    compute kappa_r=0.1,gamma_r=0.1,theta_r=0.5
    nlls(parmset=rollparms,frml=rollpi,noprint) pi time-(span-1) time
    compute kappa_re(time)=kappa_r
    compute theta_re(time)=theta_r
    compute gamma_re(time)=gamma_r
end do time
*
set kappa_0 = %if(t<=rstart,kappa_re(rstart),kappa_re)
set theta_0 = %if(t<=rstart,theta_re(rstart),theta_re)
set gamma_0 = %if(t<=rstart,gamma_re(rstart),gamma_re)
*
* Compute variances of changes in the coefficients to use as upper
* bound on coefficient shock variance.
*
sstats(mean) rstart+1 send (kappa_0-kappa_0{1})^2>>sigsqup_kappa $
                                (theta_0-theta_0{1})^2>>sigsqup_theta $
                                (gamma_0-gamma_0{1})^2>>sigsqup_gamma

*
linreg ugap_0
# ugap_0{1}
compute rho=%beta(1)
compute swugap=%sigmasq
*
dec frml[vect] muf
dec frml[rect] cf
dec frml[symm] svf
dec frml[symm] swf
*
frml muf = ||+kappa_0*ugap_0+pistar{1},0.0||
*
frml cf = ||-kappa_0      ,1.0|$
           0.0           ,1.0|$
           -ugap_0       ,0.0|$
           (pibar-pistar{1}),0.0|$

```

```

                                pim          ,0.0||
*
dec vector swtvc(4)
*
frml svf = %diag(||sv,0.0||)
frml swf = %diag(swugap~~swtvc)
*
nonlin(parmset=base) rho sv swugap swtvc
*
* These are the two variance ratio pegs
*
nonlin(parmset=stable)    swugap=15.0*swtvc(1)
nonlin(parmset=flexible) swugap=5.0*swtvc(1)
*
* A is an identity matrix other than the 1,1 element. %%DLMSsetup
* will reset that to the current test value of rho.
*
dec rect a(5,5)
compute a=%na~\%identity(4)
*
function %%DLMSsetup
compute a(1,1)=rho
end
*
nonlin(parmset=limits) swtvc(1)>=0.0 rho<=.95 $
                        swtvc(2)>=0.0 swtvc(2)<=sigsqup_kappa $
                        swtvc(3)>=0.0 swtvc(3)<=sigsqup_theta $
                        swtvc(4)>=0.0 swtvc(4)<=sigsqup_gamma

compute swtvc(1)=.1
compute swtvc(2)=.5*sigsqup_kappa
compute swtvc(3)=.5*sigsqup_theta
compute swtvc(4)=.5*sigsqup_gamma
*
* Do one Kalman smooth pass to get expansion points for future
* non-linear approximations.
*
dlm(start=%%DLMSsetup(),a=a,c=cf,y=||pi,u||,mu=muf,sv=svf,sw=swf,$
    presample=ergodic,type=smooth) sstart send xstates vstates
set ugap_0 = xstates(t)(1)
set kappa_0 = xstates(t)(3)
*
dec hash[parmset] pegs
compute pegs("stab")=stable
compute pegs("flex")=flexible
*
dec hash[hash[hash[series]]] h_ustar h_kappa h_theta h_gamma
dec hash[hash[series]] h_pie
dec hash[series[vect]] h_xstates
dec hash[series[symm]] h_vstates
*
dec string root ktype
do for root = "stab" "flex"
    dlm(start=%%DLMSsetup(),a=a,c=cf,y=||pi,u||,mu=muf,sv=svf,sw=swf,$
        type=filter,presample=ergodic,$

```



```

    parmset=base+pegs(root)+limitsmethod=bfgs,condition=10) $
    sstart send h_xstates("kf") h_vstates("kf")
*
dlim(start=%DLMSetup(),a=a,c=cf,y=|pi,u|,mu=muf,sv=svf,sw=swf,$
    type=smooth,presample=ergodic) $
    sstart send h_xstates("sm") h_vstates("sm")
*
dofor ktype = "kf" "sm"
*
* NAIRU estimates
*
set h_ustar(root)(ktype)("est") = h_xstates(ktype)(t)(2)
set h_ustar(root)(ktype)("hi") = h_xstates(ktype)(t)(2)+$
                                sqrt(h_vstates(ktype)(t)(2,2))
set h_ustar(root)(ktype)("lo") = h_xstates(ktype)(t)(2)-$
                                sqrt(h_vstates(ktype)(t)(2,2))
*
* Kappa
*
set h_kappa(root)(ktype)("est") = h_xstates(ktype)(t)(3)
set h_kappa(root)(ktype)("hi") = h_xstates(ktype)(t)(3)+$
                                sqrt(h_vstates(ktype)(t)(3,3))
set h_kappa(root)(ktype)("lo") = h_xstates(ktype)(t)(3)-$
                                sqrt(h_vstates(ktype)(t)(3,3))
*
* Theta
*
set h_theta(root)(ktype)("est") = h_xstates(ktype)(t)(4)
set h_theta(root)(ktype)("hi") = h_xstates(ktype)(t)(4)+$
                                sqrt(h_vstates(ktype)(t)(4,4))
set h_theta(root)(ktype)("lo") = h_xstates(ktype)(t)(4)-$
                                sqrt(h_vstates(ktype)(t)(4,4))
*
* Gammas
*
set h_gamma(root)(ktype)("est") = h_xstates(ktype)(t)(5)
set h_gamma(root)(ktype)("hi") = h_xstates(ktype)(t)(5)+$
                                sqrt(h_vstates(ktype)(t)(5,5))
set h_gamma(root)(ktype)("lo") = h_xstates(ktype)(t)(5)-$
                                sqrt(h_vstates(ktype)(t)(5,5))
*
* Inflation expectations (uses time-varying weights on
* observable data).
*
set h_pie(root)(ktype) = h_theta(root)(ktype)("est")*pibar+$
                        (1-h_theta(root)(ktype)("est"))*pistar{1}
end do ktype
end do root
*
compute title="TVC Unconstrained"
source msgraphs.src

```

This is the source file to do graphics for the Matheson & Stavrev (2013) programs. This does not do the actual/predicted inflation—there is something

clearly wrong with the graph in the paper. First off, the smoothed "prediction" should be (by definition) equal to the data. Second, the one-step ahead prediction would never hit outliers that well.

This uses **TABLE** instructions on most of the batches of graphs to get the overall mean and maximum from the series so the filtered and smoothed graphs use a common scale. This looks at the two upper and lower bounds series plus the rolling estimates, as the extreme values will have to be in one of those.

```
spgraph(footer="Unemployment and Inflation("+title+")",vfields=2,hfields=2,$
  xlabel=||"FILTERED","SMOOTHED"||,fillby=rows)
*
do for ktype = "kf" "sm"
  graph(shading=nber,header="Unemployment and the NAIRU") 7
  # u
  # h_ustar("stab")(ktype)("est") sstart * 2
  # h_ustar("stab")(ktype)("lo") sstart * 12
  # h_ustar("stab")(ktype)("hi") sstart * 12
  # h_ustar("flex")(ktype)("est") sstart * 3
  # h_ustar("flex")(ktype)("lo") sstart * 13
  # h_ustar("flex")(ktype)("hi") sstart * 13
end do ktype
*
do for ktype = "kf" "sm"
  graph(shading=nber,header="Inflation and Inflation Expectations") 4
  # pi
  # h_pie("stab")(ktype)
  # h_pie("flex")(ktype)
  # pibar
end do ktype
spgraph(done)
*
spgraph(footer="Estimated Parameters("+title+")",vfields=3,hfields=2,$
  xlabel=||"FILTERED","SMOOTHED"||,fillby=rows)
*
do for ktype = "kf" "sm"
  *
  * Get the max and min across the filtered stable and flexible
  * models to put the two columns on common scale.
  *
  table(noprint) sstart+10 send theta_re $
    h_theta("stab")("kf")("lo") h_theta("stab")("kf")("hi") $
    h_theta("flex")("kf")("lo") h_theta("flex")("kf")("hi")
  graph(shading=nber,header="theta",max=%maximum,min=%minimum) 7
  # theta_re rstart send
  # h_theta("stab")(ktype)("est") sstart+10 * 2
  # h_theta("stab")(ktype)("lo") sstart+10 * 12
  # h_theta("stab")(ktype)("hi") sstart+10 * 12
  # h_theta("flex")(ktype)("est") sstart+10 * 3
  # h_theta("flex")(ktype)("lo") sstart+10 * 13
  # h_theta("flex")(ktype)("hi") sstart+10 * 13
end do ktype
*
```

```

do for ktype = "kf" "sm"
  table(noprint) sstart+10 send kappa_re $
    h_kappa("stab")("kf")("lo") h_kappa("stab")("kf")("hi") $
    h_kappa("flex")("kf")("lo") h_kappa("flex")("kf")("hi")
  graph(shading=nber,header="kappa",max=%maximum,min=%minimum) 7
  # kappa_re rstart send
  # h_kappa("stab")(ktype)("est") sstart+10 * 2
  # h_kappa("stab")(ktype)("lo") sstart+10 * 12
  # h_kappa("stab")(ktype)("hi") sstart+10 * 12
  # h_kappa("flex")(ktype)("est") sstart+10 * 3
  # h_kappa("flex")(ktype)("lo") sstart+10 * 13
  # h_kappa("flex")(ktype)("hi") sstart+10 * 13
end do ktype
do for ktype = "kf" "sm"
  table(noprint) sstart+10 send gamma_re $
    h_gamma("stab")("kf")("lo") h_gamma("stab")("kf")("hi") $
    h_gamma("flex")("kf")("lo") h_gamma("flex")("kf")("hi")
  graph(shading=nber,header="gamma",max=%maximum,min=%minimum) 7
  # gamma_re rstart send
  # h_gamma("stab")(ktype)("est") sstart+10 * 2
  # h_gamma("stab")(ktype)("lo") sstart+10 * 12
  # h_gamma("stab")(ktype)("hi") sstart+10 * 12
  # h_gamma("flex")(ktype)("est") sstart+10 * 3
  # h_gamma("flex")(ktype)("lo") sstart+10 * 13
  # h_gamma("flex")(ktype)("hi") sstart+10 * 13
end do ktype
spgraph(done)

```

Example 9.4 Particle Filtering

Based upon Durbin & Koopman (2012), illustration from their section 12.4.4. This is the example from Section 9.4.

```
*seed 55329
open data nile.dat
calendar 1871
data(format=free,org=columns,skips=1) 1871:1 1970:1 nile
*
* Standard Kalman filter
*
dlm(a=1.0,c=1.0,x0=0.0,sx0=1.e+7,sv=15099.0,sw=1469.1,presample=x1,$
    y=nile,vhat=vhat,svhat=fhat) / xstates vstates
*
* Get the KF estimates of the state and variance
*
set ahatkf = xstates(t)(1)
set phatkf = vstates(t)(1,1)
*
* Bootstrap filter without resampling
*
compute sigmaeta=sqrt(1469.1)
compute sigma_x0 =sqrt(1.e+7)
compute sigmaepssq=15099.0
*
* Set up the particles
*
compute nstate=1
compute npart=10000
*
* This can be done without saving the entire histories of alpha and logw
* - just keeping the vectors from the last time period.
*
dec vect[vect] alphanew(npart)
dec series[vect[vect]] alpha
ewise alphanew(i)=%zeros(nstate,1)
gset alpha = alphanew
*
dec series[vect] logw
dec vect wt(npart)
gset logw = %zeros(npart,1)
*
dec series[vect] alphahat
dec series[symm] alphahatxx
*
gset alphahat = %zeros(nstate,1)
gset alphahatxx = %zeros(nstate,nstate)
*
set ess = 0.0
*
do time=1871:1,1970:1
    do part=1,npart
```

```

    if time==1871:1 {
        compute alpha(time) (part)=sigmax0*%ranmat(1,1)
        compute laggedlogw=0.0
    }
    else {
        compute alpha(time) (part)=alpha(time-1) (part)+%ran(sigmaeta)
        compute laggedlogw=logw(time-1) (part)
    }
    compute logw(time) (part)=laggedlogw+$
        %logdensity(sigmaepssq,nile(time)-alpha(time) (part) (1))
end do part
*
* Do overflow-safe normalization of weights
*
compute maxlogw=%maxvalue(logw(time))
compute logw(time)=logw(time)-maxlogw
compute wt=%exp(logw(time))
compute wt=wt/%sum(wt)
*
do part=1,npart
    compute alphahat(time)=alphahat(time)+$
        wt(part)*alpha(time) (part)
    compute alphahatxx(time)=alphahatxx(time)+$
        wt(part)*%outerxx(alpha(time) (part))
end do part
compute ess(time)=1.0/%normsqr(wt)
end do time
*
set ahatpf = alphahat(t) (1)
set phat   = alphahatxx(t) (1,1)-ahatpf(t)^2
set lowerpf = ahatpf+sqrt(phat)*%invnormal(.05)
set upperpf = ahatpf+sqrt(phat)*%invnormal(.95)
*
spgraph(vfields=2,hfields=2,$
    footer="Figure 12.1 Bootstrap filter with N=10000 without resampling")
graph(footer="Filtered Estimates, 90% Band and Data",overlay=dots,ovsame) 4
# ahatpf
# lowerpf / 2
# upperpf / 2
# nile
graph(row=1,col=2,footer="Kalman filter and Particle filter estimates") 2
# ahatkf
# ahatpf
graph(row=2,col=1,footer="Kalman filter and Particle filter variances") 2
# phatkf
# phat
graph(row=2,col=2,footer="Effective Sample Size") 1
# ess
spgraph(done)
*
* With resampling
*
* Clear the accumulators
*

```

```

gset alphahat    = %zeros(nstate,1)
gset alphahatxx = %zeros(nstate,nstate)
set  ess        = 0.0
*
do time=1871:1,1970:1
  do part=1,npart
    if time==1871:1 {
      compute alpha(time)(part)=sigmax0*%ranmat(1,1)
      compute laggedlogw=0.0
    }
    else {
      compute alpha(time)(part)=alpha(time-1)(part)+%ran(sigmaeta)
      compute laggedlogw=logw(time-1)(part)
    }
    compute logw(time)(part)=laggedlogw+$
      %logdensity(sigmaepssq,nile(time)-alpha(time)(part)(1))
  end do part
  compute maxlogw=%maxvalue(logw(time))
  compute logw(time)=logw(time)-maxlogw
  compute wt=%exp(logw(time))
  compute wt=wt/%sum(wt)
  *
  do part=1,npart
    compute alphahat(time)=alphahat(time)+$
      wt(part)*alpha(time)(part)
    compute alphahatxx(time)=alphahatxx(time)+$
      wt(part)*%outerxx(alpha(time)(part))
  end do part
  compute ess(time)=1.0/%normsqr(wt)
  *
  * Resample, using %RANBRANCH to stratify based upon the current
  * weights.
  *
  do i=1,npart
    compute redraw=%ranbranch(wt)
    compute alphanew(i)=alpha(time)(redraw)
  end do i
  compute alpha(time)=alphanew
  *
  * Reset the probabilities to be even across particles
  *
  compute logw(time)=%zeros(npart,1)
end do time
*
set phat    = alphahatxx(t)(1,1)-alphahat(t)(1)^2
set ahatpf  = alphahat(t)(1)
set lowerpf = ahatpf+sqrt(phat)*%invnormal(.05)
set upperpf = ahatpf+sqrt(phat)*%invnormal(.95)
*
spgraph(vfields=2,hfields=2,$
  footer="Figure 12.2 Bootstrap filter with N=10000 with resampling")
graph(footer="Filtered Estimates, 90% Band and Data",overlay=dots,ovsame) 4
# ahatpf
# lowerpf / 2

```

```
# upperpf / 2
# nile
graph(row=1,col=2,footer="Kalman filter and Particle filter estimates") 2
# ahatkf
# ahatpf
graph(row=2,col=1,footer="Kalman filter and Particle filter variances") 2
# phatkf
# phat
graph(row=2,col=2,footer="Effective Sample Size") 1
# ess
spgraph(done)
```

Example 9.5 Constrained Kalman Filter

This estimates the local level model with state governed by inequality constraint using various techniques. This is described in Section 9.5.

```

compute sigsqp=0.04
compute sigsqv=0.01
compute sigsqw=0.01
*
* This generates a set of data which spend a considerable amount of
* time above 1, so forces the boundary to be in play for large
* stretches.
*
seed 50032
set(first=0.0) y 1 400 = y{1}+%ran(.10)
graph
# y
*
* Constrained KF, using constrained optimum with unconstrained
* filtered state variance.
*
function WithStdSigma xt vt
type vector *xt
type symm *vt
*
if xt(1)>1.0
compute xt(1)=1.0
end
*
dlm(fpost=WithStdSigma,y=y,c=1.0,sw=sigsqw,sv=sigsqv,$
presample=diffuse,type=filter) / xstates_std vstates_std
set x_std = xstates_std(t) (1)
*
* Constrained KF, using constrained optimum with constrained
* filtered state variance (which is zero in this case).
*
function WithConSigma xt vt
type vector *xt
type symm *vt
*
if xt(1)>1.0
compute xt(1)=1.0,vt(1,1)=0.0
end
*
dlm(fpost=WithConSigma,y=y,c=1.0,sw=sigsqw,sv=sigsqv,$
presample=diffuse,type=filter) / xstates_con vstates_con
set x_con = xstates_con(t) (1)
*
* Constrained KF, using mean and variance of truncated distribution.
*
function WithTruncation xt vt
type vector *xt
type symm *vt

```



```

*
local real beta sigma mills
compute sigma=sqrt(vt(1,1))
compute beta=(1.0-xt(1))/sigma
compute mills=%mills(beta)
compute xt(1)--=sigma*mills
compute vt(1,1)=vt(1,1)*(1-beta*mills-mills^2)
end
*
dlim(fpost=WithTruncation,y=y,c=1.0,sw=sigsqw,sv=sigsqv,$
    presample=diffuse,type=filter) / xstates_trunc vstates_trunc
set x_trunc = xstates_trunc(t)(1)
*
* Constrained KF, using exponential approximation with predicted
* state as expansion point.
*
set xtilde = 0.0
*
function EKFPredicted xt
type vector *xt
*
compute xtilde(t)=%min(xt(1),3.0)
end
*
frml muf = 1-exp(-xtilde)-xtilde*exp(-xtilde)
frml cf = exp(-xtilde)
frml swf = exp(+2.0*xtilde)*sigsqw
*
dlim(fpre=EKFPredicted,y=y,a=1.0,mu=muf,c=cf,sw=swf,sv=sigsqv,$
    presample=diffuse) / xstates_exp0 vstates_exp0
set x_exp0 = 1-exp(-xstates_exp0(t)(1))
*
* Constrained KF, using exponential approximation with smoothed
* state as expansion point. Uses the same muf, cf and swf as
* above, just changes ttilde.
*
dlim(fpre=EKFPredicted,y=y,a=1.0,mu=muf,c=cf,sw=swf,sv=sigsqv,$
    presample=diffuse,type=smooth) / xstates_sm
set xtilde = %min(xstates_sm(t)(1),3.0)
dlim(y=y,a=1.0,mu=muf,c=cf,sw=swf,sv=sigsqv,$
    presample=diffuse) / xstates_exp1 vstates_exp1
set x_exp1 = 1-exp(-xstates_exp1(t)(1))
*
* Particle filter
*
compute npart=10000
dec vect xpart(npart) wt(npart) xresample(npart)
ewise xpart(i)=%uniform(-1.0,1.0)
set x_pf 1 400 = 0.0
do time=1,400
    do i=1,npart
        *
        * Do blind draw for X(t)|X(t-1) (i.e. draw without taking value
        * of y into account). g(X|X(t-1),y) is then equal to f(X|X(t-1))

```

```

    * so those cancel.
    *
    compute xtest=%rantruncate(xpart(i),sqrt(sigsqw),%na,1.0)
    compute ptest=%logdensity(sigsqv,y(time)-xtest)
    compute wt(i)=ptest
    compute xpart(i)=xtest
end do i
compute maxlogw=%maxvalue(wt)
ewise wt(i)=exp(wt(i)-maxlogw)
compute wt=wt/%sum(wt)
compute x_pf(time)=%dot(wt,xpart)
*
* Resample
*
ewise xresample(i)=xpart(%ranbranch(wt))
compute xpart=xresample
end do time
*
graph(key=below,klabels=||"Particle Filter",$
      "KF with Constrained Mean/Unconstrained Sigma"||) 2
# x_pf
# x_std
*
graph(key=below,klabels=||"Particle Filter",$
      "KF with Exponential Approximation"||) 2
# x_pf
# x_exp1
*
graph(key=below,klabels=||"Particle Filter",$
      "KF with Truncated Estimates"||) 2
# x_pf
# x_trunc

```

DSGE: Setting Up and Solving Models

DSGE stands for **D**ynamic **S**tochastic **G**eneral **E**quilibrium. The **DSGE** instruction in RATS takes a set of formulas and solves them (either exactly or approximately) to create a state-space model of the form (1.1). Input to it are the formulas (in the form of a `MODEL`), and list of endogenous variables. Output from it are the system matrices for the state-space representation: `A`, `F` and `Z`. The algorithm used is the QZ method from Sims (2002). However, internally RATS uses a “real” Schur decomposition, rather than the complex Schur decomposition described in the paper.

We’ll start with a simple example which has no expectation terms.

$$\begin{aligned}x_t &= \rho x_{t-1} + \varepsilon_{xt} \\c_t &= c_{t-1} + \mu_c + x_{t-1} + \varepsilon_{ct}\end{aligned}$$

The equations are input using a set of `FRML`’s which are grouped into a model. We need to define the variables `x` and `c` as series so they will be recognized as such in analyzing the model. We also will need to provide actual values for the parameters ρ and μ_c . **DSGE** can’t do symbolic analysis in terms of the deep parameters; it can only solve for the representation given a specific set of values. The initial setup is:

```
dec series x c
dec real rho mu_y
frml f1 = x - rho*x{1}
frml f2 = c - (c{1} + mu_y + x{1})
group BansalYaron f1 f2
```

Note that the `FRML`’s do not have a specific dependent variable. Instead, they’re written as an expression in the form $f(\dots) = 0$ if the `IDENTITY` option is used, or $f(\dots) = \varepsilon_t$ for expressions like these without the `IDENTITY` option. To solve for a specific set of parameter settings, we can do something like:

```
compute rho=.98,mu_c=.005
dsge (model=BansalYaron,a=a,f=f,z=z) x c
```

The series listed on the **DSGE** instruction line are the endogenous variables in the model. The number of these must match the number of equations. Because the model has two non-identities, there are two shocks. The first two states will

be x_t and c_t —the endogenous variables in the order listed. **DSGE** will examine the formulas, looking for occurrences of the endogenous variables. These will be slotted into the form:

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_t \\ c_t \end{bmatrix} = \begin{bmatrix} \rho & 0 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x_{t-1} \\ c_{t-1} \end{bmatrix} + \begin{bmatrix} 0 \\ \mu_c \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \varepsilon_{xt} \\ \varepsilon_{ct} \end{bmatrix}$$

where, while we've written them symbolically, these will all have numeric values. This is now trivially solved to get

$$\mathbf{A} = \begin{bmatrix} \rho & 0 \\ 1 & 1 \end{bmatrix}, \mathbf{Z} = \begin{bmatrix} 0 \\ \mu_c \end{bmatrix}, \mathbf{F} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Let's now look at a more complicated example. The first order conditions for the problem:

$$\max E_0 \sum \beta^t (u_1 - u_2 c_t)^2 \text{ subject to } y_t = f k_{t-1} + \varepsilon_t, k_t + c_t = y_t \quad (10.1)$$

can be written:

$$\begin{aligned} E_t \beta f \times (u_1 - u_2 c_{t+1}) &= (u_1 - u_2 c_t) \\ k_t + c_t &= f k_{t-1} + \varepsilon_t \end{aligned}$$

The expectation of c_{t+1} given t is written using the standard series lead notation $c\{-1\}$. The first equation is an identity. We rearrange that to make the model:

```
dec real beta f u1 u2
dec series c k
frml(identity) f1 = beta*f*(u1-u2*c{-1})-(u1-u2*c)
frml          f2 = k+c-f*k{1}
```

An example setting up and solving this is:

```
compute beta    =.99
compute f       =1.02
compute u1      =1.0
compute u2      =0.5
group bliss f1 f2
dsge(model=bliss,a=a,f=f,z=z) c k
```

This generates the intermediate representation:

$$\begin{bmatrix} u_2 & 0 & -\beta f u_2 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_t \\ k_t \\ E_t c_{t+1} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} c_{t-1} \\ k_{t-1} \\ E_{t-1} c_t \end{bmatrix} + \begin{bmatrix} \beta f u_1 - u_1 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \eta_t + \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \varepsilon_t$$

where η_t is a process satisfying $E_t\eta_{t+1} = 0$. The Sims' method does a decomposition of the system which, as part of the solution, eliminates the η_t term, leaving only a backwards looking state-space model in the three state variables: the two endogenous variables plus E_tc_{t+1} .

Now this problem is simple enough that it can be solved exactly (and symbolically) by, for instance, the method of undetermined coefficients, to get the representation:

$$\begin{bmatrix} c_t \\ k_t \end{bmatrix} = \frac{1}{\beta f} \begin{bmatrix} 0 & \beta f^2 - 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} c_{t-1} \\ k_{t-1} \end{bmatrix} + \frac{u_1(1 - \beta f)}{\beta f u_2(1 - f)} \begin{bmatrix} -1 \\ 1 \end{bmatrix} + \frac{1}{\beta f} \begin{bmatrix} \beta f^2 - 1 \\ 1 \end{bmatrix} \varepsilon_t$$

DSGE produces the equivalent representation (for any given setting of the parameters), but it ends up with the redundant state E_tc_{t+1} , which (in the solved version) just passively reacts to the other two states. You should look at **DSGE** as being something of a “black box”, which produces a solution to the problem, just not necessarily in the most obvious form. However, because the first block of states are *always* the dependent variables in the order you specify, you can easily determine the dynamics for the series of interest—you do the calculations with the full state-space model, and then extract out that top block.

10.1 Requirements

While **DSGE** operates numerically on the parameters, it works symbolically with the series. In order to do this, it needs to be able to recognize when a series is used and what (specific) lag or lead is needed. Series have to be referenced by individual names. You can't use elements of `VECT[SERIES]`; you can't use something like `I{0}` where `I` is an integer handle for a series. The lag or lead has to be a hard number, not a variable, that is, `C{N}` isn't permitted.

The model has to be a “general equilibrium” model—any series used in the equations of the model have to be dependent variables for the model. The only “exogenous” processes are the shocks which are implicit (in the `FRML`'s input as non-identities).

All expectations are given information through t . These are represented as leads, so `C{-1}` is E_tc_{t+1} and `C{-2}` is E_tc_{t+2} . Expectations through other information sets (such as $t - 1$) can be handled by adding states, as we'll see in section 10.2.

In the example above, the capital stock is “time-stamped” so that k_t is determined at time t (by the decision regarding c_t). In many descriptions of this type of model, the capital stock is instead dated so k_t is what is used in production at t . The resource constraint with this type of date scheme will usually be written:

$$k_{t+1} + c_t = y_t + (1 - \delta)k_t$$

(δ is the depreciation rate). You *can* keep that date scheme as long as you re-date that specific equation. If included as written, the k_{t+1} will be interpreted

as an expected value rather than a realized number. Instead, you want to use this in the form:

$$k_t + c_{t-1} = y_{t-1} + (1 - \delta)k_{t-1}$$

All other occurrences of k in the model will naturally be dated t and earlier.

10.2 Adapting to different information sets

Minford & Peel (2002) have the following small example on page 44:

$$\begin{aligned} m_t &= \bar{m} + \varepsilon_t \\ p_t &= E_{t-1}p_t + \delta(y_t - y^*) \\ m_t &= p_t + y_t \end{aligned}$$

Because this has the expectation with respect to $t - 1$ rather than t , it can't be translated directly into the correct form. And we can't do something simple like shifting the date on the second equation, since that would result in the p on the left side and the y on the right being converted into expectations as well. (The equation still holds when shifted forwards in time, it's just that it no longer has much content, since it reduces to $E_t y_{t+1} = y^*$). Instead, we need to define a new variable $w_t = E_t p_{t+1}$ and add that definition as an identity. Then $w_{t-1} = E_{t-1} p_t$ (you redate both the variable and the information set when you lag an expectation), so we can use w_{t-1} in the second equation. The revised model is:

$$\begin{aligned} m_t &= \bar{m} + \varepsilon_t \\ p_t &= w_{t-1} + \delta(y_t - y^*) \\ m_t &= p_t + y_t \\ w_t &= E_t p_{t+1} \end{aligned}$$

The code from setting up the model is:

```
dec real delta ystar mbar
dec series m p y w
*
frml          f1 = m - mbar
frml(identity) f2 = m - (p+y)
frml(identity) f3 = p - (w{1}+delta*(y-ystar))
frml(identity) f4 = w - p{-1}
group modell1 f1 f2 f3 f4
```

and a specific solution is:

```
compute delta=0.3,ystar=100.0,mbar=10.0
dsge(model=modell1,a=a,f=f,z=z) m p y w
```

Note that, even though it's just a constructed variable, you have to include w in your listing of the dependent variables. When **DSGE** generates an additional

state variable for its own bookkeeping, you don't have to worry about it, but if *you* have to augment the model in order to produce a model in the required form, you must include the new variables in your list. For instance, if you read the Sims paper, he includes as an example the following (his equation (2)):

$$w_t = \frac{1}{3} E_t [W_t + W_{t+1} + W_{t+2}] - \alpha (u_t - u^*) + \nu_t$$

$$W_t = \frac{1}{3} (w_t + w_{t-1} + w_{t-2})$$

$$u_t = \theta u_{t-1} + \gamma W_t + \mu + \varepsilon_t$$

and notes how to expand the state vector to accomodate the extra lags and leads. Note well that you *do not* have to do that yourself. Lags higher than 1 and expectations with respect to t for time $t + 2$ and above are recognized by **DSGE** and handled exactly as described in the paper. You only need to create your own augmenting states if you need expectations with respect to time periods other than t . The setup for this model is:

```
dec series w bigW u
dec real alpha ustar theta gamma mu
frml f1          = w      - $
      ((1.0/3.0*(bigW+bigW{-1}+bigW{-2})-alpha*(u-ustar))
frml f2(identity) = bigW - (1.0/3.0*(w+w{1}+w{2}))
frml f3          = u      - (theta*u{1}+gamma*bigW+mu)
group overlap f1 f2 f3
```

10.3 Non-linear models

The optimization (10.1) produces a set of linear equations only because it uses a quadratic utility function¹ and a linear production function. With a log utility function and production function $y_t = f k_{t-1}^\alpha \theta_t$, the first order conditions become:

$$E_t \beta R_{t+1} c_t / c_{t+1} = 1$$

$$R_t = \alpha f k_{t-1}^{\alpha-1} \theta_t$$

$$y_t = f k_{t-1}^\alpha \theta_t$$

$$k_t + c_t = y_t$$

$$\log \theta_t = \varepsilon_t$$

where R_t is the gross rate of return on capital and θ_t is a productivity shock. It's possible to substitute out for R and y , but at a substantial loss in clarity. Those types of substitutions are often necessary if you're trying to simplify the model to a small enough set of states to solve analytically. However, if **DSGE** is doing the hard work, you're better off writing the model in short, easy-to-read formulas with extra variables.

¹This has a "bliss point" at $c = u_1/u_2$ with utility declining for consumption above that.

Because the equations aren't linear, the Sims solution method can't apply directly. Instead, we can get an approximate solution by linearizing the model. The standard way of handling this in the literature is to log-linearize around the steady state. However, **DSGE** offers the possibility of doing a simple linearization. You need to include either `EXPAND=LOGLINEAR` or `EXPAND=LINEAR` to get one of the expansions.

In order to log-linearize, all series need to be strictly positive. With this in mind, the technology shock is multiplicative rather than additive. Log-linearization involves writing each series in the form $x_t = \bar{x} \exp(\tilde{x}_t)$, where \bar{x} is the expansion point. We want the equations written in terms of \tilde{x}_t . While there are certain tricks to log-linearizing a model manually, **DSGE** just writes the full vector of dependent variables as $X_t = \bar{X} \circ \exp(\tilde{X}_t)$ where \circ is the element by element (Hadamard) product, and produces the linearization:

$$f(X_t) \approx f(\bar{X}) + \left(\frac{\partial f}{\partial X}(\bar{X}) \circ \bar{X} \right) \bullet \tilde{X}_t = 0$$

from each of the conditions. If the expansion point \bar{X} is, in fact, the steady state, then $f(\bar{X}) = 0$, so the state-shift component Z will be zero.

The log-linearization is quite straightforward once you have the expansion point. RATS has a symbolic differentiation routine which can handle almost any of its (scalar) functions, including probability functions like `%CDF`. You *can*, however, run into problems calculating the steady state. By default, **DSGE** starts with guess values of 1.0 for each variable and uses a log-linear version of Newton's method to solve the system of non-linear equations. That generally works acceptably if the variables aren't bounded above. There are, however, models where, for instance, labor is parameterized in terms of fraction of available time, with labor + leisure = 1. Under those circumstances, the initial guess values aren't even valid. There are two ways to feed in alternative guess values. The simplest usually is to use `<<` on an individual series to override the default value of 1. For instance:

```
dsge(model=growth,a=a,f=f,z=zc,expand=loglinear,trace) $
  c k y lab<<.5 xi
```

uses values of 1 for `C`, `K`, `Y` and `XI`, but `.5` for `LAB`. You can use the `TRACE` option to see how the iterations are going. That will sometimes help you isolate any series which are causing problems.

You can also use the `INITIAL` option to input the full vector (one per series). That will mainly be used in situations where we need to repeatedly solve a system with changes in the parameters, when, for instance, we're estimating them. Once you've solved the system once, you can fetch the steady state values for that with the option `STEADY=VECTOR` for steady state estimates. You can then feed that back into future **DSGE** instructions using the `INITIAL` option. Since the steady state generally won't change much with small changes

to the parameters, you'll reduce the amount of time required to locate the new expansion point.

If you want to set the expansion point rather than have it estimated, you can use either the `<<` or `INITIAL` option to put in the guess values, then include the option `ITERS=0`.

10.4 Unit Roots

As we've seen, unit roots aren't a problem for the **DLM** instruction, which can handle a mixture of unit and stationary roots using the `PRESAMPLE=ERGODIC` option. They do, however, pose several problems for solving a model into state-space form.

First, the solution procedure is to solve “unstable” roots forwards and “stable” roots backwards. For the purpose of solution, a unit root is considered to be stable—only roots greater than 1 are solved forwards. In the world of computer arithmetic, however, a test like $\lambda \leq 1$ can be dangerous. Due to roundoff error, it's possible for a value which should be *exactly* 1 to come out slightly larger (or smaller, though that won't be a problem). **DSGE** has a `CUTOFF` option which determines the boundary between the blocks of roots. The default value for that is 1.0, but it actually is $1.0 + \text{a small positive number } (10^{-7})$. This gives a cutoff which is close enough to 1.0 that it's highly unlikely for a model to have a root that is, in fact, unstable, but less than that value, but is still far enough from 1 that true unit roots will generally have a computed value lower than it.² It's possible (though unlikely) that you'll have a model where the “fuzziness” added by **DSGE** isn't enough. If your model doesn't solve correctly, you can put in something like `CUTOFF=1.001` to give a much higher tolerance for detection of a unit root.

A more significant problem occurs in expanding a non-linear model. A model with unit roots has no single steady state. For instance, if you have a persistent technology shock such as:

$$\log \theta_t = \rho \log \theta_{t-1} + \varepsilon_t$$

the calculation of a steady state turns this into $(1 - \rho) \log \bar{\theta} = 0$. If $\rho \neq 1$, we get $\log \bar{\theta} = 0$. When $\rho = 1$, we get $0 = 0$ —the model does nothing to tack down a value for $\bar{\theta}$. Instead of having *one* steady state, the model has a different solution for the other variables for each value of $\bar{\theta}$. To get an expansion point, you'll have to do one of two things:

1. Input the entire expansion point as described at the end of section 10.3.
2. Provide values for the series which produce the unit root(s), and add the option `SOLVEBY=SVD` to **DSGE**.

²Roundoff errors in the type of calculation done by **DSGE** tend to be on the order of 10^{-12} .

`SOLVEBY` controls the way Newton's method solves the system of equations at each iteration. With the unit root(s), the system of equations is singular. `SOLVEBY=SVD` lets this solve a partial system which is non-singular, while leaving unchanged the part that's singular.³ If you try to solve a non-linear system with a unit root without using `SOLVEBY=SVD`, you'll get a message like:

```
## F016. Newton's method had singularity. If model is non-stationary, use option SOLVEBY=SVD.
Error was in FRML F7
On iteration 1 of solution for steady state
```

This suggests the fix and points out the equation that produced the singularity.

10.5 Dynare scripts

Dynare is a pre-processor and a collection of MATLAB routines for solving DSGE's. It was developed at CEPREMAP in Paris, France. Dynare scripts are quite similar to the inputs used for **DSGE**. In particular, the endogenous variables have to be referenced by name, only hard values for the lags and leads are permitted, expectation terms are all given information through t and parameters need values. Dynare uses the same arithmetic operators as used in RATS formulas, though it allows only `^` for exponentiation.⁴ The functions `LOG(x)`, `EXP(x)` and `SQRT(x)` are the same as in RATS. Variable names have similar formats. The key differences in terms of organization are:

1. Dynare uses `(k)` rather than `{k}` for lags and leads and uses the opposite sign convention, so `C(+1)` means $E_t c_{t+1}$ and `C(-1)` means c_{t-1} .
2. Dynare requires that the exogenous variables be declared explicitly, while the formulas for **DSGE** have the exogenous shocks implicit in the formulas which don't have the `IDENTITY` option.

In translating a Dynare script for use in RATS, the first of these can be handled by a straight substitution. The second is more easily handled by "endogenizing" the exogenous shocks by putting the `IDENTITY` option on all the model equations, then adding a trivial equation:

```
frml f11 = eps
```

This isn't a bad idea in general, as it makes it easier to control the definitions, positions and sign of the shocks in the model. In the `F` matrix output by **DSGE**, the shocks are in their order of appearance in the equations of the model. The

³SVD stands for Singular Value Decomposition. This decomposes a matrix into the form UWV' where U and V are unitary matrices ($UU' = I$, $VV' = I$) and W is diagonal with non-negative elements. The elements of W are known as the *singular values*. The number of non-zero elements of W is the rank of the original matrix.

⁴RATS accepts both `^` and `**`.

first non-identity on the **GROUP** instruction will define the first shock, the second defines the second, etc.

We have a PERL script on our web site which translates Dynare scripts into the roughly equivalent model setup code for RATS. An example of the translation is to take the (partial) Dynare script:

```
var y c k i l y_1 w r z;
varexo e;
parameters beta psi delta alpha rho sigma epsilon;
model;
(1/c) = beta*(1/c(+1))*(1+r(+1)-delta);
psi*c/(1-l) = w;
c+i = y;
y = (k(-1)^alpha)*(exp(z)*l)^(1-alpha);
w = y*((epsilon-1)/epsilon)*(1-alpha)/l;
r = y*((epsilon-1)/epsilon)*alpha/k(-1);
i = k-(1-delta)*k(-1);
y_1 = y/l;
z = rho*z(-1)+e;
end;
```

to the RATS code:

```
dec series y c k i l y_1 w r z
dec series e
dec real beta psi delta alpha rho sigma epsilon
frml(identity) eqn1 = (1/c) - ( beta*(1/c{-1})*(1+r{-1}-delta))
frml(identity) eqn2 = psi*c/(1-l) - ( w)
frml(identity) eqn3 = c+i - ( y)
frml(identity) eqn4 = y - ( (k{1}^alpha)*(exp(z)*l)^(1-alpha))
frml(identity) eqn5 = w - ( y*((epsilon-1)/epsilon)*(1-alpha)/l)
frml(identity) eqn6 = r - ( y*((epsilon-1)/epsilon)*alpha/k{1})
frml(identity) eqn7 = i - ( k-(1-delta)*k{1})
frml(identity) eqn8 = y_1 - ( y/l)
frml(identity) eqn9 = z - ( rho*z{1}+e)
frml eqn10 = e
group dsge eqn1 eqn2 eqn3 eqn4 eqn5 eqn6 eqn7 eqn8 eqn9 eqn10
dsge(model=dsge,a=adlm,f=fdlm,z=zdlm) y c k i l y_1 w r z e
```

In the Dynare code, `var` vs `varexo` distinguishes the endogenous variables from the shocks. On the **DSGE** instruction, the RATS code lists the endogenous variables in the same order as they were on the `var` line, and appends the exogenous variables in order as well. The equations defining the shock variables are put into the model in the same order as the shocks are listed on the `varexo` line, so that will be the placement order for the shocks in the **F** matrix.

DSGE: Applications

Given an input model, **DSGE** solves it into a state-space model representation. So what can we do with this? Many models aren't designed to be estimated with actual data. In order to take a state-space model to data, the model needs at least as many shocks as there are observable series. Almost all the models in Chapter 10 have (many) fewer shocks than endogenous variables, even if we don't count series like capital that really aren't easily observed. Instead, most solved models are used to generate simulated economies and study the serial correlation patterns implied by the model. Simulations can be done easily using **DLM**; the calculation of responses and serial correlation require some additional procedures.

In order to solve the model in the first place, you need to provide values for the deep parameters of the model. One thing that you did not need before, but will need now, are the variances of the shocks. As mentioned earlier, if you have more than one exogenous shock, they will be ordered based upon the listing on the **GROUP** instruction which created the model. The first formula in that list that isn't an identity defines the first shock, the second defines the second shock, etc. The shock is signed to be give a positive shift to the function as written. If you write a formula:

```
frml f2 = (k+c)-f*k{1}
```

the interpretation will be $k_t + c_t - f k_{t-1} = \varepsilon_t$, which means that it's a positive shock to output. The same formula written in the opposite order:

```
frml f2 = f*k{1}-(k+c)
```

would produce a shock that's actually negative to output. This is another argument for endogenizing any shock that occurs in any of the more complicated formulas—you can make sure it has the sign that you want.

11.1 Simulations

DLM with the option `TYPE=SIMULATE` can create a simulated path for the complete state vector. Now as we've seen above, the full state vector can include quite a few states which are of no direct interest, having been added to the

model solely to allow a solution in the proper form. However, since the original endogenous variables are in a known position in the state vector, we can extract them from the `SERIES[VECT]` that **DLM** produces.

One thing you must keep in mind is that if you log-linearize the model, the states generated must be interpreted as simulated values for $\log(x_t/\bar{x})$. If you're interested in simulated series for the data itself, you'll have to save the steady state (using the `STEADY` option on **DSGE**) and transform back.

Example 11.1 solves a Brock-Mirman growth model and uses the state-space representation to generate a realization of such an economy. This is similar to, but slightly more complicated than, the model in section 10.3, allowing for more general constant relative risk aversion utility (not just log utility), partial depreciation and persistent shocks. The variables are the same as before.

```
dec series c R y k theta
dec real    A alpha beta delta rho sigma
*
frml(identity) f1 = 1 - (beta*R{-1}*c^{sigma}/c{-1}^{sigma})
frml(identity) f2 = y - (A*theta*k{1}^{alpha})
frml(identity) f3 = R - (A*theta*alpha*k{1}^{(alpha-1)+1-delta})
frml(identity) f4 = k + c - (y + (1-delta)*k{1})
frml          f5 = log(theta) - rho*log(theta{1})
*
group brock f1 f2 f3 f4 f5
dsge(model=brock,a=adlm,f=fdlm,z=zdlm,expand=loglin,steady=ss) $
    c y k r theta
```

In order to back out variables to their levels, we add the `STEADY` option to save the steady state. The simulation is done by setting a value for the variance of the productivity shock and doing **DLM** with `TYPE=SIMULATE`. The variance for the shocks goes in by the `SW` option. In this case, there's only one shock. If you have more than one, you'll need to use `%DIAG(|v1,v2,...,vr|)` (assuming the shocks are independent).

```
compute sigmaeaps=.001
dlm(type=simulate,presample=ergodic,a=adlm,f=fdlm,sw=sigmaeaps^2) $
1 538 xstates
```

Since there isn't any actual data, you need to give an explicit range on the **DLM** instruction. Up to this point, we've worked with series as, in effect, symbols.

```
set c 1 538 = exp(xstates(t)(1))*ss(1)
set y 1 538 = exp(xstates(t)(2))*ss(2)
set k 1 538 = exp(xstates(t)(3))*ss(3)
set r 1 538 = exp(xstates(t)(4))*ss(4)
```

The elements of `SS` are in exactly the same order as the states. However, the steady state vector is only for the endogenous series themselves; it doesn't ex-

tend into the augmenting states. In many cases, you'll want to keep the data in log form. After all, you would typically convert variables like consumption and output to logs anyway. To get that, just copy the state out.

```
set logy 1 538 = xstates(t) (2)
```

11.2 Impulse Responses

You can compute and graph impulse response functions for a state-space model using the procedure `@DLMIRF`. Note that this was recently substantially rewritten, with some changes to the option names, so if you have any old uses of it, you might need to rewrite them.

`@DLMIRF` takes as input the **A** and **F** matrices from the state-space model. (The state shifts **Z** don't affect the results). It then computes and graphs the responses to the variables of the system to the shocks. Note that it uses a unit size for each shock. That might give responses that are unnaturally large. For instance, a unit shock to ε_t in $\log \theta_t = \rho \log \theta_{t-1} + \varepsilon_t$ will increase θ by a factor of e , with correspondingly enormous changes to the variables. You can change the scale of the shocks by multiplying **F** by a small number. If you have just one shock, or would like the same scaling to apply to all, you can just use the option `F=.01*FDLM`, for instance. If you want different scales to apply to each shock, you can do `F=FDLM*%DIAG(scale factors)`. The scale factors here should be standard errors, not variances.

There are several options for controlling the organization of the graphs. Aside from standard **HEADER** and **FOOTER** options, the **PAGE** option allows you to arrange pages **BYSHOCK** (one shock per page, all variables), **BYVARIABLE** (one variable per page, all shocks), **ALL** (everything on one page) and **ONE** (one combination of shock and variable per page). There's also a **COLUMNS** option which allows you to better arrange the graphs on a single page. The shocks are given labels using the **SHOCKS** option and the endogenous variables are labeled using the **VARIABLES** option.¹ Because the state vector usually includes augmenting variables (and often endogenized shocks), you generally don't want to include the entire state vector in the graphs. If you just restrict the **VARIABLES** option to list a subset of the states, only those will be shown. Since you have freedom to list the variables in any order on the **DSGE** instruction, you can arrange the variables there so the ones of interest come first.

Example 11.2 solves Hansen's (1985) Real Business Cycle model with indivisible labor and graphs responses to the technology shock. The setup for the model is similar to what we've seen, except that the Lagrange multiplier isn't substituted out. (It appears in both a consumption and labor/leisure decision).

¹The changes with the rewriting of this were to replace the old **LABELS** option with **SHOCKS**, **VLABELS** with **VARIABLES**, and to add the **PAGE** option, replacing options which just read **BYSHOCK** and **BYVARIABLE** rather than **PAGE=...**

```

dec series c y n r k z lambda
*
frml(identity) eqn1 = 1.0-c*lambda
frml(identity) eqn2 = a-lambda*(1-theta)*y/n
frml(identity) eqn3 = r-theta*y/k{1}-1+delta
frml(identity) eqn4 = lambda-beta*lambda{-1}*r{-1}
frml(identity) eqn5 = y-gammabar*z*k{1}^theta*n^(1-theta)
frml(identity) eqn6 = c+k-y-(1-delta)*k{1}
frml          eqn7 = log(z)-rho*log(z{1})

```

A novelty with this example is that one of the parameters is set indirectly based upon a desire to have the labor variable hit a particular steady state value. That's done by using a **FIND** instruction where the **DSGE** is solved inside the function evaluation. This is used to find a point where the steady state for N (variable 3) is $1/3$.

```

group hansenrbc eqn1 eqn2 eqn3 eqn4 eqn5 eqn6 eqn7
*
nonlin a
compute a=1.0
dec real nbar
find root 3.0*nbar-1.0
    dsge(expand=loglin,steadystate=ss,model=hansenrbc) $
        c y n r k z lambda
    compute nbar=ss(3)
end find

```

The impulse responses are created using **@DLMIRF** and graphed for the first five states, leaving out the technology and Lagrange multiplier series. Note that the responses are all going to be in log form. The shocks are scaled by **SIGMAEPS** to give more realistic sizes.

```

dsge(expand=loglin,model=hansenrbc,a=adlm,f=fdlm,z=zdlim) $
    c y n r k z lambda
@dlimirf(a=adlm,f=fdlm*sigmaeps,steps=33,$
    page=byshock,shocks=||"Technology"||,$
    vars=||"Consumption","Output"," Hours","Interest","Capital"||)

```

Example 11.1 DSGE Simulation

This is a fairly general specification for the Brock-Mirman stochastic growth model (Brock & Mirman (1972)). It sets up and solves the model and generates a single simulated economy.

```

dec series c R y k theta
dec real    A alpha beta delta rho sigma
*
frml(identity) f1 = 1 - (beta*R{-1}*c^sigma/c{-1}^sigma)
frml(identity) f2 = y - (A*theta*k{1}^alpha)
frml(identity) f3 = R - (A*theta*alpha*k{1}^(alpha-1)+1-delta)
frml(identity) f4 = k + c - (y + (1-delta)*k{1})
frml          f5 = log(theta) - rho*log(theta{1})
*
compute A=1.0,sigma=0.5,delta=.1,rho=.90,alpha=.33,beta=.90
*
group brock f1 f2 f3 f4 f5
dsge(model=brock,a=adlm,f=fdlm,z=zdlm,expand=loglinear,steady=ss) $
    c y k r theta
*
compute sigmaeps=.001
*
dlm(type=simulate,presample=ergodic,a=adlm,f=fdlm,sw=sigmaeps^2) $
    1 538 xstates
*
* Generate simulated levels
*
set c 1 538 = exp(xstates(t)(1))*ss(1)
set y 1 538 = exp(xstates(t)(2))*ss(2)
set k 1 538 = exp(xstates(t)(3))*ss(3)
set r 1 538 = exp(xstates(t)(4))*ss(4)
*
* Generate simulated logy
*
set logy 1 538 = xstates(t)(2)
*
* Compute its autocorrelations
*
corr logy
*
* Compute sample expectational error using f1. Since that's written in
* terms of the levels, it needs the fully transformed data. Look at its
* sample statistics and autocorrelations. This is scaled up considerably
* because of the very small magnitude.
*
set error = 1.e+6*f1
stats error
corr error

```


Example 11.2 DSGE Impulse Response Functions

This solves Hansen's RBC model with indivisible labor and graphs the responses of the variables to the technology shock.

```

dec real beta a theta delta gammabar rho
dec real sigmaeps
*
* Endogenous series
*
dec series c y n r k z lambda
*
frml(identity) eqn1 = 1.0-c*lambda
frml(identity) eqn2 = a-lambda*(1-theta)*y/n
frml(identity) eqn3 = r-theta*y/k{1}-1+delta
frml(identity) eqn4 = lambda-beta*lambda{-1}*r{-1}
frml(identity) eqn5 = y-gammabar*z*k{1}^theta*n^(1-theta)
frml(identity) eqn6 = c+k-y-(1-delta)*k{1}
frml          eqn7 = log(z)-rho*log(z{1})
*
* Calibration
*
compute theta    =.4
compute delta    =.012
compute rho      =.95
compute sigmaeps=.007
compute beta     =.987
compute gammabar=1.0
*
* Determine A so that nbar is 1/3
*
group hansenrbc eqn1 eqn2 eqn3 eqn4 eqn5 eqn6 eqn7
*
nonlin a
compute a=1.0
dec real nbar
find root 3.0*nbar-1.0
    dsge(expand=loglin,steadystate=ss,model=hansenrbc) $
        c y n r k z lambda
    compute nbar=ss(3)
end find
*
dsge(expand=loglin,model=hansenrbc,a=adlm,f=fdlm,z=zdlm) $
    c y n r k z lambda
@dmlirf(a=adlm,f=fdlm*sigmaeps,steps=33,$
    page=byshock,shocks=||"Technology"||,$
    variables=||"Consumption","Output","Hours","Interest","Capital"||)

```

DSGE: Estimation

There's a relatively thin literature on formal estimation of the deep parameters in a DSGE using actual data. An article frequently cited on estimation of DSGE's (Ruge-Murcia (2007)) works entirely with a simulated economy. The primer on the use of Dynare (Barillas et al. (2007)) has estimation code for most of its models, but the only two that use real world data (rather than data simulated using the model being estimated) aren't DSGE's—they're simple state-space models.

Conceptually, estimation shouldn't be that difficult. Given the model, we have a method which generates either an exact state-space model (for linear models) or an approximate one (for non-linear models). For any collection of parameters, we can generate a history of states, which can be compared with actual data. Unfortunately, it's not that easy. The main problem can be seen by looking at Example 11.2. This is a perfectly reasonable looking model, which generates predictions for output, consumption, labor and interest rates. However, it has only one exogenous shock. As a result, the errors in the four predictions have a rank one covariance matrix: once you know one, you should be able to compute exactly the others. In sample, that will never happen. In order to do a formal estimation, we will have to do one of the following:

- Reduce the number of observables to match the number of fundamental shocks.
- Add measurement errors to increase the rank of the covariance matrix.
- Add more fundamental shocks.

Reducing the number of observables makes it much more likely that some of the deep parameters really can't be well estimated. For instance, because the capital stock is effectively unobservable, the depreciation rate δ can't easily be determined from the data—a high value of δ with a high level for the capital stock produces almost the same predictions for the observable variables as a low value for both. Adding fundamental shocks requires a more complex model, such as the well-known Smets-Wouters model (Smets & Wouters (2003)).

Before improvements in the algorithms for solving DSGE's and increases in compute power made larger models feasible, the standard technique for handling models was *calibration*. In this, you accept the fact that the model isn't a complete generating process for the data, and instead look for more informal

ways to calibrate or tune the parameters to match with important behavior of the observed data (such as impulse responses or covariances). We'll concentrate here on the two main methods for estimating models which *do* admit a complete description of the observables: maximum likelihood and Bayesian methods.

12.1 Maximum Likelihood

We'll start with a simple case from the *Practicing Dynare* guide, since it demonstrates the setup needed for more complicated models. The original source is Sargent (1977). The two observables are money growth (μ_t) and inflation (x_t). With two observables, we need two shock processes: one will be to money demand, the other to money supply. The model (reduced to growth rates) can be written:

$$\begin{aligned}x_t &= x_{t-1} + a_{1t} - \lambda a_{1t-1} \\ \mu_t &= (1 - \lambda) x_{t-1} + \lambda \mu_{t-1} + a_{2t} - \lambda a_{2t-1} \\ a_{1t} &= (\lambda + (1 - \lambda) \alpha)^{-1} (\varepsilon_t - \eta_t) \\ a_{2t} &= (\lambda + (1 - \lambda) \alpha)^{-1} ((1 + \alpha(1 - \lambda)) \varepsilon_t - (1 - \lambda) \eta_t)\end{aligned}$$

The underlying shocks are ε_t and η_t , which are assumed to be independent. Since those are both tangled up in the definitions of a_1 and a_2 , we'll use separate equations to put them into the model. Thus, we have:

```
frml(identity) f1 = x - (x{1}+a1-lambda*a1{1})
frml(identity) f2 = mu- ((1-lambda)*x{1}+lambda*mu{1}+a2-lambda*a2{1})
frml(identity) f3 = a1-1.0/(lambda+(1-lambda)*alpha)*(eps-eta)
frml(identity) f4 = a2-(1.0/(lambda+(1-lambda)*alpha)*$
    ((1+alpha*(1-lambda))*eps-(1-lambda)*eta))
frml          d1 = eps
frml          d2 = eta
*
group cagan f1 f2 f3 f4 d1 d2
```

In the example from *Practicing Dynare*, the contemporaneous a variables are substituted out of F1 and F2. There's no reason to do that in RATS, since **DSGE** will handle that automatically when it solves for the fully backwards representation. The model is linear, so the model can be solved for a state-space representation rather easily with:

```
dsge(model=cagan,a=adlm,f=fdlm) x mu a1 a2 eps eta
```

(There won't be a Z term since there are no additive constants anywhere.) However, the A and F matrices depend upon the underlying parameters: λ and α . While it's possible here to solve analytically for the state-space representation,

which would give closed form expressions for A and F , we can instead use **DSGE** to do that calculation for each setting for the parameters.

The likelihood function can be evaluated by **DLM** given the system matrices. To do this, we need a C matrix to map states to observables. In this case, there will be just six states; however, in general, you don't know before solving the model how many states needed to be added. (There will always be some if you have expectational variables). The simplest way to set up the C matrix is to first solve the model for your guess values for the parameters, then use

```
compute cdlm=%identity(2)~~%zeros(%rows(adlm)-2,2)
```

CDLM will take the first state as the first observable and the second as the second. For simplicity, you should arrange your variable list on **DSGE** to put the observables in the initial positions. The 2's in this are the number of observables. The same will work for a different number of observables by replacing 2 in all three locations.

We can use **DLM** to do the estimation, but we need to run **DSGE** with each new setting of parameters to get the new A and F matrices. It's simplest to write a small function which does this, and put it into the **START** option.

```
function EvalModel
dsge(model=cagan,a=adlm,f=fdlm) x mu a1 a2 eps eta
end EvalModel
```

One other complication is that the process has a unit root: x_t and μ_t are cointegrated with $x_t - \mu_t$ being stationary. Fortunately, the **PRESAMPLE=ERGODIC** option on **DLM** can handle this automatically. The shocks are both in the state equation, so there is no direct measurement error. The model can be estimated with:

```
dlm(start=%(EvalModel()),sw=%diag(||sig_eps^2,sig_eta^2||),$
    a=adlm,f=fdlm,y=||x,mu||,c=cdlm,sw=sw,presample=ergodic,$
    pmethod=simplex,piters=5,method=bfgs)
```

The `%(...)` allows you to include several calculations in the **START** option. You need to enclose these in this placeholder function so the comma inside it is interpreted as a separator for the calculations, rather than as a separator for option fields.

The second example is from Ireland (2004). This estimates the Hansen RBC using three observables: output, consumption and labor. Since the model itself has only one shock, the approach used was to add measurement errors. The programming for this would be relatively straightforward if the measurement errors were assumed to be serially uncorrelated; it would be very similar to the previous example, except there would now be an **SV** option to handle the measurement errors. However, the simple measurement errors proved to be

insufficient to produce a good fit. Instead, Ireland assumed that a VAR(1) model was needed. As we've seen before, you can only accomodate serially correlated measurement errors by adding them to the state vector. The state vector will be the one created by **DSGE** concatenated with the measurement error vector. The **A** and **F** matrices will have the usual diagonal concatenation of the DSGE model and the measurement error VAR. The loading matrix **C** will be a bit more complicated here. The observable will need to be equal to the sum of the prediction for the state coming out of the DSGE and the error component from the VAR.

With the observables, the measurement process is a 3 variable VAR. This needs a complete 3×3 matrix for the lag coefficients and a 3×3 symmetric for the covariance matrix. To avoid problems with the covariance matrix going non-positive definite, it's parameterized in Cholesky factor (lower triangular) form. To do that, declare the matrix as type **PACKED** (short for packed lower triangular) and use the function **%LTOUTERXX** to expand it to a symmetric matrix.

```
dec rect d(3,3)
dec symm sigmav(3,3)
dec packed lsigmav(3,3)
```

The data are converted to per capita values, but aren't detrended, since the model itself implies a particular trend. As is typical, the discount factor β and the depreciation rate δ are pegged. The other parameters for the model and those for the VAR for the measurement error will be estimated.

In order to speed up computation time, we use the steady state at the guess values as the initial guess for the expansion point. In most cases, it should take only a few iterations to refine that to get the steady state at the new set of parameters.

```
sstats(mean) 1948:1 * yt>>ymean ct>>cmean ht>>hmean
*
group dsge model f1 f2 f3 f4 f5 f6
dsge(model=dsge model, expand=loglin, a=ax, f=fx, steady=basesteady) $
y<<ymean c<<cmean h<<hmean in<<ymean-cmean k a
```

The only system matrix which we can compute before knowing the parameters is the **C** matrix. This is similar to before, but now needs the extra identity matrix to add in the measurement errors.

```
compute cfull=%identity(3)~~%zeros(%rows(ax)-3,3)~~%identity(3)
```

The function that we need to execute in doing the setup for **DLM** is more complicated, since we now have to glue together the full system matrices. We also need to save the new value of the steady state, since we will need it to convert the natural values of the data into the same terms as the log-linearized states.

```

function SolveModel
*
* Guard against solving the model with problem values
*
if eta<1.0.or.eta>1.05.or.rho>=1.0
    return
dsge(model=dsgemodel,expand=loglinear,a=ax,f=fx,$
    steady=steady,initial=basesteady) y c h in k a
compute afull=ax~\d,ffull=fx~\%identity(3)
compute [symmetric] swfull=sigma^2~\%ltouterxx(lsigmav)
end SolveModel

```

The observables need to be computed on the fly. The data values for output and consumption (YT and CT) are detrended by the exponential trend implied by the technology. All variables are then converted into log deviations from the steady state.

```

dec frml[vect] yf
frml yf = ||log(yt*eta^(-t)/steady(1)), $
          log(ct*eta^(-t)/steady(2)), $
          log(ht/steady(3))||

```

The instruction for estimating is:

```

dlm(startup=SolveModel(),y=yf,$
    a=afull,f=ffull,c=cfull,sw=swfull,presample=ergodic,$
    pmethod=simplex,piters=5,method=bfgs,itors=100,$
    reject=(eta<1.0.or.eta>1.05.or.rho>=1.0) 1948:1 2002:2

```

This has a REJECT option to avoid ineligible values: negative growth ($\eta < 1$) and a unit root technology shock. The $\eta > 1.05$ condition guards against evaluations with growth rates that are way too high—this is quarterly data, so that would be 20% growth rate per annum. It's possible for really strange values to be tested in early iterations, and values like that can cause the DSGE solution to fail. If you get a problem like that, take a look at the current values for the parameters (for instance, with `DISPLAY ETA`). If you need to put REJECT conditions in to protect against those, do it. Note that you shouldn't have to worry about things like that until you run into a problem. If the model solves, the bad values will naturally give low likelihoods. The REJECT option is described in greater detail in this chapter's *Tips and Tricks*, section 12.3.

12.2 Bayesian Methods

When it comes to estimating DSGE's, Bayesian methods are more popular than straight maximum likelihood. In some sense, all estimation exercises with these models are at least partially Bayesian—when parameters such as the discount rate and depreciation rate are pegged at commonly accepted values, that's Bayesian. Those are parameters for which the data have little information, so we impose a point prior.

As with maximum likelihood, the heart of Bayesian methods is the evaluation of the likelihood of the derived state-space model. So what accounts for their popularity? Part of this is due to the types of problems seen in the Ireland example. Variational methods of optimization (like the BFGS algorithm used by **DLM**) often try to evaluate the likelihood at what are apparently very strange sets of parameters. This is part of the process of a “black box” routine discovering the shape of the likelihood. For simpler functions, there is usually no harm in this—if a test set of parameters requires the log or square root of a negative number, the function returns an NA and the parameter set is rejected. If a root goes explosive, the residuals get really large and the likelihood will be correspondingly small. However, the function evaluation in a DSGE is quite complicated. If the model is non-linear, we first do the following:

1. solve for a steady state expansion point
2. solve for the backwards representation
3. solve for the ergodic mean and covariance of the state-space model

before we can Kalman filter through to evaluate the likelihood. In the Ireland example, the values of the growth rate η for which all those steps are well-behaved forms a relatively tight band. On the first few iterations (before much is known about the shape of the function), it's quite easy for a test set of parameters to go outside that. By imposing a prior which excludes values which we know to be nonsensical, we can avoid those types of problems.

The other main reason is that the likelihood function can be quite flat with respect to parameters besides β and δ ; those are just the two most common parameters that are poorly identified from the data. Including a prior allows for those other parameters to be restricted to a region that makes economic sense.

To employ Bayesian methods, we need to specify a prior density $p(\theta)$ for the set of parameters θ which we want to estimate. We combine the likelihood $p(Y|\theta)$ with the prior to create the posterior density:

$$p(\theta|Y) \propto p(Y|\theta)p(\theta)$$

$p(\theta)$ can be chosen to, for instance, exclude certain regions. If $p(\theta) = 0$, then the posterior has to be zero there as well. We can therefore avoid even computing the likelihood function at such points. The priors are generally specified

independently across the components of θ . If $\theta = \{\theta_1, \dots, \theta_k\}$, then the posterior in log form can be written:

$$\log p(\theta|Y) = \text{const.} + \log p(Y|\theta) + \sum_{i=1}^k \log p_i(\theta_i)$$

$\log p(Y|\theta)$ is just the log likelihood. $\log p_i(\theta_i)$ are generally quite easy to calculate if the priors are chosen from one of the well-known densities, so we can fairly easily compute the log posterior density for any test set of θ . If θ had just one or two components, we could graph the posterior density or its contours and could tell quite a bit at a glance. Unfortunately, in 10 or 20 dimensions, that isn't feasible.

Since the log likelihood itself for a model like this never takes the form of a well-known distribution, the posterior is also not of a known form, regardless of the choice of prior. Thus, we can't simply describe it as say, a Normal or t with a given mean and covariance matrix. Instead, we need to use some form of simulation technique to explore the shape of the posterior. The method generally used is known as *Random Walk Metropolis-Hastings*. This is one of a more general set of methods known as *Markov Chain Monte Carlo* or MCMC, which is described briefly in Appendix F and covered in greater detail in the *Bayesian Econometrics* course. The procedure followed is:

1. Pick an initial set of θ . An obvious choice is the maximizer for the log posterior density. Call this $\theta^{(0)}$.
2. For draw i , create a test set θ^* by adding to $\theta^{(i-1)}$ a randomly chosen increment.
3. Evaluate the log posterior density at the test set. Use that to compute $\alpha = p(\theta^*|Y)/p(\theta^{(i-1)}|Y)$.
4. With probability α , accept the new draw, so $\theta^{(i)} = \theta^*$; otherwise, stay put: $\theta^{(i)} = \theta^{(i-1)}$.
5. Go to step 2 and repeat as often as necessary.

Under fairly general circumstances, this produces a sequence which (eventually) creates draws from the posterior density itself. We can describe that density using various functions of the sequence of $\theta^{(i)}$, such as means and variances.

Everything in this is fairly mechanical except step 2. What randomly chosen increment should be used? Remember that this is in k space. If we start at the maximizer for the posterior density, and just take a k vector of independent standard Normals, we may take millions of draws before getting one accepted at step 4 if the unit variance of the draws is way out of line with the scale of the parameters. A multivariate Normal is likely to be a good choice, but it needs to be chosen based upon the scale of the parameters. Given that step 1 starts

with an optimization problem, using the final inverse second derivative matrix from that is generally a good start. If the percentage of acceptances is too low, you can usually just scale that down to improve the behavior.

The setup for the simulation is largely the same regardless of the rest of the problem. First, we need to set the number of draws that we want and the number of “burn-in” draws. Because the process above only *eventually* produces the correct distribution, we discard a certain number of early draws when that might not yet be true.

```
compute nburn =5000
compute ndraws=25000
compute accept=0
```

The following is used to allow us to keep entire histories of the draws. While you can get means and variances by using running sums, histograms and percentiles need full histories.¹ NBETA here is the number of parameters.

```
dec series[vect] bgibbs
gset bgibbs 1 ndraws = %zeros(nbeta,1)
```

We’ll look first at the general frame of the simulation loop, ignoring the logic for acceptance and rejection.

```
infobox(action=define,progress,lower=-nburn,upper=ndraws) $
  "Random Walk MH"
do draw=-nburn,ndraws
  *
  infobox(current=draw) $
    %strval(100.0*accept/(draw+nburn+1),"##.##")
  if draw<=0
    next
  *
  *   Do the bookkeeping here.
  *
  compute bgibbs(draw)=%parmspeek(pset)
end do draw
infobox(action=remove)
```

The **INFOBOX** instructions show the progress for the loop, which can take quite a while for large models or parameter sets. The nice thing about this is that it shows a running percentage of acceptances. If that’s too low (below 20%), or too high (close to 100%), you may want to try adjusting the variance on the increment.² You can hit the “Cancel” button on the progress box without having to wait for it to finish.

¹The Gibbs in the names refers to Gibbs sampling, which is another MCMC technique.

²You might wonder why too high a value is bad. In Random Walk M-H, you will usually only get that if the increments are so small that you’re never leaving the immediate neighborhood.

PSET is the PARMSET (parameter set) which is being used to organize the parameters. The loop runs from `-NBURN` to `+NDRAWS`. As long as `DRAW` is non-positive, we're in the burn-in draws that we don't want to save. (We go through the same draw and test procedure, we just don't save the results). As written, this actually discards `NBURN+1` draws which simplifies the coding.

Inside the actual loop, we have the following:

```
compute parmlast=%parmspeek(pset)
compute %parmspoke(pset,parmlast+%ranmvnormal(fxx))
compute logptest=EvalPosterior()
```

The first line retrieves the current settings; the second adds the random Normal increment and replaces the parameter set with the new value. The third line calculates the log posterior—we'll have to write a function which computes that.

This is the acceptance code:

```
if %valid(logptest)
  compute %a=exp(logptest-logplast)
else
  compute %a=0.0

if %a>1.0.or.%uniform(0.0,1.0)<%a {
  compute accept=accept+1
  compute logplast=logptest
}
else
  compute %parmspoke(pset,parmlast)
```

`%VALID` checks for missing values. If the log density isn't computable, we want the acceptance probability to be zero. If we at least have a value, we compute the probability of acceptance. The calculation is kept in log form up until now because it's quite possible for the probabilities themselves to be out-of-range of the floating point representation on the computer.³ If we accept the new value, we increment the `ACCEPT` counter and save the function value so we don't have to compute it again. Otherwise, we put the previous value back into the parameter set.

All of that is basically the same regardless of the model. Different applications will use different values of `NBURN` and `NDRAWS`, but everything else stays as is. What we need for a specific application is to create the `EvalPosterior`

The idea behind step 4 is that, while you will usually stay in areas of high density, you at least have some chance of moving to areas with lower density, allowing a more complete examination of the shape.

³A log likelihood of +900 has a likelihood that would overflow a typical computer, and a log likelihood of -900 has a likelihood that "underflows" so it could be represented only as zero.

function, do the maximization to get the initial settings and choose the `FXX` matrix.

Our first example is the same model as **12.1**. The first fifty or so lines, which read the data, set up the model and define the functions to solve it are the same as before. We now need to add the function:

```
function EvalModel
compute SolveModel()
dlm(a=adlm,f=fdlm,sw=%diag(||sig_eps^2,sig_eta^2||),y=||x,mu||,$
    c=cdlm,g=gdlm,method=solve)
compute EvalModel=%logl
end
```

This solves the model into state-space form, and evaluates it (using **DLM** with `METHOD=SOLVE`) at the current settings. This uses uniform priors on all parameters. This isn't really a good idea (particularly for the standard deviations), but it simplifies some of the demonstration. The obvious way to indicate a uniform prior is to use the upper and lower bounds; however, this is replicating an example where the priors are put in as mean and standard deviation. The `%UniformParms` function backs out a 2-vector with the upper and lower bounds using the mean and standard deviation as inputs.

```
source uniformparms.src
compute alphaparms =%UniformParms(-5.0,2.0)
compute lambdaparms =%UniformParms(0.68,0.5)
compute sig_etaparms=%UniformParms(0.5,0.25)
compute sig_epsparms=%UniformParms(0.5,0.25)
```

For a uniform prior, the density is a constant inside the supported range. So the only thing that `EvalPosterior` needs to do is reject the values that are out of range, returning `%NA`. The actual log prior density can be ignored since it will just add to the unknown constant that we're already ignoring. This includes an extra check for a combination of α and λ at which the function can't be computed because of a division by (near) zero. This really isn't part of the "prior", just a protection against bad behavior in the function. Since the likelihood there is zero, the posterior is zero.

```

function EvalPosterior
if abs(lambda+(1-lambda)*alpha)<1.e-3.or.$
    alpha<alphaparms(1).or.alpha>alphaparms(2).or.$
    lambda<lambdaparms(1).or.lambda>lambdaparms(2).or.$
    sig_eta<sig_etaparms(1).or.sig_eta>sig_etaparms(2).or.$
    sig_eps<sig_epsparms(1).or.sig_eps>sig_epsparms(2) {
        compute EvalPosterior=%na
    }
    return
}
compute EvalPosterior=EvalModel()
end

```

We can't use **DLM** to maximize a general posterior density. We *could* use it here because the posterior is the same as the likelihood within the 4-cube support of the prior, but that won't be true with any other type of prior. Since the posterior is a general function of the parameters, the instruction that we use to maximize it is **FIND**. Because we've already written the code to evaluate the function of interest, this is quite simple. As with maximum likelihood, this is doing a first set of iterations to improve the estimates of the variances before doing the full set. This is where we define the **PARMSET** used in the Monte Carlo procedure.

```

nonlin sig_eta sig_eps
find(method=simplex, iters=5, noprint) maximum EvalPosterior()
end find
nonlin(parmset=pset) alpha lambda sig_eta sig_eps
find(method=bfgs, parmset=pset, stderrs) maximum EvalPosterior()
end find

```

The **STDERRS** option is needed because we want the inverse Hessian, which will be in **%XX**. You also need **METHOD=BFGS**, which is *not* the default for **FIND**, in order to compute that.⁴

The **FXX** matrix used in drawing the increment needs to be a factor (matrix square root) of the covariance matrix for the multivariate Normal. The first try for this will usually be **%DECOMP(%XX)**. If you do that, you get an acceptance rate of about 28%. That's not bad, but by reducing the scale to half that, you get a percentage more like 50%. If you want to get a feel for how Random Walk M-H works, try playing with the .5 multiplier on the next line. Make sure that you rerun starting with **ACCEPT=0**.

```
compute [rect] fxx=.5*%decomp(%xx)
```

The post-loop processing takes the generated histories and generates a histogram using the **DENSITY** instruction. **DENSITY** has a *wizard*⁵ which does the

⁴The default is **METHOD=SIMPLEX**. **FIND** with **METHOD=SIMPLEX** can be used for any continuous function, while **METHOD=BFGS** requires a function which is twice continuously differentiable.

⁵(*Kernel*) *Density Estimation* on the *Statistics* menu

DENSITY and **SCATTER** instructions. You might need to make a change to the default band width (**BAND** option), as the default tends to be a bit too narrow for use in graphics. **BGIBBS** is a **VECT[SERIES]** just like the ones used for state vectors, so extracting one component should be a familiar instruction by now.

```
set alphas 1 ndraws = bgibbs(t) (1)
density(grid=automatic,maxgrid=100,band=.75) $
  alphas 1 ndraws xalpha falpha
scatter(style=line,vmin=0.0,footer="Posterior for alpha")
# xalpha falpha
```

12.3 Tips and Tricks

Most RATS instructions which do non-linear optimization include a `REJECT` option. With this, you provide a function or formula that evaluates to “true” (non-zero) for any situation where you want the overall function to return an NA. This is done immediately (before any other calculations) and so can be helpful if you want to avoid doing a calculation that is nonsensical, or if you need to protect some part of your calculation that might fail dramatically (that is, give an error) under some circumstances. We had the following code:

```
dlim(startup=SolveModel(),y=yf,$
      a=afull,f=ffull,c=cfull,sw=swfull,presample=ergodic,$
      pmethod=simplex,piters=5,method=bfgs,itors=100,$
      reject=(eta<1.0.or.eta>1.05.or.rho>=1.0)) 1948:1 2002:2
```

ETA is a (gross) growth rate. If it's less than 1.0, the whole solution process makes no sense. If it's too large (it's a quarter to quarter rate, so 1.05 is 20+% per year), there are numerical problems solving for the steady state of the DSGE. In the early iterations of a non-linear estimation, it's quite possible for rather extreme values to come up for evaluation. In most cases, you'll just get a really low likelihood (or a natural NA, if, for instance, they would require taking the log of a negative number), and they'll be rejected on that basis. `REJECT` can (and should) be used if something worse happens than a really low likelihood.

You can't use `REJECT` as a cheap way to impose a restriction when the boundary is feasible. If, for instance, your function is computable at $x \geq 1$ and you want to impose $x \leq 1$, you need to do a parameter set that includes

```
x x<=1.0
```

The option `REJECT=(X>1.0)` will cause the optimization to fail to converge if the function is increasing at $x = 1$. Given the shape of the function, the optimization will naturally be testing values of x larger than 1. When it's told that the value there is NA (as a result of the `REJECT`), it will cut the step size, but will probably have to cut it quite a few times before getting down below 1.0. The next iteration will likely need even more subiterations in order to get under 1, etc. Eventually, it will hit a subiterations limit.

The proper course is to use the constrained optimization, which, in this case, will come in from above $x = 1$. It uses a penalty function for values above 1.0, but that penalty function starts relatively small, so at least initially (given the shape of the function), it will have a higher total (likelihood less penalty) for values above 1. The penalty function increases as you iterate more, eventually dominating the function value and forcing x towards 1. You'll probably end up with the final estimated x being something like 1.000001.

Example 12.1 Maximum Likelihood: Hyperinflation Model

```

open data cagan_data.prn
data(format=prn,org=cols) 1 34 mu x
*
declare series a1 a2 eps eta
declare real    alpha lambda sig_eta sig_eps
*
* There's no particular reason to substitute out the current a1 and a2
* in the first two equations.
*
frml(identity) f1 = x -(x{1}+a1-lambda*a1{1})
frml(identity) f2 = mu-((1-lambda)*x{1}+lambda*mu{1}+a2-lambda*a2{1})
frml(identity) f3 = a1-1.0/(lambda+(1-lambda)*alpha)*(eps-eta)
frml(identity) f4 = a2-(1.0/(lambda+(1-lambda)*alpha)*$
                    ((1+alpha*(1-lambda))*eps-(1-lambda)*eta))
frml          d1 = eps
frml          d2 = eta
*
group cagan f1 f2 f3 f4 d1 d2
*
* lambda+(1-lambda)*alpha needs to stay clear of its zero point. It
* appears that it needs to be negative, so alpha must be less than
* -lambda/(1-lambda) on the guess values.
*
compute alpha=-3.00,lambda=.7,sig_eta=.001,sig_eps=.001
*****
function EvalModel
dsge(model=cagan,a=adlm,f=fdlm) x mu a1 a2 eps eta
end EvalModel
*****
compute EvalModel()
compute cdlm=%identity(2)~~%zeros(%rows(adlm)-2,2)
compute gdlm=%dlmgfoma(adlm)
*
* Because we really have no idea what the scale is on the variances, we
* first estimate the model with the alpha and lambda fixed. This uses
* only a small number of simplex iterations to get the sigmas into the
* right zone.
*
nonlin sig_eta sig_eps
dlm(start=%(EvalModel()),sw=%diag(||sig_eps^2,sig_eta^2||)), $
    a=adlm,f=fdlm,y=|x,mu|,c=cdlm,sw=sw,g=gdlm,exact,$
    method=simplex,itors=5,noprint)
*
nonlin alpha lambda sig_eta sig_eps
dlm(start=%(EvalModel()),sw=%diag(||sig_eps^2,sig_eta^2||)), $
    a=adlm,f=fdlm,y=|x,mu|,c=cdlm,sw=sw,g=gdlm,exact,$
    pmethod=simplex,pitors=5,method=bfgs)

```

Example 12.2 Maximum Likelihood: Hansen RBC

```

calendar(q) 1948
all 2002:02
*
* Data file has data transformed for final use
*
open data ych.dat
data(format=free,org=columns) 1948:1 2002:2 yt ct ht
*
* We have to use <<in>>, since <<i>> is a reserved name. We use yt, ct
* and ht for the actual data to keep them separate from the formal
* series being used in the model.
*
dec series y c h k in a
dec real a0 theta eta delta gamma beta rho sigma
*
* This is written with k lagged a period from the model in the paper,
* so that k(t) is determined at t.
*
frml(identity) f1 = y-(a*k{1}^theta*h^(1-theta))
frml(identity) f2 = y-(c+in)
frml(identity) f3 = eta*k-((1-delta)*k{1}+in)
frml(identity) f4 = gamma*c*h-(1-theta)*y
frml(identity) f5 = eta/c-(beta*(1/c{-1})*(theta*(y{-1}/k)+(1-delta)))
frml f6 = log(a)-((1-rho)*log(a0)+rho*log(a{1}))
*
* These are pegged
*
compute beta =.99
compute delta=.025
*
* Initial guess values (quite close to optimum)
*
compute gamma=.0045
compute theta=.2000
compute eta =1.0051
compute a0 =6.0000
compute rho =.9975
compute sigma=.0055
*
* Coefficients on the VAR(1) for the measurement errors
*
dec rect d(3,3)
input d
1.4187 0.2251 -0.4441
0.0935 1.0236 -0.0908
0.7775 0.3706 0.2398
*
* The covariance matrix of the measurement errors is estimated in
* Cholesky factor form.
*
dec symm sigmav(3,3)
dec packed lsigmav(3,3)

```



```

input lsigmav
    0.0072
    0.0040 0.0057
    0.0015 0.0010 0.0000
*
* Do an initial solution to the model to get initial guesses for the
* steady state. We'll use <<basesteady>> as the guess values when
* repeatedly solving the model.
*
sstats(mean) 1948:1 * yt>>ymean ct>>cmean ht>>hmean
*
group dsgemodel f1 f2 f3 f4 f5 f6
dsge(model=dsgemodel,expand=loglinear,a=ax,f=fx,steady=basesteady) $
    y<<ymean c<<cmean h<<hmean in<<ymean-cmean k a
*
* The full state space model includes both the linearized DSGE variables
* and the measurement errors. Those need to be part of the state vector
* because they follow an VAR(1) process. If they were serially
* uncorrelated, they could just be handled with the SV option on DLM.
*
compute cfull=%identity(3)~~%zeros(%rows(ax)-3,3)~~%identity(3)
*
* This function is called at the start of each function evaluation. It
* solves out the DSGE model for the current set of parameters, and
* appends the matrices for the VAR(1) representation.
*
function SolveModel
*
* Guard against solving the model with problem values
*
if eta<1.0.or.eta>1.05.or.rho>=1.0
    return
dsge(model=dsgemodel,expand=loglinear,a=ax,f=fx,$
    steady=steady,initial=basesteady) y c h in k a
compute afull=ax~\d,ffull=fx~\%identity(3)
compute [symmetric] swfull=sigma^2~\%ltouterxx(lsigmav)
end SolveModel
*
nonlin gamma theta eta a0 rho sigma d lsigmav
*
* The observables are log(variable/steadystate) with y and c also
* logarithmically detrended. Values of eta less than 1 and rho bigger
* than one are rejected to avoid bad behavior in the solution process.
* We also reject etas that are too big, which turn out to cause problems
* computing the steady state.
*
dec frml[vect] yf
frml yf = ||log(yt*eta^(-t)/steady(1)), $
        log(ct*eta^(-t)/steady(2)), $
        log(ht/steady(3))||
*
* Full sample estimates. Note that the <<D>> matrix is shown in a
* different order than in Table I in the paper. Our <<D>> is listed by
* columns, rather than by rows. This also shows the estimated covariance

```

```

* matrix in its Cholesky factor form. The paper shows the standard
* errors and covariances instead. We do the recalculated versions below.
*
dln(startup=SolveModel(),y=yf,$
    a=afull,f=ffull,c=cfull,sw=swfull,presample=ergodic,$
    pmethod=simplex,piters=5,method=bfgs,itters=100,$
    reject=(eta<1.0.or.eta>1.05.or.rho>=1.0)) 1948:1 2002:2

```

Example 12.3 Bayesian Estimation: Hyperinflation Model

```

open data cagan_data.prn
data(format=prn,org=cols) 1 34 mu x
*
declare series a1 a2 eps eta
declare real    alpha lambda sig_eta sig_eps
*
* There's no particular reason to substitute out the current a1 and a2
* in the first two equations.
*
* Note that the model in this form has no expectational terms. It
* could be coded directly for input into DLM; however, DSGE can
* produce the state-space model as well.
*
frml(identity) f1 = x-(x{1}+a1-lambda*a1{1})
frml(identity) f2 = mu-((1-lambda)*x{1}+lambda*mu{1}+a2-lambda*a2{1})
frml(identity) f3 = a1-1.0/(lambda+(1-lambda)*alpha)*(eps-eta)
frml(identity) f4 = a2-1.0/(lambda+(1-lambda)*alpha)*$
                    ((1+alpha*(1-lambda))*eps-(1-lambda)*eta)
frml          d1 = eps
frml          d2 = eta
*
group cagan f1 f2 f3 f4 d1 d2
compute alpha=-2.344,lambda=.5921,sig_eta=.10,sig_eps=.10
*****
*
* This solves the model to produce the state space form
*
function SolveModel
dsge(model=cagan,a=adlm,f=fdlm) x mu a1 a2 eps eta
end SolveModel
*****
*
* Since the dimension of A can increase as part of the solution by
* DSGE, this solves the model once, then sets up the C matrix for DLM
* based upon the size of the generated A matrix.
*
compute SolveModel()
compute cdlm=%identity(2)~~%zeros(%rows(adlm)-2,2)
compute gdlm=%dlmgfoma(adlm)
compute gdlm=|1.0,-1.0|~\%identity(%rows(adlm)-2)
*****
*
* This solves the model, then evaluates the log likelihood

```

```

*
function EvalModel
compute SolveModel()
dlim(a=adlim,f=fdlm,sw=%diag(||sig_eps^2,sig_eta^2||),y=||x,mu||,$
    c=cdlim,g=gdlm,method=solve)
compute EvalModel=%logl
end
*****
*
* Get the ranges for the priors. If you directly specify uniforms, just
* use the upper and lower bounds. We are using this because the original
* code specified the uniforms based upon mean and standard deviation.
*
source uniformparms.src
compute alphaparms =%UniformParms(-5.0,2.0)
compute lambdaparms =%UniformParms(0.68,0.5)
compute sig_etaparms=%UniformParms(0.5,0.25)
compute sig_epsparms=%UniformParms(0.5,0.25)
*****
*
* This solves the model, then evaluates the posterior. Since all
* components of the prior are uniform, we don't even have to worry
* about the densities; we just need to reject out of range values.
* This also discards values which come close to a zero divisor in f3
* and f4.
*
function EvalPosterior
if abs(lambda+(1-lambda)*alpha)<1.e-3.or.$
    alpha<alphaparms(1).or.alpha>alphaparms(2).or.$
    lambda<lambdaparms(1).or.lambda>lambdaparms(2).or.$
    sig_eta<sig_etaparms(1).or.sig_eta>sig_etaparms(2).or.$
    sig_eps<sig_epsparms(1).or.sig_eps>sig_epsparms(2) {
compute EvalPosterior=%na
return
}
}
*
* Since all the priors are uniform, the log densities will be constant
* once we've excluded out of range values.
*
compute EvalPosterior=EvalModel()
end
*****
*
* Maximize the posterior. As with the ML for this model, we do this in
* two stages, first estimating the two standard deviations, then doing
* the full parameter set. Because the prior is flat, in this case,
* this is the same as ML.
*
nonlin sig_eta sig_eps
find(method=simplex, iters=5, noprint) maximum EvalPosterior()
end find
nonlin(parmset=pset) alpha lambda sig_eta sig_eps
find(method=bfgs, parmset=pset, stderrs) maximum EvalPosterior()
end find

```

```

*
* Start at the posterior mode
*
compute nbeta    =%nreg
compute logplast=%funcval
*
compute nburn    =5000
compute ndraws=25000
compute accept=0
*
dec series[vect] bgibbs
gset bgibbs 1 ndraws = %zeros(nbeta,1)
*
* Acceptance rate with a full step is a bit low, so we take a reduced
* size increment.
*
compute [rect] fxx=0.5*%decomp(%xx)
*
infobox(action=define,progress,lower=-nburn,upper=ndraws) $
  "Random Walk MH"
do draw=-nburn,ndraws
  compute parmlast=%parmspeek(pset)
  compute %parmspoke(pset,parmlast+%ranmvnormal(fxx))
  compute logptest=EvalPosterior()
  if %valid(logptest)
    compute %a=exp(logptest-logplast)
  else
    compute %a=0.0

  if %a>1.0.or.%uniform(0.0,1.0)<%a {
    compute accept=accept+1
    compute logplast=logptest
  }
  else
    compute %parmspoke(pset,parmlast)
*
infobox(current=draw) $
  %strval(100.0*accept/(draw+nburn+1),"##.##")
if draw<=0
  next
*
* Do the bookkeeping here.
*
compute bgibbs(draw)=%parmspeek(pset)
end do draw
infobox(action=remove)
*
@mcmcpostproc(ndraws=ndraws,mean=bmean,stderrs=bstderrs,cd=bcd) bgibbs
*
set alphas 1 ndraws = bgibbs(t)(1)
density(grid=automatic,maxgrid=100,band=.75) $
  alphas 1 ndraws xalpha falpha
scatter(style=line,vmin=0.0,footer="Posterior for alpha")
# xalpha falpha

```

```
*
set lambdas 1 ndraws = bgibbs(t) (2)
density(grid=automatic,maxgrid=100,band=.10) $
  lambdas 1 ndraws xlambda flambda
scatter(style=line,vmin=0.0,footer="Posterior for lambda")
# xlambda flambda
*
set sigetas 1 ndraws = bgibbs(t) (3)
density(grid=automatic,maxgrid=100,band=.05) $
  sigetas 1 ndraws xsigeta fsigeta
scatter(style=line,vmin=0.0,footer="Posterior for sig_eta")
# xsigeta fsigeta
*
set sigepss 1 ndraws = bgibbs(t) (4)
density(grid=automatic,maxgrid=100,band=.003) $
  sigepss 1 ndraws xsigeps fsigeps
scatter(style=line,vmin=0.0,footer="Posterior for sig_eps")
# xsigeps fsigeps
```

Probability Distributions

A.1 Uniform

Parameters	Lower (a) and upper (b) bounds
Kernel	1
Support	$[a, b]$
Mean	$\frac{a + b}{2}$
Variance	$\frac{(b - a)^2}{12}$
Main uses	priors and approximate posteriors for parameters that have a limited range.
Density function	trivial
Moment Matching	<code>%UniformParms(mean, sd)</code> (external function) returns the 2-vector of parameters (a, b) for a uniform with the given mean and standard deviation.
Random Draws	<code>%UNIFORM(a, b)</code> draws one or more (depending upon the target) independent $U(a, b)$.
Notes	$U(0, 1)$ is a special case of the beta distribution (beta parameters are 1 and 1).

A.2 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.3 Beta distribution

Parameters	2 (called α and β below)
Kernel	$x^{\alpha-1} (1-x)^{\beta-1}$
Support	$[0, 1]$. If $\alpha > 1$, the density is 0 at $x = 0$; if $\beta > 1$, the density is 0 at $x = 1$.
Mean	$\frac{\alpha}{\alpha + \beta}$
Variance	$\frac{\alpha\beta}{(\alpha + \beta)^2 (\alpha + \beta + 1)}$
Main uses	priors and approximate posteriors for parameters that measure fractions, or probabilities, or autoregressive coefficients (when a negative value is unreasonable).
Density function	<code>%LOGBETADENSITY (x, a, b)</code>
Random Draws	<code>%RANBETA (a, b)</code> draws one or more (depending on the target) independent Betas.
Moment Matching	<code>%BetaParms(mean,sd)</code> (external function) returns the 2-vector of parameters for a beta with the given mean and standard deviation.
Extensions	If $x \sim \text{Beta}(\alpha, \beta)$ then $ax + b$ is distributed on $[a, a + b]$

A.4 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.5 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.7 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.6 Chi-Squared Distribution

Parameters	Degrees of freedom (ν).
Kernel	$x^{(\nu-2)/2} \exp(-x/2)$
Range	$[0, \infty)$
Mean	ν
Variance	2ν
Main uses	Asymptotic distribution. Prior, exact and approximate posterior for the precision (reciprocal of variance) of residuals or other shocks in a model.
Density function	<code>%CHISQRDENSITY(x, nu)</code> is the (non-logged) density with <code>nu</code> degrees of freedom at <code>x</code> .
Tail Probability	<code>%CHISQR(x, nu)</code> returns the probability that a chi-squared with <code>nu</code> degrees of freedom exceeds <code>x</code> .
Inverse Tail Probability	<code>%INVCHISQR(p, nu)</code> returns the critical value for probability p .
Random Draws	<code>%RANCHISQR(nu)</code> draws one or more (depending upon the target) independent chi-squareds with <code>NU</code> degrees of freedom.

A.7 (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.5) 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 sd 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.8 Bernoulli Distribution

Parameters	probability of success (p)
Kernel	$p^x(1 - p)^{1-x}$
Range	0 or 1
Mean	p
Variance	$p(1 - p)$
Main uses	Realizations of 0-1 valued variables (usually latent states).
Random Draws	<p><code>%RANBRANCH(p1, p2)</code> draws one or more independent trials with (integer) values 1 or 2. The probability of 1 is $\frac{p1}{p1 + p2}$ and the probability of 2 is $\frac{p2}{p1 + p2}$. (Thus, you only need to compute the relative probabilities of the two states). The 1-2 coding for this is due to the use of 1-based subscripts in RATS .</p>

A.9 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.10 Wishart Distribution

Parameters	Scaling \mathbf{A} (symmetric $n \times n$ matrix) and degrees of freedom (ν). This only has a proper density if $\nu > n - 1$ and \mathbf{A} is positive definite.
Kernel	$\exp\left(-\frac{1}{2}\text{trace}(\mathbf{A}^{-1}\mathbf{X})\right) \mathbf{X} ^{\frac{1}{2}(\nu-n-1)}$
Support	Positive definite symmetric matrices
Mean	$\nu \mathbf{A}$
Main uses	Prior, exact and approximate posterior for the precision matrix (inverse of covariance matrix) of residuals in a multivariate regression, though that is mainly in the inverse form (Appendix A.11) since that would be the distribution of the covariance matrix itself.
Random Draws	<p><code>%RANWISHART (n, nu)</code> draws a single $n \times n$ Wishart matrix with $\mathbf{A} = \mathbf{I}$ and degrees of freedom ν.</p> <p><code>%RANWISHARTF (F, nu)</code> draws a single $n \times n$ Wishart matrix with $\mathbf{A} = \mathbf{F}\mathbf{F}'$ and degrees of freedom ν. \mathbf{F} can be any factor of \mathbf{A}, but would typically be computed as the Cholesky factor using <code>%DECOMP</code>.</p>

A.11 Inverse Wishart Distribution

Parameters	Scaling Ψ (symmetric $n \times n$ matrix) and degrees of freedom (ν). This only has a proper density if $\nu > n - 1$ and Ψ is positive definite.
Kernel	$\exp\left(-\frac{1}{2}\text{trace}(\Psi\mathbf{X}^{-1})\right) \mathbf{X} ^{-\frac{1}{2}(\nu+n+1)}$
Support	Positive definite symmetric matrices
Mean	$\frac{1}{\nu-n-1}\Psi$
Main uses	Prior, exact and approximate posterior for the covariance matrix of residuals in a multivariate regression with Gaussian residuals.
Diffuse versions	The density function is improper if $\nu < n - 1$, but the improper prior with $\nu = 0$ and $\Psi = 0$ has kernel $ \mathbf{X} ^{-\frac{1}{2}(n+1)}$ which forms the Jeffrey's prior for inference on the covariance matrix.
Combining Densities	If $\mathbf{X} \sim IW(\Psi_1, \nu_1)$ and $\mathbf{X} \sim IW(\Psi_2, \nu_2)$ are inverse Wishart densities for \mathbf{X} , then the posterior from combining them has $\mathbf{X} \sim IW(\Psi_1 + \Psi_2, \nu_1 + \nu_2 + n + 1)$.
Random Draws	<code>%RANWISHARTI(F, nu)</code> draws a single $n \times n$ inverse Wishart matrix with $\mathbf{F}\mathbf{F}' = \Psi^{-1}$ and degrees of freedom ν . Note that \mathbf{F} needs to be a factor of the <i>inverse</i> . \mathbf{F} can be any factor matrix, but is typically the Cholesky factor, computed using <code>%DECOMP</code> .
Notes	The basic result has the data evidence on the covariance matrix of Gaussian residuals summarized as an inverse Wishart with $\Psi = T\hat{\Sigma}$, where T is the number of observations and $\hat{\Sigma}$ the sample covariance matrix of residuals (thus Ψ itself is the sum of the outer products of the residuals). The degrees of freedom for the inverse Wishart from the data itself are typically $T - (n + 1)$ (sometimes less some additional adjustments for regressors, depending upon the form of conditioning). The $n + 1$ is needed because you don't really have the ability to estimate a covariance matrix until you have that many observations. Combining data with the Jeffrey's prior "corrects" the degrees of freedom so the posterior value of $\nu = T$.

An informative prior is generally based upon a prior belief on the value of \mathbf{X} . Because \mathbf{X} is typically the covariance matrix Σ , call this Σ_0 . The corresponding value of Ψ for this is $\alpha\Sigma_0$ where the prior degrees of freedom are $\alpha + n + 1$. Combined with data, this gives an inverse Wishart with $T + \alpha$ degrees of freedom and Ψ matrix which is $T\hat{\Sigma} + \alpha\Sigma_0$, thus (roughly) sample and non-sample information on Σ weighted by the number of actual and “dummy” observations.

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

$$Q(\beta) = (\beta - \hat{\beta})' \hat{\mathbf{H}} (\beta - \hat{\beta}) + (\beta - \beta^*)' \mathbf{H} (\beta - \beta^*) \quad (\text{B.1})$$

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

$$Q(\beta) = (\beta - \bar{\beta})' \bar{\mathbf{H}} (\beta - \bar{\beta}) + Q(\bar{\beta}) \quad (\text{B.2})$$

The first term in (B.2) has value 0 at $\bar{\beta}$, so using this and (B.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.

Non-Standard Matrix Calculations

Non-standard matrix inversion was introduced into the statistics literature by Koopman (1997).¹ The goal is to compute an exact (limit) inverse of an input (symmetric) matrix of the form²

$$\mathbf{A} + \kappa \mathbf{B} \text{ as } \kappa \rightarrow \infty$$

The result will be the matrix of the form

$$\mathbf{C} + \mathbf{D}\kappa^{-1} + \mathbf{E}\kappa^{-2} \quad (\text{C.1})$$

Note that the κ^{-2} term isn't needed in many applications. With the help of this expansion, it's possible to compute exact limits as $\kappa \rightarrow \infty$ for calculations like:

$$(\mathbf{F} + \mathbf{G}\kappa)(\mathbf{A} + \mathbf{B}\kappa)^{-1} \approx (\mathbf{F} + \mathbf{G}\kappa)(\mathbf{C} + \mathbf{D}\kappa^{-1}) \rightarrow \mathbf{F}\mathbf{C} + \mathbf{G}\mathbf{D}$$

You might think that you could just use a guess matrix of the form (C.1) multiply, match terms and be done quickly. Unfortunately, because matrices generally don't commute, it's very easy to get a left inverse or a right inverse, but not a true inverse. The matching terms idea is correct, but it has to be done carefully. We will start with a case that's more manageable. Note that this derivation is simpler than that in Koopman's paper. Here, the "infinite" matrix is isolated into an identity block, and we look only at the expansion through κ^{-1} .

$$\left(\begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}'_{12} & \mathbf{A}_{22} \end{bmatrix} + \kappa \begin{bmatrix} \mathbf{I}_r & 0 \\ 0 & 0 \end{bmatrix} \right) \times \quad (\text{C.2})$$

$$\left(\begin{bmatrix} \mathbf{C}_{11} & \mathbf{C}_{12} \\ \mathbf{C}'_{12} & \mathbf{C}_{22} \end{bmatrix} + \kappa^{-1} \begin{bmatrix} \mathbf{D}_{11} & \mathbf{D}_{12} \\ \mathbf{D}'_{12} & \mathbf{D}_{22} \end{bmatrix} \right) = \begin{bmatrix} \mathbf{I}_r & 0 \\ 0 & \mathbf{I}_{n-r} \end{bmatrix} + \kappa^{-1} \text{Rem}$$

Since the κ term in the product has to zero out, $\mathbf{C}_{11} = 0$ and $\mathbf{C}_{12} = 0$. Using that and collecting the $O(1)$ terms, we get

$$\left(\begin{bmatrix} \mathbf{D}_{11} & \mathbf{A}_{12}\mathbf{C}_{22} + \mathbf{D}_{12} \\ 0 & \mathbf{A}_{22}\mathbf{C}_{22} \end{bmatrix} \right) = \begin{bmatrix} \mathbf{I}_r & 0 \\ 0 & \mathbf{I}_{n-r} \end{bmatrix}$$

¹*Non-standard analysis* was introduced into mathematics in the 1970's using results from "model theory" to embed the real numbers within a framework that includes "infinite" and "infinitesimal" numbers. It was hoped that this would allow simpler descriptions of limit calculations in real analysis, but never really caught on. See Blass (1978) for a brief survey of the ideas behind this.

²We're using conventional choices of \mathbf{A} , \mathbf{B} for the names of the matrices. These are not intended to match with the use of those names elsewhere in the paper.

So we get $\mathbf{D}_{11} = \mathbf{I}_r$, $\mathbf{C}_{22} = \mathbf{A}_{22}^{-1}$ and $\mathbf{D}_{12} = -\mathbf{A}_{12}\mathbf{A}_{22}^{-1}$. \mathbf{D}_{22} is arbitrary; it just ends up in the remainder, so for simplicity we can make it zero. If we check the reverse multiplication

$$\left(\begin{bmatrix} 0 & 0 \\ 0 & \mathbf{C}_{22} \end{bmatrix} + \kappa^{-1} \begin{bmatrix} \mathbf{I}_r & \mathbf{D}_{12} \\ \mathbf{D}'_{12} & 0 \end{bmatrix} \right) \left(\begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}'_{12} & \mathbf{A}_{22} \end{bmatrix} + \kappa \begin{bmatrix} \mathbf{I}_r & 0 \\ 0 & 0 \end{bmatrix} \right)$$

we can verify that it also will be the identity + a term in κ^{-1} .

In general, we won't have an input matrix with the nice form in (C.2). However, for a p.s.d. symmetric matrix \mathbf{B} , we can always find a non-singular matrix \mathbf{T} such that $\mathbf{T}\mathbf{B}\mathbf{T}'$ has that form.³ So, in general, we can compute the inverse with:

$$(\mathbf{A} + \kappa\mathbf{B})^{-1} = \mathbf{T}'(\mathbf{T}\mathbf{A}\mathbf{T}' + \kappa\mathbf{T}\mathbf{B}\mathbf{T}')^{-1}\mathbf{T} \quad (\text{C.3})$$

For an input matrix pair $\{\mathbf{A}, \mathbf{B}\}$, this will produce an output pair $\{\mathbf{C}, \mathbf{D}\}$.

To derive the more accurate expansion in the case with the simpler form of \mathbf{B} , our test inverse is

$$\left(\begin{bmatrix} \mathbf{C}_{11} & \mathbf{C}_{12} \\ \mathbf{C}'_{12} & \mathbf{C}_{22} \end{bmatrix} + \kappa^{-1} \begin{bmatrix} \mathbf{D}_{11} & \mathbf{D}_{12} \\ \mathbf{D}'_{12} & \mathbf{D}_{22} \end{bmatrix} + \kappa^{-2} \begin{bmatrix} \mathbf{E}_{11} & \mathbf{E}_{12} \\ \mathbf{E}'_{12} & \mathbf{E}_{22} \end{bmatrix} \right)$$

Everything from above goes through, except we now can't make \mathbf{D}_{22} arbitrary. When we multiply out, the κ^{-1} terms are

$$\begin{aligned} & \begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}'_{12} & \mathbf{A}_{22} \end{bmatrix} \begin{bmatrix} \mathbf{D}_{11} & \mathbf{D}_{12} \\ \mathbf{D}'_{12} & \mathbf{D}_{22} \end{bmatrix} + \begin{bmatrix} \mathbf{I}_r & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{E}_{11} & \mathbf{E}_{12} \\ \mathbf{E}'_{12} & \mathbf{E}_{22} \end{bmatrix} \text{ or} \\ & \begin{bmatrix} \mathbf{A}_{11}\mathbf{D}_{11} + \mathbf{A}_{12}\mathbf{D}'_{12} & \mathbf{A}_{11}\mathbf{D}_{12} + \mathbf{A}_{12}\mathbf{D}_{22} \\ \mathbf{A}'_{12}\mathbf{D}_{11} + \mathbf{A}_{22}\mathbf{D}'_{12} & \mathbf{A}'_{12}\mathbf{D}_{12} + \mathbf{A}_{22}\mathbf{D}_{22} \end{bmatrix} + \begin{bmatrix} \mathbf{E}_{11} & \mathbf{E}_{12} \\ 0 & 0 \end{bmatrix} \end{aligned}$$

The bottom left element in the \mathbf{AD} matrix is zero because of the first order solution. Since \mathbf{D}_{22} was arbitrary from before, we can now solve for it as

$$\mathbf{D}_{22} = -\mathbf{A}_{22}^{-1}\mathbf{A}'_{12}\mathbf{D}_{12} = \mathbf{A}_{22}^{-1}\mathbf{A}'_{12}\mathbf{A}_{12}\mathbf{A}_{22}^{-1}.$$

With that, we also get

$$\mathbf{E}_{11} = -\mathbf{A}_{11}\mathbf{D}_{11} - \mathbf{A}_{12}\mathbf{D}'_{12} = -\mathbf{A}_{11} + \mathbf{A}_{12}\mathbf{A}_{22}^{-1}\mathbf{A}'_{12}$$

and

$$\mathbf{E}_{12} = -\mathbf{A}_{11}\mathbf{D}_{12} - \mathbf{A}_{12}\mathbf{D}_{22} = (\mathbf{A}_{11} - \mathbf{A}_{12}\mathbf{A}_{22}^{-1}\mathbf{A}'_{12})\mathbf{A}_{12}\mathbf{A}_{22}^{-1}$$

\mathbf{E}_{22} is now arbitrary. In terms of the input matrix, this is:

$$\begin{aligned} & \begin{bmatrix} 0 & 0 \\ 0 & \mathbf{A}_{22}^{-1} \end{bmatrix} + \kappa^{-1} \begin{bmatrix} \mathbf{I} & \mathbf{A}_{12}\mathbf{A}_{22}^{-1} \\ \mathbf{A}_{22}^{-1}\mathbf{A}'_{12} & \mathbf{A}_{22}^{-1}\mathbf{A}'_{12}\mathbf{A}_{12}\mathbf{A}_{22}^{-1} \end{bmatrix} + \\ & \kappa^{-2} \begin{bmatrix} -\mathbf{A}_{11} + \mathbf{A}_{12}\mathbf{A}_{22}^{-1}\mathbf{A}'_{12} & (\mathbf{A}_{11} - \mathbf{A}_{12}\mathbf{A}_{22}^{-1}\mathbf{A}'_{12})\mathbf{A}_{12}\mathbf{A}_{22}^{-1} \\ \mathbf{A}_{22}^{-1}\mathbf{A}'_{12}(\mathbf{A}_{11} - \mathbf{A}_{12}\mathbf{A}_{22}^{-1}\mathbf{A}'_{12}) & 0 \end{bmatrix} \quad (\text{C.4}) \end{aligned}$$

³The simplest to compute is based upon a modified version of the Cholesky factorization.

The extension of this to the more general \mathbf{B} matrix is as before. In the typical situation, the transforming \mathbf{T} matrix will take the form

$$\begin{bmatrix} \mathbf{T}_1 \\ \mathbf{T}_2 \end{bmatrix}$$

where \mathbf{T}_1 is $r \times n$ and \mathbf{T}_2 is $(n-r) \times n$. If that's the case, then the component matrices in the formula (C.4) are $\mathbf{A}_{11} = \mathbf{T}_1 \mathbf{A} \mathbf{T}'_1$, $\mathbf{A}_{12} = \mathbf{T}_1 \mathbf{A} \mathbf{T}'$ and $\mathbf{A}_{22} = \mathbf{T}_2 \mathbf{A} \mathbf{T}'_2$. The expanded inverse will be (C.4) premultiplied by \mathbf{T}' and postmultiplied by \mathbf{T} .

A General Result on Smoothing

Proposition. Let y and x be random variables defined on a common probability space (Ω, \mathcal{F}, P) . Let \mathcal{I} and \mathcal{J} be sub-sigma fields (information sets) of \mathcal{F} with $\mathcal{I} \subseteq \mathcal{J}$. If

$$f(x|y, \mathcal{I}) = f(x|y, \mathcal{J}) \quad (\text{D.1})$$

then

$$\frac{f(x|\mathcal{J})}{f(x|\mathcal{I})} = \int f(y|x, \mathcal{I}) \frac{f(y|\mathcal{J})}{f(y|\mathcal{I})} dy \quad (\text{D.2})$$

Proof

By Bayes' rule

$$f(x|y, \mathcal{I}) = \frac{f(y|x, \mathcal{I})f(x|\mathcal{I})}{f(y|\mathcal{I})}$$

By (D.1), $f(x|y, \mathcal{I}) = f(x|y, \mathcal{J})$, so we have

$$f(x|y, \mathcal{J}) = \frac{f(y|x, \mathcal{I})f(x|\mathcal{I})}{f(y|\mathcal{I})} \quad (\text{D.3})$$

By standard results,

$$f(x|\mathcal{J}) = \int f(x, y|\mathcal{J}) dy = \int f(x|y, \mathcal{J})f(y|\mathcal{J}) dy \quad (\text{D.4})$$

Substituting (D.3) into (D.4) yields

$$f(x|\mathcal{J}) = \int \frac{f(y|x, \mathcal{I})f(x|\mathcal{I})}{f(y|\mathcal{I})} f(y|\mathcal{J}) dy$$

which can be rearranged to yield (D.2).

Generalized Ergodic Initialization

We start with the general representation (with fixed system matrices):

$$\mathbf{X}_t = \mathbf{A}\mathbf{X}_{t-1} + \mathbf{Z} + \mathbf{F}\mathbf{W}_t$$

For any full-rank matrix \mathbf{Q} , the following is an equivalent state-space representation:

$$\mathbf{X}_t^{(q)} \equiv \mathbf{Q}\mathbf{X}_t = \mathbf{Q}\mathbf{A}\mathbf{Q}^{-1}\mathbf{X}_{t-1}^{(q)} + \mathbf{Q}\mathbf{Z}_t + \mathbf{Q}\mathbf{F}\mathbf{W}_t$$

If we can compute the ergodic mean and variance of $\mathbf{X}^{(q)}$ as $E\mathbf{X}^{(q)}$ and $\Sigma^{(q)}$, then we can transform back to the original space with $E\mathbf{X} = \mathbf{Q}^{-1}E\mathbf{X}^{(q)}$ and $\text{var}\mathbf{X} = \mathbf{Q}^{-1}\Sigma^{(q)}\mathbf{Q}'^{-1}$. We would like to find choices of \mathbf{Q} which will simplify the calculation of the ergodic variance.

Suppose that we are fortunate enough to have a \mathbf{Q} such that $\Lambda = \mathbf{Q}\mathbf{A}\mathbf{Q}^{-1}$ is diagonal. This will be the case if \mathbf{A} is diagonalizable with real eigenvalues. If we write $\mathbf{M} = \mathbf{Q}\mathbf{F}\Sigma_{\mathbf{W}}\mathbf{F}'\mathbf{Q}'$, then the ergodic variance of the transformed problem solves:

$$\Lambda\Sigma^{(q)}\Lambda + \mathbf{M} = \Sigma^{(q)}$$

or

$$\sigma_{ij}^{(q)} = m_{ij}/(1 - \lambda_i\lambda_j) \tag{E.1}$$

where λ_i is diagonal element i of Λ . Because computing eigenvalues and eigenvectors is, in general, an $O(n^3)$ calculation,¹ this is quite a bit faster than the “textbook” solution for the variance, even for n as small as six. It has the further advantage that it isolates the non-stationary roots. If we look at (E.1), if $|\lambda_i| \geq 1$, $\sigma_{ii}^{(q)}$ has either a divide by zero, or is negative. The extension to handle non-stationary roots is as follows:

1. If $|\lambda_i| < 1$ and $|\lambda_j| < 1$, solve for $\sigma_{ij}^{(q)}$ using (E.1).
2. If $|\lambda_i| \geq 1$, make $\sigma_{ij}^{(q)}$ (and, by symmetry $\sigma_{ji}^{(q)}$) zero. In a parallel “diffuse” matrix, make $\sigma_{ii}^{(\infty)} = 1$.
3. All non-diagonal elements of $\sigma^{(\infty)}$ are equal to zero.

¹Because the final step in computing eigenvalues is iterative, the computational complexity isn’t precisely known. The $O(n^3)$ is based upon a fixed number of iterations per eigenvalue.

Two things to note with this. First, (E.1) *has* a solution for $\sigma_{ij}^{(q)}$ when $|\lambda_i| = 1$ but $|\lambda_j| < 1$. However, according to rule 2, we make those zero. Non-zero values for those (covariances between stationary and non-stationary states) end up not mattering; they drop out of the limit calculations. If you think about it, those are finite (fixed) covariances, but the variance of one of the two variables is “infinite”. Thus, the correlation between the components is (in effect) zero. Second, if both λ_i and λ_j are 1, then (E.1) gives an apparent value of ∞ . According to rule 3, we make the diffuse i, j component zero. This is for convenience. The “shape” of the diffuse component doesn’t matter—as long as we have a full rank matrix covering the correct subspace, we get the same results once we’ve seen enough data. This is proved in Doan (2010).

The diffuse matrix (if it’s non-zero) computed in this way is transformed back to the original \mathbf{X} space just as we do with the finite covariance matrix. We will call the combined stationary-diffuse initialization computed this way the *Generalized Ergodic Initialization*.

The only problem with this treatment is that it won’t apply to all matrices. However, there is an alternative which accomplishes the same improvement in speed and handles the non-stationary roots in the same fashion. This is the Schur Decomposition. Every square matrix \mathbf{A} has a (not necessarily unique) representation as $\mathbf{A} = \mathbf{Q}\mathbf{R}\mathbf{Q}^{-1}$ where \mathbf{Q} is a unitary matrix² and \mathbf{R} is an upper (or “right”) triangular matrix. This will likely produce complex matrices—to avoid that, it’s possible to use the “real” Schur decomposition, which also always exists. For that, \mathbf{Q} is a (real) unitary matrix, and \mathbf{R} is a real matrix that is block-upper triangular, where the diagonal blocks are either 1×1 (for real eigenvalues) or 2×2 for pairs of complex eigenvalues.

The Schur decomposition is an $O(n^3)$ calculation, but is less ambitious than the calculation of eigenvalues and eigenvectors, and so takes less time. On the other hand, the solution for the ergodic covariance matrix is quite a bit more involved than (E.1)—it’s also an $O(n^3)$ operation. In the end, it appears that the Schur method is somewhat faster than the eigenvalue method when both can be applied.

The `PRESAMPLE=ERGODIC` option will do the initialization using the method described here and we would recommend that as the quickest and simplest handling for the state-space model with time-invariant components. There is an alternative which allows the user himself to isolate the non-stationary states. This is the `G` option..

Suppose that we have an $r \times N$ matrix \mathbf{G} which transforms the states to stationarity. This has to be full rank with r equal to the number of stationary eigenvalues of \mathbf{A} . Let \mathbf{H} be a full rank $(N - r) \times N$ matrix that satisfies $\mathbf{G}\mathbf{H}' = 0$. Such a matrix can always be constructed (using, for instance, singular value

²The inverse of a unitary matrix is its conjugate transpose, or, for a matrix with only real elements, its simple transpose.

decomposition), but we don't actually need to construct it. The transforming matrix for the states is then

$$\mathbf{Q} \equiv \begin{bmatrix} \mathbf{G} \\ \mathbf{H} \end{bmatrix}$$

If the \mathbf{G} matrix has been chosen correctly, $\mathbf{A}^* = \mathbf{Q}\mathbf{A}\mathbf{Q}^{-1}$ will be block lower triangular where the top left $r \times r$ will have no unit roots. It can be verified that

$$\mathbf{Q}^{-1} = \begin{bmatrix} \mathbf{G}'(\mathbf{G}\mathbf{G}')^{-1} & \mathbf{H}'(\mathbf{H}\mathbf{H}')^{-1} \end{bmatrix}$$

so the submatrix of \mathbf{A}^* for the stationary part is $\mathbf{G}\mathbf{A}\mathbf{G}'(\mathbf{G}\mathbf{G}')^{-1}$. The transformed system now has the states blocked into stationary and non-stationary. Its Generalized Ergodic variance is block diagonal with the solution to the stationary block in the top left and (for convenience) an identity matrix in the bottom right. Transforming back gives the mixed stationary-diffuse initialization for the original model.

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

$$\alpha = \frac{f(\theta)}{f(\theta^{(i-1)})} \times \frac{q(\theta^{(i-1)})}{q(\theta)} \quad (\text{F.1})$$

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 (F.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:

$$\alpha = \frac{f(\theta)}{q(\theta|\theta^{(i-1)})} \bigg/ \frac{f(\theta^{(i-1)})}{q(\theta^{(i-1)}|\theta)} \quad (\text{F.2})$$

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

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.

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.

Importance Sampling

Monte Carlo integration is fairly straightforward if the density f is an easy one from which to generate draws. However, many models do not produce convenient distributions. One method which can be used for more difficult densities is known as importance sampling. For technical details beyond those provided here, see Geweke (1989).

Importance sampling attacks an expectation over an unwieldy density function using

$$\begin{aligned} E_f(h(x)) &= \int h(x) f(x) dx \\ &= \int h(x) (f(x)/g(x)) g(x) dx \\ &= E_g(hf/g) \end{aligned}$$

where f and g are density functions, with g having convenient Monte Carlo properties. If f and g are true density functions (that is, they integrate to 1), we can use

$$\hat{h} = (1/n) \sum h(x_i) f(x_i)/g(x_i) \quad (\text{G.1})$$

to estimate the desired expectation, where the x_i are drawn independently from g . If $\int |h(x)| f(x) dx < \infty$, then $\int |h(x) (f(x)/g(x))| g(x) dx < \infty$ so the strong law of large numbers applies to (G.1), and \hat{h} will converge a.s. to $E_f(h(x))$.

This simple calculation (G.1) is based upon f and g being true density functions. In reality, the integrating constants are either unknown or very complicated. If we don't know f/g , just $w = f^*/g^* = cf/g$, where c is an unknown constant, then

$$\begin{aligned} (1/n) \sum h(x_i) w(x_i) &= (1/n) \sum h(x_i) cf(x_i)/g(x_i) \rightarrow cE_f(h(x)) \\ (1/n) \sum w(x_i) &= (1/n) \sum cf(x_i)/g(x_i) \rightarrow cE_f(1) = c \\ (1/n) \sum (h(x_i) w(x_i))^2 &= (1/n) \sum (h(x_i) cf(x_i)/g(x_i))^2 \rightarrow c^2 E_g(h(x) f(x)/g(x))^2 \end{aligned}$$

Using the second of these to estimate c gives the key results

$$\begin{aligned} \hat{h} &= \sum h(x_i) w(x_i) / \sum w(x_i) \\ s_h^2 &= \left[\sum (h(x_i) w(x_i))^2 / \left(\sum w(x_i) \right)^2 \right] - \hat{h}^2 / n \end{aligned} \quad (\text{G.2})$$

The main task in importance sampling is choose a proper g . In most applications, it's crucial to avoid choosing a g with tails that are too thin relative to f . If the variance of hf/g doesn't exist, the convergence of the sample means may be extremely slow. If, on the other hand, that variance *does* exist, the Central Limit Theorem will apply, and we can expect convergence at the rate $n^{1/2}$.

To take a trivial example, where the properties can be determined analytically, suppose $h(x) = x$, and f is a Normal with mean zero and variance 4. Suppose we choose as g a standard Normal. Then $f/g = \frac{1}{2} \exp(+\frac{3}{8}x^2)$ and

$$\int (h(x) f(x)/g(x))^2 g(x) dx = \int \frac{x^2}{4\sqrt{2\pi}} \exp\left(+\frac{1}{4}x^2\right) dx = \infty$$

Convergence of \hat{h} with this choice of g will be painfully slow.

Suppose now that f is a Normal with mean zero and variance 1/4 and again we choose as g a standard Normal. Now

$$\int (h(x) f(x)/g(x))^2 g(x) dx = \int \frac{4x^2}{\sqrt{2\pi}} \exp\left(-\frac{7}{2}x^2\right) dx = \frac{4}{7\sqrt{7}} \approx .216$$

For this particular h , not only does the importance sampling work, but, in fact, it works *better* than independent draws from the true f density. The standard error of the importance sampling estimate is $\sqrt{.216/n}$, while that for draws from f would be $\sqrt{.25/n}$. This result depends on the shape of the h function—in this case, the importance function gives a lower variance by oversampling the values where h is larger. If h goes to zero in the tails, sampling from g will still work, but won't do better than draws from f . (A thin-tailed g works respectably only for such an h .)

The lesson to be learned from this is that, in practice, it's probably a good idea to be conservative in the choice of g . When in doubt, scale variances up or switch to fatter-tailed distributions or both. For instance, the most typical choice for an importance function is the asymptotic distribution from maximum likelihood estimates. You're likely to get better results if you use a fatter-tailed t rather than the Normal.

There's one additional numerical problem that needs to be avoided in implementing this. Particularly for large parameter spaces, the omitted integrating constants in the density functions can be huge. As a result, a direct calculation of w can produce machine overflows or underflows. (The typical computer can handle real numbers up to around 10^{300}). To avoid this, we would advise computing w by

$$\exp(\log f^*(x) - \log f_{\max}^* - \log g^*(x) + \log g_{\max}^*)$$

where f_{\max}^* and g_{\max}^* are the maximum values taken by the two kernel functions.

Now all of the above shows how to compute the expectation of a measurable function of the random variable x . If you want to compute quantiles, you need

to use the function `%WFRACILES`. Quantiles are estimated by sorting the generated values and locating the smallest value for which the cumulated normalized weights exceeds the requested quantile. The proof of this is in the Geweke article. If you want to estimate a density function, add the `WEIGHT` option to your **DENSITY** instruction.

Appendix H

VAR Likelihood Function

Most of what we will do here applies not just to VAR's, but to any linear systems regression with identical explanatory variables in each equation. For instance, in finance it's common to have several returns regressed upon a common set of factors. If those factors are observable (or are constructed separately), we have exactly this type of model.

It's most convenient to write this in terms of the precision matrix $\mathbf{H} \equiv \Sigma^{-1}$. With identical explanatory variables, the model can be written in either the form:

$$y_t = (\mathbf{I}_m \otimes x_t) \beta + \varepsilon_t, \varepsilon_t \sim N(0, \mathbf{H}^{-1}), i.i.d.$$

or

$$y_t = \mathbf{B} x_t' + \varepsilon_t \tag{H.1}$$

where x_t is the k vector of explanatory variables, \otimes is the Kroneker product, β is the km vector of coefficients for the full system and \mathbf{B} is the reshaped $m \times k$ matrix of the same. \mathbf{B} “vec’ed” by rows gives β , that is, the first k elements of β are the first row of \mathbf{B} , the next k elements are the second row, etc. The first form will be more convenient for analyzing the β , while the second will be better for analyzing the precision matrix, and for computing the likelihood function. For analyzing β , we’re best off keeping the likelihood for a data point t in the form:

$$p(y_t | x_t, \beta, \mathbf{H}) \propto |\mathbf{H}|^{1/2} \exp \left(-\frac{1}{2} (y_t - (\mathbf{I}_m \otimes x_t) \beta)' \mathbf{H} (y_t - (\mathbf{I}_m \otimes x_t) \beta) \right)$$

When we multiply across t , we get the following as the quadratic form involving β :

$$\sum (y_t - (\mathbf{I}_M \otimes x_t) \beta)' \mathbf{H} (y_t - (\mathbf{I}_M \otimes x_t) \beta)$$

or

$$\sum y_t \mathbf{H} y_t' - 2\beta' \sum (\mathbf{I}_m \otimes x_t)' \mathbf{H} y_t + \beta' \sum (\mathbf{I}_M \otimes x_t)' \mathbf{H} (\mathbf{I}_m \otimes x_t) \beta$$

(Since this is a scalar, the two cross terms are equal, hence the $2 \times$)

One of the convenient properties of the Kroneker product is that

$$(\mathbf{A} \otimes \mathbf{B})(\mathbf{C} \otimes \mathbf{D}) = \mathbf{AC} \otimes \mathbf{BD}$$

if the dimensions work.

In particular,

$$(\mathbf{I} \otimes \mathbf{B})(\mathbf{C} \otimes \mathbf{I}) = \mathbf{C} \otimes \mathbf{B}$$

allowing certain types of matrices to “commute”. Shrewdly using \mathbf{I}_1 matrices (that is, the constant 1), allows us to do the following:

$$(\mathbf{I}_m \otimes x_t)' \mathbf{H} (\mathbf{I}_m \otimes x_t) \Rightarrow (\mathbf{I}_m \otimes x_t)' (\mathbf{H} \otimes \mathbf{I}_1) (\mathbf{I}_m \otimes x_t) \Rightarrow \mathbf{H} \otimes x_t' x_t$$

and

$$\begin{aligned} \sum (\mathbf{I}_m \otimes x_t)' \mathbf{H} y_t &\Rightarrow \sum (\mathbf{H} \otimes x_t)' y_t \Rightarrow \\ \sum (\mathbf{H} \otimes \mathbf{I}_k) (\mathbf{I}_m \otimes x_t)' (y_t \otimes \mathbf{I}_1) &\Rightarrow \sum (\mathbf{H} \otimes \mathbf{I}_k) (y_t \otimes x_t') \end{aligned} \quad (\text{H.2})$$

which gives us

$$\sum y_t' \mathbf{H} y_t - 2\beta' \sum (\mathbf{H} \otimes \mathbf{I}_k) (y_t \otimes x_t') + \beta' \sum (\mathbf{H} \otimes x_t' x_t) \beta$$

or (after pulling sums through)

$$\sum y_t' \mathbf{H} y_t - 2\beta' (\mathbf{H} \otimes \mathbf{I}_k) \sum (y_t \otimes x_t') + \beta' \left(\mathbf{H} \otimes \sum x_t' x_t \right) \beta$$

From the properties of multivariate Normals (Appendix B), we read off that the precision of β is

$$\hat{\mathbf{H}} = \mathbf{H} \otimes \sum x_t' x_t$$

and the mean is

$$\hat{\beta} = \left(\mathbf{H} \otimes \sum x_t' x_t \right)^{-1} (\mathbf{H} \otimes \mathbf{I}_k) \sum (y_t \otimes x_t') = \left(\mathbf{I}_m \otimes \sum x_t' x_t \right)^{-1} \sum (y_t \otimes x_t') \quad (\text{H.3})$$

Since $\sum (y_t \otimes x_t')$ is just a vec'ed version of the equation by equation $\sum x_t' y_{it}$ vectors, $\hat{\beta}$ is just equation by equation least squares, independent of \mathbf{H} . If we *didn't* have identical explanatory variables, that is, if we have a more general SUR model, we don't get this result because it's the properties of the Kroneker product that allow us to pull the \mathbf{H} out from between the x and y factors in (H.2).

The classical (non-Bayesian) result here is that the maximum likelihood estimator is OLS equation by equation with covariance matrix

$$\hat{\mathbf{H}}^{-1} = \Sigma \otimes \left(\sum x_t' x_t \right)^{-1} \quad (\text{H.4})$$

If we use the matrix form (H.1), we can rewrite the likelihood as:

$$p(\mathbf{Y}|\mathbf{X}, \mathbf{B}, \mathbf{H}) \propto |\mathbf{H}|^{T/2} \exp \left(-\frac{1}{2} \text{trace} \mathbf{H} (T \times \Sigma(\mathbf{B})) \right) \quad (\text{H.5})$$

where we define

$$\Sigma(\mathbf{B}) = \frac{1}{T} \sum (y_t - \mathbf{B}x_t') (y_t - \mathbf{B}x_t')'$$

which is the covariance matrix evaluated at the matrix of coefficients \mathbf{B} . For the classical result, we can take the log of this to get

$$\frac{T}{2} \log |\mathbf{H}| - \frac{T}{2} \text{trace } \mathbf{H} \Sigma(\mathbf{B}) \quad (\text{H.6})$$

The derivative of $\log |\mathbf{H}|$ with respect to its elements is \mathbf{H}'^{-1} and the derivative of $\text{trace } \mathbf{H} \Sigma(\mathbf{B})$ with respect to the elements of \mathbf{H} is just $\Sigma(\mathbf{B})$. Since \mathbf{H} is symmetric, the solution for the first order conditions is $\mathbf{H}^{-1} = \Sigma(\mathbf{B})$ or $\Sigma = \Sigma(\mathbf{B})$. For a Bayesian analysis of this, part of (H.5) will be the density for $\hat{\beta}$. That needs an integrating constant of

$$\left| \mathbf{H} \otimes \sum x'_t x_t \right|^{1/2} = |\mathbf{H}|^{k/2} \left| \sum x'_t x_t \right|^{m/2}$$

(This is another property of Kroneker products). The $|\sum x'_t x_t|$ is just data, so we can ignore it. We do, however, need to allow for the $|\mathbf{H}|^{k/2}$ factor. The exponent in the kernel for β has to be zero at the mean $\hat{\beta}$, so the “leftover” exponent has to be the non-zero part at $\hat{\beta}$, allowing us to rewrite (H.5) as:

$$p(\mathbf{Y}|\mathbf{X}, \mathbf{B}, \mathbf{H}) \propto |\mathbf{H}|^{(T-k)/2} \exp \left(-\frac{1}{2} \text{trace } \mathbf{H} (T \times \Sigma(\hat{\mathbf{B}})) \right) \times p(\beta|\mathbf{Y}, \mathbf{H})$$

The standard non-informative (“flat”) prior for this model takes the form

$$p(\beta, \mathbf{H}) \propto |\mathbf{H}|^{-\frac{m+1}{2}}$$

allowing us to read off the posterior for \mathbf{H} as

$$p(\mathbf{H}|\mathbf{Y}) \propto |\mathbf{H}|^{\frac{T-k-(m+1)}{2}} \exp \left(-\frac{1}{2} \text{trace } \mathbf{H} (T \times \Sigma(\hat{\mathbf{B}})) \right)$$

This is a Wishart with $T - k$ degrees of freedom and scale matrix $(T \times \Sigma(\hat{\mathbf{B}}))^{-1}$.

The scale matrix on this is a function just of the data (it depends upon $\hat{\beta}$, not \mathbf{B}), so we can do direct draws from it. What we end up needing for all other purposes is actually $\Sigma \equiv \mathbf{H}^{-1}$, so we’ll actually use the `%RANWISHARTI` function, which draws an inverse Wishart.

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)

$$\mathbf{B}_T = \frac{1}{T} \sum_{t=1}^T \left(\frac{\partial \log f_t}{\partial \theta} \right)' \left(\frac{\partial \log f_t}{\partial \theta} \right) \quad (\text{I.1})$$

with the derivatives evaluated at $\hat{\theta}$.¹ Serial correlation in the gradients is handled by a Newey-West type calculation in (I.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.

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

- Bai J 2004 *Journal of Econometrics* **122**(1), 137–183.
- Barillas F, Colacito R, Kitao S, Matthes C, Sargent T J & Shin Y 2007 *Practicing Dynare*.
- Bernanke B, Boivin J & Elias P 2005 *Quarterly Journal of Economics* **120**(1), 387–422.
- Blass A 1978 *Bulletin of the American Mathematical Society* **84**(1), 34–41.
- Bloem A M, Dippelsman R J & Maehle N 2001 Quarterly national accounts manual: Concepts, data sources and compilation Technical report International Monetary Fund.
URL: <http://www.imf.org/external/pubs/ft/qna/2000/Textbook/ch1.pdf>
- Brock W A & Mirman L J 1972 *Journal of Economic Theory* **4**(3), 479–513.
- Brockwell P J & Davis R A 2002 *Introduction to Time Series Forecasting* 2nd edn New York: Springer-Verlag.
- Brown R, Durbin J & Evans J M 1975 *JRSS-B* **37**(2), 149–192.
- Carter C & Kohn R 1994 *Biometrika* **81**(3), 541–553.
- Chow G C & Lin A L 1971 *Review of Economics and Statistics* **53**(4), 372–375.
- Clark P K 1987 *Quarterly Journal of Economics* **102**, 797–814.
- Commandeur J J F & Koopman S 2007 *An Introduction to State Space Time Series Analysis* Oxford: Oxford University Press.
- Diebold F X, Rudebusch G D & Aruoba S B 2006 *Journal of Econometrics* **131**(1), 309–338.
- Doan T A 2010 *Estima Technical Paper* (1).
- Durbin J & Koopman S 2012 *Time Series Analysis by State Space Methods* 2nd edn Oxford: Oxford University Press.
- Fernandez R B 1981 *Review of Economics and Statistics* **63**(3), 471–478.
- Geweke J 1989 *Econometrica* **57**(6), 1317–1339.
- Hamilton J 1994 *Time Series Analysis* Princeton: Princeton University Press.

- Hansen G D 1985 *Journal of Monetary Economics* **16**(3), 309–327.
- Harvey A C 1989 *Forecasting, structural time series models and the Kalman filter* Cambridge: Cambridge University Press.
- Harvey A C, Ruiz E & Shepard N 1994 *Review of Economic Studies* **61**(2), 247–264.
- Hodrick R J & Prescott E C 1997 *Journal of Money, Credit and Banking* **29**(1), 1–16.
- Ireland P 2004 *Journal of Economic Dynamics and Control* **28**(6), 1205–1226.
- Jones R H 1980 *Technometrics* **20**(3), 389–395.
- Kim C J & Nelson C R 1999 *State-Space Models with Regime Switching* MIT Press.
- Koopman S 1997 *Journal of American Statistical Association* **92**(6), 1630–1638.
- Koopman S, Shepard N & Doornik J A 1999 *Economics Journal* **2**, 113–166.
- Kozicki S 1999 *Journal of Economic Dynamics and Control* **23**(7), 997–1028.
- Litterman R B 1983 *Journal of Business and Economic Statistics* **1**(2), 169–173.
- Matheson T & Stavrev E 2013 *Economics Letters* **120**(3), 468–472.
- Minford P & Peel D 2002 *Advanced Macroeconomics: A Primer* Cheltenham: Elgar.
- Novalés A, Fernández E & Ruiz J 2009 *Economic Growth: Theory and Numerical Solution Methods* Springer.
- Perron P & Wada T 2009 *Journal of Monetary Economics* **56**(6), 749–765.
- Ruge-Murcia F J 2007 *Journal of Economic Dynamics and Control* **31**(8), 2599–2636.
- Sargent T J 1977 *International Economic Review* **18**(1), 59–82.
- Simon D J 2010 *IET Control Theory and Applications* pp. 1–16.
- Sims C 2002 *Computational Economics* **20**(1), 1–20.
- Smets F & Wouters R 2003 *Journal of the European Economic Association* **1**(5), 1123–1175.
- Stock J & Watson M 1991 in K Lahiri & G Moore, eds, ‘Leading Economic Indicators: New Approaches and Forecasting Records’ Cambridge: Cambridge University Press chapter 4.

West M & Harrison J 1997 *Bayesian Forecasting and Dynamic Models* New York: Springer.

White H 1994 *Estimation, Inference and Specification Analysis* Cambridge: Cambridge University Press.

Index

- ~ operator, 19
- ~~ operator, 19
- ~\ operator, 19

- A option, 4
- Absorbing state, 152, 156, 288
- Additive seasonal model, 34
- Airline model, 53
- @**ARMADLM** procedure, 29
- Aruoba, S., 115

- Bai, J., 118
- Basic structural model, 53
- Bernanke, B., 16
- Bernoulli distribution, 270
- Beta distribution, 265
- %**BICDF** function, 271
- Boivin, J., 16
- Brock, W., 241
- Burn-in, 152, 285

- C option, 4
- Calibration, 243
- Carter, C., 149
- Carter-Kohn simulation, 149
- %**CDF** function, 264
- Chi-squared distribution, 268, 269
- Chow, G., 144
- Chow-Lin algorithm, 144
- Clark model, 93
- Clark, P. K., 56, 93
- CONDITION option, 52
- @**CUSUMTests** procedure, 42
- CUTOFF option, 234

- Damped trend model, 67
- %**DENSITY** function, 264
- DENSITY** instruction, 291
- %**DIAG** function, 40
- Diebold, F., 115
- @**DISAGGREGATE** procedure, 144
- Distribution (of data series), 143

- @**DLMIRF** procedure, 239
- Dummy variable seasonal model, 34
- Dynare, 235, 243

- EKF, 188
- Eliasz, P., 16
- %**EQNXVECTOR** function, 16
- Ergodic solution, 47, 282
- EXACT option, 17
- Extended Kalman filter, 145, 184, 188
- Extreme value distribution, 185

- F option, 17
- Factor models, 113
- Fernandez, R., 144
- Fourier seasonal model, 34

- G option, 283
- Gamma distribution, 266, 267
- Gap model
 - bivariate, 123, 189
 - univariate, 66
- Geweke, J., 289
- Gibbs sampling, 150, 285
- Gumbel distribution, 185

- Hamilton, J., 12, 77
- Hansen, G., 239
- Harmonic seasonal model, 34
- Harvey, A., 53, 186
- Hodrick-Prescott filter, 31
 - multivariate, 111
- Hyperparameter, 37, 85

- Importance sampling, 199, 289
- Independence chain M-H, 286
- INFOBOX** instruction, 154
- INITIAL option, 233
- Interpolation, 143
- Inverse Wishart distribution, 273
- %**INVNORMAL** function, 264
- Ireland, P., 245

- Jones, R.H., 29
- Kalman filter, 4
 - extended, 188
 - non-linear, 188
 - with state restrictions, 205
- Kalman gain, 4, 49
- Kalman smoothing, 12
- Kim, C.-J., 58, 81
- KLIC, 296
- Kohn, R., 149
- Koopman, S. J., 49, 278
- Kozicki, S., 111
- Kullback-Liebler Information Criterion, 296
- Lin, A., 144
- Litterman, R., 144
- Local trend model, 29, 30
- @LocalDLM** procedure, 31, 40
- @LocalDLMInit** procedure, 40
- %LOGDENSITY** function, 264, 271
- %LTOUTERXX** function, 66
- M-H, *see* Metropolis-Hastings
- Markov Chain Monte Carlo, 150, 285
- Matheson, T., 189, 215
- MCMC, *see* Markov Chain Monte Carlo
- @MCMCPOSTPROC** procedure, 287
- Measurement equation, 1
- Metropolis-Hastings, 286
- %MID** function, 116
- Minford, P., 231
- Mirman, L., 241
- MU option, 5
- Multivariate Normal distribution, 271, 275
- Nelson, C. R., 58, 81
- Non-linear Kalman filter, 184, 188
- Normal distribution, 264
 - multivariate, 271, 275
- Observation equation, 1
- Overflow, 287
- PACKED matrix, 66
- Particle filter, 196, 206
- Peel, D., 231
- Perron, P., 60
- Phillips curve, 123, 189
- Pile-up effect, 86
- Precision, 276
- PRESAMPLE=ERGODIC** option, 52, 283
- PRESAMPLE=X0** option, 5
- PRESAMPLE=X1** option, 5
- Principal components, 117
- @PRINCOMP** procedure, 118
- @PRINFACTORS** procedure, 118
- Probability distributions
 - Bernoulli, 270
 - beta, 265
 - chi-squared, 268
 - gamma, 266
 - inverse chi-squared, 269
 - inverse gamma, 267
 - inverse Wishart, 273
 - multivariate normal, 271, 275
 - normal, 264
 - uniform, 263
 - Wishart, 272
- Proportional Denton method, 146
- Pseudo-code, 153, 201
- QMLE, 295
- Quasi-Maximum Likelihood Estimation, 295
- %RAN** function, 264
- %RANBRANCH** function, 270
- Random walk M-H, 287
- Rank of Observables, 59
- %RANMAT** function, 264, 271
- %RANMVNORMAL** function, 271
- %RANWISHART** function, 272
- %RANWISHARTF** function, 272
- %RANWISHARTI** function, 273, 294
- Recursive residuals, 42
- REJECT** option, 255
- RLS** instruction, 81
- Rudebusch, G., 115
- Ruge-Murcia, R., 243

- Ruiz, E., 186
- Sargent, T., 244
- %SCALAR function, 6
- Schur decomposition, 228
- @SeasonalDLM procedure, 35
- SH0 option, 51
- Shephard, N., 186
- Simon, D., 205
- Sims, C., 228
- Singular value decomposition. , 235
- Smets, F., 243
- SOLVEBY option, 235
- SSFPack, 49
- @STAMPDiags procedure, 42
- Standardized predictive residuals, 42
- START option, 18
- State equation, 1
- State variable, 1
- Stavrev, E., 189, 215
- STEADYSTATE option, 233
- Stochastic volatility model, 184
- Stock, J., 106
- SV option, 5
- SVHAT option, 6
- SW option, 5
- Sweep
 - in MCMC, 285
- SWEEP** instruction, 119
- SWHAT option, 13
- SX0 option, 5, 51
- Time-varying coefficients model, 9, 81, 156
- Transition equation, 1
- Trigonometric seasonal model, 34
- UC model, 2
- Underflow, 287
- Uniform distribution, 263
- %UNIFORM function, 263
- %UNIFORMPARMS function, 252, 263
- Unobservable components, 2
- VHAT option, 6
- Wada, K., 60
- Watson, M., 106
- %WFRACTILES function, 291
- WHAT option, 13
- White, H., 295
- Wishart distribution, 272
 - inverse, 273
- Wouters, R., 243
- X0 option, 5
- X11 seasonal adjustment, 32
- X12-ARIMA seasonal adjustment, 32
- Y option, 5
- YHAT option, 7
- %ZTEST function, 264