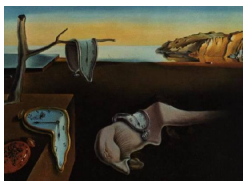


Journal of Statistical Software
May 2011, Volume 41, Issue 11. http://www.jstatsoft.org/

State Space Methods in gretl

Riccardo Lucchetti
Università Politecnica delle Marche

Abstract

gretl is a general-purpose econometric package, whose most important characteristic is being *free software*. This ensures that its source code is freely available under the general public license (GPL) and, like most GPL software, that it can be used free of charge. As of version 1.8.1 (released in May 2009), it offers a mechanism for handling linear state space models in a reasonably general and efficient way. This article illustrates its main features with two examples.

Keywords: Kalman filter, state space methods, unobserved components, free software.

1. Introduction

1.1. gretl and the free software movement

gretl (Cottrell and Lucchetti 2011) is an econometric package which aims to implement the widest possible array of statistical procedures. One of its main characteristics is that gretl is *free software*, in the sense that it is released under the general public license (GPL), like more famous software projects such as the Linux kernel, other statistical software such as R, or the GNU project. In fact, gretl is part of the GNU project; for details on the GPL and the GNU project, see Yalta and Lucchetti (2008) or Deek and McHugh (2008).

As most GPL-licensed software, gretl is available free of charge (it can be downloaded from `http://gretl.sourceforge.net/`), which explains at least in part its growing popularity (see Lucchetti 2009 for some statistics). Allin Cottrell, the founder and leader of the project, provides a history of gretl and its future prospects in Cottrell (2009).

To the end user, it may seem that the main consequence of gretl being free software is that its price is zero; however, there are many more aspects to take into consideration. For example, free software projects can (and, as a rule, do) benefit from one another in terms of code re-use: parts of gretl's source code were taken and adapted from several other free projects,

such as R (R Development Core Team 2011) and **Gnumeric** (The **Gnumeric** Team 2010) and in the same way other free projects (R, for one) incorporate some of gretl's code. gretl uses a number of free libraries which spare the development team from re-implementing some algorithms: for example, gretl employs the **FFTW** library (itself a GPL-licensed project — see Frigo and Johnson 2005) for the discrete Fourier transform, which ensures not only that the algorithms are among the best available, but also that any future improvement in **FFTW** will be automatically passed to gretl.

Apart from code reuse, another common trait which gretl shares with the rest of the free software world is the extreme friendliness and vivacity of the online community, which effectively acts as a 24/7 online helpdesk.

1.2. State-space modelling

Since its 1.8.1 version, which was released on 2009-05-21, gretl offers a mechanism for handling linear state space models in a reasonably general way. Given that gretl's friendly and intuitive graphical user interface (GUI) is commonly considered as one of its strongest points (see for example Smith and Mixon 2006), the reader may find it surprising that the only way to access the program's features discussed here is through scripting. In fact, creating a GUI which is intuitive yet general enough is not trivial and there has been some debate among the developers as to the best way to expose the state space modelling functionalities through the GUI, but a consensus has not been found.[1]

As a consequence, in order to describe the facilities that gretl offers for working with state space models, it is best to think of gretl as a programming language. The main idea behind gretl's implementation of state space models is that the user can define a state space model which becomes "the" model within the context in which it is defined and all subsequently called functions. The `kalman` environment is used to set it up and two specialized functions, `kfilter()` and `ksmooth()`,[2] are used to perform the actual filtering.

As an example, consider this very minimal script:

```
nulldata 10
set seed 12345
y = normal()

kalman
  obsy y
  obsymat 1
  obsvar 1
  statemat 1
  statevar 1
end kalman

series v
kfilter(&v)
print y v -o
```

[1] In fact, the gretl development team would welcome suggestions.

[2] An additional function, `ksimul()` is available for using state space models to generate simulated data, but will not be described here. See Cottrell and Lucchetti (2009) for details.

Leaving details aside for a moment, the structure of the script should be rather clear. The first three lines of code generate a white noise process 10 observations long. The next seven lines of code set up the state space model, which in this example is

$$y_t = \alpha_t + \varepsilon_t, \qquad \varepsilon_t \sim \text{NID}(0,1), \tag{1}$$

$$\alpha_{t+1} = \alpha_t + \eta_t, \qquad \eta_t \sim \text{NID}(0,1), \qquad t = 1, \ldots, 10. \tag{2}$$

The last three lines of code perform a forward pass of the filter, store the prediction errors, and print them together with the original data. Running the code above produces[3]:

```
              y           v

 1     1.954669    1.954669
 2     0.652640   -1.302028
 3    -0.168688   -1.255338
 4     0.394389    0.092325
 5    -0.055069   -0.414286
 6    -1.658005   -1.761118
 7    -0.464892    0.520464
 8     1.832629    2.496318
 9     1.530098    0.650977
10     1.711905    0.430458
```

If a more "structural" programming approach is needed, so that the algorithm is split into subprograms, it is important to note that a `kalman` environment is local to a function. So for example, suppose that the basic structure of a gretl script is as follows:

```
function do_stuff()
  ...
  kfilter()
  ...
end function

function ss1()
  kalman
     ...
  end kalman
  do_stuff()
end function

function ss2()
  kalman
     ...
  end kalman
```

[3]The actual numerical results for this example were produced with gretl version 1.9.1. Earlier versions of gretl may produce different output, since the algorithm for generating pseudo-random normal variates has been changed, switching from the Box-Muller algorithm to the ziggurat method (see Marsaglia and Tsang 2000) since gretl 1.8.7; a compatibility mode is provided for replicating results obtained with earlier versions.

```
  do_stuff()
end function
```

When function `ss1` is called (which fits state space model 1, say), the `do_stuff` function (which does the filtering, estimation, and so forth) will act on the state space model defined in that function, which may be completely different from the one specified in the function `ss2` (which fits state space model 2, say). The implementation of the forwards and backwards filtering algorithm is completely written in C (like the rest of gretl), which ensures excellent numerical performance.[4] If parameter estimation is necessary, it can be performed via the `mle` command, which uses numerical methods to maximize a likelihood function.

Hence, a highly idealized flowchart for implementing a statistical model amenable to a state space representation (such as (V)AR(I)MA, structural time series models and others) in gretl runs like this:

1. set up the model by defining its state space representation in a `kalman` block;
2. set up an `mle` block, in which the system matrices are filled with the unknown parameters and the likelihood, computed by running the forward pass filter via the `kfilter()` function, is maximized;
3. after estimation is done, the function `ksmooth()` can be used to perform fixed-interval smoothing and the quantities of interest, such as states, can be accessed by the user.

The two main ingredients, the `kalman` block and the `mle` block, can be briefly illustrated as follows.

1.3. The `kalman` block

A `kalman` block is a series of statements that describe the system matrices of the state space model. For example, the code snippet

```
kalman
  ...
  obsymat foo
  ...
end kalman
```

would tell gretl that a matrix called `foo` should be used as the system matrix which premultiplies the states in the observation equation (Z_t in the terminology adopted in Commandeur, Koopman, and Ooms 2011). The contents of the matrix `foo` can then be modified via ordinary gretl commands, without the need for re-declaring the whole model. Any element of the system matrices can therefore be assigned a value at any time in a very convenient and intuitive way, simply by modifying the corresponding matrix. Special syntax constructs are also available for specifying time-varying matrices. Table 1 translates gretl keywords into the notation adopted here.

[4] A handy way to explore gretl's source code is through its web-based CVS access: http://gretl.cvs.sourceforge.net/gretl/gretl/. Most of the code for the state space algorithms discussed here is in the `lib/src/kalman.c` file.

Keyword	Symbol
`obsy`	y_t
`obsymat`	Z_t
`obsvar`	H_t
`statemat`	T_t
`statevar`	Q_t
`inistate`	a_1
`inivar`	P_1

Table 1: `Kalman` block keywords.

Note that gretl's syntax does not provide the equivalent of the R_t matrix; however, this is not a serious limitations, since the `statevar` matrix needs not be invertible, so it is more accurate to say that the keyword `statevar` is used to specify $R_t Q_t R_t^\top$.[5]

Moreover, the additional keywords `obsx` and `obsxmat` can be used for introducing exogenous variables in the observation equation. Some defaults are available: for example, if the `inistate` keyword is omitted, a_1 is understood to be a zero vector (see Cottrell and Lucchetti 2009 for a complete list).[6] As of version 1.9.1, it is also possible to introduce an "intercept" into the state transition equation via the `stconst` keyword: that is, the term c_t in models for which the state transition equation can be written as

$$\alpha_{t+1} = c_t + T_t \alpha_t + \eta_t;$$

of course, this could have been accomplished without a special keyword by re-defining the state vector appropriately, so that it includes one or more zero-variance states, but this could be rather inefficient numerically.

Once the model is set up, the `kfilter()` function performs a forward pass and several quantities of interest become available: the one-step-ahead prediction errors v_t with associated variances F_t and also a series containing the individual contributions to the log-likelihood

$$\ell_t = -\frac{p}{2}\ln(2\pi) - 0.5\left(\ln|F_t| + v_t^\top F_t^{-1} v_t\right),$$

which may be accessed via the `$kalman_llt` keyword; their sum is available via the `$lnl` accessor.

In Section 2 we provide commented estimation of the local level model on the Nile dataset. Section 3 contains a more complex model.

1.4. The `mle` block

The `mle` command is gretl's generic procedure for maximum likelihood estimation. By default,

[5]Special syntax is available for state space models in which η_t and ε_t are contemporaneously correlated, so to match the general notation used, for example, in Harvey and Proietti (2005). For reasons of space, it will not be illustrated here; the interested reader will find more details in the gretl's User Guide.

[6]As the state space code is a relatively recent addition, the precise details of the syntax may change somewhat in the future, to accommodate new features or simplify its syntax. If they do, the gretl development team will make every effort to minimize the inconvenience of backward-incompatible changes.

it uses the quasi-Newton BFGS algorithm, although an option is available to use L-BFGS as an alternate algorithm (see Liu and Nocedal 1989).[7]

Although the user can supply analytical derivatives, gretl can use a numerical approximation which is based on a 4-step Richardson extrapolation algorithm. This algorithm choice ensures that the numerically computed score is remarkably accurate for standard problems.

A miniature example of the usage of `mle` can be given as follows. Consider the estimation of an AR(1) model by conditional ML. The log-likelihood for one observation can be written as

$$\ell_t = \ln \varphi \left(\frac{y_t - \rho y_{t-1}}{\sigma} \right) - \ln(\sigma),$$

where $\varphi()$ is the standard normal density function. This translates into the following code:

```
sigma = 1
rho = 0
mle loglik = ln(dnorm(e/sigma)) - ln(sigma)
  e = y - rho * y(-1)
  params rho sigma
end mle
```

Once the parameters are initialized, the `mle` code block comprises an expression for the log-likelihood, an indication of which parameters have to be estimated, via the `params` keyword, and any intermediate computation (in this case calculation of the forecast errors, `e`).

The `mle` command offers the user three choices for the computation of the covariance matrix of the estimate: the outer product of gradients (the default), the inverse Hessian (numerical), or the robust "sandwich" estimator (see for example Gourieroux, Monfort, and Trognon 1984).

2. Case 1: The local level model applied to the Nile data

gretl does not provide a built-in command for estimation of the local level model. However, such a model is reasonably simple to set up. In order to show a possible way to implement the local level model in gretl and use the Nile data as an example, we will first write a function performing the "core" task of estimating the model, and then call it from a "main" file.

2.1. The core model function

The local level model (described in more detail in the introductory article Commandeur *et al.* 2011 of this volume) can be written as

$$y_t = \mu_t + \varepsilon_t, \qquad \varepsilon_t \sim \text{NID}(0, \sigma_\varepsilon^2), \tag{3}$$

$$\mu_{t+1} = \mu_t + \xi_t, \qquad \xi_t \sim \text{NID}(0, \sigma_\xi^2), \tag{4}$$

where the only two parameters that must be estimated are the two variances σ_ε^2 and σ_ξ^2.

[7] Although some additional methods are used internally for several estimators (Newton-Raphson, BHHH, simulated annealing), gretl's `mle` command does not currently give the user the choice of maximization algorithms other than BFGS or L-BFGS. However, since BFGS is widely regarded as the method of choice for regular estimation problems, the `mle` command is already quite powerful as is.

In practice, a gretl script for estimating a local level model would have at its core a function more or less like the following (with comments interspersed):

```
function matrix local_level (series y, series *u[null], series *v[null])
```

This is the standard way of defining a function in gretl: in this case, the function `local_level` returns a matrix holding the estimated variances and takes three arguments. The first one, `y`, contains a data series. The other two should contain pointers[8] to data series (as indicated by the `*` modifier) and are used to return the one-step-ahead prediction error v_t and its variance F_t, respectively. The `null` keyword indicates that they may be omitted, in which case the two series are discarded.

```
scalar ss1 ss2 scale
sy = init_y(y, &ss1, &ss2, &scale)
```

Here, the user-defined function `init_y` (not shown here, but available in the accompanying files) is used to compute a preliminary estimate of the variances and to rescale `y` so to facilitate numerical maximization of the log-likelihood.[9]

```
kalman
   obsy sy
   obsymat 1
   obsvar ss1
   statemat 1
   statevar ss2
end kalman --diffuse
```

Here the state space representation is set up. The `--diffuse` option is used to indicate that a diffuse-prior algorithm will be used when filtering, setting $P_1 = 10^7 \cdot I$. The gretl manual states:

> [Diffuse i]nitialization of the Kalman filter [...] has been the subject of much discussion in the literature—see for example de Jong (1991); Koopman (1997). At present gretl does not implement any of the more elaborate proposals that have been made.

[8]The usage of pointers in gretl is a much simplified application of the concept of pointers in the C programming language. gretl's concept of pointers is quite similar to the one found in the Ox programming language (see Doornik 2007). Usage of pointers is the standard way in gretl to have a function return more than a single object.

[9]The algorithm used for initializing the two variances is based on the reduced form of the local level model:

$$\Delta y_t = \xi_{t-1} + \Delta\varepsilon_t,$$

so that Δy_t is a MA(1) process; by a standard method-of-moments reasoning, consistent estimators of the two variances can be obtained from γ_0 and γ_1, the sample autocovariances of Δy_t, as follows:

$$\tilde{\sigma}^2_\varepsilon = -\gamma_1 \quad \text{and} \quad \tilde{\sigma}^2_\xi = \gamma_0 - 2\gamma_1.$$

These two estimators can then be used as initial values.

However, the P_1 matrix can be set by the user via the `inivar` keyword, so more sophisticated alternatives are possible if necessary.

Note that the scalars `ss1` and `ss2` are specified as σ_ε^2 and σ_ξ^2, respectively, via the `obsvar` and `statevar`. Assigning new values to these scalars would (and, in fact, will) modify the contents of the relevant system matrices in the state space representation.

```
matrix theta = {ln(ss1),ln(ss2)}
mle loglik = ERR ? NA : misszero($kalman_llt)
    ss1 = exp(theta[1])
    ss2 = exp(theta[2])
    ERR = kfilter()
    params theta
end mle --hessian --quiet
```

Here ML estimation of the parameters is performed. Since gretl does not provide a constrained optimization routine, the actual parameters that are used for numerical maximization of the log-likelihood are the two natural logarithms of the variances, which in this example are contained in the vector `theta`.[10] Points to note:

- the construct `loglik = ERR ? NA : misszero($kalman_llt)` must be interpreted as follows[11]: the `kfilter()` returns a scalar (0 for no error, nonzero otherwise) and fills a series called `$kalman_llt` which contains the per-observation likelihood contributions. So, in case of error, the loglikelihood is set to missing, which would force `mle` to abort; otherwise, missing values in the loglikelihood contributions (because of missing values in y_t) are set to 0.
- The scalars `ss1` and `ss2` get filled with $\exp(\theta_1)$ and $\exp(\theta_2)$. The optimization is performed in terms of the vector `theta`, as indicated by the `params` keyword.
- The `--hessian` option instructs `mle` to compute the variance-covariance matrix of $\hat{\theta}$ from the numerical Hessian.

```
matrix variances = exp(theta)'
matrix ret = scale * variances
series ll = $kalman_llt
T = sum(ok(ll))

matrix V = $vcv .* (variances*variances')
print_results(variances~V, scale, T, $lnl)
```

[10] In fact, estimation may be based on the so-called "concentrated likelihood" (see Durbin and Koopman 2001, p. 31) to make estimation numerically more efficient; however, given the aim of this paper, I chose not to complicate the code.

[11] Perhaps a few readers may be unfamiliar with the syntax `y = A ? x0 : x1`. This construct is used in several programming languages and sets `y` to `x0` if `A` is nonzero and to `x1` otherwise.

Here, the results obtained with ML estimation are re-transformed to the scale of the original y_t variable and stored into the `ret` matrix, to make them available to the caller function; the variance-covariance matrix of the estimated parameters is computed via the delta method: The Jacobian is

$$J = \frac{\partial}{\partial \theta} \begin{bmatrix} \sigma_\varepsilon^2 \\ \sigma_\xi^2 \end{bmatrix} = \begin{bmatrix} \exp(\theta_1) & 0 \\ 0 & \exp(\theta_2) \end{bmatrix} = \begin{bmatrix} \sigma_\varepsilon^2 & 0 \\ 0 & \sigma_\xi^2 \end{bmatrix},$$

so

$$V \begin{pmatrix} \hat{\sigma}_\varepsilon^2 \\ \hat{\sigma}_\xi^2 \end{pmatrix} = JV(\hat{\theta})J^\top = V(\hat{\theta}) \odot \left(\begin{bmatrix} \sigma_\varepsilon^2 \\ \sigma_\xi^2 \end{bmatrix} \begin{bmatrix} \sigma_\varepsilon^2 & \sigma_\xi^2 \end{bmatrix} \right)$$

where $\odot$ indicates the Hadamard product (the corresponding gretl operator is `.*`) and the scalar `T`, the actual sample size, is computed as the number of non-missing observations via gretl's internal function `ok()`. The `$vcv` matrix is automatically generated after successful completion of the `mle` command. Then, the user-written function `print_results` formats the estimation output and prints it out.

```
    matrix f_err f_var
    kfilter(&f_err, &f_var)

    series u = f_err * sqrt(scale)
    series v = f_var * scale

    return ret
end function
```

Finally, the forward pass filter is run once more to retrieve the one-step-ahead prediction error v_t and its variance F_t. These are rescaled, and stored into the two series `u` and `v`. As a last step, the estimates are returned.

2.2. The main script

Data input

The "main" script would look something like

```
open nile.gdt
series uhat Ft
matrix Vars = local_level(nile, &uhat, &Ft)
```

where first the dataset is opened, next the series for the prediction error and its variance are declared, and then parameter estimation is performed.

Estimation

After reading the data, the function `local_level` defined in Section 2.1 is called and the estimation performed, yielding the results shown in Table 2. The estimated parameters are

```
                coefficient   std. error   z-stat   p-value
-------------------------------------------------------------
sigma^2_eps      15098.5       2859.67     5.280    1.29e-07 ***
sigma^2_eta      1469.19       1238.34     1.186    0.2355

sample size = 100
log-lik = -646.013
```

Table 2: Maximum likelihood estimates.

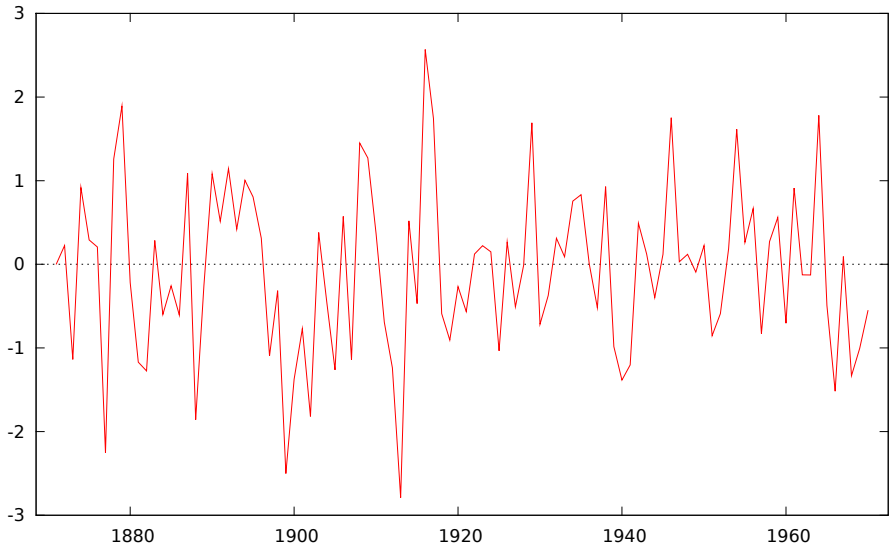


Figure 1: Standardized prediction errors.

saved into the `Vars` matrix for later use. A plot of the observation equation standardized prediction errors (that is, $v_t/\sqrt{F_t}$) is visible in Figure 1.

Smoothing

The next step is the computation of the smoothed state vector: the relevant line in the main input file is

```
sm_nile = loclev_sm(nile, Vars[1], Vars[2], &smv)
```

in which the task is delegated to the user-written function `loclev_sm`, which is also provided in the accompanying files, but is worth showing here. The function first sets up a new `kalman` block with the estimated variances (in the first seven lines of code), and then invokes the `ksmooth` function to obtain and return the estimate of the smoothed states μ_t (and optionally their variances):

```
function series loclev_sm (series y, scalar s1, scalar s2, series *v[null])
```

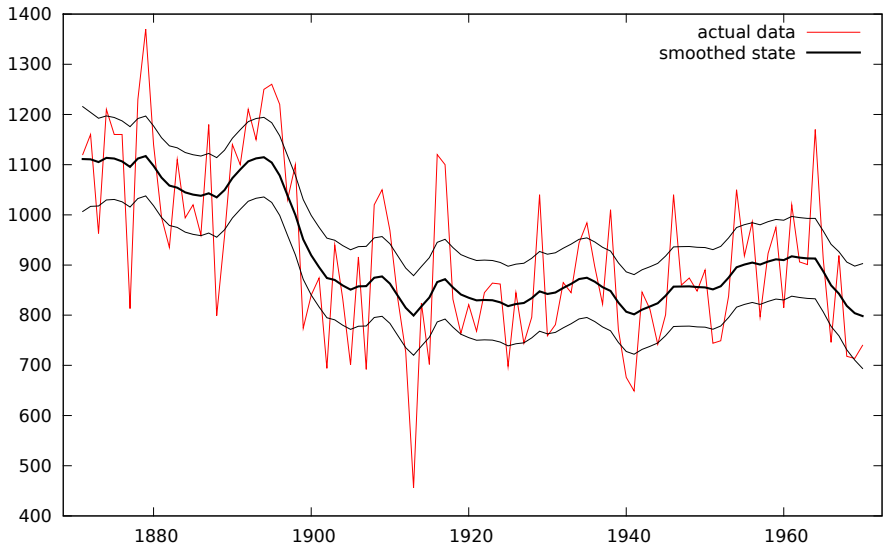


Figure 2: Original data and smoothed state with 90% confidence band.

```
 kalman
    obsy y
    obsymat 1
    statemat 1
    statevar s2
    obsvar s1
end kalman --diffuse

if isnull(v)
    series ret = ksmooth()
else
    matrix V
    series ret = ksmooth(&V)
    series v = V
endif

return ret
end function
```

The `ksmooth()` function returns the smoothed states. If given a matrix-pointer to an argument, as shown, this is filled with the variance for the smoothed state. Figure 2 shows the original data and the estimated μ_t series, together with a 90% confidence band.

Auxiliary residuals

At present, gretl lacks a native algorithm for performing disturbance smoothing, so an automatic mechanism for retrieving generalized residuals and their variances is not provided yet. However, they can be computed quite easily since all the necessary building blocks are

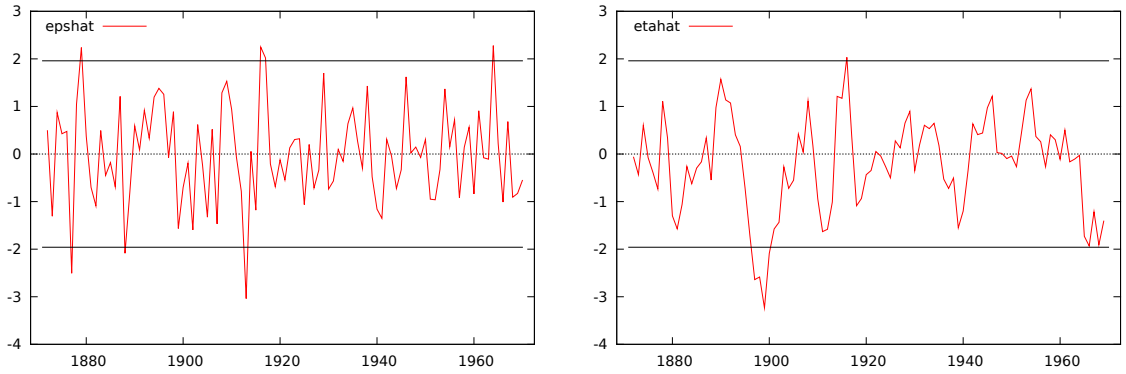


Figure 3: Auxiliary residuals with 95% confidence bands.

available. The relevant formulae are

$$\hat{\varepsilon}_t = y_t - \hat{\mu}_t \tag{5}$$

$$\hat{\xi}_t = \hat{\mu}_{t+1} - \hat{\mu}_t \tag{6}$$

For the variance of $\hat{\xi}_t$, it is necessary to compute the quantity N_t as defined in (Durbin and Koopman 2001, Equation 2.29), namely as a function of the variance of the prediction errors F_t and the Kalman gain K_t. These quantities can be obtained by running the `kfilter()` with pointer arguments. Again, this is not shown here for space reasons, but an example is available in the accompanying files. The code used for the computation and plotting of the auxiliary residuals is:

```
epshat = (nile - sm_nile)
etahat = diff(sm_nile)

nt = Nt(Ft, kgain)
Dt = 1/Ft + nt * kgain^2
Dt[1] = NA

seps = Vars[1] * sqrt(Dt)
auxres1 = (nile - sm_nile) / seps

seta = Vars[2] * sqrt(nt)
auxres2 = etahat / seta
series up =  1.96
series lo = -1.96
plot3a <- gnuplot auxres1 up lo time --with-lines --single-yaxis
plot3b <- gnuplot auxres2 up lo time --with-lines --single-yaxis
```

The thus obtained plots of the auxiliary residuals are shown in Figure 3.

Forecasts

For the local level model, forecasts can be obtained quite simply by appending missing observations at the end of the sample and re-running the smoother. Since gretl automatically

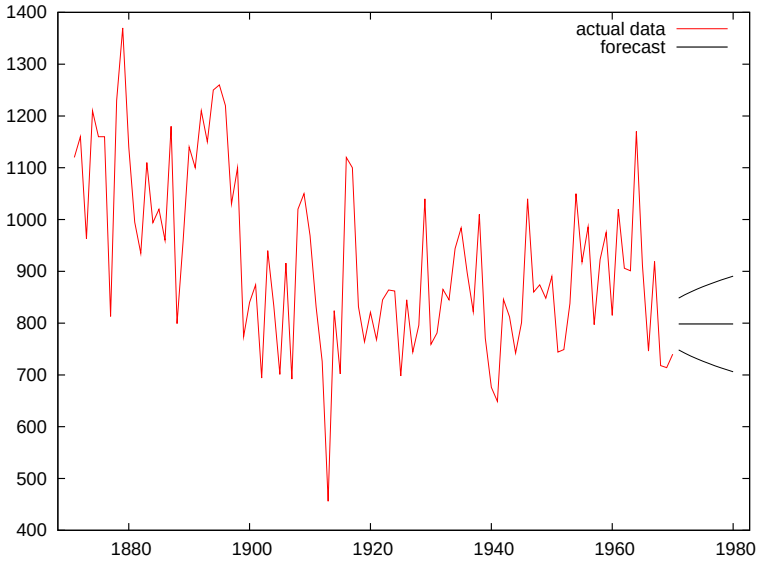


Figure 4: Forecast with 50% confidence band.

takes missing data into account, the necessary code is:

```
dataset addobs 10

sm_nile = loclev_sm(nile, Vars[1], Vars[2], &smv)
series fore  = NA
series fctop = NA
series fcbot = NA
smpl 1971 ;
    confint(sm_nile, smv, 0.50, &fctop, &fcbot)
    series fore = sm_nile
smpl full
```

Here the user function `loclev_sm` (described above) was used again to extract the smoothed states from the `kalman` structure. The `confint` function (provided in the accompanying files as well) is mainly a convenience function to compute the "top" and "bottom" limits of the confidence interval. The forecasts and their 50% confidence band are shown in Figure 4.

3. Case 2: A multivariate unobserved components model

3.1. The model

In this section, we will use a labour market model to exemplify how the setup examined in the previous section extends very naturally to the multivariate case. This model was first used in Lucchetti and Staffolani (1996) and later revisited in Russo and Veredas (2000). It is a bivariate unobserved-components model whose purpose is to provide an empirical counterpart to the "buffer stock" theory on working hours.

The theoretical model on which the empirical model is based is a derivation of the ideas in Bentolila and Bertola (1990) and can be briefly sketched as follows: the total demand for labour can be written as $L_t = N_t H_t$, where N_t is the number of employees and H_t is the number of hours worked. Firms face hiring and firing costs, so when a firm's desired use of labour factor changes, it may be advantageous to spread ΔN_t through time and vary the amount of hours worked while adjustment is taking place. In other terms, if a firm needs to increase its usage of labour, it will increase working hours per worker while new workers are recruited.

Assuming there is a standard weekly working time $\bar{H}$, if no adjustment costs were present, an increment in the demand for labour would lead to a parallel increase in N_t simultaneously. If, however, the adjustment takes place through time, H_t will deviate temporarily from $\bar{H}$ in order to compensate for the extra workers. If we assume that firms optimize their level of labour input intertemporally, then it makes sense to treat the relative variation in the desired level of labour as an unanticipated shock with constant variance $\Delta \ln(L_t) = \eta_t \sim \mathrm{NID}(0, \sigma_\eta^2)$. Upon defining $h_t \equiv \ln(H_t/\bar{H})$ and $n_t \equiv \ln(N_t)$, the relation $\Delta \ln(L_t) = \Delta h_t + \Delta n_t = \eta_t$ holds by definition. If adjustment takes place over time, then, the time path of the hours and of the variation in labour force can be described as:

$$\begin{aligned} \Delta n_t &= B(L)\eta_t, \\ h_t &= A(L)\eta_t, \end{aligned}$$

where $B(L) = 1 - \Delta A(L)$, so the parameters of the two polynomials $A(L)$ and $B(L)$ are linked via $b_i = b_{i-1} - a_i$. A firm which faces an unanticipated labour demand shock η_t will adjust L_t instantaneously by varying h_t while hiring; in the long term, h_t goes back to 0 and n_t is fully adjusted. The polynomial $A(L)$ is assumed to be of some finite order p, so full adjustment takes place in a finite time[12].

In order to match the actual features of observed data series, these two equations must be modified to take three additional facts into account: first, the size of the economy and capital/labour composition of the production technology vary exogenously through time, so Δn_t must contain some additional term to account for long-term trends. Moreover, institutional factors introduce sizeable seasonality in both h_t and Δn_t. Finally, the actual number of hours worked per week could deviate from the firm's plan because of random factors, such as weather, power cuts and so forth.

The first effect will not be modelled explicitly, but can be captured via a local level component; seasonality is modelled via two separate seasonal unobserved components (see Durbin and Koopman 2001, p. 40). As a consequence, the empirical model can be cast as follows:

$$\Delta n_t = \mu_t + B(L)\eta_t + \gamma_t^n, \tag{7}$$

$$h_t = A(L)\eta_t + \gamma_t^h + \varepsilon_t, \tag{8}$$

where μ_t is the local level component, γ_t are the seasonal factors and ε_t is the noise component

[12] Note that neither $A(0)$ nor $B(0)$ are assumed to equal 1; identification is attained via the conditions $B(1) = 1$ and $a_0 = 1 - b_0$.

in the hours equation. In matrix notation,

$$\begin{bmatrix} \Delta n_t \\ h_t \end{bmatrix} = \begin{bmatrix} 1 & \boldsymbol{b}^\top & \mathbf{e}_1^\top & 0 \\ 0 & \boldsymbol{a}^\top & 0 & \mathbf{e}_1^\top \end{bmatrix} \begin{bmatrix} \mu_t \\ \boldsymbol{\eta}_t \\ \boldsymbol{\gamma}_t^n \\ \boldsymbol{\gamma}_t^h \end{bmatrix} + \begin{bmatrix} 0 \\ \varepsilon_t \end{bmatrix}, \tag{9}$$

$$\begin{bmatrix} \mu_{t+1} \\ \boldsymbol{\eta}_{t+1} \\ \boldsymbol{\gamma}_{t+1}^n \\ \boldsymbol{\gamma}_{t+1}^h \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \mathbf{L} & 0 & 0 \\ 0 & 0 & \mathbf{C} & 0 \\ 0 & 0 & 0 & \mathbf{C} \end{bmatrix} \begin{bmatrix} \mu_t \\ \boldsymbol{\eta}_t \\ \boldsymbol{\gamma}_t^n \\ \boldsymbol{\gamma}_t^h \end{bmatrix} + \begin{bmatrix} u_t \\ \eta_t \mathbf{e}_1 \\ \omega_t^n \mathbf{e}_1 \\ \omega_t^h \mathbf{e}_1 \end{bmatrix}, \tag{10}$$

where $\boldsymbol{a}$ and $\boldsymbol{b}$ are vectors containing the parameters of the polynomials $A(L)$ and $B(L)$ in equations (8) and (7), respectively; $\boldsymbol{\eta}_t$ is the vector $[\eta_t \cdots \eta_{t-p}]$; $\mathbf{e}_1$ is a vector containing 1 as its first element and 0 elsewhere (with appropriate size depending on context). The $\mathbf{L}$ and $\mathbf{C}$ matrices are

$$\mathbf{L} = \begin{bmatrix} 0 & 0 \\ I & 0 \end{bmatrix}, \qquad \mathbf{C} = \begin{bmatrix} -\mathbf{u}^\top & -1 \\ I & 0 \end{bmatrix},$$

and $\mathbf{u}$ is a vector of ones.

The five structural disturbance terms refer to the local level component (u_t), the "buffer stock" component (η_t), the seasonal terms (ω_t^n and ω_t^h) and the transitory hours shock (ε_t); they are assumed to be jointly normal and mutually uncorrelated. The unknown parameters are the coefficients of the polynomial $B(L)$ (or, equivalently, $A(L)$) and the five variances of the structural disturbances.

3.2. Implementation and results

To exemplify the actual implementation of the above model, I used Australian data from February 1978 to June 2009. To be specific, the raw data were taken from the Australian Bureau of Statistics database, cat no. 6291.0.55.001, Table 09. For the employment series and hours series, I used the monthly seasonally unadjusted data series labelled A84378T (average weekly actual hours worked) and A84376L (total employment in thousands).

After compacting the data to a quarterly frequency through averaging, logs were taken. To take into account institutional factors that have driven $\bar{H}$ down through time, the log hours series was then regressed against a constant and a linear trend and the residuals were used as h_t.[13] The basic infrastructure for estimating the parameters in equations (9) and (10) is given by the following function[14]:

```
function matrix Giuditta_estimate(series dn, series h, scalar order)
    bhat = ones(order,1)/order
    vars = ones(5,1)
    theta = bhat | vars
```

[13]This is equivalent to assuming that the time path of $\ln(\bar{H})$ can be adequately described by a linear time trend; alternative choices were also possible, including the joint estimation of the constant and trend parameters together with the rest of the model, or the incorporation of $\ln(\bar{H})$ into the model as an additional element of the state vector, but in my opinion they would have complicated the example without yielding a real benefit.

[14]At the time of its conception, this model was nicknamed the "Giuditta" model by its creators, with a reference to the film "The Little Devil", by Roberto Benigni. As often happens, the name has stuck.

```
    matrix H R F Q P
    nstates = make_matrices(theta, $pd, &H, &R, &F, &Q, &P)
    matrix Y = { dn h }

    kalman
        obsy Y
        obsymat H
        obsvar R
        statemat F
        statevar Q
        inivar P
    end kalman

    series LL
    mle LL = errcode ? NA : $kalman_llt
        nstates = make_matrices(theta, $pd, &H, &R, &F, &Q, &P)
        errcode = kfilter()
        params theta
    end mle --hessian

    theta[order+1:] = abs(theta[order+1:])
    print_results(theta, $vcv, $nobs, $kalman_lnl)
    return theta
end function
```

Comments:

- Initialization in the first three lines of code is kept intentionally naïve: the coefficients of the $B()$ polynomial are set to 0 and all variances to 1.
- The `make_matrices` function (not described here, but provided in the accompanying files) in the next three lines of code simply fills the system matrices with the elements of the vector θ. In this case, we initialize P_1 to $1000 \cdot I$, as diffuse initialization causes numerical problems.
- Estimation is carried out by using the `kalman` block in the next eight lines of code and an `mle` block in the following six lines of code in a manner very similar to the Nile example in Section 2.
- The function `print_results` in the last three lines of code, apart from printing out the estimates, computes the variance matrix of the estimated parameters.
- Estimates for the Australian data are shown in Table 3. The pattern of the buffer stock adjustment appears to be quite reasonable: 53.3% of the adjustment seems to occur in the same quarter as the shock (parameter `b_0`) and a year after the shock the effect on working hours is small (7.3%, parameter `a_4`), but still significant.

```
            coefficient   std. error   z-stat   p-value
  ------------------------------------------------------
  a_0         0.466131     0.0547856     8.508   1.77e-17 ***
  a_1         0.187928     0.0426714     4.404   1.06e-05 ***
  a_2         0.132266     0.0268418     4.928   8.32e-07 ***
  a_3         0.113411     0.0314670     3.604   0.0003   ***
  a_4         0.0731748    0.0286753     2.552   0.0107   **
  s_eps       1.59895      0.124880     12.80    1.56e-37 ***
  s_u         0.115993     0.0616542     1.881   0.0599   *
  s_eta       0.965748     0.125454      7.698   1.38e-14 ***
  s_gdn       0.0281485    0.0258615     1.088   0.2764
  s_gh        0.107907     0.0591376     1.825   0.0681   *


            coefficient   std. error   z-stat   p-value
  ------------------------------------------------------
  b_0         0.533869     0.0547856    9.745    1.94e-22 ***
  b_1         0.345942     0.0719262    4.810    1.51e-06 ***
  b_2         0.213676     0.0666455    3.206    0.0013   ***
  b_3         0.100265     0.0431129    2.326    0.0200   **
  b_4         0.0270898    0.0171383    1.581    0.1140

T = 124
log-likelihood = -368.025
```

Table 3: Giuditta model estimates.

The main file contains a few obvious commands for reading the data, performing basic transformations and other trivial tasks. Then, there is the following code block:

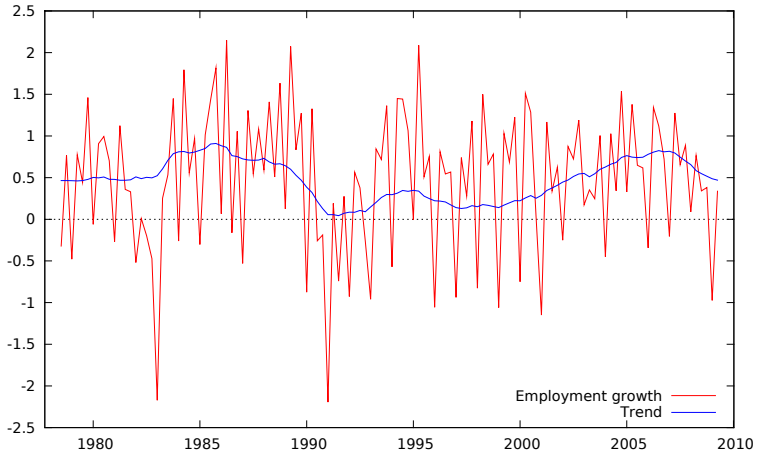
```
theta = Giuditta_estimate(dn, h, 5)

series eta mu sdn sh bufdn bufh
S = Giuditta_sm(theta, dn, h, &mu, &eta, &sdn, &sh, &bufdn, &bufh)
```

where the function `Giuditta_sm` is structurally very similar to `Giuditta_estimate`: the elements of the vector θ are used to fill up the system matrices and, after having set the model up, instead of using `mle` to perform estimation, `ksmooth` is run to return the smoothed estimates of the quantities of interest[15] (shown in Figure 5).

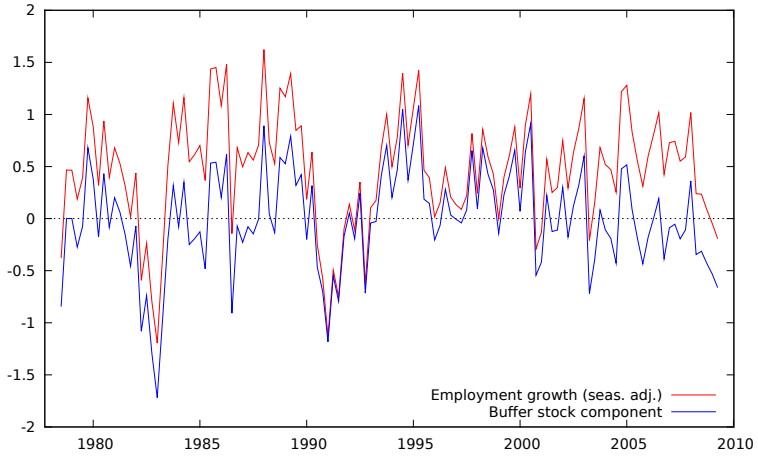
3.3. Diagnostics

Diagnostic tests based on standardized prediction errors can be implemented in a similar vein

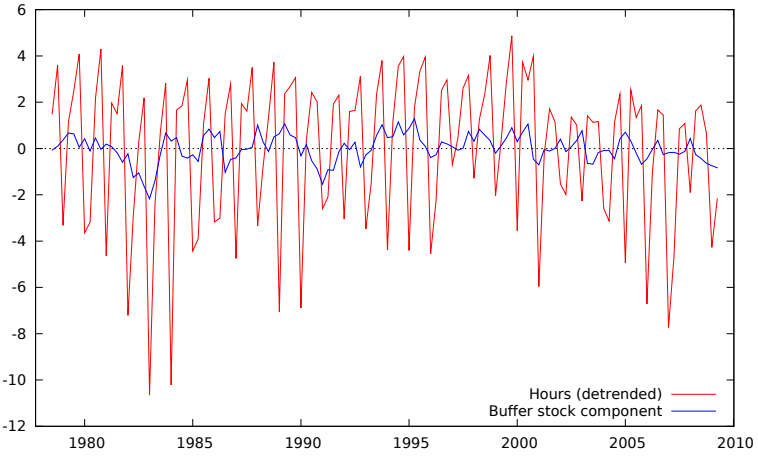
[15]Of course, it is possible to enrich the example by computing confidence intervals for these series, but this has been skipped for the sake of conciseness.



Δn_t and the smoothed estimate of μ_t



Seasonally adjusted series for Δn_t and the smoothed estimate of its buffer stock component.



Seasonally unadjusted series for hours worked and smoothed estimate of its buffer stock component.

Figure 5: Giuditta model: Smoothed estimates.

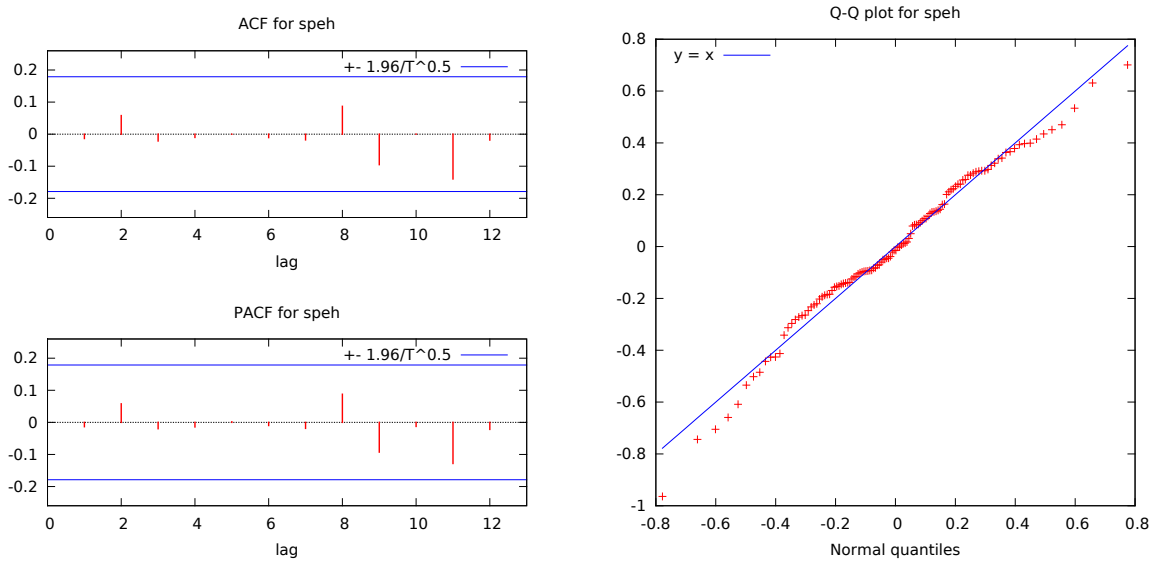


Figure 6: Giuditta model: Correlogram and QQ plot for standardized prediction errors.

as in Section 2.2.4. Here we concentrate on the disturbances to (8), the equation for hours worked (the equation for Δn_t does not have a disturbance term).

The computations are delegated to a function called `Giuditta_prederr` (provided in the accompanying files), which produces the standardized prediction errors. These are, in turn, analysed[16] via internal gretl commands, as follows:

```
series speh = Giuditta_prederr(theta, V)

smpl +4 ;
qqplot speh
corrgm speh 12
normtest speh --all
```

and the output reads:

```
Test for normality of speh:

 Doornik-Hansen test = 3.72497, with p-value 0.155286
 Shapiro-Wilk W = 0.983974, with p-value 0.165903
 Lilliefors test = 0.0570351, with p-value ~= 0.43
 Jarque-Bera test = 4.23413, with p-value 0.120384
```

The correlogram and QQ-plot are shown in Figure 6. As can be seen, the correlogram does not show significant persistence of the residuals; non-normality seems not to be a problem either.

[16]The first four observations are discarded.

4. Conclusions

In addition to the examples reported here, gretl offers several other facilities to deal with state space models: for example, it is possible to specify a model with time-varying system matrices with little more effort than shown in the examples above, so techniques like the univariate treatment of multivariate time series (see Durbin and Koopman 2001, Chapter 6) is possible.

Moreover, it is also possible to set up a state space model and use it to simulate data, rather than filter existing time series. This may be very useful in those situations when inference via simulation is required.

Of course, gretl's implementation is still rather young, and there are several areas which offer room for improvement:

1. The syntax may benefit from some revision: some keywords may not be necessary for simulation; moreover, it would be useful if auxiliary residuals could be obtained automatically.

2. Native methods for disturbance smoothing or simulation smoothing are not provided yet. Since gretl provides access to all the system matrices and the state vectors, those methods can be implemented via user-level functions. However, the gretl development team is considering the possibility of adding native functions in the interest of computational speed in the future.

3. Structural time series models are known to have likelihoods with potential multiple maxima. An enlargement of the algorithm menu offered by the `mle` command could prove extremely useful in those cases.

However, the current implementation of state space models in gretl is already capable of handling the large majority of situations currently faced by practitioners. Most importantly, the source is *open to inspection*, as scientific software should be (some say *all* software). This ensures, at a minimum, that possible bugs or other shortcomings are, at least in principle, spotted and eradicated quickly. Ideally, the improvement in quality of gretl's state space modelling apparatus may become an effort undertaken by the scientific community at large, with all the obvious benefits that this would entail.

Acknowledgments

First, I would like to thank Allin Cottrell: apart from making the whole gretl project possible, he contributed many useful comments to an early draft. Thanks are also due to Ignacio Díaz-Emparanza, Giulio Palomba, Sven Schreiber and an anonymous referee for their valuable remarks. Jacques Commandeur and Marius Ooms also provided me with helpful observations. Of course, no one is to blame for errors but myself. Research assistance by Barbara Ermini is gratefully acknowledged.

References

Bentolila S, Bertola G (1990). "Firing Costs and Labour Demand: How Bad Is Eurosclerosis?" *Review of Economic Studies*, **57**(3), 381–402.

Commandeur JJF, Koopman SJ, Ooms M (2011). "Statistical Software for State Space Methods." *Journal of Statistical Software*, **41**(1), 1–18. URL http://www.jstatsoft.org/v41/i01/.

Cottrell A (2009). *gretl: Retrospect, Design and Prospect*, chapter 1, pp. 3–13. EHUCHAPS. Universidad del País Vasco - Facultad de Ciencias Económicas y Empresariales.

Cottrell A, Lucchetti R (2009). *The gretl User Manual*. Version 1.9.1, URL http://sourceforge.net/projects/gretl/files/manual/gretl-guide-a4.pdf/download.

Cottrell A, Lucchetti R (2011). *gretl User's Guide – Gnu Regression, Econometrics and Time-Series Library*. Version 1.9.4, URL http://gretl.sourceforge.net/.

de Jong P (1991). "The Diffuse Kalman Filter." *The Annals of Statistics*, **19**, 1073–83.

Deek FP, McHugh JAM (2008). *Open Source*. Cambridge University Press, Cambridge.

Doornik J (2007). *Object-Oriented Matrix Programming Using Ox*. 3rd edition. Timberlake Consultants Press, London.

Durbin J, Koopman SJ (2001). *Time Series Analysis by State Space Methods*. Number 24 in Oxford Statistical Science Series. Oxford University Press, Oxford.

Frigo M, Johnson SG (2005). "The Design and Implementation of **FFTW**3." *Proceedings of the IEEE*, **93**(2), 216–231.

Gourieroux C, Monfort A, Trognon A (1984). "Pseudo Maximum Likelihood Methods: Applications to Poisson Models." *Econometrica*, **52**(3), 701–20.

Harvey AC, Proietti T (eds.) (2005). *Readings in Unobserved Components Models*. Oxford University Press, Oxford.

Koopman SJ (1997). "Exact Initial Kalman Filtering and Smoothing for Nonstationary Time Series Models." *Journal of the American Statistical Association*, **92**, 1630–38.

Liu DC, Nocedal J (1989). "On the Limited Memory BFGS Method for Large Scale Optimization." *Mathematical Programming*, **45**(3), 503–528.

Lucchetti R (2009). *Who Uses gretl? An Analysis of the SourceForge Download Data*, chapter 3, pp. 45–55. EHUCHAPS. Universidad del País Vasco - Facultad de Ciencias Económicas y Empresariales.

Lucchetti R, Staffolani S (1996). "Domanda di Lavoro, Orari e Occupazione Nella Grande Industria Italiana." *Politica Economica*, **1**.

Marsaglia G, Tsang WW (2000). "The Ziggurat Method for Generating Random Variables." *Journal of Statistical Software*, **5**(8), 1–7. URL http://www.jstatsoft.org/v05/i08/.

R Development Core Team (2011). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.R-project.org/.

Russo G, Veredas D (2000). "Institutional Rigidities and Employment Rigidity in the Italian Large Industrial Firms." *CORE Discussion Papers 2000048*, Université Catholique de Louvain, Center for Operations Research and Econometrics (CORE). URL http://ideas.repec.org/p/cor/louvco/2000048.html.

Smith RJ, Mixon JW (2006). "Teaching Undergraduate Econometrics with gretl." *Journal of Applied Econometrics*, **21**(7), 1103–1107.

The **Gnumeric** Team (2010). *The **Gnumeric** Manual, Version 1.10*. GNOME Documentation Project. URL http://projects.gnome.org/gnumeric/.

Yalta AT, Lucchetti R (2008). "The GNU/Linux Platform and Freedom Respecting Software for Economists." *Journal of Applied Econometrics*, **23**(2), 279–286.

Affiliation:

Riccardo (Jack) Lucchetti
Università Politecnica delle Marche
Piazzale Martelli, 8
I-60121 Ancona, Italy
E-mail: r.lucchetti@univpm.it

Journal of Statistical Software http://www.jstatsoft.org/
published by the American Statistical Association http://www.amstat.org/
Volume 41, Issue 11 *Submitted:* 2010-01-13
May 2011 *Accepted:* 2010-12-01