EvalEst Guide

In *R*, the functions in this package are made available with

```
> library("EvalEst")
```

The code from the vignette that generates this guide can be loaded into an editor with *edit(vignette("EvalEst"))*. This uses the default editor, which can be changed using *options()*. Also, it should be possible to view the pdf version of the guide for this package with *print(vignette("EvalEst"))* and the guide for the dse package with *print(vignette("dse-guide"))*.

1 Evaluating Estimation Methods

One way to test estimation techniques is to specify a "true" model which is used to produce simulated data and then examine how well an estimation technique finds the true model. This is not as general as theoretical results, since it is really only valid at the "true" parameter values and for the sample size tested, however, it can be illustrative and theoretical results for small samples are very difficult to obtain. It also provides a very good cross check of the simulation and estimation code. Also, equivalent representations may have effects which are not yet fully appreciated in the literature. The following models from Gilbert (1995) [1] will be used to illustrate.

```
> mod1 <- ARMA(A=array(c(1,-.25,-.05), c(3,1,1)),
               B=array(1,c(1,1,1)))
> mod2 <- ARMA(A=array(c(1,-.8, -.2 ), c(3,1,1)),
               B=array(1,c(1,1,1)))
> mod3 <- ARMA(
   A=array(c(
    1.00,-0.06,0.15,-0.03,0.00,0.02,0.03,-0.02,0.00,-0.02,-0.03,
    -0.02,0.00,-0.07,-0.05,0.12,1.00,0.20,-0.03,-0.11,0.00,-0.07,
    -0.03,0.08,0.00,-0.40,-0.05,-0.66,0.00,0.00,0.17,-0.18,1.00,
    -0.11,-0.24,-0.09), c(4,3,3)),
   B=array(diag(1,3), c(1,3,3)))
```

mod2 has a unit root, as can be verified with *roots(mod2)* or *stability(mod2)*.

The function *MonteCarloSimulations* runs simulate repeatedly to give many data samples.

```
> z <-  MonteCarloSimulations(mod1,
                              simulation.args=list(sampleT=100))
> tfplot(z)
> distribution(z)
```

[1] P.D. Gilbert, 1993. "State Space and ARMA Models: An Overview of the Equivalence", Bank of Canada working paper 93–4. Also available at www.bank-banque-canada.ca/pgilbert

Usually it is not necessary to use *MonteCarloSimulations* and actually save all the simulations since the seed and other information about the random number generator (RNG) can be used to reproduce the samples. Thus functions for testing estimation methods can produce the same samples when they are needed.

The function *EstEval* simulates and then estimates models:

```
> e.ls.mod1 <- EstEval( mod1, replications=100,
    simulation.args=list(sampleT=100, sd=1),
    estimation="estVARXls",
    estimation.args=list(max.lag=2),
    criterion="TSmodel"
  #  rng=list(kind="default", normal.kind="default",
  #  seed=c(13,44,1,25,56,0,6,33,22,13,13,0))# Splus - see below

    )
```

In this example simulation and estimation will be repeated 100 times with samples of size 100 and the standard deviation of the model noise will be set to 1. *simulation.args* are passed to the function *simulate*, which may take different arguments depending on the class of the model. Estimation is done with the function *estVARXls* and *estimation.args* are passed to it. The argument criterion specifies what should be returned from the estimation. In this case the model is returned (An object of class *TSmodel*) but not additional information as is usually returned in the object *TSestModel*. It is also possible to specify *coef* or *roots* to return only that specific information, but that information can be extracted from the *TSmodel* as illustrated below. In general *EstEval* will work with any estimation method which will take the results of simulate applied to the supplied model and returns something that criterion can extract. That is, if *criterion(estimation(simulate(model)))* returns something (with *criterion* and *estimation* replaced by the functions you supply and model replaced by the model you supply), then *EstEval* should work with your functions. This does not mean that plots described below will necessarily work or make sense.

An optional argument *rng* can be specified. If supplied, the RNG and seed will be set. This is useful if an experiment is to be reproduced. Using Splus 3.2 and 3.3 the settings indicated by comments in the examples in this section will reproduce the results in Gilbert (1995). It was possible to generate similar random experiments in *S* and in *R*, but not using the *Splus* default generator. (I have not tested in *Splus* for several years now.) If the argument *rng* is given as

```
> rng=list(kind="Wichmann-Hill", seed=c(979,1479,1542),
   normal.kind="Box-Muller")
```

then the uniform RNG is set to Wichmann-Hill, the normal transformation is set to Box-Muller, and the initial seed is set. With the RNG set in this way both *Splus* and *R* will produce similar results. These settings are reset to their

previous values when the function completes. They can be set so that they do not revert using the function

```
> setRNG(kind="Wichmann-Hill", seed=c(979,1479,1542),
   normal.kind="Box-Muller")
```

The argument seed is optional (and other values can be supplied but they should be consistent with the generator). An initial seed will be generated if it is omitted. Typically the seed should be set only when trying to reproduce previous results.

The following uses mod2 as the true model.

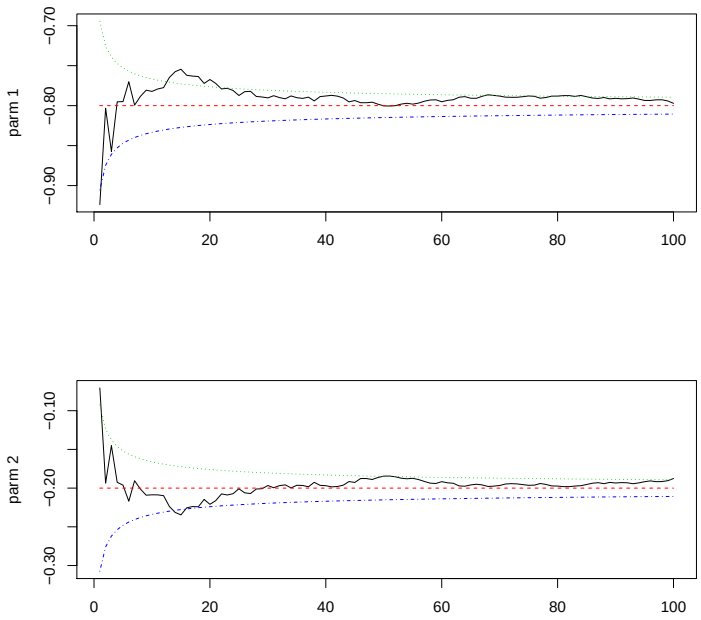
```
> e.ls.mod2 <- EstEval( mod2, replications=100,
                    simulation.args=list(sampleT=100, sd=1),
                    estimation="estVARXls",
                    estimation.args=list(max.lag=2),
                    criterion="TSmodel"
                    #rng=list(kind="default", normal.kind="default",
                    #seed=c(13,43,7,57,62,3,30,29,24,54,47,2))#Splus
   )
```

To plot a line chart of the cumulative average of the estimated parameters use *coef* to extract the parameters (coefficients) from the *TSmodel*:

```
> par(mfcol=c(2,1)) # set the number of plots on the graphics device
> tfplot(coef(e.ls.mod1))
```

The plot from *mod2* looks like this:

```
> tfplot(coef(e.ls.mod2))
```

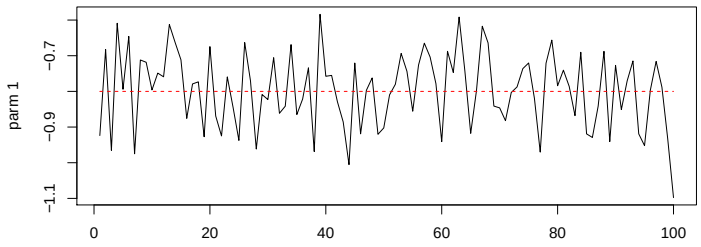


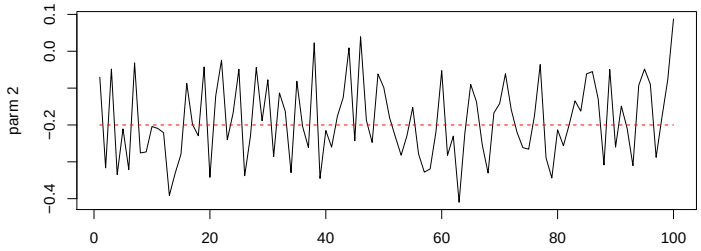
The straight line indicates the true value. To plot a line chart of the estimated parameters use *coef* to extract the parameters from the *TSmodel*:

```
> par(mfcol = c(2,1)) #set number of plots on graphics device
> tfplot(coef(e.ls.mod1), cumulate = FALSE, bounds = FALSE)
```

bounds controls whether or not estimated one standard deviation bounds are plotted. The plot from *mod2* looks like this:

```
> tfplot(coef(e.ls.mod2), cumulate = FALSE, bounds = FALSE)
```



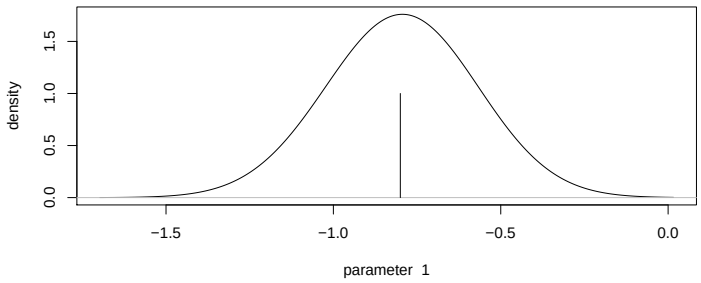


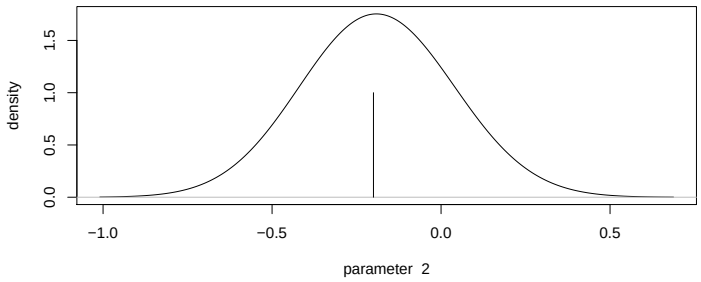
To plot the distribution of estimates:

```
> distribution(coef(e.ls.mod1), bandwidth=.2)
```

The plot from *mod2* looks like this:

```
> distribution(coef(e.ls.mod2), bandwidth=.2)
```



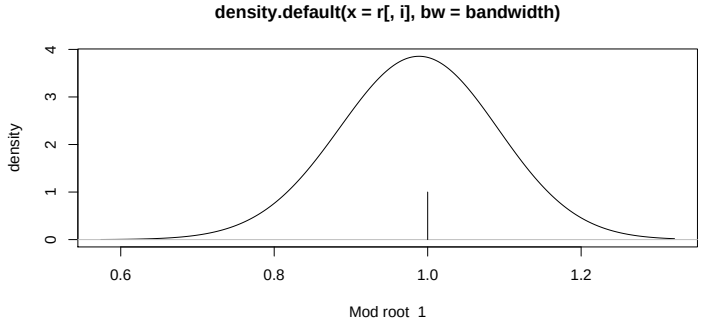


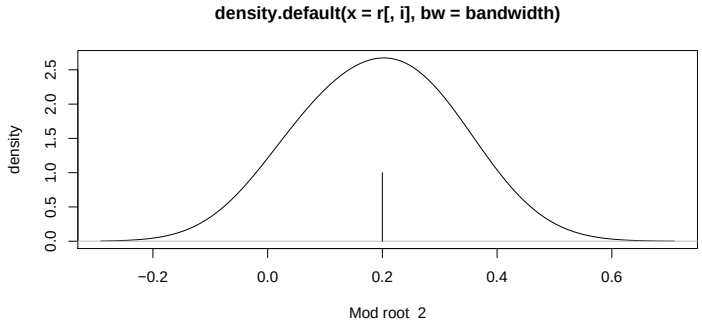
To plot the roots of the estimated model use *roots* to extract the roots from the *TSmodel*:

```
> e.ls.mod1.roots <- roots(e.ls.mod1)
> plot(e.ls.mod1.roots)
> plot(e.ls.mod1.roots, complex.plane=F)
> plot(roots(e.ls.mod2), complex.plane=F)
> distribution(e.ls.mod1.roots, bandwidth=.2)
```

bandwidth is an argument passed to the kernel estimator used to generate the plot. The plot from *mod2* looks like this:

```
> distribution(roots(e.ls.mod2), bandwidth=.1)
```





Some attention to the equivalence of different model representations is necessary when evaluating estimation methods. For example, if the state space equivalent of a VAR model is used as the true model for simulation and *estVARXls* is used for estimation then parameter estimates will be very different from those of the state space model (but root estimates should still be similar). Many estimation techniques may also do some model selection (such as *estBlackBox* does), so the returned models may have different numbers of parameters and/or lags.

Evaluating models based on their forecast performance avoids some of these difficulties. In any case, since forecasting is often the end objective, it is useful to evaluate models directly on their forecasting performance. The function *forecastCovEstimatorsWRTtrue()* evaluates estimation methods using a given true model for simulation. It calculates the covariance of forecast errors of the estimated models relative to the output of the true model:

```
> pc <- forecastCovEstimatorsWRTtrue(mod3,
          estimation.methods = list(estVARXls=list(max.lag=6)),
          est.replications=2, pred.replications=10
         # rng=list(kind="default", normal.kind="default",
         # seed=c( 53,41,26,39,10,1,19,25,56,32,28,3))#Splus
  )
```

The names of the elements in the list *estimation.methods* specify the estimation methods and their value is a list of the arguments to the method. If no

arguments are required then the value should be specified as *NULL*. The covariance for forecasts of zero and a simple trend are also calculated. These are useful benchmarks. *est.replications* controls the number of times a sample is generated and used for estimating a model with each estimation method. *pred.replications* controls how many times the forecasts from the estimated model are compared with output from the true model. Thus the total number of simulations is *est.replications* + *est.replications* * *pred.replications*, so 22 in the above example.

A similar function is available which applies a model reduction procedure after the estimation:

```
> pc.rd <- forecastCovReductionsWRTtrue(mod3,
                estimation.methods = list(estVARXls=list(max.lag=3)),
                est.replications=2, pred.replications=10
                # rng = list(kind = "default", normal.kind="default",
                # seed=c(29,55,47,18,33,1,15,15,34,46,13,2))
  )
```

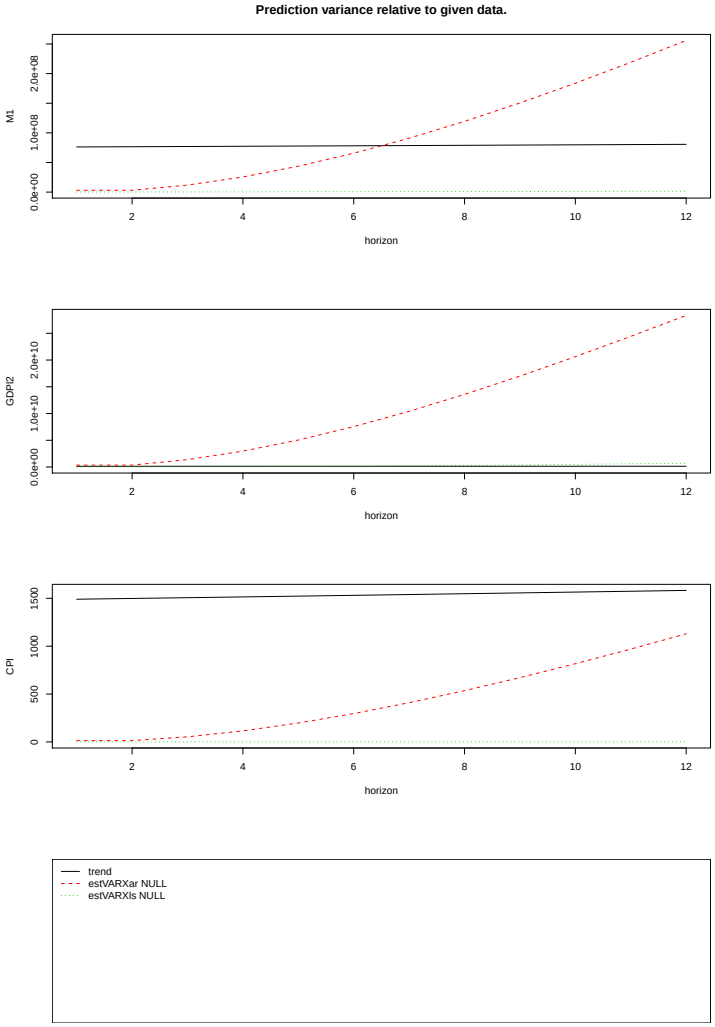
The reduction procedure used is *MittnikReducedModels*. An optional argument *criteria* can be specified. This controls the model selection criteria used by the reduction technique.

It is possible to compare different estimation techniques on the basis of their out-of-sample forecasting error with respect to a data sample. In the following example *estimation.sample* controls the portion of the sample used for estimation. It can be a fraction indicating a portion of the sample, or it can be an integer in which case it will be treated as the number of periods to use for estimation.

```
> data(eg1.DSE.data, package="dse")
> z <-outOfSample.forecastCovEstimatorsWRTdata(trimNA(eg1.DSE.data),
  estimation.sample=.5,
  estimation.methods = list(estVARXar=NULL, estVARXls=NULL),
  trend=T)
```

The plot looks like this:

```
> tfplot(z)
```



In the example below the number of lags is limited (the default is 12 for *estBlackBox4*) and printing of intermediate results is suppressed.

```
> z <-outOfSample.forecastCovEstimatorsWRTdata( trimNA(eg1.DSE.data),
                        estimation.sample=.5,
                        estimation.methods = list(
                        estBlackBox4=list(max.lag=3, verbose=F),
                        estVARXls=list(max.lag=3)),
                        trend=T, zero=T)
> tfplot(z)
```

The object returned by *outOfSample.forecastCovEstimatorsWRTdata()* contains the estimated models so it is possible to extract the models and use *l*, *horizonForecasts* and *featherForecasts*. In the above example the model estimated with *estBlackBox4* is the first model and that estimated with *estVARXls* is the second, so

```
> zz <- horizonForecasts(TSmodel(z, select=1), TSdata(z),
                         horizons=c(1,3,6))
```

would generate an object with the actual forecasts for the model estimated with *estBlackBox4* (rather than the covariance of the forecast errors) and *forecasts(zz)[3,30,]* will then be the prediction made for the 30th period from 6 (the third element of *horizons)* periods previous. The generic function *horizonForecasts()* can also be applied directly to *z* and the appropriate information will be extracted to generate forecasts for all the estimated models.