1 dse1 Guide

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

```
> library("dse1")
```

Several data sets are included with this library and will be used in examples in this guide. In S these are available when the library is attached. In R they are made available by

```
> data(package = "dse1")
> data(eg1.DSE.data, package = "dse1")
> data(egJofF.1dec93.data, package = "dse1")
```

1.1 Defining a TSdata Structure

This section describes how to construct a *TSdata* structure if you have other data you would like to use. Section 10 discusses adding new kinds of *TSdata* classes. Some installations may have an online database and it may be possible to connect directly to this data. See the padi and dsepadi packages as on one possibility for doing this.

For many people the situation will be that the data is in some ASCII file. This can be loaded into session variables with a number of standard S/R functions, the most useful of which are probably *scan()* and *read.table()*. Following is an example which reads data from an ASCII file called "eg1.dat" and puts it in the variable called *eg1.DSE.data* (which is also one of the available data sets). The file has five columns of numbers and 364 rows. The first column just enumerates the rows and is discarded.

This matrix can be used to form a *TSdata* object by

The matrix and the resulting *TSdata* object do not have a good time scale associated with points. A better time scale can be added by

```
> eg1.DSE.data <- tframed(eg1.DSE.data, list(start = c(1961, 3),
      frequency = 12))
```

There are several different possibilities for representing time in S/R objects. The most common is the ts matrix object, which is used in the above default tframed method. (ts is a class in R. In S it is not a class of object, but the default representation of time series which existed before classes were introduced.) The above tframed method and ts can also be used directly on the matrix before the TSdata object is formed. However, [,] in Splus results in the time scale being

lost, so it would need to be reassigned to the input and output matrices of the TSdata object. The methods from the tframe library are used extensively in the DSE library because they provide a common way to proceed in Splus and R, extend to other time representations in addition to ts, and provide a mechanism for extending methods to other objects like TSdata and TSmodels.

Names can be given to the series with

```
> seriesNamesInput(eg1.DSE.data) <- "R90"
> seriesNamesOutput(eg1.DSE.data) <- c("M1", "GDPl2", "CPI")
```

Setting the series names is not necessary but many functions can use the names if they are available. (This overlaps somewhat with S/R dimnames, but is the preferred method in this library as it extends to data which is not a matrix.) The *TSdata* object with elements input and output is the structure which the functions in this library expect. More details on this structure are available in the help for TSdata. The input and output elements can be defined in a number of different ways and new representations can be fairly easily added. For example, when the data is on a remote database as used by TSPADI, the S/R object is just a description of where to get the data, rather than the data itself. In this case the *freeze()* function is used automatically by many functions in the DSE library in order to get a copy of the data when calculations are to be performed.

Once data is available a model can be estimated:

```
> model1 <- est.VARX.ls(eg1.DSE.data)
> model2 <- est.SS.Mittnik(eg1.DSE.data, n = 4)
```

(Note: these models are not the same as those reported in Gilbert,1993. In that paper a variant of *est.VARX.ar* was used.) The scale of the series in *eg1.DSE.data* are very different, with the result that the covariance matrix of the residuals from the estimation is nearly singular. This is detected during the calculation of residual statistics. Statistics are then calculated using only the non-degenerate subspace and a warning message is printed. A better model might be obtained if the data were scaled differently.

Information about the estimated models can be displayed, for example:

```
> summary(model1)
> summary(model2)
> model1
> model2
> stability(model1)
> stability(model2)
> information.tests(model1, model2)
```

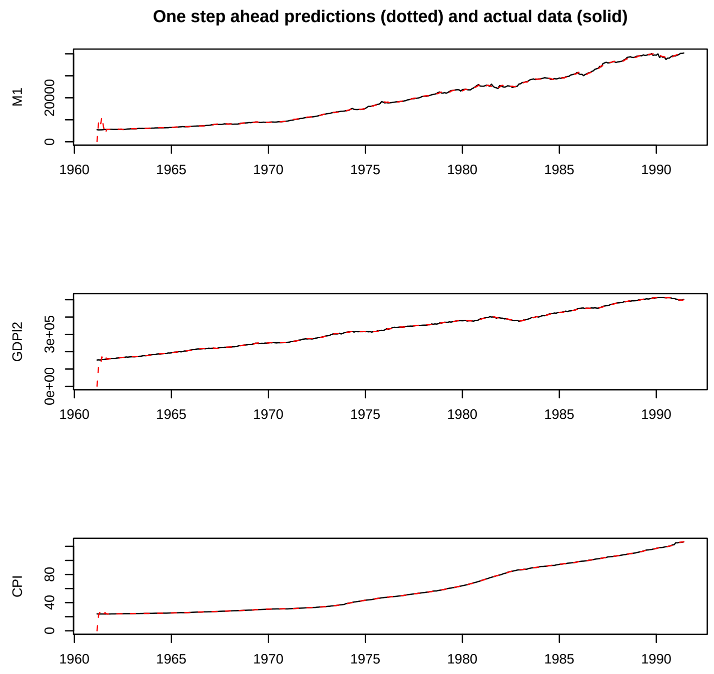
Typing the name of an object in S/R results in the object being printed. To display plots it is first necessary to open a graphics window:

```
> x11()
```

Once a graphics display is active then plots can be viewed:

```
> tfplot(model1)
> tfplot(model2)
> tfplot(eg1.DSE.data)
> check.residuals(model1)
> check.residuals(model2)
```

The function *tfplot* produces separate graphs for each series. The first



***tfplot* command produces this graphic**

Note that initial conditions have been set to zero, but the effect of this dies out quickly. (Also note that the graph labels may be slightly

different depending on which version DSE and of R or S you are using.)

1.2 ARMA and State Space TSmodels

Specifying ARMA and SS models is described below, but first their definition is outlined. The linear time-invariant ARMA representation is

$$A(L)y_t = B(L)e_t + C(L)u_t \tag{1}$$

where y_t is a p dimensional vector of observed output variables, u_t is an m dimensional vector of input variables, e_t is a p dimensional unobserved disturbance vector process and A, B and C are matrices of the appropriate dimension in the lag (back shift) operator L. VAR models can be thought of as a special case of ARMA models with B(L)=I. ARIMA models are also a special case of ARMA models.

Note that the time convention here implies that the input variable u_t can influence the output variable y_t in the same time period. This convention is not always used in time-series models but is important for economics data, especially at annual frequencies.

A linear time-invariant state space representation in innovations form is given by

$$\begin{aligned} z_t &= Fz_{t-1} + Gu_t + Ke_{t-1} \\ y_t &= Hz_t + e_t \end{aligned} \tag{2}$$

where z_t is the unobserved underlying n dimensional state vector, F is the state transition matrix, G, the input matrix, H, the output matrix, and K, the Kalman gain. The library also has some limited capabilities to work with the more general non-innovations form

$$\begin{aligned} z_t &= Fz_{t-1} + Gu_t + Qn_t \\ y_t &= Hz_t + Re_t \end{aligned} \tag{3}$$

where n_t is the system noise, Q, the system noise matrix, and R the output (measurement) noise matrix.

Models are specified by setting up the arrays that define the model and grouping them into a TSmodel object. Here is an example ARMA model with two series, a second order AR polynomial, a first order MA polynomial and no exogenous variable:

```
> AR <- array(c(1, 0.5, 0.3, 0, 0.2, 0.1, 0, 0.2, 0.05, 1, 0.5, 0.3), c(3, 2,
      2))
> MA <- array(c(1, 0.2, 0, 0.1, 0, 0, 1, 0.3), c(2, 2, 2))
> arma <- ARMA(A = AR, B = MA, C = NULL)
> rm(AR, MA)
> arma
> stability(arma)
> data.arma.sim <- simulate(arma)
> arma <- l(arma, data.arma.sim)
> summary(arma)
> tfplot(data.arma.sim)
> tfplot(arma)
```

Note that arrays are filled in the order of their dimensions, which may not be what you expect. The internal representation of TSmodels may be described in the help for the specific model constructors, but in general it should be considered "opaque" and an understanding of the internal data structure should not be necessary to use the models. The function *l()* evaluates the model with the simulated data. Functions generally use default values for some arguments. For example, the length of the simulation and the covariance of the noise can be specified. The above example uses the default values. See the help on simulate for more details. In the example above, arma is initially assigned an object of class TSmodel, but it is then re-assigned the value returned by *l()*, which is an object of class TSestModel. Also, many functions work with different classes of objects, and do different things depending on the class of the argument. The function *tfplot()* works with objects of class *TSdata* and *TSestModel*.

Here is an example of a state space model:

```
> f <- array(c(0.5, 0.3, 0.2, 0.4), c(2, 2))
> h <- array(c(1, 0, 0, 1), c(2, 2))
> k <- array(c(0.5, 0.3, 0.2, 0.4), c(2, 2))
> ss <- SS(F = f, H = h, K = k)
> print(ss)
> stability(ss)
> data.ss.sim <- simulate(ss)
> ss <- l(ss, data.ss.sim)
> summary(ss)
> tfplot(ss)
```

Data which has been generated with simulate is a TSdata object and can be used with estimation routines. This provides a convenient way to generate data for estimation algorithms, but remember that estimation will not necessarily get back to the model you start with,

since there are equivalent representations (see Gilbert, 1993). However, a good estimate will get close to the likelihood and predictions of the original model.

Here is an example of changing between state space and ARMA representations using the models defined in the previous example:

```
> ss.from.arma <- l(to.SS(arma), data.arma.sim)
> arma.from.ss <- l(to.ARMA(ss), data.ss.sim)
> summary(ss.from.arma)
> summary(arma)
> summary(arma.from.ss)
> summary(ss)
> stability(arma)
> stability(ss.from.arma)
```

The function *roots()* is used by *stability()* and can be used by itself to return the roots but not evaluate their magnitude footnoteBy default the roots of an ARMA model are calculated by converting the model to state space form, for reasons explained in Gilbert (2000). By specifying by.poly=T the method can be changed to use an expansion of the polynomial determinant.. When their arguments are *TSmodel*s the functions *to.SS()* and *to.ARMA()* return objects of class *TSmodel* which are not assigned to a variable in the above example, but used in the evaluation of *l()*. The models are returned as part of the *TSestModel* returned by *l()*.

1.3 Model Estimation

The example data *eg1.DSE.data* and *egJofF.1dec93.data* are available with the DSE library and are used in examples in this section.

To estimate an AR model with the default number of lags:

```
> model.eg1.ls <- est.VARX.ls(trim.na(eg1.DSE.data))
```

In this example trim.na removes NA padding from the ends of the data, since the estimation method cannot handle missing values. This padding may not be present, depending on how the data was retrieved. This data is highly correlated and highly parameterized models result in a degenerate covariance matrix. When this happens a warning is produced in this and other examples.

It is also possible to select a subsample of the data:

```
> subsample.data <- tfwindow(eg1.DSE.data, start = c(1972, 1), end = c(1992, 12))
```

This creates a new variable with data starting in January 1972 and ending in December 1992. The S/R function window also usually works, however the function tfwindow is typically used in the DSE library and this guide because it has occasionally been necessary to correct some problems with window. Various functions can be applied to the estimation result

```
> summary(model.eg1.ls)
> print(model.eg1.ls)
> tfplot(model.eg1.ls)
> check.residuals(model.eg1.ls)
```

Other estimation techniques are available

```
> model.eg1.ar <- est.VARX.ar(trim.na(eg1.DSE.data))
> model.eg1.ss <- est.SS.from.VARX(trim.na(eg1.DSE.data))
> model.eg1.bft <- bft(trim.na(eg1.DSE.data))
> model.eg1.mle <- est.max.like(est.VARX.ls(trim.na(eg1.DSE.data), max.lag = 1))
```

tfplot **can put multiple similar objects on a plot**

```
> tfplot(model.eg1.ls, model.eg1.ar)
> tfplot(model.eg1.ls, model.eg1.ar, start. = c(1990, 1))
```

Because of this, the argument ***start.*** **requires a period at the end so it is not confused with an object to be plotted.**

Most of the estimation techniques have several optional parameters which control the estimation. Consult the help for the individual functions. ***est.max.like*** **extracts data from a TSestModel and uses the model structure and initial parameter values for the estimation. (Note: Maximum likelihood estimation can be very slow and may not converge in the default number of iterations. It also tends to over fit unless used with care, so that out-of-sample performance is not good. I do not generally recommend it, although it does offer possibilities for constraining the structure in specific ways (e.g. fixing some model matrix entries to zero or one). You might consider comparing mle to other estimation techniques using functions discussed in the following sections.) In the above** ***est.max.like*** **example a smaller (one lag) model is used. Be prepared for the estimation to take some time when models have a large number of parameters.**

An important point to note is that the one-step-ahead predictions and related statistics returned by these estimation techniques are calculated by evaluating l(model, data) as the final step after the model has been estimated. This can give different results than might

be expected using the estimation residuals, particularly with respect to initial condition effects. (For stable models initial condition effects should not be too important. If they are an important factor check the documentation for specific models regarding the specification of initial conditions.)

Also remember when estimating a model that, if you want to predict future values of a variable, it will need to be an output in the TSdata object.

For the next example a four variable subset of the data in *egJofF.1dec93.data* will be used. This subset is extracted by

```
> eg4.DSE.data <- egJofF.1dec93.data
> output.data(eg4.DSE.data) <- output.data(eg4.DSE.data, series = c(1, 2, 6, 7))
```

which selects the 1st, 2nd, 6th, and 7th series of the output data. The following uses the currently preferred automatic estimation procedure:

```
> model.eg4.bb <- est.black.box(trim.na(eg4.DSE.data), max.lag = 3)
```

An optional argument *verbose=F* will make the function print much less detail about the steps of the procedure. The optional argument, *max.lag=3*, specifies the maximum lag which should be considered. The default *max.lag=12* may take a very long time for models with several variables. *est.black.box* currently uses *est.black.box*4, also known as *bft(..., standardize=T)* which is called the brute force technique in Gilbert (1995).

The traditional model information criteria tests can be performed to compare models:

```
> information.tests(model.eg1.ar, model.eg1.ss)
```

An arbitrary number of models can be supplied. The generated table lists several information criteria. For state space models the calculations are done with both the number of parameters (the number of unfixed entries in the model arrays) and the theoretical parameter space dimension. See Gilbert (1993, 1995) for a more extensive discussion of this subject.

Note that converting among representations produces input-output equivalent models, so that predictions, prediction errors, and any statistics calculated from these, will be the same for the models. However, different estimation techniques produce different models with

different predictions. So, *est.VARX.ls(data)* and *to.SS(est.VARX.ls(data))* will produce equivalent models and *est.SS.Mittnik(data)* and *to.ARMA(est.SS.Mittnik(data))* will produce equivalent models, but the first two will not be equivalent to the second two.