The `StrucTiSM` package for *gretl*

Riccardo (Jack) Lucchetti Sven Schreiber

version 0.7

1 Introduction

This package implements Harvey-style structural time series models. Its name stands for **Struc**tural **Ti**me **S**eries **M**odels. The classic obligatory reference is Harvey (1989), but more modern excellent treatments abound, such as Commandeur and Koopman (2007). One we particularly like is Pelagatti (2015). At present, the package is rather limited (single-series only, no cycles) but, we believe, useful for standard tasks.

The basic idea is that an observed time series y_t can be thought of as the sum of some components, also known as state variables, or *states* for short:

$$y_t = \mu_t + s_t + \varepsilon_t \tag{1}$$

where μ_t is a trend component known as the "level", s_t is a seasonal component and ε_t is white noise. Each component could be absent. A particular version of a model depends on the characteristics those components are assumed to have.

The trend is basically a random walk process with a drift: the drift is called "slope" and can be 0, or a constant, or itself a random walk. In formulae:

$$\mu_t = \mu_{t-1} + \beta_{t-1} + \nu_t \tag{2}$$
$$\beta_t = \beta_{t-1} + \zeta_t \tag{3}$$

where ν_t and ζ_t are uncorrelated white noise processes, possibly with zero variance.

The seasonal component can be specified in several ways, which are largely equivalent in practice, but which may make a difference in some cases. The most common ways are to model seasonality either via seasonal dummies (possibly constant, or themselves evolving as random walks), or via trigonometric terms (see Pelagatti (2015), section 3.4 for further details).

In most cases, estimation is performed via numerical maximum likelihood, which is a relatively easy task since these models have a very natural state-space representation so the Kalman filter can be used (for technical details, see the *Gretl User's Guide*; for the statistical underpinnings of the procedure, see Hamilton (1994) or Pollock (1999)). However, the spcial cases when all the components are deterministic are handled separately, because maximum likelihood estimators are available in closed form, and are computed via OLS.

Notable sub-cases are:

- The Local Level model (basically, a random walk plus noise):[1]

$$y_t = \mu_t + \varepsilon_t \qquad \mu_t = \mu_{t-1} + \nu_t$$

- The Local Linear Trend model

$$y_t = \mu_t + \varepsilon_t \qquad \mu_t = \mu_{t-1} + \beta_{t-1} + \nu_t$$

  where β_t is itself a random walk process; therefore, this is an I(2) process plus noise.

- The basic structural model

$$y_t = \mu_t + \gamma_t + \varepsilon_t \qquad \mu_t = \mu_{t-1} + \beta_{t-1} + \nu_t$$

  Same as the LLT model plus some form of seasonality (usually stochastic dummies).

In this context, the object of estimation are the variances of the structural disturbances. This is performed via ML under a joint normality assumption. Once the variances are estimated, applying the Kalman smoother produces unbiased and efficient predictors of the unobserved states, which can be analysed separately.

2 Package basics

The package is organised in a similar way to other gretl packages, such as `SVAR`, `gig`, `DPB` etcetera. The stages of the work are

- First, you set up a model, as a bundle.
- Next, you call a function which performs estimation of the unknown parameters; this adds to your bundle their estimates plus other relevant quantities.
- Finally, you extract from the bundle the information you need.

You can do this via scripting, or via the GUI.

2.1 The scripting interface

The functions for the first basic step is called `STSM_setup()`. It returns an initialised bundle, given the characteristics of the model you want to estimate. Its arguments are:

1. the series you want to analyse;
2. a Boolean entry if you want ε_t or not

[1] An example is provided in Lucchetti (2011) on how to set up a LL model in gretl. Don't read it. It refers to an ancient version of gretl and none of the information is accurate, or even applicable, any longer. The only value it has is historical.

3. the trend specification: 1 means "stochastic", 2 "deterministic" (default = 1)

4. the slope specification: 0 means "none", 1 "stochastic", 2 "deterministic" (default = 1)

5. the seasonal specification: here you have 0 for "none"; 1 and 2 allow for a stochastic specification (with trigonometric terms or dummies, respectively); 3 gives you deterministic dummy seasonals (default = 2)

6. an optional list of further exogenous regressors, see below for details (default = null)

So for example you'd set up a LLT model via

```
bundle MyModel = STSM_setup(y, 1, 1, 1, 2)
```

Estimation is carried out via the `STSM_estimate()` function, whose compulsory parameter is the reference to the bundle you set up previously, as in `STSM_estimate(&MyModel)`. Optionally, you can add two other arguments: a degree of verbosity, to check on the ML estimating progress in case something goes wrong, and an integer called "mapping". The mapping parameter enables you to choose what reparametrisation of the variances is used internally for the ML algorithm. 0 corresponds to "no mapping", 1 to "standard deviations" and 2 to "log variances"; for example, if 2 is chosen, the log-likelihood function is expressed as a function of the *logs* of σ_ν^2, σ_ε^2 etcetera. You may want to use different settings if you experience convergence problems, but in standard cases the default choice (1, ie standard deviations) should work quite well. A fourth, optional parameter controls the technique used for estimating the covariance matrix of the parameters (by default, the inverse Hessian).

In order to fetch information from your bundle after estimation, you just use ordinary bundle syntax, with one exception: you can extract the smoothed estimates of the components as a list[2] by the `STSM_components()` function, as in

```
list COMPS = STSM_components(Model)
```

This function will perform automatically a series of boring tasks, such as giving the series proper names. For example, if the name of the variable you're analysing is `foo`, the estimated level will bear the name `foo_level`. If you want to do this by hand, however, the states are available as individual series in the bundle; the full matrix of smoothed states is also available, under the name of `St`.

The function `STSM_components()` also accepts a second, optional, Boolean parameter (the default is 0). If set to 1, the output list will also include the time-varying standard errors for the estimated states. These are taken from the matrix that, inside the model bundle, is labelled as `stSE`. The name of the estimated standard errors are the same as those for the states, with the extra suffix "`_se`".

If you just want to decompose a series into components by using an "off-the-shelf"model, we provide two convenience functions called `LLT` and `BSM`,

[2] At present, there's no provision for auxiliary residuals. They can be added later.

respectively, which take a series as input and return the states as a list. The optional Boolean parameter for extracting the states' standard errors applies here too. A third, optional parameter can be used to retrieve the model bundle for later usage. That is, the lines

```
bundle model = null
list comps = LLT(y, 0, &model)
```

are equivalent to

```
bundle model = STSM_setup(y, 1, 1, 1, 2)
STSM_estimate(&model)
list comps = STSM_components(&model)
```

2.2 An example

In the following example we use the famous "airline" dataset from Box and Jenkins (provided among gretl's sample datasets as `bjg.gdt`) to estimate a Basic Structural model and plot the smoothed states. The code

```
include StrucTiSM.gfn
open bjg.gdt # The immortal "airline" dataset

# set up the model

scalar irregular = 1 # yes
scalar trend = 1     # stochastic
scalar slope = 2     # deterministic
scalar seasonal = 2  # stochastic dummies
Airline = STSM_setup(lg, irregular, trend, slope, seasonal)

# perform estimation

STSM_estimate(&Airline)

# extract the states and plot them

comps = STSM_components(Airline)
gnuplot lg lg_level --time-series --with-lines \
    --single-yaxis --output=display
scatters comps --time-series --output=display
```

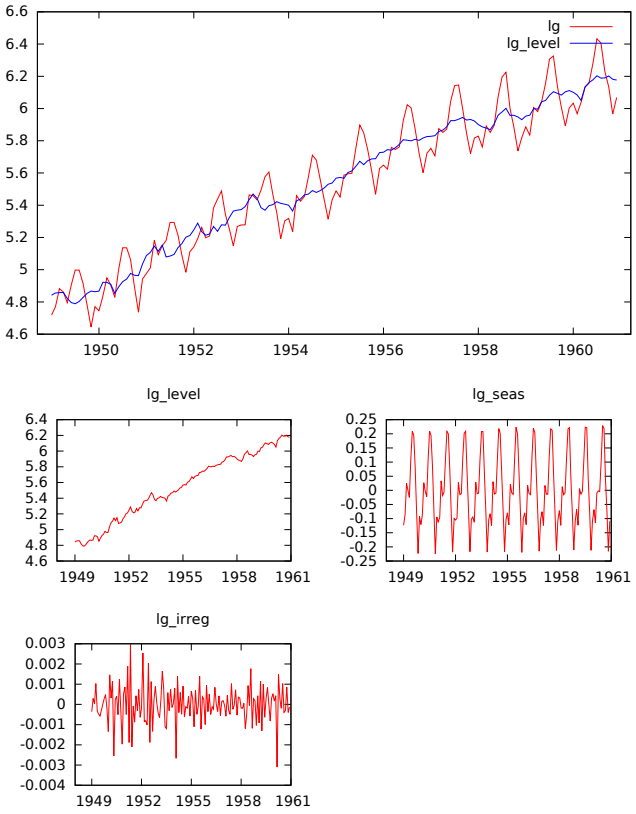
produces the listing below.

Note that, differently from other programs, StrucTiSM reports standard deviations rather than variances in the estimation output.[3]

```
Structural model for lg, 1949:01 - 1960:12 (T = 144)

                    coefficient   std. error     z     p-value
```

[3]It should also be noted that special care must be taken when reading the output, as the p-value reported cannot be interpreted as a proper test for zeroing the corresponding parameter. The reason is that a test for $\sigma = 0$ implies the evaluation of the log-likelihood and its derivatives at a point on the frontier of the parameter space, so usual asymptotics don't apply. See Pelagatti (2015), page 131.

Figure 1: BSM on the airline data



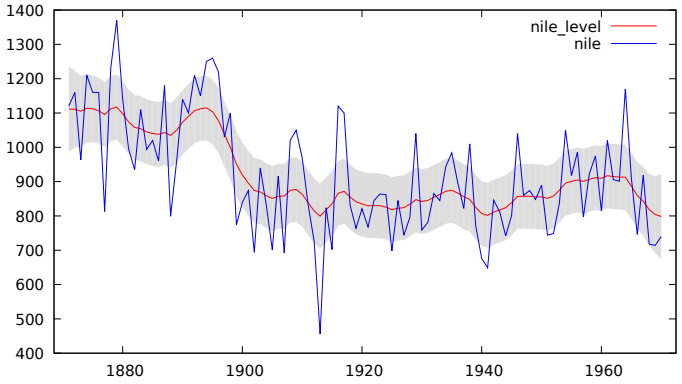

```
  ------------------------------------------------------------
  Irregular          0.0113924     0.00567347    2.008    0.0446    **
  Trend              0.0264475     0.00359817    7.350    1.98e-13  ***
  Seasonal (dums)    0.00800572    0.00273532    2.927    0.0034    ***

Average log-likelihood = 1.27187

Specification:

Stochastic trend, no slope, dummy seasonals (stoch.),
irregular component
```

and the plots are shown in figure 1.

The following example, instead, analyses the Nile data, the canonical example employed in all the articles contained in the special issue of the *Journal of Statistical Software* devoted to state-space modelling (see Commandeur et al. (2011)). It shows how to build a confidence band around a smoothed state and draw it:

```
include StrucTiSM.gfn
open nile.gdt
LL = STSM_setup(nile,1,1,0,0)
```

Figure 2: LL model on the Nile data




```
STSM_estimate(&LL)
X = STSM_components(LL, 1)
list Plot = nile_level nile
plot Plot
    options time-series with-lines
    option band=nile_level,nile_level_se,1.96
    option band-style=fill
end plot --output=display
```

which produces what you see in figure 2.

2.3 Forecasting

For a thorough exposition of the approach we follow, the best reference is probably section 4.11 in Durbin and Koopman (2012). In brief, forecasting on the basis of an already estimated model is performed by running the Kalman filter on that model, in which a suitable number of missing observaions are appended to the dependent variable.

The function that the package provides for this purpose is called `STSM_fcast` and it works by adding two new items to a model bundle: both are vectors with h elements, where h is the desired forecast horizon. They are contained under the names `fcast` and `fcastvar`, and contain the point forecasts and their variances, respectively. (Forecast variances do not take into account parameter uncertainty.)

By default, the function prints out the point forecasts and the corresponding standard errors; this behaviour can be turned off by providing 0 as the third parameter to the function (see section 3).

For example, the following script

```
set verbose off
include StrucTiSM.gfn
open nile.gdt --quiet

# estimate a local level model on the Nile data
model = STSM_setup(nile, 1, 1, 0, 0)
# estimate silently
```

```
scalar err = STSM_estimate(&model, 0)

# forecast
STSM_fcast(&model, 4)
```

should produce this output

```
Out of sample forecast for nile

   horizon    forecast    std.err.

         1     798.368     143.527
         2     798.368     148.557
         3     798.368     153.422
         4     798.368     158.138
```

Slightly more elaborate examples are provided in the `forecasting.inp` script, in the examples directory. If necessary, you can also obtain forecasts for the latent states, by passing a fourth Boolean parameter. Again, see section 3 for more details.

2.4 Interpolation of missing values

One thing state-space models are quite good at is interpolating missing values (provided, of course, that the statistical model fits the data reasonably well). As a consequence, it is easy to use a structural model as a fancier alternative to crude interpolation methods like the ones provided in the `gap_filler` function, that is contained in the `extra` addon to gretl.

The script below provides an example with Box and Jenkins' "airline" series, in which we artificially create a few gaps first and then fit a Basic Structural Model to recover the missing entries.

```
set verbose off
include StrucTiSM.gfn

# use Box and Jenkins' airline series
open bjg.gdt --quiet

# create a few artificial gaps
gappy = lg
smpl 1954:3 1955:6
gappy = NA
smpl full

# fit a Basic Structural model to the gappy series
m = BSM(gappy)

# fill in the gaps
gappy_filled = ok(gappy) ? gappy : gappy_level + gappy_seas

# compare original and reconstructed series
gnuplot gappy_filled lg --with-lines --time-series --output=display
```

Running the above script above should produce the following output and the plot depicted in figure 3.

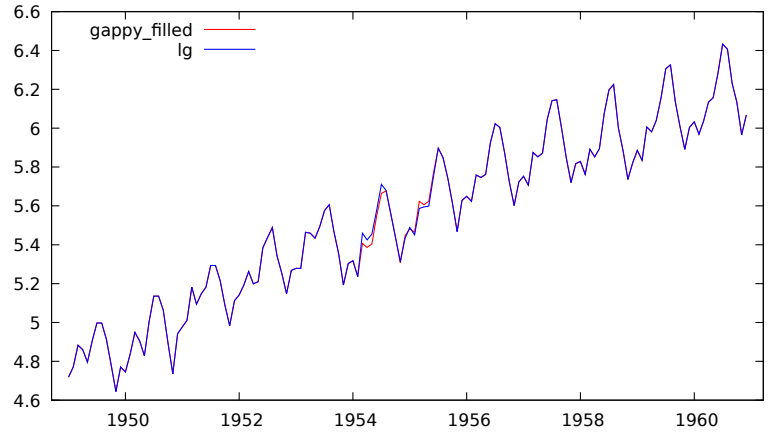


Figure 3: Original and reconstructed airline data

```
Structural model for gappy, 1949:01 - 1960:12 (T = 128)

                  coefficient  std. error       z      p-value
  ------------------------------------------------------------
  Irregular       0.0110440    0.00656742   1.682      0.0926   *
  Trend           0.0293437    0.00415215   7.067      1.58e-12 ***
  Slope           4.17805e-08  0.000568305  7.352e-05  0.9999
  Seasonal (dums) 0.00718025   0.00271764   2.642      0.0082   ***

Average log-likelihood = 1.60974


Specification:

Stochastic trend, stochastic slope, dummy seasonals (stoch.),
irregular component
```

2.5 Exogenous variables

You may include exogenous regressors in your model. In this case equation (1) becomes

$$y_t = x_t'\beta + \mu_t + s_t + \varepsilon_t \tag{4}$$

An example is given in the dedicated directory, where we use the dataset made popular by Harvey and Durbin (1986) on the effects of seat belt legislation in UK road fatalities. The file's name is `roadacc.inp`, and it contains a specification akin to the one presented in Commandeur and Koopman (2007), section 7.2.

Please note that, as yet, models without stochastic components cannot include exogenous variables. Use OLS for that.

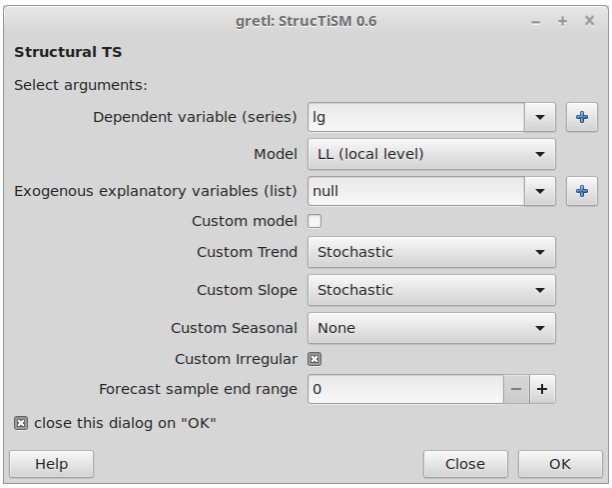
Forecasting with exogenous variables When the model includes exogenous variables, forecasting is somewhat complicated by the requirement of having

valid observations for those non-modelled varaibles in the range of the forecasting horizon. How this data handling is performed depends on whether the GUI or the scripting interface is used.

As explained in the forecasting section 2.3, when using the GUI, the initially active sample must contain the forecast range. The StrucTiSM package then shortens the sample for estimation automatically, and this approach also covers the inclusion of exogenous variables by taking their forecast range values from the initial full sample. Obviously, those values have to exist to make forecasting with such a model possible.

The situation is a little bit different for the scripting interface. Here estimation is always carried out for the currently active sample range, because there is no way for the estimation function `STSM_estimate()` to know that a forecast is wanted later on. When calling the `STSM_forecast()` function afterwards, the user must make sure that the active sample at that point also covers the desired forecast range as indicated by the horizon argument. This is different from the situation without exogenous variables, the reason being that the needed exogenous values for the forecast range would not be available for the function otherwise. Hence, the shortening of the active sample for estimation and the subsequent re-expansion for forecasting is the responsibility of the scripting user in this case. (The forecasting horizon can of course be shorter than the available post-estimation sample range, so that argument must still be specified.)

2.6 The GUI



You access this window via the *Model* > *Time Series* > *Structural TS* menu. Things to note:

- You can select one of the three stock models from the drop-down menu, or put together your own combination, by ticking the "custom model" tick box. You may add exogenous variables via the appropriate box, either for stock models or custom ones.
- After pressing "OK", estimation will be performed and the results displayed. If you click on the Graph button, a summary graph of the states

will be produced.

- The "Save bundle content" icon gives you access to the individual bundle elements, which include the smoothed states, the system matrices, and more.
- You can save the model bundle by clicking the Save button in the output window. After that, you can extract the components by applying the `STSM_components` function to the bundle you just saved.
- Since v0.6 there is an additional integer argument specifying the forecast horizon. If you set that to a positive value, the estimation sample will be shortened by that amount, and forecasts will be produced for the trailing observations. Thus, if you want to calculate truly out-of-sample forecasts via the GUI, you would need to add extra (empty) observations to your dataset (menu Data/Add observations; or through the scripting command `dataset addobs`) before calling the StrucTiSM package.

3 Function list (in alphabetical order)

```
BSM(series y, bool se, model *mod)
```

Estimates a Basic Structural Model of y_t and returns a list holding the components.

- `y`, the dependent variable
- `se`, include in the list also the estimated standard errors for the smoothed states (optional, default=no)
- `mod`, if not null, contains the estimated model upon successful completion

---

```
LLT(series y, bool se, model *mod)
```

Estimates a Local Linear Trend model of y_t and returns a list holding the components.

- `y`, the dependent variable
- `se`, include in the list also the estimated standard errors for the smoothed states (optional, default=no)
- `mod`, if not null, contains the estimated model upon successful completion

---

```
STSM_components(bundle mod, bool se)
```

Extracts the states to a list of series.

- `mod`, the estimated model
- `se`, include in the list also the estimated standard errors for the smoothed states (optional, default=no)

`STSM_estimate(bundle *mod, int verbose, int mapping, int vcvmethod)`

Estimates the variances of the model and performs the state smoothing. The arguments are:

- `mod`, pointer to a bundle created via `STSM_setup`
- `verbose`, verbosity (optional, default=1),
- `mapping` type of reparametrisation: 0 = Variances, 1 = Std. Dev (default), 2 = logarithm
- `vcvmethod` 0 = OPG, 1 = Hessian (default), 2 = QML

Returns an error code.

`STSM_fcast(bundle *mod, int horizon, bool verbose, bool do_states)`

Forecasts the dependent variable and stores results into the bundle as `fcast` and `fcastvar`. The arguments are:

- `mod`, pointer to a bundle created via `STSM_setup`; it must contain estimation results.
- `horizon`, an integer with the forecasting horizon. If 0 or omitted, a "sensible" default based on current periodicity is used.
- `verbose` Boolean. If non-zero, a printout is given with point forecasts and associated standard errors (default = yes).
- `do_states` Boolean. If non-zero, the predicted values for the unobservable components are also stored into the bundle `mod` under the names `sfcast` and `sfcastvar`.

Returns an error code.
NOTE: Models with exogenous variables are supported since version 0.65, under the obvious condition that those variables must have known values over the forecasting horizon.

`STSM_printout(bundle *mod)`

Prints out the estimates.

```
STSM_setup(series y, bool epsilon, int trend, int slope, int seasonal,
list X)
```

Returns an initialised bundle. The arguments are:

- `y`, the dependent variable
- `epsilon`, Boolean, presence of ε_t in eq. (1) (optional, default=yes),
- `trend` type of trend: 1 = stochastic, 2 = deterministic (optional, default=1),
- `slope` 0 = none, 1 = stochastic, 2 = deterministic (optional, default=1),
- `seasonal` Seasonal, 0 = none, 1 = stochastic with trigonometric terms, 2 = stochastic with dummies, 3 = deterministic dummies (optional, default = 2)
- `X` list, exogenous variables in the measurement equation (optional, default = null)

References

Commandeur JJ, Koopman SJ. 2007. *An Introduction to State Space Time Series Analysis*. Oxford University Press.

Commandeur JJ, Koopman SJ, Ooms M. 2011. Statistical Software for State Space Methods. *Journal of Statistical Software* **41**.

Durbin J, Koopman SJ. 2012. *Time Series Analysis by State Space Methods*. Oxford University Press, 2 edition.

Hamilton JD. 1994. *Time Series Analysis*. Princeton, NJ: Princeton University Press.

Harvey AC. 1989. *Forecasting, Structural Time Series Models and the Kalman Filter*. Cambridge University Press.

Harvey AC, Durbin J. 1986. The effects of seat belt legislation on British road casualties: A case study in structural time series modelling. *Journal of the Royal Statistical Society. Series A (General)* : 187–227.

Lucchetti R. 2011. State Space Methods in gretl. *Journal of Statistical Software* **41**.

Pelagatti M. 2015. *Time Series Modelling with Unobserved Components*. CRCpress.

Pollock DSG. 1999. *A Handbook of Time-Series Analysis, Signal Processing and Dynamics*. New York: Academic Press.

Changelog

Version	Changes
0.1	Initial release
0.2	Several bug fixes (thanks to Silvia Rodriguez for her bug report); exogenous variables in the measurement equation; optional export of the variance series for the smoothed states; switch to LBFGS for optimisation.
0.3	Handling models with no stochastic latent states via OLS as "special cases".
0.4	Bump version requirement to avoid windows bug; changed plot titles in GUI; added some error checking.
0.5	Add `STSM_fcast` and make it possible to retrieve the model bundle from shortcut functions.
0.51	Fix bug with forecasting with models with no irregular component.
0.52	Add subsection 2.4.
0.53	Fix bug with forecasting when horizon = 1.
0.6	Attempt forecasting support in the GUI.
0.7	(Nov 2021) Extend forecasting support to models with exogenous variables; require gretl 2020c for internal reasons.