Studies in Nonlinear Dynamics and Econometrics

Quarterly Journal
April 1996, Volume 1, Number 1
The MIT Press

Studies in Nonlinear Dynamics and Econometrics (ISSN 1081-1826) is a quarterly journal published electronically on the Internet by The MIT Press, Cambridge, Massachusetts, 02142. Subscriptions and address changes should be addressed to MIT Press Journals, 55 Hayward Street, Cambridge, MA 02142; (617)253-2889; e-mail: journals-orders@mit.edu. Subscription rates are: Individuals $40.00, Institutions $130.00. Canadians add additional 7% GST. Prices subject to change without notice.

Estimation of the Stochastic Volatility Models by Simulated Maximum Likelihood: C++ Code

Jón Daníelsson*
Department of Economics
University of Iceland
101 Reykjavík, Iceland
jond@hag.hi.is

Abstract. *This is documentation for a C++ implementation of the simulated maximum likelihood (SML) estimation method, where the SML algorithm is applied to the stochastic volatility (SV) model. The algorithm and code can easily be adapted to a richer class of SV models, as well as to more general dynamic latent-variable models.*

1 Introduction

The stochastic volatility (SV) model has been estimated by a variety of estimation methods. The only known exact-likelihood method is simulated maximum likelihood (SML), and this is the documentation for C++ code for estimation of the univariate SV model by SML. The version of the model implemented below is a basic, three-parameter log-normal SV model, but it can easily be extended to a richer class of SV models, such as allowing for mean or leverage effects. The code also provides the building blocks for implementing a general dynamic latent-variable model.

The code is C++, and is designed so that it will compile and run on most systems with minimal effort. Executables for several systems are also available. This code was originally used by Daníelsson and Richard (1993) and Daníelsson (1994), but had to be extensively modified to make it suitable for public release. The code has been tested on an HP UX system using the HP optimizing compiler, a SUN OS system using GNU g++, and Windows 95 using Visual C++ 4.0.

The code is divided into two main parts: the main algorithm and the library functions. The implementation of SML along with various supporting routines is self-contained. The code is dependent upon library calls for random-number generation and optimization. The code was developed with the aid of the NAG library, however, equivalent routines had to be used instead so the code would compile on any system. See Section 7 for more details.

2 The Stochastic Volatility (SV) Model

The simplest version of the SV model is:

$$\begin{aligned} y_t &\sim N\left(0, \sigma_t^2\right) \\ \log \sigma_t^2 &\sim N\left(\omega + \delta \log \sigma_{t-1}^2, \upsilon^2\right). \end{aligned} \tag{2.1}$$

*I would like to thank Birgir Runolfsson and an anonymous referee for useful comments. I benefitted from the HCM fellowship of the EU, and the Tinbergen Institute, Erasmus University Rotterdam provided facilities to finish this project. Updated versions of the program and executables for HP UX, SUN OS, and WIN32 are available through my homepage *http://www.hag.hi.is/~jond*. Executables for more systems and updates may become available at a later date at this site.

The model in (2.1) is the most typical representation of the SV model. However, several extensions have been proposed in the literature. In most cases the SML code can easily be adapted to include those extensions.

2.1 Mean component

It is easy to estimate model (2.1) with a fixed mean. The most efficient way would be to subtract the mean from the data before any estimation, since known estimation methods are rather complex and the end result would be the same. If the mean, μ_t, is time dependent, the parameters of the mean process have to be estimated directly. While most researchers do not find significant mean components, it is easy to implement them, both in the theoretical model and the SML algorithm. For example, the mean might be assumed to be a function of expected volatility:

$$\begin{aligned}\mu_t &= \alpha + \beta E\left[\sigma_t^2\right] \\ &= \alpha + \beta\sigma_{t-1}^2\end{aligned}$$

which is the specification tried and rejected in Daníelsson (1995). This requires changing line 168 in file **sml.cc** from

```
yyy = x2[t]/exp(ly)+ly;
```

to

```
mu = alpha+beta*exp(lagy1[n]);
yyy = (data[t]-mu)*(data[t]-mu)/exp(ly)+ly;
```

where `mu` has to be defined as double.

2.2 Leverage effects

Leverage effects imply that the shock to returns, ϵ_t, is correlated to the shock to volatility, η_t. This type of model was tried and rejected in Daníelsson (1995). This can easily be implemented by changing line 166 in file **sml.cc**.

3 Simulated Maximum Likelihood

Estimation of dynamic latent-variable (DLV) models is complicated and typically requires simplifying assumptions for the underlying model. The Simulated Maximum Likelihood (SML) method was developed by Daníelsson and Richard (1993) for estimation of a general DLV model, and Daníelsson (1994) applied SML to the SV model.

Since σ_t^2 is latent, it has to be integrated out of the joint density of y_t and σ_t^2 as in:

$$f\left(Y_T \mid \theta\right) = \int_{\Delta} f\left(X_T,\, Y_T\right) dX_T \tag{3.1}$$

where X_T contains all σ_t^2, Y_T contains all y_t, θ is the vector of parameter, and Δ is the support $(\Re_+^T)$. The marginal density $f\left(Y_T \mid \theta\right)$ is obtained by simulation. Define an importance sampling function (ISF) as:

$$\mu\left(X_T \mid\, Y_T, \theta\right) \equiv \prod_{t=1}^{T} f\left(\sigma_t^2 \mid \sigma_{t-1}^2, \theta\right) \tag{3.2}$$

and a remainder function (RF) as:

$$h\left(X_T,\, Y_T \mid \theta\right) \equiv \prod_{t=1}^{T} f\left(y_t, \sigma_t^2 \mid \theta\right). \tag{3.3}$$

A Monte Carlo (MC) estimate of the marginal density $f\left(Y_T \mid \theta\right)$ is obtained by drawing simulated Y_T from the ISF and evaluating the RF with those random values. If the n-th simulated vector of Y_T is denoted as $Y_T^{(n)}$, then one simulated likelihood value is:

$$h\left(X_T,\, Y_T^{(n)} \mid \theta\right) \tag{3.4}$$

and the SL estimate, conditional on the parameters, is:

$$\overline{I}_N(\theta) = \sum_{n=1}^{N} h\left(X_T,\, Y_T^{(n)} \mid \theta\right) \tag{3.5}$$

where N is the number of simulations. Parameter estimates are obtained by maximizing $\log(\overline{I}_N(\theta))$.

3.1 Acceleration

Estimation by the importance and remainder functions obtained by (3.2) and (3.3) is very inefficient, and the accelerated Gaussian importance sampling (AGIS) algorithm (Daníelsson and Richard [1993]) is used to speed up the simulation.

Multiply the ISF and divide the RF by the auxiliary function (AF):

$$\mathcal{A}\left(Q^{(n)},\, Y^{(n)}\right) \equiv \prod_{t=1}^{T} \exp\left[a_t^{(n)} + b_t^{(n)} y_t^{(n)} + c_t^{(n)} \left(y_t^{(n)}\right)^2\right]$$

where the coefficients $Q^{(n)}$ are obtained by regressing $\log(f(y_t^{(n)}, \sigma_t^2 \mid \theta))$ on constants $y_t^{(n)}$ and $(y_t^{(n)})^2$. See class `regress` for the details of the regression. The ISF and the AF are multiplied to get the accelerated ISF, which is rewritten as a density function. See function `sml1::get_isd()` for the details of that operation.

3.1.1 Implementation of SML-AGIS The optimizer calls the function `smaxf1()` with the parameter vector θ. All `smaxf1()` does is call the class function `sml1::OptFunc()`, which checks for valid parameters and calls the function `sml1::func()` and then reports and returns the result to the optimizer. These functions are in the file `con.cc`.

The implementation of the SML-AGIS algorithm is contained in the file `sml.cc.` There the first stop is the function `sml1::func()`. Note that the algorithm uses Q, where the code uses the variables `bb1`, `bb2`, and `bb3`. The algorithm is outlined in Figure 1.

`regress::init()`, which allocates memory for the `regress` class and calculates the expected value of $Q^{(n)}$.

1. This value of Q is used to evaluate the coefficients of the accelerated ISF in `sml1::get_isd()`.
2. A loop is created that iterates through the AGIS iterations, default 30 times, or if `diffindex` < `STOP_AGIS`. `diffindex` measures the change in Q between iterations.
   1) Evaluate likelihood in function `sml1::lik1_func()` which:
      (a) Loops through time,
      (b) generates vectors of `N`-sized random numbers:
      - loops through N
      - evaluates log likelihood
      - collects values for regression in `regress::add()`,

      (c) after exiting simulation loop, does regression with `regress::run`,
      (d) continues looping through time, and
      (e) calculates log likelihood, applying constant.
   2) If `diffindex` < `STOP_AGIS` or iteration > 30 exit, else evaluate `sml1::get_isd()`.
3. Check for errors and return function value.

Figure 1
One evaluation of the likelihood, conditional on θ.

4 Numerical Issues

There are a few numerical issues that require special attention.

4.1 Estimation of ω by auto scale

When ω is very different from zero, e.g., it may be -1.0 for weekly data, the algorithm tends to underestimate ω by a large amount, but the estimates of δ and υ are not affected. This is corrected for by multiplying the data by a scaling factor, *scale*. This provides ω estimates that are close to zero, and the corresponding "adjusted" ω, called ω^*, estimate can then be rescaled to the original ω. The best way to do this is using δ and υ to get the scaling factor, i.e.:

$$E\left[y_t\right] = \exp\left[\frac{\omega}{1-\delta} + \frac{1}{2}\frac{\upsilon^2}{1-\delta^2}\right]$$

and since the target value of ω is 0, we let:

$$scale = \sqrt{\frac{\exp\left[\frac{1}{2}\frac{\upsilon^2}{1-\delta}\right]}{\mathrm{Var}\left(y_T\right)}}.$$

If we then define $y_t^* = scale \cdot y_t$, the δ and υ estimates will be unbiased by using y_t^*, and ω^* is rescaled back by

$$\omega = \omega^* - \log\left(scale^2\right)(1-\delta)\,.$$

To get the Hessian in correct units, the values are converted at each function evaluation. This method requires knowing δ and υ. They can be obtained by iteration, i.e., use the initial values to scale the data, get δ and υ estimates, use those to rescale the original data, re-estimate, and iterate. In general, four iterations seems enough. This is done in the code by the flag `Ascale`, which by default is set equal to `YES`, but can be reset on the command line (see Section 5.2). To get the correct likelihood value, the program makes one final run with the unscaled data and optimal parameters. See the function `sml1::optimize()` for the exact implementation.

4.2 Fast and slow regression

There are two ways to do the regression in class `regress`. We can employ a least-squares method that uses singular value decomposition, or we can invert the $X'X$ matrix by hand (it is a 3×3). The latter method is much faster, but less robust. It is also the default method, but the user can override it by the command line option **-reg**.

5 Using the Program

The program is called up from the command line. The default name of the executable is **svm**, and at least one more argument is needed, i.e., the name of the input file, e.g.,

svm ibm.dat.

Note that the first argument must be the name of the input file, and if the program is called up without any command line options it will print out all the options and exit. Default values, e.g., initial values, can be modified on the command line.

5.1 Input file

The input file must have one column that contains the data in log-difference form. See, e.g., the sample file sp500.dat. The program will automatically determine the number of observations in the file.

5.2 Command line

The program sets a number of default values which can be overridden on the command line. Probably the three options most likely to be changed are the initial parameter values, auto scaling, and the amount of output. The command line options are the following:

-n	Number of simulations (default 25)
-i	Number of AGIS iterations (default 30)
-ascale	Turns autoscaling off (see Section 4.1)
-h	Turns off Hessian estimation
-p	Tolerance (default 0.002)
-print	And one integer in $[1, 4]$; determines the amount of output (default 1)
-seed	Sets random number generator seed (default 324)
-par omega delta nu	Sets the initial values of the parameters (default 0.0, 0.95, 0.4)
-reg	Turns on robust and slow regression (see Section 4.2)

5.3 Amount of output

The amount of printed output is controlled by the command line option **-print**. This sets the variable `Print`, which has the default value of 1. The other possible values are:

0	Print only maximum log-likelihood value
1	Print header and footer
2	Additionally print a report on autoscale
3	Additionally print each iteration
4	Additionally print messages on each iteration

5.4 Sample output

If the default values are used, then the output from using SP-500 data is:

```
Simulated Maximum Likelihood Estimation
    v. 1.2 February 15, 1996
Stochastic Volatility Model
Jon Daníelsson (c) 1996
Send comments to: jond@hag.hi.is
Wed Feb 14 17:17:05 1996
Initial Value: omega = 0, delta = 0.95, nu = 0.4
    Inputfile: sp2022
    Input Data: nobs = 2022, mean = 0.0421258, var = 1.25686
Function Value = -2898.803
Running Time = 00:08:39
Parameter    Value   s.e.
Omega       -0.001   (0.004)
Delta        0.961   (0.008)
Nu           0.158   (0.015)
```

6 The Code

All code is written in standard C++. It contains both core SML files and library routines. The library routines are kept separate, because a user might prefer to supply her own routines.

6.1 Core code

6.1.1 Files The source code consists of one header file **svm.h** and five program files.

main.cc	Sets up the program classes
setup.cc	Parses the command line, sets default options, and contains utility routines

con1.cc	Sets up the optimization, prints main output, calls optimizer, and contains the objective function
sml.cc	The main part of the program, evaluates the likelihood function
regress.cc	Does the regression work for the auxiliary function; this is a separate class
Makefile	Generic makefile (may need local modification)

6.1.2 Classes The program classes are:

`command_line`	Provides default values and command line options
`sml1`	Where the simulated likelihood evaluation takes place
`regress`	Calculates the default values for the auxiliary function, collects simulated y's during the simulation, and provides the parameters of the auxiliary function

7 Libraries

I use the NAG library for all "black box" numerical work. In general, one gets very high quality routines with minimum effort by using NAG. This is especially important in random number generation, which is at the center of all simulation work. The NAG library routines are also very useful in optimization and estimating the Hessian.

The NAG library is sold on a commercial basis, and publicly available routines had to be substituted for this release. I added the public-domain RANLIB library for random number generation to this release along with some routines for optimization and Hessian estimation. These routines are of inferior quality to the commercial NAG routines, and the NAG library is plug-compatible to the code. If I find better public-domain routines, I will include them in later versions.

The interface to the library routines is through the classes `opt` and `normal`, which are declared in header files **optimize.h** and **random.h**. The library-related files are:

optimize.h	Declares the interface class between optimization library routines and the main code
random.h	Declares the interface class between random number library routines and the main code
common.h	Defines macros that direct the code toward NAG or other library calls
com.cc	Part of the RANLIB release
ranlib.cc	Part of the RANLIB release
ranlib.h	Declarations for the RANLIB library
simplex.cc	Contains optimization and Hessian estimation code
optimize.cc	The interface code between the core code and library calls

References

Daníelsson, Jon (1994), "Stochastic Volatility in Asset Prices, Estimation with Simulated Maximum Likelihood," *Journal of Econometrics* 64, 375–400.

Daníelsson, Jon (1995), "Multivariate Stochastic Volatility Models and GARCH Models," mimeo, University of Iceland.

Daníelsson, Jon and J.F. Richard (1983), "Quadratic Acceleration for Simulated Maximum Likelihood Estimation," *Journal of Applied Econometrics* 8, 153–173.

Editorial Policy

The *SNDE* is formed in recognition that advances in statistics and dynamical systems theory may increase our understanding of economic and financial markets. The journal will seek both theoretical and applied papers that characterize and motivate nonlinear phenomena. Researchers will be encouraged to assist replication of empirical results by providing copies of data and programs online. Algorithms and rapid communications will also be published.

ISSN 1081-1826