Estimation and Inference with the Efficient Method of Moments: With Applications to Stochastic Volatility Models and Option Pricing
van der Sluis, P.J.

Citation for published version (APA):
van der Sluis, P. J. (1999). Estimation and Inference with the Efficient Method of Moments: With Applications to Stochastic Volatility Models and Option Pricing Amsterdam: Thela Thesis. TI Research Series nr. 204

General rights
It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

Disclaimer/Complaints regulations
If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: http://uba.uva.nl/en/contact, or a letter to: Library of the University of Amsterdam, Secretariat, Singel 425, 1012 WP Amsterdam, The Netherlands. You will be contacted as soon as possible.
Appendix A

Description of Code and Score Generators

Even with the current state of computing power the estimation of stochastic volatility models using simulation techniques such as the efficient method of moments may still be very time-consuming. This is amplified by the fact that many econometricians use for their applied work matrix-oriented programming languages, such as GAUSS, MATLAB, S-PLUS or more recently Ox (Doornik (1996)). See e.g. Cribari-Neto (1997) for a comparison of GAUSS, Ox and S-PLUS. As long as the estimation involves mainly matrix manipulations without any substantial for-next or do-while loops, which is the case for linear models such as regression, AR and VAR models, there is no loss of computing power. There is also no loss in computing power in case the developers of the matrix language have provided the researcher with a hard coded built-in procedure for doing a certain time consuming job. However, there is a big loss of computing time as soon as the solution to a certain problem involves loops, such as the Kalman filter, where a system of Riccati equations is evaluated recursively, or the EMM estimation technique that we are dealing with in this thesis.

One faces a substantial loss of computational speed for two reasons: (i) A matrix oriented language is not good at doing loops. (ii) Kalman filter techniques or EMM are not yet standard procedures of these matrix oriented programming languages. One solution may be to hard code the whole program in C, FORTRAN, MODULA 2 or PASCAL. This has the drawback that the programmer has the continuous feeling that he or she is reinventing the wheel. The econometrician has to do many things that do not belong to his or her core competence, therefore from an economic perspective the marginal benefits of less computing costs may not be on equal footing with the marginal losses in human capital\(^1\). One possible solution to

\(^1\) An example where C/C++ code is provided in this context is Danielsson (1996a) where C/C++
this is to *link* hard coded procedures that contain the loops, to the matrix programming language, combining the facilities of the matrix programming language with the speed of a (relatively) low level programming language. Under Ox this link can be implemented *dynamically* using dynamic link libraries (DLL’s) on Windows based platforms (Windows 3.xx, Windows 95/98 and Windows NT) or *statically* on mainframe computers under AIX, UNIX, SUN and many others. These libraries basically extend the matrix language with some procedures. For this has been done for the Kalman filter by Koopman, Shephard and Doornik (1999) with SFFPack, and for several Bayesian procedures to estimate stochastic volatility models by Shephard (1996b) with SvPack, among others. Here time consuming procedures were hard coded in C/C++ and linked to the matrix programming language Ox. Exactly this is done here for the EMM estimation technique for a wide class of stochastic volatility models using dynamic link libraries for Windows and statically on the RS/6000 under AIX.

The program is called EmmPack. Unlike its name it does not only provide code to estimate stochastic volatility models via EMM. As a consequence of the EMM methodology one can also use the package to estimate by maximum likelihood the auxiliary SemiNonParametric models, for which, in the case of stochastic volatility models, we have specified an EGARCH leading term. This means that this program can also be used to quickly estimate EGARCH models with disturbance terms that follow a variety of non-normal distributions.

The contents of this appendix previously appeared in van der Sluis (1997a). The current version of EmmPack is 1.04.

### A.1 Description of EmmPack and Ox

The Ox matrix programming language is new to the market of econometric software. It is by far the fastest matrix oriented programming language see Cribari-Neto (1997). Several other advantages are: it is a very open language, its syntax is very similar to C/C++ and there are currently versions for DOS, Windows 3.xx, Windows 95/98, Windows NT, AIX, SunOs, Solaris, HP-UX, Irix, Linux and Unix. Most of these versions are free. There are no differences between these versions, except that only in the Windows version one can use Ox’s sister program

source code for the estimation of a basic stochastic volatility model is given.

For an extensive list of different platforms for which Ox is available, see section A.1. Since the list is still growing, for an up to date list see

http://www.nuff.ox.ac.uk/Users/Doornik

This code is available from the author on request. For other platforms contact the author.
A.1. DESCRIPTION OF EMMPACK AND OX

GiveWin which provides several graphic capabilities. Another small difference is that for some versions one cannot dynamically link C/C++ code to Ox, for these platforms one has to resort to statical linking which is not more difficult but less elegant. One may also try to link the library to the more common programs as GAUSS or MATLAB. One should probably have to write a DLL which translates exported C/C++ functions from the Ox DLL to C/C++ functions that can be imported from GAUSS or MATLAB. How the C/C++ functions are exported from an Ox DLL can be found in Doornik (1996, pp. 286–328). It should be mentioned that the DLL in EmmPack uses mathematical C/C++ function from Ox, so one should own a copy Ox. Therefore one may better save the trouble and directly use Ox.

The DLL is optimized for the Intel Pentium. For the RS/6000 the author has used the IBM compiler which should generate very efficient code on the RS/6000. In spite of this it is worth mentioning that the gain in speed on both the Intel and the RS/6000 was immense, with the highest relative gain on the Intel. On a P5-166 with 32 MB under Windows 95/98 it took in pure Ox 1.20a code 117.7 seconds to evaluate an EGARCH(1,1)-H(5,0) for $2 \times 5,000$ antithetic variables. With the C/C++ code in a DLL it took only 9.5 seconds under the same configuration: a twelve-fold improvement. Whereas this gain in speed is considerable this does not mean that one should always program in C/C++ or any other low level programming language. Only for the types of problems that we are dealing with in this particular problem, namely a loop with allocations, hard-coding gives an enormous speed improvements.

Currently version 1.04 of EmmPack can be downloaded from

http://center.kub.nl/staff/sluis/

The files are zipped as emmpack.zip. One can unzip them with a utility such as pkunzip. This file contains the following: s&p500.mat, xr.mat, emm.dll, sv_model.oxo, sv_model.h, maxlsid.oxo, maxlsid.h, emm.h, and the central program, emm.ox, containing modifiable Ox source code. Unless you are an oxpert, it is wise to create a directory \ox\packages\emm\ and move the emm.ox file to this directory. Next, place the files sv_models.oxo, sv_models.h, maxlsid.oxo and maxlsid.h to the directory \ox\include\ and move the files emm.dll to the directory \ox\bin\). Additional description of the program can be found in the source code emm.ox.

A few things to note here: emm.dll contains a dynamic link library for the Windows 3.xx, Windows 95/98 and Windows NT operating systems. These require Ox version 1.20a. The AIX version is available on request from the author. The sv_models.oxo and maxlsid.oxo contain compiled Ox code. The maxlsid.oxo is a modification of the maximize.oxo code that is included in the official release of Ox, except that instead of two-sided derivatives only one-sided derivatives are taken in
the BFGS algorithm. Although less accurate the resulting optimization is twice as fast and the loss of accuracy seems to be irrelevant for the problem at hand. The file sv_models.oxo contains the Ox compiled code of several procedures to generate antithetic series from stochastic volatility models. There is no need to hard code these procedures, because the author extensively used hard-coded built in procedures from Ox. So there will be virtually no gain, maybe even a loss, in hard coding these procedures. These are not all the stochastic volatility models that can be estimated with this program. The user can use his own. In the module sv_models.oxo the following procedures have been currently provided:

- `sarmav10(const total,const theta)`
- `sarmav20(const total,const theta)`
- `sarmav30(const total,const theta)`
- `sarmav11(const total,const theta)`
- `sarmav12(const total,const theta)`
- `asarmav10(const total,const theta)`
- `asarmav20(const total,const theta)`
- `asarmav30(const total,const theta)`
- `asarmav11(const total,const theta)`
- `asarmav12(const total,const theta)`

**in:** an integer `total` denoting the number of variables you want to simulate, `theta` a vector of parameters. The first element of `theta` denotes the \( \omega \) variable, the next variables denote the ARMA variables. Next follows the variable \( \sigma_p \). If applicable the last element is the asymmetry variable \( \lambda \). **out:** returns a `total`\( \times \)2 series of antithetic variables\(^4\) from the specified process.

\(^4\)For an explanation and motivation of antithetic variables see Appendix A.6.
Before estimating the structural stochastic volatility model, one has to specify an auxiliary EGARCH-H model. As mentioned before, one can also use this program for estimating EGARCH-H models. As described above the auxiliary models are taken from the SNP densities with an EGARCH\((p, q)\) leading term. In the program the integers \(k_x, k_z, p\) and \(q\) refer to the variables \(K_x, K_z, p\) and \(q\) respectively, as defined in (3.19). The order of input and output of the variables is follows: \(\{\xi, \alpha_1, \ldots, \alpha_q, \rho_1, \ldots, \rho_p, \kappa_0, \kappa_1, \Theta\}\) where \(\Theta\) is a matrix defined as

\[
\Theta = \begin{bmatrix}
1 & a_{10} & \cdots & a_{K_x0} \\
a_{01} & \ddots & \ddots & \vdots \\
\vdots & \ddots & \ddots & \vdots \\
a_{0K_x} & a_{1K_x} & \cdots & a_{K_xK_x}
\end{bmatrix}
\]

(A.1)

In the program this matrix is vectorized as \((1, a_{01}, \ldots, a_{0K_x}, a_{10}, \ldots, a_{1K_x}, \ldots, a_{K_xK_x})\) by the procedure `decomp`. Other procedures that are included in the source-code of `emm.ox` include:

- **likeli(const vP, const adFunc, const avScore, const amHessian)**
  This function is the likelihood of the auxiliary EGARCH-H model, with in and output in the generic format as on page 114 of Doornik (1996)

- **gradproc(const vP)**
  This function returns the score of the auxiliary model, the auxiliary parameter vector `vP` denotes the parameters of the auxiliary model in which the score has to be evaluated.

- **createin(const vP)**
  This function returns the outer product of the scores in the auxiliary parameter vector `vP`.

- **diagnos(const l, const paras, const diags, const file.diagnos)**
  The function writes several diagnostic statistics for the auxiliary model to an open file `file.diagnos`. Here `l` denotes the loglikelihood in the optimum, `paras` denotes the parameters in the optimum and `diags` is the outer product of the scores in the optimum. Details are also given in the next section.
Appendix A. Description of Code and Score Generators

- **dist(const vP, const adFunc, const avScore, const amHessian)**

This function returns the value of the minimum chi-squared criterium of the EMM estimation of the structural model. Its format is the same as on page 114 of Doornik (1996).

- **jtest(const I, const inv_l, const tot, const m_hat, const M_hat)**

The function J test returns a vector of which the first element denotes the value of the J test and the other elements the individual t−values as described in Section 3. The inputs are l, inv_l, tot, m_hat and M_hat which denote $J_T J^{-1}_T$, $T$, $m_N(\theta_T, \beta_T)$ and $M(\theta_T, \beta_T)$ respectively.

- **gradproc_theta2 (const adFunc, const sv.theta)**

This function returns the score of auxiliary model at the parameters of the structural model sv.theta, i.e. $m(\theta, \beta_T)$. It is of the same format as on page 125 and 126 in Doornik (1996) and therefore by using this function with the Ox procedure NumJacobian we may calculate the numerical derivative of $m(\theta, \beta_T)$ with respect to $\theta$, i.e. $M(\theta, \beta_T)$.

After the auxiliary model is fitted the global variable z contains the $z_t(\beta)$ from the auxiliary model. These may be used for specification tests. More information on the procedures and the variables can be found in the source file emm.ox. The file s&p500.mat contains the S&P500 series and the file xr.mat contains the exchange rate series that have been analysed in Chapter 4.

As for all nonlinear optimization problems: be aware of local optima. The author encountered some in this context. One can check this by trying at different parameter values, possibly with a lower value of $N$. A sensible choice may be to set $N = 20,000$.

A.2 Description of the Auxiliary Model

In this appendix the class of auxiliary models that is used in EmmPack will be described. We use the set-up of Section 2.3.1. Let the score function be $S_T(y_1, ..., y_T; \psi) = \sum_{t=1}^{T} \nabla_{\psi} l_t(y_t; \psi)$

For conditional mean and variance parameters we get

$$\nabla_{\beta} l_t(y_t; \psi) = f[z_t(\beta); \eta]^{-1} f'[z_t(\beta); \eta] \nabla_{\beta} z_t(\beta) - 0.5 h_t^{-2}(\beta) \nabla_{\beta} h_t^{2}(\beta)$$  \hspace{1cm} (A.2)

here \( f'[z_t(\beta); \eta] = \frac{df[z_t(\beta); \eta]}{dz_t(\beta)} \) and \( \nabla_{\beta} z_t(\beta) = -\nabla_{\beta} \mu_t(\beta) h_t^{-1}(\beta) - 0.5 \sigma_t(\beta) h_t^{-3}(\beta) \nabla_{\beta} h_t^{2}(\beta) \)
In EmmPack we specify the standard normal density for $z_t$. Now there are no nuisance parameters so $\psi = \beta$. The gradient becomes

$$\nabla_\beta l_t(y_t; \beta) = \nabla_\beta \mu_t(\beta) \epsilon_t(\beta) h_t^{-2}(\beta) + 0.5 \nabla_\beta h_t^2(\beta) h_t^{-2}(\beta) [\epsilon_t(\beta)^2 h_t^{-2}(\beta) - 1]$$

(A.3)

It is known that in many models the normal distribution cannot capture all the excess kurtosis that is often observed in financial markets. An alternative may be to use Student's $t$ distribution or the Generalized Error Distribution (GED) of Nelson (1991). However since we will use a SNP density for the $z_t$ we follow the easiest way and use the Gaussian density.

In this version of EmmPack, the leading term of the SNP model, specified by $m_t(\beta)$ and $h_t^2(\beta)$ will be, $m_t(\beta) = 0$ and $h_t^2(\beta)$ following the EGARCH model of Nelson (1991) as given in (2.5). Note that in the implementation of EmmPack the absolute function is replaced by $b(z)$ which denotes a two times differentiable approximation to the absolute value function, $|z|$.

The SNP density based on the Gaussian distribution is given by formula (3.18). For identification we need $a_{00} = 1$. Let $K = \max(K_z, K_x)$. For computational ease we employ polynomials that are of Hermite form:

$$He_i(z) = \sum_{j=0}^{[i/2]} (-1)^j \frac{i!}{j!(i - 2j)!} z^{i - 2j}$$

(A.4)

$$\int He_i He_j \exp\{-1/2 z^2\} dz = \begin{cases} 0 & i \neq j \\ \sqrt{2\pi i} & i = j \end{cases}$$

(A.5)

Let

$$He_i(z) = (i!)^{-1/2} He_i(z)$$

(A.6)

A nice result of this normalization is that

$$\int \overline{He_i(u)} \overline{He_j(u)} \phi(u) du = \begin{cases} 0 & i \neq j \\ 1 & i = j \end{cases}$$

(A.7)

The polynomial is now taken as

$$P_K(z, x) = \sum_{i=0}^{K_z} \gamma_i(x) \overline{He_i(z)}$$

(A.8)

---

5A good choice is e.g. $b(z) = |z|$ for $|z| \geq \pi/2K$ and $b(z) = (\pi/2 - \cos(Kz))/K$ for $|z| < \pi/2K$ and set e.g. $K = 100$.

The derivatives are $b'(z) = -1$ for $z < -\pi/2K$, $b'(z) = 1$ for $z > \pi/2K$, $b'(z) = \sin(Kz)$ for $|z| < \pi/2K$. 


where $\gamma_i(x)$ is a monomial in $x$, e.g. $\gamma_i(x) = \gamma_{i0} + \gamma_{i1}x$

\[
\int P_K(z, x)^2 \phi(z) dz = \sum_{i=0}^{K_x} \gamma_i(x)^2
\]

Later on we will need
\[
\frac{\partial z_t}{\partial \psi_k} = \frac{\partial}{\partial \psi_k} \left( \frac{y_t - \mu_t}{\sqrt{h_t^2}} \right) = -\left( \frac{1}{h_t^2} \frac{\partial \mu_t}{\partial \psi_k} + \frac{z_t}{2h_t^2} \frac{\partial h_t^2}{\partial \psi_k} \right)
\]

and the recurrence relation
\[
\begin{align*}
\text{He}_{i+1}(z) &= z \text{He}_i(z) - i \text{He}_{i-1}(z) \\
\text{He}_0(z) &= 1 \\
\text{He}_1(z) &= z
\end{align*}
\]

A differential relation that is useful is given by
\[
\frac{\partial \text{He}_i(z)}{\partial z} = i \text{He}_{i-1}(z)
\]

An individual component of prediction error decomposition of the likelihood function now reads
\[
\ln f_K(y_t|x_t, \psi) = \sum_{t=0}^{K_x} \frac{\gamma_t(x)^2}{2} + \ln \phi(z_t) - 0.5 \ln(h_t^2) - \ln \left[ \sum_{i=0}^{K_x} \gamma_i(x)^2 \right]
\]

Only I and III require special treatment as regards the analytic expression of the derivative. The remaining part of the score is simple and well-known. For completeness, expressions of the derivative of part II is given in Appendix A.3. What is left are some derivatives which are needed in the score generator or that can be used as analytical derivatives. Differentiation of part III
\[
\frac{\partial \ln \sum_{t=0}^{K_x} \gamma_t(x)^2}{\partial \psi_k} = \frac{2}{\sum_{i=0}^{K_x} \gamma_i(x)^2} \sum_{i=0}^{K_x} \gamma_i(x) \frac{\partial \gamma_i(x)}{\partial \psi_k}
\]

note
\[
\frac{\partial \gamma_i(x)}{\partial \psi_k} = 0 \text{ if } \psi_k \text{ is a leading term parameter}
\]

Differentiation of part I
\[
\frac{\partial \ln P_K(z, x)^2}{\partial \psi_k} = \frac{2 \cdot \partial P_K(z, x)}{P_K(z, x)}
\]
A.3. DERIVATIVES OF EGARCH PROCESS

\[
\frac{\partial P_\kappa(z, x)}{\partial \psi_k} = \frac{\partial}{\partial \psi_k} \left\{ \sum_{i=0}^{K_x} \gamma_i(x) \mathbb{H}_{i}(z) \right\} = \sum_{i=0}^{K_x} \frac{\partial \gamma_i(x)}{\partial \psi_k} \mathbb{H}_{i}(z) + \sum_{i=0}^{K_x} \gamma_i(x) \frac{\partial \mathbb{H}_{i}(z)}{\partial \psi_k}
\]

where

\[
\frac{\partial \mathbb{H}_{i}(z)}{\partial \psi_k} = (i!)^{-1/2} \frac{\partial \mathbb{H}_{i}(z)}{\partial z} \frac{\partial z}{\partial \psi_k}
\]

(A.19)

(A.20)

About the Jacobian \( \frac{\partial \gamma_i(x)}{\partial \psi_k} \) in case \( \psi_k \) is not a leading term parameter. For example in the case \( K_x = 1 \) and all non-leading term parameters are vectorized as \( \gamma = (\gamma_00, \gamma_01, \gamma_{10}, \gamma_{11}, \ldots, \gamma_{K_20}, \gamma_{K_21})' \) we have

\[
\frac{\partial \gamma_i(x)}{\partial \gamma} = \begin{pmatrix}
0 & \cdots & 0 & 1 & 0 & \cdots & 0 \\
\vdots & & \vdots & & \vdots & & \vdots \\
0 & & 1 & & x & & 0 \\
\vdots & & \vdots & & \vdots & & \vdots \\
0 & & \cdots & & 0 & & 0
\end{pmatrix}
\]

where the 1 occurs at position \( 2i - 1 \) and the \( x \) at \( 2i \) (A.21)

A.3 Derivatives of EGARCH Process

If \( h_t^2 \) follows an EGARCH\((p, q)\) process then

- \( \nabla \ell_t h_t^2(\beta) = h_t^2(\beta) \left[ 1 + \sum_{i=1}^{p} \frac{\rho_i}{h_{i-1}^2(\beta)} \nabla \ell_t h_{i-1}^2(\beta) + (1 + \sum_{i=1}^{q} \alpha_i L^i) \right] (\kappa_1 \nabla \ell_t z_{t-1} + \kappa_2 b'(z_{t-1}) \nabla \ell_t z_{t-1}) \]

- \( \nabla_{\alpha_j} h_t^2(\beta) = h_t^2(\beta) \left[ \sum_{i=1}^{p} \frac{\rho_i}{h_{i-1}^2(\beta)} \nabla_{\alpha_j} h_{i-1}^2(\beta) + \kappa_1 z_{t-j-1} + \kappa_2 \{ b(z_{t-j-1}) - \sqrt{2/\pi} \} + (1 + \sum_{i=1}^{q} \alpha_i L^i) \right] (\kappa_1 \nabla_{\alpha_j} z_{t-1} + \kappa_2 b'(z_{t-1}) \nabla_{\alpha_j} z_{t-1}) \]

- \( \nabla_{\rho_j} h_t^2(\beta) = h_t^2(\beta) \left[ \sum_{i=1}^{p} \frac{\rho_i}{h_{i-1}^2(\beta)} \nabla_{\rho_j} h_{i-1}^2(\beta) + (1 + \sum_{i=1}^{q} \alpha_i L^i) \right] (\kappa_1 \nabla_{\rho_j} z_{t-1} + \kappa_2 b'(z_{t-1}) \nabla_{\rho_j} z_{t-1}) \]

- \( \nabla_{\kappa_1} h_t^2(\beta) = h_t^2(\beta) \left[ \sum_{i=1}^{p} \frac{\rho_i}{h_{i-1}^2(\beta)} \nabla_{\kappa_1} h_{i-1}^2(\beta) + (1 + \sum_{i=1}^{q} \alpha_i L^i) \right] (\kappa_1 \nabla_{\kappa_1} z_{t-1} + \kappa_2 b'(z_{t-1}) \nabla_{\kappa_1} z_{t-1}) \]

- \( \nabla_{\kappa_2} h_t^2(\beta) = h_t^2(\beta) \left[ \sum_{i=1}^{p} \frac{\rho_i}{h_{i-1}^2(\beta)} \nabla_{\kappa_2} h_{i-1}^2(\beta) + (1 + \sum_{i=1}^{q} \alpha_i L^i) \right] (\kappa_1 \nabla_{\kappa_2} z_{t-1} + \kappa_2 b'(z_{t-1}) \nabla_{\kappa_2} z_{t-1}) \]

The recursive formulas must be started up. In the program we set \( h_{-1}^2, \ldots, h_p^2 = z'z/T \), where \( z \) is based on the current parameter estimates.
A.4 Derivatives of EGARCH-t Process

For \( \nu > 2 \) the standardized \( t \)-distribution is given by

\[
 f(z_t(\beta); \nu) = \frac{\Gamma[0.5(\nu + 1)]}{\Gamma(0.5\nu)} \frac{1}{\sqrt{\pi(\nu - 2)}} \left[ 1 + \frac{z_t^2(\beta)}{\nu - 2} \right]^{-(\nu + 1)/2} \tag{A.22}
\]

The loglikelihood of \( y_t \) is given by

\[
 \ln f(y_t; \zeta) = \ln \Gamma[0.5(\nu + 1)] - \ln \Gamma(0.5\nu) - \frac{1}{2} \ln[\pi(\nu - 2)] - \frac{\nu + 1}{2} \ln[1 + \frac{z_t^2(\beta)}{\nu - 2}] - 0.5 \ln h_t^2(\beta) \tag{A.23}
\]

For the gradients we have

\[
 \nabla_\beta \ln f(y_t; \zeta) = f^{-1}(z_t(\beta); \nu) f'(z_t(\beta); \nu) \nabla_\beta z_t(\beta) - 0.5 h_t^{-2}(\beta) \nabla_\beta h_t^2(\beta) \tag{A.24}
\]

\[
 \nabla_\nu \ln f(y_t; \zeta) = f^{-1}(z_t(\beta); \nu) \frac{\partial}{\partial \nu} f(z_t(\beta); \nu) \tag{A.25}
\]

where \( f'(z_t(\beta); \nu) = \frac{d}{dz} f(z_t(\beta); \nu) \)

\[
 f'(z_t(\beta); \nu) = \frac{\Gamma(0.5\nu + 0.5)}{\Gamma(0.5\nu) \sqrt{(\pi(\nu - 2))}} \left[ 1 + \frac{z_t^2(\beta)}{\nu - 2} \right]^{-\frac{\nu - 1}{2}} \left[ -z_t(\nu + 1) \right] \frac{1}{(\nu - 2 + z_t^2)} \tag{A.26}
\]

\[
 \nabla_\beta z_t(\beta) = -0.5 z_t(\beta) h_t^{-2}(\beta) \nabla_\beta h_t^2(\beta) \tag{A.27}
\]

\[
 \frac{\partial}{\partial \nu} f(z_t(\beta); \nu) = \frac{\Gamma(0.5\nu + 0.5)}{\Gamma(0.5\nu) \sqrt{(\pi(\nu - 2))}} \left[ 1 + \frac{z_t^2(\beta)}{\nu - 2} \right]^{-\frac{\nu - 1}{2}} \frac{1}{2} \psi \left( \frac{\nu + 1}{2} \right) - \frac{1}{2} \psi \left( \frac{\nu}{2} \right) - \frac{1}{2} \ln \left[ 1 + \frac{z_t^2(\beta)}{\nu - 2} \right] + \frac{(\nu + 1) z_t^2}{2(\nu - 2)^2 (1 + \frac{z_t^2}{\nu - 2})} \tag{A.28}
\]
so we have

\[ f^{-1}(z_t(\beta); \nu) f'(z_t(\beta); \nu) \nabla_\beta z_t(\beta) = -\nabla_\beta z_t(\beta) z_0(\nu + 1) \nu - 2 \left(1 + \frac{z_0^2}{\nu - 2}\right)^{-1} \]  
\[ f^{-1}(z_t(\beta); \nu) \frac{\partial}{\partial \nu} f(z_t(\beta); \nu) = 0.5 \psi\left(\nu + \frac{1}{2}\right) - 0.5 \psi\left(\nu \right) - \frac{1}{2(\nu - 2)} \]
\[ -\frac{1}{2} \ln \left[1 + \frac{z_0^2}{\nu - 2}\right] + \frac{(\nu + 1) z_0^2}{2(\nu - 2)^2 \left(1 + \frac{z_0^2}{\nu - 2}\right)} \]  

(A.29)  

(A.30)

Note that \( \psi \) denotes the psi or digamma function as defined in Abramowitz and Stegun (1972) (hence AS), formula 6.3.1. Numerically, this function was evaluated using the recurrence relation 6.3.6 and the asymptotic expansion 6.3.18 in AS. This resulted in a precision of about \( 10^{-14} \). The terms \( -0.5 h_t^{-2}(\beta) \nabla_\beta h_t^2(\beta) \) are the same as those given in Section A.3.

A.5 Derivatives of MEGARCH Process

This appendix provides analytical expressions for the likelihood and score of the MEGARCH model of (3.30) to (3.32). Consider the \( n \)-variate stochastic process \( y_t(\theta_0) \). The \( n \)-variate model for the conditional mean is

\[ \mu_t(\beta) = E_{t-1}[y_t] \]  

(A.31)

This conditional mean is set zero for simplicity. The \( n \)-variate zero mean process is then defined as

\[ \epsilon_t(\beta) = y_t - \mu_t(\beta) \]  

(A.32)

The model for the \( n \)-variate conditional variance-covariance is defined as

\[ H_t(\beta, r) = E_{t-1}[\epsilon_t(\beta, r) \epsilon_t'(\beta, r)] \]  

(A.33)

where \( r \) denotes nuisance parameters. This leads to the \( n \)-variate standardized process:

\[ z_t(\beta) = R_t^{-1}(\beta) \epsilon_t(\beta) \]  

(A.34)

The \( R_t \) will be chosen the Choleski decomposition of the matrix \( H_t \). The matrix \( R_t \) will typically be a lower or upper triangular matrix; see (3.18). Let \( f(z_t; \beta) \) be the density for \( z_t(\beta) \).
APPENDIX A. DESCRIPTION OF CODE AND SCORE GENERATORS

We set \( \mu_t(\beta) = 0 \) and \( H_t(\beta) \) following the MEGARCH model as described in Section 7.2.

Let \( \psi = (\beta', \tau')' \). The density of \( z_t \) is the Gaussian density. From this we deduce that \( H_t \) is given by, suppressing the dependency on \( t \)

\[
H_t = \begin{bmatrix}
    h_0^2 & h_0 r_{01} h_1 & \cdots & h_0 r_{0n-1} h_{n-1} \\
    h_0 r_{01} h_1 & h_1^2 & \cdots & h_1 r_{1n-1} h_{n-1} \\
    \vdots & \vdots & \ddots & \vdots \\
    h_0 r_{0n-1} h_{n-1} & h_1 r_{1n-1} h_{n-1} & \cdots & h_{n-1}^2
\end{bmatrix} \tag{A.35}
\]

Let

\[
\Phi_t = \begin{bmatrix}
    1/h_0 & 0 & 0 & 0 \\
    0 & \ddots & \ddots & \vdots \\
    \vdots & \ddots & \ddots & 0 \\
    0 & 0 & 0 & 1/h_{n-1}
\end{bmatrix} \tag{A.36}
\]

and

\[
\Theta = \begin{bmatrix}
    1 & r_{01} & \cdots & r_{0n-1} \\
    r_{01} & \ddots & \ddots & \vdots \\
    \vdots & \ddots & \ddots & r_{n-2, n-1} \\
    r_{0n-1} & \cdots & r_{n-2, n-1} & 1
\end{bmatrix} \tag{A.37}
\]

since \( f(\Phi_t y_t) = f(\zeta_t) \) we have that \( f(y_t) = \det \Phi_t \cdot f(\zeta_t) \).

Finally, we find the following expression for the loglikelihood of \( y_t \) for \( t = 1, 2, \ldots, T \)

\[
l_t(y; \tau, \beta) \propto \ln |\Phi_t| - \frac{1}{2} \ln |\Theta| - \frac{1}{2} \zeta_t \Theta^{-1} \zeta_t
\]

\[
= \ln |\Phi_t| - \frac{1}{2} \ln |\Theta| - \frac{1}{2} \text{tr} \Theta^{-1} \zeta_t \zeta_t' \tag{A.38}
\]

By the prediction error decomposition we get the following expression for the loglikelihood of the full sample

\[
L_T(y_1, \ldots, y_T; \psi) = \sum_{t=1}^{T} l_t(y_t; \psi) \tag{A.39}
\]

Let the score function be

\[
S_T(y_1, \ldots, y_T; \psi) = \sum_{t=1}^{T} \nabla_{\psi} l_t(y_t; \psi) \tag{A.40}
\]

Let \( \tau = \text{vech}(\Theta) \) and let \( a \) and \( A \) be a vector and matrix respectively defined by

\[
\text{vec}(\Theta) = a + Ar \tag{A.41}
\]
A.6. ANTITHETIC VARIABLES

Then we can show using rules from matrix differential calculus as in Magnus and Neudecker (1988):

\[ \nabla_{\beta} t(y_t; \tau, \beta) = A' \text{vec}(\Theta^{-1}[\zeta_t \zeta_t' - \Theta] \Theta^{-1}) \quad (A.42) \]

\[ \nabla_{\beta} l_t(y_t; \tau, \beta) = \nabla_{\beta} \ln |\Phi_t| - \Theta^{-1} \zeta_t \nabla_{\beta} \zeta_t' \quad (A.43) \]

Note that \( \nabla_{\beta} \zeta_t(\beta) = -0.5 \zeta_t^2 h_{it}^2 \nabla_{\beta} h_{it}(\beta) \) and \( \nabla_{\beta} \ln |\Phi_t| \) is easy since \( \Phi_t \) is a diagonal matrix. What is left are expressions for \( \nabla_{\beta} h_{it}(\beta) \). These can be found in Section A.3.

A.6 Antithetic Variables

In this context using antithetic variables as a variance reduction technique is very important. See e.g. Chapter 8 of Ross (1990), among others, for a review of variance reduction techniques. In short the antithetic variates technique is based on the fact that if we wish to estimate by simulation \( \theta = \mathbb{E}[X] \) on basis of two generated variables \( X_1 \) and \( X_2 \). We are better off in case \( X_1 \) and \( X_2 \) are negatively correlated than independent, because

\[ \text{Var}(\frac{X_1 + X_2}{2}) = \frac{1}{4} [\text{Var}(X_1) + \text{Var}(X_2) + 2\text{Cov}(X_1, X_2)] \quad (A.44) \]

In the context of this thesis this means e.g. that for generating the stochastic volatility model in (2.13), we employ the generated series \( \{z_t, \eta_{t+1}\}_{t=1}^N \) together with \( \{-z_t, -\eta_{t+1}\}_{t=1}^N \). This will have the effect that two score generators from these series will be negatively correlated. Through experimentation we found the correlation between two antithetic score generators to be about \(-0.6\) for realistic parameter values of the SV model.
Appendix A. Description of Computer Programs

Let \( \Phi = (\theta, \phi) \). The density of \( \Phi \) is the Gaussian density. From this we deduce that \( H_j \) is given by, suppressing the dependency on : 

\[
\begin{align*}
H_j &= \frac{1}{\sqrt{2\pi\sigma_j^2}} e^{-\frac{(\theta - \mu_j)^2}{2\sigma_j^2}} \\
&= \frac{1}{\sqrt{2\pi\sigma_j^2}} e^{-\frac{(\theta - \mu_j)^2}{2\sigma_j^2}}
\end{align*}
\]

(8.1)

Section 7.2

A.A.

In our model of the universe consider a small solid angle \( \Delta \Omega \) and in the volume of \( \Delta V \) one may count an average of \( \mu \) particles. Let us consider a small sphere of radius \( a \) and in this sphere there is an average of \( \mu \) particles. 

(8.2)

A.A.

By the Poisson error decomposition we get the Poisson distribution for the likelihood of the full sample:

\[
\chi^2(\theta) = -2 \log L(\theta) = \sum \frac{(y_i - \mu_i)^2}{\mu_i}
\]

(8.3)

Let the score function be

\[
S_p(\mu_i) = \sum \frac{y_i - \mu_i}{\mu_i}
\]

(8.4)

Let \( F \) be \( n \times 1 \) and let \( a \) and \( b \) be vectors and matrices, respectively, defined by

\[
F(a) = a + b
\]

(8.5)