SNP: A Program for Nonparametric Time Series Analysis

Version 9.0

User's Guide [1]

A. Ronald Gallant
Duke University
Fuqua School of Business
Durham NC 27708-0120 USA

George Tauchen
Duke University
Department of Economics
Durham NC 27708-0097 USA

December 1990
Last Revised December 2010

[1]Research supported by the National Science Foundation. The code and this guide are available at http://econ.duke.edu/webfiles/arg/snp.

Abstract

SNP is a method of nonparametric time series analysis. The method employs an expansion in Hermite functions to approximate the conditional density of a multivariate process. An appealing feature of this expansion is that it is a nonlinear nonparametric model that directly nests the Gaussian VAR model, the semiparametric VAR model, the Gaussian ARCH model, the semiparametric ARCH model, the Gaussian GARCH model, and the semiparametric GARCH model. The unrestricted SNP expansion is more general than any of these models. The SNP model is fitted using conventional maximum likelihood together with a model selection strategy that determines the appropriate order of expansion.

The program has switches that allow direct computation of functionals of the fitted density such as conditional means, conditional variances, and points for plotting the density. Other switches generate simulated sample paths which can be used to compute nonlinear functionals of the density by Monte Carlo integration, notably the nonlinear analogs of the impulse-response mean and volatility profiles used in traditional VAR, ARCH, and GARCH analysis. Simulated sample paths can also be used to set bootstrapped sup-norm confidence bands on these and other functionals.

The purpose of this Guide is to review the underlying methodology and to walk the user through an application. Our intent is that the Guide be self contained and that little reference to the cited literature will be required to use the program and the SNP method.

What is new in Version 9.0 is implementation in C++ via a matrix class, generalizing the variance function to the BEKK, provisions for leverage and level effects in the variance function, allowing scalar, diagonal, or full matrices for each component of the variance function, removing all nesting restrictions for estimates started from a previous fit, and elimination of third party optimization software.

The code and this guide are available at http://econ.duke.edu/webfiles/arg/snp.

Contents

1 Introduction **1**

1.1 The SNP Method . . . 2

1.2 Refinements and Extensions . . . 3

2 Building and Running SNP **4**

2.1 Availability . . . 4

2.2 Building and Running SNP . . . 4

3 Estimation and Model Selection **5**

3.1 Estimation . . . 5

3.2 Model Selection . . . 16

4 Fitting SNP Models: A Walk Through Example **22**

4.1 Fitting Strategy . . . 22

4.2 Using the Program . . . 23

4.3 Running on a Parallel Machine . . . 40

5 Adapting the SNP Code **42**

5.1 Plots of the Conditional Variance Function . . . 42

6 References **49**

1 Introduction

For a stationary, multivariate time series, the one-step-ahead conditional density represents the process. The conditional density incorporates all information about various characteristics of the series including conditional heteroskedasticity, non-normality, time irreversibility, and other forms of nonlinearities. These properties are now widely considered to be important features of many time series processes. Since the conditional density completely characterizes a process, it is thus naturally viewed as the fundamental statistical object of interest.

SNP is nonparametric method, based on an expansion in Hermite functions, for estimation of the conditional density. The method was first proposed by Gallant and Tauchen (1989) in connection with an asset pricing application. Estimation of SNP models entails using a standard maximum likelihood procedure together with a model selection strategy that determines the appropriate degree of the expansion. Under reasonable regularity conditions, the estimator is consistent.

The method has undergone a number of refinements and extensions, all of which are available as features of a C++ program that is in the public domain. The tasks of model fitting and specification testing have been greatly simplified, and, to a large extent, automated within the program. For a given data set, these tasks are now no more demanding in terms of total computational effort than those of typical nonlinear methods.

The program also incorporates many additional features related to prediction, residual analysis, plotting, and simulation. These capabilities are designed to facilitate subsequent analysis and interpretation. Predicted values and residuals, for instance, are of central importance for undertaking diagnostic analysis and calculating measures of fit. Density plots are useful for visualizing key characteristics of the process such as asymmetries and heavy tails. Simulation has numerous potential applications. One is Monte Carlo analysis, in particular setting bootstrapped confidence intervals, as described in Gallant, Rossi, and Tauchen (1992). Another is nonlinear error shock analysis, described in Gallant, Rossi, and Tauchen (1993), which develops the nonlinear analog of conventional error shock analysis for linear VAR models. An important new application is reprojection, which is a form of nonlinear Kalman filtering, that can be used to forecast the unobservables of nonlinear latent variables models; the leading example is forecasting the volatility process of continuous-time stochastic volatility models (Gallant and Tauchen, 1998).

This guide provides a general overview of the method along with detailed instructions for using the computer program. The topics covered include model formulation, estimation, and specification testing. Also included is a worked example using a realistic data set. The example shows how to tailor the code for a specific application and it takes the reader through all aspects of model fitting, including pointing out the pitfalls. The example also shows how to utilize each of the extended features of the program related to prediction, residual analysis, plotting, and simulation.

1.1 The SNP Method

The method is termed SNP, which stands for **S**emi**N**on**P**arametric, to suggest that it lies halfway between parametric and nonparametric procedures. The leading term of the series expansion is an established parametric model known to give a reasonable approximation to the process; higher order terms capture departures from that model. With this structure, the SNP approach does not suffer from the curse of dimensionality to the same extent as kernels and splines. In regions where data are sparse, the leading term helps to fill in smoothly between data points. Where data are plentiful, the higher order terms accommodate deviations from the leading term and fits are comparable to the kernel estimates proposed by Robinson (1983).

The theoretical foundation of the method is the Hermite series expansion, which for time series data is particularly attractive on the basis of both modeling and computational considerations. In terms of modeling, the Gaussian component of the Hermite expansion makes it easy to subsume into the leading term familiar time series models, including VAR, ARCH, and GARCH models (Engle, 1982; Bollerslev, 1986). These models are generally considered to give excellent first approximations in a wide variety of applications. In terms of computation, a Hermite density is easy to evaluate and differentiate. Also, its moments are easy to evaluate because they correspond to higher moments of the normal, which can be computed using standard recursions. Finally, a Hermite density turns out to be very practicable to sample from, which facilitates simulation.

1.2 Refinements and Extensions

A sequence of empirical applications, beginning with Gallant and Tauchen (1989), has stimulated extensions and refinements of the SNP methodology. The original asset-pricing application was a limited information maximum likelihood situation where both the likelihood (which is the product of one-step-ahead conditional densities) and the Euler conditions (structural equations) had to have nonparametric properties and be mathematically convenient. This naturally lead to a series expansion type of approach so that standard algorithms could be used to optimize the likelihood subject to the Euler conditions.

Extensions to better adapt the method to markedly conditionally heteroskedastic processes such as exchange rate data were developed by Gallant, Hsieh, and Tauchen (1991). Further extensions to robustify the methodology against extremely heavy tailed processes such as real interest rate data were reported Gallant, Hansen, and Tauchen (1990). Processes such as bivariate stock price and volume series can require a high degree Hermite polynomial to fit them which generates a plethora of irrelevant interactions. Gallant, Rossi, and Tauchen (1992) described filters to remove them. Efficient method of moments (EMM), applications (Gallant and Tauchen, 1996, 2004; Ahn, Dittmar, and Gallant 2002, Ahn, Dittmar, Gallant, and Gao, 2003; Chernov, Gallant, Ghysels, and Tauchen, 2003) lead to the development of further robustifications and the addition of a GARCH leading term (Gallant, Hsieh, and Tauchen, 1997; Gallant and Long 1997; Gallant and Tauchen, 1997; Gallant and Tauchen, 1998). The SNP density is also useful for synthesizing a likelihood in Bayesian applications (Gallant and McCulloch, 2009; Aldrich and Gallant, 2011).

Our description of the SNP nonlinear time series methodology in Section 3 incorporates the above refinements.

2 Building and Running SNP

2.1 Availability

The code and this guide are available at http://econ.duke.edu/webfiles/arg/snp. It has run under the Sun, GNU, and Microsoft C++ compilers.

2.2 Building and Running SNP

Download `snp.tar` from `http://econ.duke.edu/webfiles/arg/snp`. On a Unix machine use `tar -xf snp.tar` to expand the tar archive into a directory that will be named `snp`. On a Windows machine use unzip; i.e., Windows recognizes a Unix tar archive as a zip file. The distribution has the following directory structure:

```
lib
   libscl
   libsnp
snpman
snprun
snpsrc
svfx
```

Often one changes the name `snp` of the parent directory to a name that represents the project one is working on. For the example in the manual `snp` was renamed `sv` as short for stochastic volatility.

First the two libraries libscl and libsnp must be built, in that order. On a Unix machine change directory to `lib/libscl/gpp` and type `make`. For Microsoft Windows, a batch file supplied by Microsoft with their compiler must be executed first. The following is an example:

```
C:"\Program Files\Microsoft Visual Studio .NET\Vc7\bin\"vcvars32.bat
```

The exact syntax will depend on where the Microsoft C++ compiler is installed. Change directory to `lib\libscl\ms` and type `nmake`. Building libsnp and libsnp is similar.

To run the SNP example that comes with the distribution on a Unix machine within the directory `snprun` copy `makefile.gpp` to `makefile`, type `make` and then ./snp. For Microsoft copy `makefile.ms` to `makefile`, type `C:"\...\"vcvars32.bat, nmake`, then `snp`, etc.

3 Estimation and Model Selection

In this section, we describe an estimation strategy for nonlinear time series analysis proposed by Gallant and Tauchen (1989) and its extensions. These extensions are: an ARCH leading term, which better adapts the method to markedly conditionally heteroskedastic processes, proposed by Gallant, Hsieh, and Tauchen (1991); a spline transformation, which imposes stationarity on fits to extremely persistent data such as interest rates (Gallant and Tauchen, 1998); and, filters, which remove the high order interactions in fits to multiple time series, proposed by Gallant, Rossi, and Tauchen (1992). New to this release is a BEKK variance function, which is a generalization of GARCH and which has not yet been tested in applications.

The derivation of the SNP model that we present here provides a fundamental understanding of the model so that one appreciate the implications of the tuning parameters. It does not provide the mathematical connection with the results of Gallant and Nychka (1987) that is required for theoretical work. For this, see Gallant, Hsieh, and Tauchen (1991). See Gallant and Tauchen (1992) for a description of the simulation algorithm.

3.1 Estimation

As stated above, the SNP method is based on the notion that a Hermite expansion can be used as a general purpose approximation to a density function. Letting z denote an

$M-$vector, we can write the Hermite density as $h(z) \propto [\mathcal{P}(z)]^2\phi(z)$ where $\mathcal{P}(z)$ denotes a multivariate polynomial of degree K_z and $\phi(z)$ denotes the density function of the (multivariate) Gaussian distribution with mean zero and the identity as its variance-covariance matrix. Denote the coefficients of $\mathcal{P}(z)$ by a, which is a vector whose length depends on K_z and M. When we wish to call attention to the coefficients, we write $\mathcal{P}(z|a)$.

The constant of proportionality is $1/\int[\mathcal{P}(s)]^2\phi(s)ds$ which makes $h(z)$ integrate to one. As seen from the expression that results, namely

$$h(z) = \frac{[\mathcal{P}(z)]^2\phi(z)}{\int[\mathcal{P}(s)]^2\phi(s)ds},$$

we are effectively expanding the square root of the density in Hermite functions of the form $\mathcal{P}(z)\sqrt{\phi(z)}$. Because the square root of a density is always square integrable over $(-\infty, \infty)$, and because the Hermite functions of the form $\mathcal{P}(z)\sqrt{\phi(z)}$ are dense for the collection of square integrable functions defined on $(-\infty, \infty)$, (Fenton and Gallant, 1996) every density has such an expansion. Because $[\mathcal{P}(z)]^2/\int[\mathcal{P}(s)]^2\phi(s)ds$ is a homogeneous function of the coefficients of the polynomial $\mathcal{P}(z)$, the coefficients can only be determined to within a scalar multiple. To achieve a unique representation, the constant term of the polynomial part is put to one.

Customarily the Hermite density is written with its terms orthogonalized, and the code is written in the orthogonalized form for numerical efficiency. But reflecting that here would lead to cluttered notation and add nothing to the ideas. The code contains methods that convert back and forth between orthogonalized and regular forms so the representation described in this User's Guide is actually available.

A change of variables using the location-scale transformation $y = Rz + \mu$, where R is an upper triangular matrix and μ is an $M-$vector, gives

$$f(y|\theta) \propto \{\mathcal{P}[R^{-1}(y-\mu)]\}^2\{\phi[R^{-1}(y-\mu)]/|\det(R)|\}$$

The constant of proportionality is the same as above, $1/\int[\mathcal{P}(s)]^2\phi(s)ds$. Because $\{\phi[R^{-1}(y-\mu)]/|\det(R)|\}$ is the density function of the $M-$dimensional, multivariate, Gaussian distribution with mean μ and variance-covariance matrix $\Sigma = RR'$, and because the leading term of the polynomial part is one, the leading term of the entire expansion is proportional to the multivariate, Gaussian density function. Denote the Gaussian density of dimension M with mean vector μ and variance-covariance matrix Σ by $n_M(y|\mu, \Sigma)$ and write

$$f(y|\theta) \propto [\mathcal{P}(z)]^2 n_M(y|\mu, \Sigma)$$

where $z = R^{-1}(y - \mu)$ for the density above.

When K_z is put to zero, one gets $f(y|\theta) = n_M(y|\mu, \Sigma)$ exactly. When K_z is positive, one gets a Gaussian density whose shape is modified due to multiplication by a polynomial in $z = R^{-1}(y - \mu)$. The shape modifications thus achieved are rich enough to accurately approximate densities from a large class that includes densities with fat, t-like tails, densities with tails that are thinner than Gaussian, and skewed densities (Gallant and Nychka, 1987).

The parameters θ of $f(y|\theta)$ are made up of the coefficients a of the polynomial $\mathcal{P}(z)$ plus μ and R and are estimated by maximum likelihood. Equivalent to maximum likelihood but more stable numerically is to estimate θ in a sample of size n by minimizing $s_n(\theta) = (-1/n)\sum_{t=1}^{n} \log[f(y_t|\theta)]$. As mentioned above, if the number of parameters p_θ grows with the sample size n, the true density and various features of it such as derivatives and moments are estimated consistently (Gallant and Nychka, 1987).

This basic approach can be adapted to the estimation of the conditional density of a multiple time series $\{y_t\}$ that has a Markovian structure. Here, the term Markovian structure is taken to mean that the conditional density of the $M-$vector y_t given the entire past y_{t-1}, y_{t-2}, ... depends only on L lags from the past. For convenience in this discussion, we will presume that the data are from a process with a Markovian structure, but one should be aware that if L is sufficiently large, then non-Markovian data can be well approximated by an SNP density (Gallant and Long, 1996). For notational convenience, we collect these lags together in a single vector denoted as x_{t-1}, which is $M \cdot L$, viz.

$$x_{t-1} = (y_{t-1}, y_{t-2}, .., y_{t-L}),$$

where L exceeds all lags in the following discussion.

To approximate the conditional density of $\{y_t\}$ using the ideas above, begin with a sequence of innovations $\{z_t\}$. First consider the case of homogeneous innovations; that is, the distribution of z_t does not depend on x_{t-1}. Then, as above, the density of z_t can be approximated by $h(z) \propto [\mathcal{P}(z)]^2\phi(z)$ where $\mathcal{P}(z)$ is a polynomial of degree K_z. Follow with the location-scale transformation $y_t = Rz_t + \mu_x$ where μ_x is a linear function that depends on L_u lags

$$\mu_x = b_0 + Bx_{t-1}.$$

(If $L_u < L$, then some elements of B are zero.) The density that results is

$$f(y|x,\theta) \propto [\mathcal{P}(z)]^2 n_M(y|\mu_x, \Sigma)$$

where $z = R^{-1}(y - \mu_x)$. The constant of proportionality is as above, $1/\int[\mathcal{P}(s)]^2\phi(s)ds$. The leading term of the expansion is $n_M(y|\mu_x, \Sigma)$ which is a Gaussian vector autoregression or Gaussian VAR.

When K_z is put to zero, one gets $n_M(y|\mu_x, \Sigma)$ exactly. When K_z is positive, one gets a semiparametric VAR density that can approximate well over a large class of densities whose first moment depends linearly on x and whose shape is constant with respect to variation in x.

To approximate conditionally heterogeneous processes, proceed as above but let each coefficient of the polynomial $\mathcal{P}(z)$ be a polynomial of degree K_x in x. A polynomial in z of degree K_z whose coefficients are polynomials of degree K_x in x is, of course, a polynomial in (z, x) of degree $K_z + K_x$ (with some of the coefficients put to zero). Denote this polynomial by $\mathcal{P}(z, x)$. Denote the mapping from x to the coefficients a of $\mathcal{P}(z)$ such that $\mathcal{P}(z|a_x) = \mathcal{P}(z, x)$ by a_x and the number of lags on which it depends by L_p. The form of the density with this modification is

$$f(y|x,\theta) \propto [\mathcal{P}(z,x)]^2 n_M(y|\mu_x, \Sigma)$$

where $z = R^{-1}(y - \mu_x)$. The constant of proportionality is $1/\int[\mathcal{P}(s,x)]^2\phi(s)ds$. When K_x is zero, the density reverts to the density above. When K_x is positive, the shape of the density will depend upon x. Thus, all moments can depend upon x and the density can, in principal, approximate any form of conditional heterogeneity. (Gallant and Tauchen, 1989; Gallant, Hsieh, and Tauchen, 1989).

Large values of M can generate a large number of interactions (cross product terms) for even modest settings of degree K_z; similarly, for $M \cdot L_p$ and K_x. Accordingly, we introduce two additional tuning parameters, I_z and I_x, to control these high order interactions. $I_z = 0$

or 1 means no interactions, $I_z = 2$ means interactions of degree 2 are included, etc.; similarly for I_x. Note that this convention differs from the early SNP literature where I_x denotes exclusion of interactions rather than inclusion.

Above, all coefficients $a = a_x$ of the polynomial $\mathcal{P}(z|a)$ are polynomials of degree K_x in x. For small K_z and I_z this is reasonable. When K_z and I_z are large, two additional tuning parameters maxK_z and maxI_z can be set to eliminate the dependence on x of coefficients of degree higher than these values.

In practice, especially in applications to data from financial markets, the second moment can exhibit marked dependence upon x. In an attempt to track the second moment, K_x can get quite large. To keep K_x small when data are markedly conditionally heteroskedastic, the leading term $n_M(y|\mu_x, \Sigma)$ of the expansion can be put to a Gaussian GARCH rather than a Gaussian VAR. We use a modified BEKK expression as described in Engle and Kroner (1995); the modifications are to add leverage and level effects. This is the most important difference between the earlier Fortran implementations of SNP, which used R-GARCH, and the C++ version. The form is

$$
\begin{aligned}
\Sigma_{x_{t-1}} = \; & R_0 R_0' \\
& + \sum_{i=1}^{L_g} Q_i \Sigma_{x_{t-1-i}} Q_i' \\
& + \sum_{i=1}^{L_r} P_i (y_{t-i} - \mu_{x_{t-1-i}})(y_{t-i} - \mu_{x_{t-1-i}})' P_i' \\
& + \sum_{i=1}^{L_v} \max[0, V_i(y_{t-i} - \mu_{x_{t-1-i}})] \max[0, V_i(y_{t-i} - \mu_{x_{t-1-i}})]' \\
& + \sum_{i=1}^{L_w} W_i \, x_{(1),t-i} x_{(1),t-i}' W_i'.
\end{aligned}
$$

Above, R_0 is an upper triangular matrix. The matrices P_i, Q_i, V_i, and W_i can be scalar, diagonal, or full M by M matrices. Which is controlled by setting switches Ptype, Qtype, Vtype, and Wtype to one of the characters 's', 'd', or 'f'. The notation $x_{(1),t-i}$ indicates that only the first column of x_{t-i} enters the computation. The $\max(0, x)$ function is applied elementwise. Because $\Sigma_{x_{t-1}}$ must be differentiable with respect to the parameters of $\mu_{x_{t-2-i}}$, the $\max(0, x)$ function actually applied is a twice continuously differentiable cubic spline approximation that agrees with the $\max(0, x)$ function except over the interval (0, 0.1) over which it lies slightly above the $\max(0, x)$ function. It is defined in header smooth.h. Often $\Sigma_{x_{t-1}}$ in factored form is required, i.e. $\Sigma_{x_{t-1}} = R_{x_{t-1}} R'_{x_{t-1}}$. The factorization and its derivative are computed by the function factor. To reduce clutter, we shall usually write Σ_x and R_x for $\Sigma_{x_{t-1}}$ and $R_{x_{t-1}}$.

GARCH models are often difficult to fit, and the SNP version of GARCH is no exception. Our suggestion is to restrict attention to models with $L_g = 1$ and $L_r = 1$ or $L_g = 2$ and $L_r = 1$, as is done in most of the applied GARCH literature. With multivariate fits $(M > 1)$, restricting P to diagonal and Q to scalar enhances stability. When both of these restrictions are imposed, SNP-GARCH is more stable than most GARCH routines. Even if an unrestricted multivariate fit is sought, it is a still good idea to fit first with the restrictions imposed and to relax them later to get an unconstrained fit rather than trying to compute an unconstrained fit directly. Similar considerations apply to V and W.

Note that when $L_g > 0$, the SNP model is not Markovian and that one must know both x_{t-1} and $R_{x_{t-2}}$ through $R_{x_{t-2-L_g}}$ to move forward to the value for y_t. Thus, x_{t-1} and $R_{x_{t-2}}$ through $R_{x_{t-2-L_g}}$ represent the state of the system at time $t-1$ and must be retained or recomputed in order to evaluate the SNP conditional density of y_t or to iterate the SNP model forward by simulation. If one wants to compute the derivatives of the SNP density with respect to model parameters, one must retain or recompute the derivatives of $R_{x_{t-2}}$ through $R_{x_{t-2-Lg}}$ with respect to model parameters as well. Previous versions of SNP used a retention strategy. Version 9.0 uses a recomputation strategy and the code has switches to facilitate this for the purpose of computing conditional means, conditional variances, plots, and simulations.

In the log likelihood computation, the state is initialized at R_0 and iterated forward the number of steps specified by the control parameter `drop` before the summands $\log f(y_t|x_{t-1}, \theta)$ are accumulated. Similarly for computations described in the paragraph above.

With R_x specified as either an ARCH or a GARCH as above, the form of the conditional density becomes

$$f(y|x, \theta) \propto [\mathcal{P}(z, x)]^2 n_M(y|\mu_x, \Sigma_x)$$

where $z = R_x^{-1}(y - \mu_x)$. The constant of proportionality is $1/\int[\mathcal{P}(s, x)]^2 \phi(s) ds$. The leading term $n_M(y|\mu_x, \Sigma_x)$ is Gaussian ARCH if $L_g = 0$ and $L_r > 0$ and Gaussian GARCH if both $L_g > 0$ and $L_r > 0$ (leaving aside the implications of L_v and L_w).

As described above each of the functions a_x, μ_x, and R_x is permitted its own lag specification. Let us review the conventions. The number of lags in the function a_x for which $\mathcal{P}(z,x) = \mathcal{P}(z|a_x)$ is L_p. The number of lags in the location function μ_x is L_u, and the number of lags in the scale function R_x is controlled by the upper indexes of summation L_g, L_r, L_v, and L_w in the equation defining Σ_x above. The number of lags actually required to compute R_x is $\max(L_r + L_u, L_v + L_u, L_w)$ due to the dependence of the variance function on the mean function. The vector x has conceptual length (as an array in memory) $M \cdot L$, where $L = \max(L_r + L_u, L_v + L_u, L_w, L_p)$. However, in the code, each object handles its own recursions using whatever data structure is convenient so an x of this length does not actually appear anywhere.

The length of a in $\mathcal{P}(z|a)$ depends on K_z and I_z. (The length of a also depends implicitly on M as do all vectors and matrices in the remainder of this paragraph.) In the code, the object (i.e. class) snpden defines a. The function a_x has parameters $[a_0|A]$ whose length (as an array in memory) is controlled by $\max K_z$, $\max I_z$, I_x, and K_x; a_0 is the subset of a that is does not depend on x (when $K_z < \max K_z$ or $I_z < \max I_z$) and A controls the mapping from x to the subset of a that does depend on x. In the code, the object afunc defines the mapping a_x from x to a and hence $[a_0|A]$. The parameters of the location function are $[b_0|B]$ whose length is controlled by L_u and a switch `iecpt` that controls whether an the intercept b_0 is present or not. In the code, ufunc defines the mapping μ_x from x to μ and hence $[b_0|B]$. The parameters of the variance function R_x that maps x to R are $[R_0|Q_1 \cdots Q_q|P_1 \cdots P_p|V_1 \cdots V_q|W_1 \cdots W_q]$. The total length is controlled by L_g, L_r, L_v, and L_w and the switches Qtype, Ptype, Vtype, and Wtype that determine whether Q, P, V, and W are scalars, diagonal matrices, or full matrices, respectively. In the code rfunc defines the mapping R_x from x to R and hence $[R_0|Q_1 \cdots Q_q|P_1 \cdots P_p|V_1 \cdots V_q|W_1 \cdots W_q]$. With the exception of A, which for technical reasons contains the coefficient of the constant term of $\mathcal{P}(z)$ as its first element, and R_0, all vectors and matrices above can be null.

These are arranged in the order

$$\theta = \text{vec}\Big[a_0|A|b_0|B|R_0|P_1 \cdots P_p|Q_1 \cdots Q_q|V_1 \cdots V_q|W_1 \cdots W_q\Big]$$

Parameter setting	Characterization of $\{y_t\}$
$L_u = 0,\ L_g = 0,\ L_r = 0,\ L_p \geq 0,\ K_z = 0,\ K_x = 0$	iid Gaussian
$L_u > 0,\ L_g = 0,\ L_r = 0,\ L_p \geq 0,\ K_z = 0,\ K_x = 0$	Gaussian VAR
$L_u > 0,\ L_g = 0,\ L_r = 0,\ L_p \geq 0,\ K_z > 0,\ K_x = 0$	semiparametric VAR
$L_u \geq 0,\ L_g = 0,\ L_r > 0,\ L_p \geq 0,\ K_z = 0,\ K_x = 0$	Gaussian ARCH
$L_u \geq 0,\ L_g = 0,\ L_r > 0,\ L_p \geq 0,\ K_z > 0,\ K_x = 0$	semiparametric ARCH
$L_u \geq 0,\ L_g > 0,\ L_r > 0,\ L_p \geq 0,\ K_z = 0,\ K_x = 0$	Gaussian GARCH
$L_u \geq 0,\ L_g > 0,\ L_r > 0,\ L_p \geq 0,\ K_z > 0,\ K_x = 0$	semiparametric GARCH
$L_u \geq 0,\ L_g \geq 0,\ L_r \geq 0,\ L_p > 0,\ K_z > 0,\ K_x > 0$	nonlinear nonparametric

Table 1. Restrictions Implied by Settings of the Tuning Parameters. The parameters L_v and L_w are set to zero. The parameters I_z, maxI_z, and I_x have no effect when $M = 1$. The parameter max$K_z = K_z$ in each instance that $K_x > 0$.

internally to the program. The code permits some of them to be fixed in the optimization and some to be active. The subset that is active is denoted by ρ and the number that are active (length of ρ) by p_ρ. Because the constant term is always fixed, ρ will always have dimension at least one less than θ. Below, we do not carefully distinguish between θ and ρ and usually use θ to mean either. When the distinction is important we shall note it.

The parameters are estimated by minimizing the active elements of θ in

$$s_n(\theta) = -(1/n) \sum_{t=1}^{n} \log[f(y_t|x_{t-1}, \theta)].$$

Putting certain of the tuning parameters to zero implies sharp restrictions on the process $\{y_t\}$, the more interesting of which are displayed in Table 1. We call particular attention to the case $L_u, L_r, K_z > 0$ and $L_g, K_x = 0$ because it generates the semiparametric ARCH class of densities proposed by Engle and Gonzales-Rivera (1991).

To improve the stability of computations, the observations $\{y_t\}_{t=1}^n$ are centered and scaled to have mean zero and identity variance-covariance matrix. The centering and scaling is accomplished by (1) computing estimates of the unconditional mean and variance

$$\bar{y} = (1/n) \sum_{t=1}^{n} \tilde{y}_t$$

$$S = (1/n) \sum_{t=1}^{n} (\tilde{y}_t - \bar{y})(\tilde{y}_t - \bar{y})'$$

where $\tilde{y}_t$ denotes the raw data, and (2) applying the methods above to

$$y_t = S^{-1/2}(\tilde{y}_t - \bar{y})$$

where $S^{-1/2}$ denotes the factored inverse of S. That is, just replace the raw data $\{\tilde{y}_t\}$ by the centered and scaled data $\{y_t\}$ throughout. Because of the location-scale transformation $y = Rz + \mu$, the consistency results cited above are not affected by the transformation from $\tilde{y}_t$ to y_t. These centering and scaling transformations are internal to the program and are transparent to the user.

To aid interpretation of results with multivariate data, one may want to make $S^{-1/2}$ above diagonal by setting the off-diagonal elements of S to zero before factorization by setting the switch diag to 1.

On the other hand, because the factorization is done using the singular value decomposition, the variables y_t computed with diag$\,=0$ actually are interpretable: they are the normalized principal componants. That is, of all variables of the form $a'(\tilde{y}_t - \bar{y})$, y_{1t} had the largest sample variance prior to being rescaled to have variance one; of all such variables that are orthogonal to y_{1t}, y_{2t} had the largest variance; of all such variables orthogonal to both y_{1t} and y_{2t}, y_{3t} had the largest; and so on.

Time series data often contain extreme or outlying observations, particularly data from financial markets. This is not a particular problem when the extreme value is considered as a y_t because it just fattens the tails of the estimated conditional density. However, once it becomes a lag and passes into x_{t-1}, the optimization algorithm can use an extreme value in x_{t-1} to fit an element of y_t nearly exactly thereby reducing the corresponding conditional variance to near zero and inflating the likelihood. This problem is endemic to procedures that adjust variance on the basis of observed explanatory variables.

One can compensate for this effect by an additional transformation

$$\hat{x}_i = \begin{cases} \frac{1}{2}\left\{x_i + \frac{4}{\pi}\arctan\left[\frac{\pi}{4}(x_i + \sigma_{\mathrm{tr}})\right] - \sigma_{\mathrm{tr}}\right\} & -\infty < x_i < -\sigma_{\mathrm{tr}} \\ x_i & -\sigma_{\mathrm{tr}} < x_i < \sigma_{\mathrm{tr}} \\ \frac{1}{2}\left\{x_i + \frac{4}{\pi}\arctan\left[\frac{\pi}{4}(x_i - \sigma_{\mathrm{tr}})\right] + \sigma_{\mathrm{tr}}\right\} & \sigma_{\mathrm{tr}} < x_i < \infty. \end{cases}$$

where x_i denotes an element of x_{t-1}. This is a trigonometric spline transformation that has a no effect on values of x_i between $-\sigma_{\mathrm{tr}}$ and σ_{tr} but progressively compresses values that

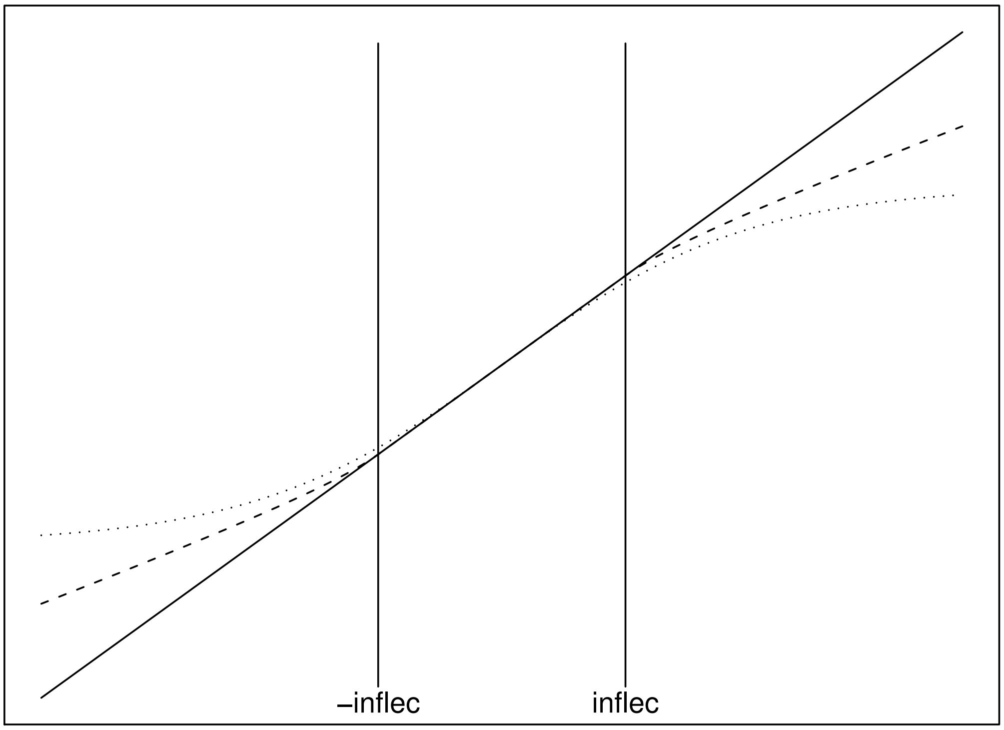

Figure 1. The squashers. The dashed line shows the trigonometric spline transformation. The dotted line shows the logistic transformation. The solid line is a 45 degree line, which represents no transformation. The two vertical lines are at $x = -\sigma_{\mathrm{tr}}$ and $x = \sigma_{\mathrm{tr}}$.

exceed $\pm\sigma_{\mathrm{tr}}$; see Figure 1. A more extreme squasher is the logistic transformation

$$\hat{x}_i = (4\sigma_{\mathrm{tr}})\,\frac{\exp(x_i/\sigma_{\mathrm{tr}})}{1+\exp(x_i/\sigma_{\mathrm{tr}})} - 2\sigma_{\mathrm{tr}}$$

It has negligible effect on values of x_i between $-\sigma_{\mathrm{tr}}$ and σ_{tr} but progressively compresses values that exceed $\pm\sigma_{\mathrm{tr}}$ so they are bounded by $\pm 2\sigma_{\mathrm{tr}}$; see Figure 1.

A switch, squash, allows a user to choose no transformation, the spline transformation, or the logistic transformation; σ_{tr} is user selectable. We recommend $\sigma_{\mathrm{tr}} = 2$ and think that the spline is the better choice when squashing is necessary. Squashing is roughly equivalent to using a variable bandwidth in kernel density estimation. Because it affects only $y_{t-1}, \ldots, y_{t-L}$ and not y_t, the asymptotic properties of SNP estimators discussed above are unaltered.

These transforms are applied to the x_{t-1} that enter $\mathcal{P}(z, x)$, μ_x, and Σ_x. In addition, the residuals $y_{t-1-i} - \mu_{x_{t-2-i}}$ that enter the ARCH terms and leverage effect terms in the expression for Σ_x are transformed. The dictum that the sum of the coefficients (sum of squares actually because we use the BEKK form) must be less than one no longer holds under transformation. For the logistic, only the autoregressive coefficients enter the sum because the forcing variables in the moving average part have bounded expectation. Although we have not verified the mathematics, it seems obvious from the expression for the spline transform that it would suffice that the sum of squares of the coefficients be less than 2 for the spline.

For data from financial markets, experience suggests that a long simulation from a fitted model will have unconditional variance and kurtosis much larger than the variance and kurtosis of the sample. When the spline transform is imposed, this anomaly is attenuated. Estimated coefficients and the value of s_n are not much affected. Thus, it seems that if simulation from the fitted SNP model is an important component of the statistical analysis, then the spline transform should definitely be imposed.

Note the order in which the transformations are applied

$$\begin{array}{ccccccccc}
\tilde{y}_t & \to & y_t & \to & x_{t-1} & \to & \hat{x}_{t-1} & \to & \mu_x,\ R_x \\
\text{raw} & \to & \text{centered,} & \to & \text{lagged} & \to & \text{spline} & \to & \text{location and} \\
\text{data} & & \text{scaled data} & & \text{data} & & \text{data} & & \text{scale transformation}
\end{array}$$

The code is written so that these transformations are transparent to the user. All input and output is in the units of the raw data $\tilde{y}_t$.

The SNP score is often used in connection with the EMM estimation method (Gallant and Tauchen, 1996), which is a simulation estimator that involves simulating from a model and averaging the SNP score over that simulation. Some of the simulated values fed to SNP by EMM can be much different than the data from which the SNP parameters were estimated which gives rise to a nuisance: The SNP density evaluated at some of these values can be smaller than the smallest value that the log function can evaluate which tends to destabilize EMM optimizations. The fix is to replace the SNP density by

$$f^*(y|x,\theta) = \frac{\{\mathcal{P}^2\,[\,R_x^{-1}(y-\mu_x),\ x\,] + \epsilon_0\}\, n_M(y|\mu_x, \Sigma_x)}{\int [\mathcal{P}(s,x)]^2 \phi(s)\, ds + \epsilon_0}$$

where the user sets the value $\epsilon_0 > 0$; effectively, $\mathcal{P}^2(z,x) + \epsilon_0$ replaces $\mathcal{P}^2(z,x)$ throughout the code and all computations are affected: estimation, moments, plots, and simulations.

L_u	L_g	L_r	L_p	K_z	I_z	K_x	I_x	p_ρ	s_n	BIC	HQ	AIC
1	0	0	1	0	0	0	0	3	1.39760	1.40969	1.40445	1.40119
2	0	0	1	0	0	0	0	4	1.39699	1.41312	1.40613	1.40179
3	0	0	1	0	0	0	0	5	1.39689	1.41705	1.40832	1.40289
4	0	0	1	0	0	0	0	6	1.39392	1.42214	1.40991	1.40231
1	0	1	1	0	0	0	0	4	1.36600	1.38213	1.37514	1.37080
1	0	2	1	0	0	0	0	5	1.35263	1.37280	1.36406	1.35863
1	0	3	1	0	0	0	0	6	1.33329	1.35748	1.34700	1.34048
1	0	4	1	0	0	0	0	7	1.33050	1.35872	1.34649	1.33889
1	0	3	1	4	0	0	0	10	1.30107	1.34139	1.32392	1.31306
1	0	3	1	5	0	0	0	11	1.30107	1.34542	1.32621	1.31426
1	0	3	1	6	0	0	0	12	1.29634	1.34473	1.32377	1.31073
1	0	3	1	4	0	1	0	15	1.29407	1.35456	1.32835	1.31205
1	0	3	1	4	0	2	0	20	1.28966	1.37031	1.33537	1.31364
1	1	1	1	0	0	0	0	5	1.32781	1.34797	1.33923	1.33380
1	1	1	1	4	0	0	0	9	1.29425	1.33054	1.31482	1.30504
1	1	1	1	4	0	1	0	14	1.29109	1.33475	1.32309	1.30788

Table 2. Optimized Likelihood and Model Selection Criteria.

We have found $\epsilon_0 = 0.001$ to be a reasonable value.

3.2 Model Selection

Some conventional model selection criteria are the Schwarz criterion, the Hannan-Quinn criterion, and the Akaike information criterion. The Schwarz Bayes information criterion (Schwarz, 1978) is computed as

$$\text{BIC} = s_n(\hat{\theta}) + (1/2)(p_\rho/n)\log(n)$$

with small values of the criterion preferred. The criterion rewards good fits as represented by small $s_n(\hat{\theta})$ but uses the term $(1/2)(p_\rho/n)\log(n)$ to penalize good fits gotten by means of excessively rich parameterizations. The criterion is conservative in that it selects sparser parameterizations than the Akaike AIC information criterion (Akaike, 1969), which uses the penalty term p_ρ/n in place of $(1/2)(p_\rho/n)\log(n)$. Schwarz is also conservative in the sense that it is at the high end of the permissible range of penalty terms in certain model selection settings (Potscher, 1989). Between these two extremes is the Hannan and Quinn (Hannan, 1987) criterion

$$\text{HQ} = s_n(\hat{\theta}) + (p_\rho/n)\log[\log(n)].$$

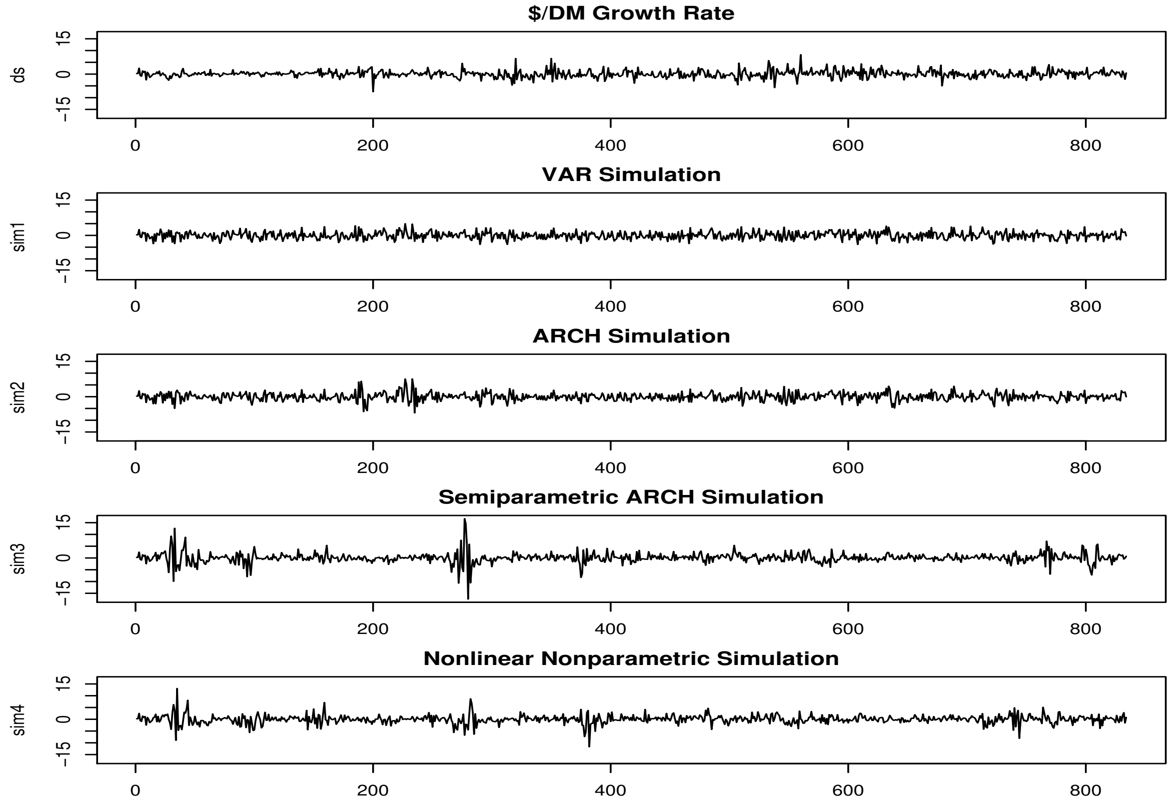

Figure 2. Changes in Weekly \$/DM Spot Exchange Rates, SNP-ARCH. The first panel is a plot of the data, which are each Friday's quote over the years 1975 to 1990 expressed as percentage change from the previous week. The second panel is a simulation from an SNP fit with $(L_u, L_g, L_r, L_p, K_z, I_z, K_x, I_x) = (1, 0, 0, 1, 0, 0, 0, 0)$; the third with (1,0,3,1,0,0,0,0), the fourth with (1,0,3,1,4,0,0,0), and the fifth with (1,0,3,1,4,0,1,0). The parameters L_v and L_w are set to zero. The parameters I_z, maxI_z, and I_x have no effect when $M = 1$. The parameter max$K_z = K_z$ in each instance that $K_x > 0$.

Our suggestion is to use the Schwarz BIC criterion to move along an upward expansion path until an adequate model is determined. BIC seems to do a good job of finding abrupt drops in integrated squared error which is the point at which one would like to truncate in EMM applications (Gallant and Tauchen, 1999; Coppejans and Gallant, 2002).

We can illustrate using data on the week to week percentage change in the US dollar to German mark exchange rate for the years 1975 through 1990 from Bansal, Gallant, Hussey, and Tauchen (1995), which are plotted in the upper panel of Figure 2. Computed BIC, AIC, and HQ criteria for several SNP specifications are shown in Table 2. Using the first block of the table, one first increases L_u to determine the Schwarz preferred VAR fit, which

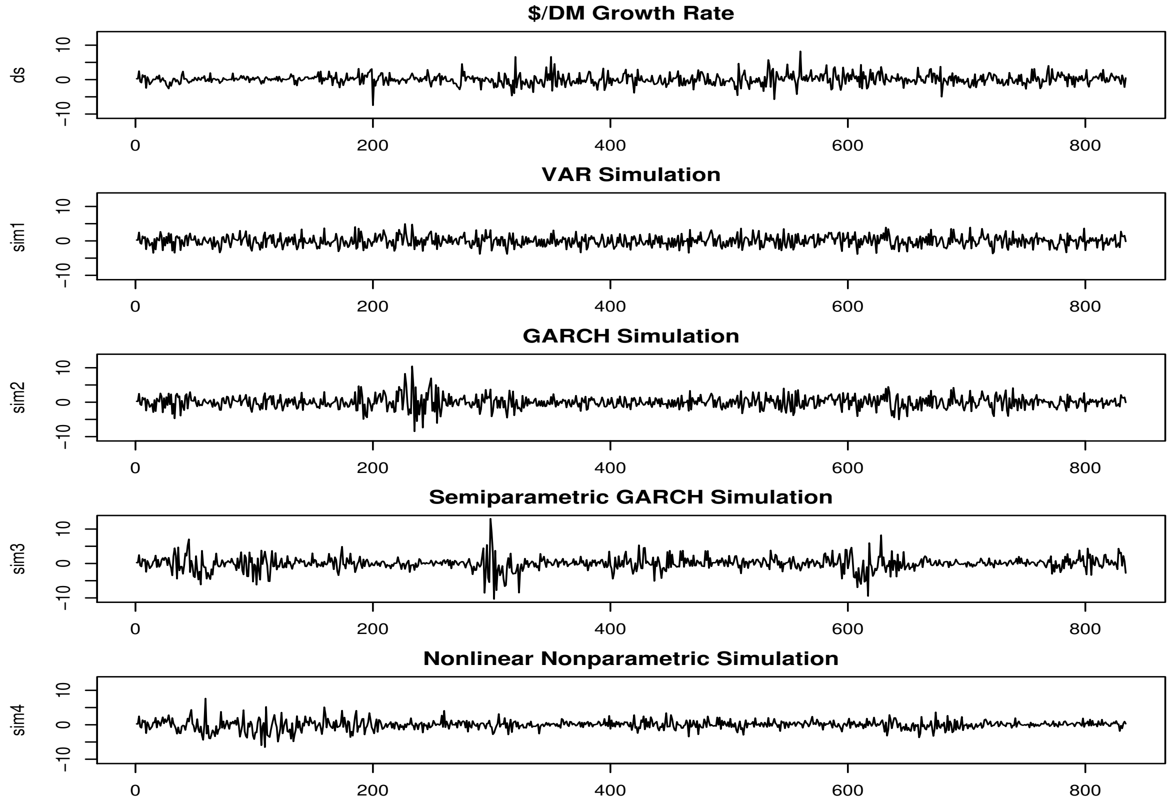

Figure 3. Changes in Weekly \$/DM Spot Exchange Rates, SNP-GARCH. The first panel is a plot of the data, which are each Friday's quote over the years 1975 to 1990 expressed as percentage change from the previous week. The second panel is a simulation from an SNP fit with $(L_u, L_g, L_r, L_p, K_z, I_z, K_x, I_x) = (1, 0, 0, 1, 0, 0, 0, 0)$; the third with (1,1,1,1,0,0,0,0), the fourth with (1,1,1,1,4,0,0,0), and the fifth with (1,1,1,1,4,0,1,0). The parameters L_v and L_w are set to zero. The parameters I_z, maxI_z, and I_x have no effect when $M = 1$. The parameter max$K_z = K_z$ in each instance that $K_x > 0$.

corresponds to $L_u = 1$. The value $L_p = 1$ shown in the table is inoperative because $K_x = 0$. The convention that $L_p > 0$ regardless of the value of K_x was adopted for programming convenience. Next one increases L_r to determine the Schwarz preferred ARCH fit, which corresponds to $L_u = 1$ and $L_r = 3$. Then K_z is increased to determine the Schwarz preferred semiparametric ARCH, which is $L_u = 1$, $L_r = 3$, and $K_z = 4$. Experience has taught us never to consider a value of $K_z < 4$. Lastly, increase K_x to determine if a fully nonlinear specification is called for. In this case BIC suggests that the answer is no.

The last lines of Table 2 are computations for GARCH specifications using $L_g = 1$ and $L_r = 1$. The specification preferable to $L_g = 0$ and $L_r = 3$, which was the Schwarz preferred

Gaussian ARCH model. Trying, as above, $K_z = 4$, we find that $L_u = 1$, $L_g = 1$, $L_r = 1$, and $K_z = 4$ is the Schwarz preferred semiparametric GARCH model. Again as above, we increase K_x to determine if a fully nonlinear specification is called for; BIC suggests that the answer is no. We terminate with the Schwarz preferred model

$$(L_u, L_g, L_r, L_p, K_z, I_z, K_x, I_x) = (1, 1, 1, 1, 4, 0, 0, 0)$$

with $p_\rho = 9$ (and $p_\theta = 10$) at a saturation ratio of (820)/9=91 observations per parameter. The parameters L_v and L_w have been set to zero. The parameters I_z, maxI_z, and I_x have no effect when $M = 1$. The parameter max$K_z = K_z$ in each instance that $K_x > 0$.

This model selection strategy has the advantage of generating an expanding sequence of interesting models, each of which is Schwarz preferred within its class. This is a considerable aid to interpreting results in applications. However, the strategy does not necessarily produce the overall Schwarz preferred model. One would have to examine fits for $(L_u, L_g, L_r, L_p, K_z, I_z, K_x, I_x)$ taking all values over a large grid to to find the overall Schwarz preferred model. In our work we explore sub branches of the upward selection tree as a precaution against missing a substantially better model. We also recompute the final fit from several nearby nodes of the selection tree to be sure that we are not stuck on a local minimum. The ability use start values from any specification, subsetted or not, is a new feature in Version 9.0. The only restriction is that one cannot change the dimension of the time series M when moving from one specification to another.

Simulations from each of the Schwarz preferred ARCH class of models along the expansion path are shown in the lower panels of Figure 2. Simulations from each of the Schwarz preferred GARCH class of models along the expansion path are shown in the lower panels of Figure 3. As seen from the figures, there are readily apparent qualitative differences in these models.

In this illustration, the values of I_z and I_x are irrelevant because the data is univariate. For multivariate applications, put I_z and I_x to zero (to eliminate all interactions initially), and determine K_z and K_x as above. Then use the Schwarz criterion to see if interactions need to be included; that is, if I_z and I_x need to be increased. Be mindful of maxK_z and maxI_z in multivariate applications. Initially one should set maxK_z = maxI_z = K_z. One should only try specifications with $\max K_z < K_z$ and $\max I_z = K_z < K_z$ late in the specification search or at least not before $K_z = 6$ is reached.

This model selection strategy is not completely satisfactory because the Schwarz criterion seems to be both too conservative along some dimensions and too agressive along others (Fenton and Gallant,1996). It has a tendency to prefer models with $K_x = 0$ when there is good evidence that $K_x > 0$ might be more reasonable. On the other hand, it has a tendency to drive K_z unreasonably large in some time series applications. For data from financial markets, one might be well advised to scrutinize models with $K_z > 6$ carefully. See, in this connection, the following applied papers: Gallant and Tauchen (1989); Hussey (1989), Tauchen and Hussey (1991); Gallant, Hsieh, and Tauchen (1991, 1996); Gallant, Hansen, and Tauchen (1990), Gallant, Rossi and Tauchen (1992, 1993). These papers recount experience with number of alternative model selection procedures that did not work well, notably Akaike's (1969) criterion, the Brock, Dechert, and Scheinkman (1987) statistic, and upward likelihood ratio testing. The tentative recommendation is to consider a battery of specification tests based on model residuals when $K_x = 0$ seems unreasonable. Gallant, Rossi and Tauchen (1992) is the most complete discussion of this method.

As mentioned earlier, without the spline transformation, simulations for data from financial markets seem to be too volatile. For this example, the statistics from the data, from the preferred 11114000 fit, and from the 11114000 fit estimated with the spline transformation at $\sigma_{\text{tr}} = 2$ and $\sigma_{\text{tr}} = 1$ are shown in Table 3. As seen in the table, simulations from the fit with the spline imposed are closer to the values for the data.

			Spline	
Statistic	Data	No Spline	$\sigma_{tr} = 2$	$\sigma_{tr} = 1$
a0[1]		-0.05161	-0.05815	-0.06017
a0[2]		0.04295	0.04841	0.04348
a0[3]		0.04028	0.04061	0.04224
a0[4]		0.11637	0.11561	0.11425
A(1,1)		1.00000	1.00000	1.00000
b0[1]		0.07282	0.08320	0.08783
B(1,1)		0.05833	0.05522	0.05643
R0[1]		0.15943	0.12217	0.10538
P(1,1)		-0.37897	-0.34999	-0.35784
Q(1,1)		-0.89804	-0.91796	-0.92280
sn		1.29426	1.29183	1.29064
bic		1.33055	1.32812	1.32693
mean	0.0551579	0.0279452	0.042668	0.0497616
std dev	1.49642	2.39219	1.74122	1.64478
variance	2.23926	5.72258	3.03184	2.70531
skewness	0.336136	-0.0104977	0.195555	0.221971
kurtosis	3.15683	107.77	6.69258	5.17968
no. obs.	834	100834	100834	100834
minimum	-7.46218	-95.1443	-20.7219	-16.8238
5th percentile	-2.23933	-2.84253	-2.58987	-2.47018
25th percentile	-0.70021	-0.787851	-0.759155	-0.741107
50th percentile	0.0126643	0.025511	0.0314289	0.0345291
75th percentile	0.817671	0.815992	0.795872	0.790886
95th percentile	2.5071	2.97132	2.76759	2.66638
maximum	8.22484	77.851	19.4012	15.1295

Table 3. Effect of the Spline Transformation. Estimates and statistics from simulations for the 11114000 model estimated with and without a spline transformation.

The plots and specification search discussed thus far do not impose the spline transform nor does the walk through example in the next section. When this Guide is revised, we shall most likely impose it.

4 Fitting SNP Models: A Walk Through Example

The C++ code that implements the SNP methodology, in addition to facilitating model estimation, makes it easy to retrieve residuals, predicted conditional means, and predicted conditional variances. These statistics are useful for diagnostic testing, model evaluation, forecasting, and related purposes. In addition, the code provides the ordinates of the SNP conditional density over a rectangular grid of points, which is useful for plotting purposes and for performing numerical integration against the SNP conditional density. Finally, it can generate Monte Carlo simulated realizations of arbitrary length from the SNP density, a capability with a variety of applications.

4.1 Fitting Strategy

As discussed in Section 3, the model selection strategy entails moving upward along an expansion path. The fitted SNP models becomes more richly parameterized at each level along the path. The expansion tentatively stops when the best model under the Schwarz criterion is obtained. One might then subject the Schwarz preferred model to a battery of specification tests on the conditional first and second moments. Often, but not always, further expansion of the model is needed in order to achieve satisfactory performance on the diagnostics.

Experience suggests that care is needed in fitting the SNP model at any particular level along the expansion path. Estimates at one level provide start values for the next, so the user should be cautious of hitting a local optimum at any level. Among other things, a false optimum could adversely affect computations at all subsequent levels in the path.

The software is thus designed to facilitate the process of checking for a local optimum, so that the user can have reasonable confidence in the computations before proceeding to the next level. On a single run, the program is capable of performing a wave of optimizations at start values read in from various files. At each optimization, the program randomly perturbs the start values in proportion to user selected scale factors and passes them to the optimizer which performs a few iterations to obtain a preliminary fit. This is repeated a number of times that the user selects. The best of these preliminary fits is retained and passed to the optimizer for iteration to a minimum.

In typical practice, between ten and twenty waves of runs, with twenty or so optimizations within each wave, might be performed to compute the SNP model at a particular level before proceeding to the next. The distributed code contains a file control.tpl that defines a set of runs that has worked well for us. In making the decision as to whether to accept the computations at one level and proceed to the next, the user should look for convergence to the same overall optimum from several start values. This agreement can sometimes be difficult to obtain for large models, and near the end of the expansion path the user might simply have to accept the best computed objective function value out of a wave of fits. In numerical experiments, we have found that, near the end of the path, this probably does little harm as the various optima differ only in their implications for extreme tail behavior.

Table 2 provides an example of a computed expansion path. Each level, that is, row in the table, was computed using the software in the above-described manner. In the next subsection we lead the reader through the steps of computing a similar table.

4.2 Using the Program

The purpose of this subsection is to walk the user through an application. The time series that we use for illustration is the weekly US dollar to German mark exchange rate series described in Subsection 3.2 and plotted in the upper panel of Figure 2. Data, code, and output for this application come with the distribution.

Program control is through a parameter file, for which we shall use naming conventions that describe the model specification within it, and another text file, control.dat.

The program loops through each line of control.dat and uses what it finds there to control its operation. The format is five blank separated fields. They are two strings specifying (1) the input parmfile filename and (2) the output filename, two floats denoted (3) fnew and (4) fold, and two integers denoted (5) nstart and (6) jseed. An example is

control.dat

```
10010000.in0 10010000.fit    0.0e+0    0.0e+0         0              454589
```

For this example, the input parmfile is 10010000.in0 and the output parmfile is 10010000.fit. The fields fnew and fold give the magnitudes by which the parameter values in the parmfile are to be randomly perturbed to generate starting values; nstart states how many times this is to be done, and jseed specifies the initial seed so that results can be replicated. By coding 0 in every numeric and integer field except jseed, we have specified that no perturbation is to be done (because, as explained below, parmfile 1001000.in0 specifies a VAR and the negative of a VAR likelihood is convex). We describe the perturbation parameters in more detail after we have described the contents of the parmfiles 10010000.in0 and 10010000.fit.

The input parmfile of which 10010000.in0 just below is an example sets all program parameters and options. When starting a project it is easiest to commence with a VAR, as we are doing here, because the amount of information required for a VAR is minimal and finding the optimum is assured. The distribution contains a copy of 10010000.in0 that can be edited. Similarly, as estimation proceeds, output parmfiles are copied to new input parmfiles that are edited to specify richer or alternative specifications or to specify tasks such as simulation. There is no need to ever construct a parmfile from scratch.

10010000.in0

```
OPTIMIZATION DESCRIPTION (required)
  SpotRate     Project name, pname, char*
       9.0     SNP version, defines format of this file, snpver, float
        15     Maximum number of primary iterations, itmax0, int
       385     Maximum number of secondary iterations, itmax1, int
  1.00e-08     Convergence tolerance, toler, float
         1     Write detailed output if print=1, int
         0     task, 0 fit, 1 res, 2 mu, 3 sig, 4 plt, 5 sim, 6 usr, int
         0     Increase simulation length by extra, int
  3.00e+00     Scale factor for plots, sfac, float
       457     Seed for simulations, iseed, int
        50     Number of plot grid points, ngrid, int
         0     Statistics not computed if kilse=1, int
DATA DESCRIPTION (required)
         1     Dimension of the time series, M, int
       834     Number of observations, n, int
        14     Provision for initial lags, must have 3<drop<n, int
         0     Condition set for plt is mean if cond=0, it-th obs if it, int
         0     Reread, do not use data from prior fit, if reread=1, int
dmark.dat      File name, any length, no embedded blanks, dsn, string
4 6 5          Read these white space separated fields, fields, intvec
TRANSFORM DESCRIPTION (required)
         0     Normalize using start values if useold=1 else compute, int
         0     Make variance matrix diagonal if diag=1, int
         0     Spline transform x if squash=1, logistic if squash=2, int
```

```
  4.00e+00     Inflection point of transform in normalized x, inflec, float
POLYNOMIAL START VALUE DESCRIPTION (required)
         0     Degree of the polynomial in z, Kz, int
         0     Degree of interactions in z, Iz, int
  0.00e+00     Zero or positive to get positive SNP for EMM, eps0, float
         1     Lags in polynomial part, Lp, int
         0     Max degree of z polynomial that depends on x, maxKz, int
         0     Max interaction of z polynomial that depends on x, maxIz, int
         0     Degree of the polynomial in x, Kx, int
         0     Degree of the interactions x, Ix, int
MEAN FUNCTION START VALUE DESCRIPTION (required)
         1     Lags in VAR part, Lu, int
         1     Intercept if icept=1, int
VARIANCE FUNCTION START VALUE DESCRIPTION (required)
         0     Lags in GARCH (autoregressive) part, may be zero, Lg, int
         s     Coded 's','d','f' for scalar, diagonal, full, Qtype, char
         0     Lags in ARCH (moving average) part, may be zero, Lr, int
         s     Coded 's','d','f' for scalar, diagonal, full, Ptype, char
         0     Lags in leverage effect of GARCH, may be zero, Lv, int
         s     Coded 's','d','f' for scalar, diagonal, full, Vtype, char
         0     Lags in additive level effect, may be zero, Lw, int
         s     Coded 's','d','f' for scalar, diagonal, full, Wtype, char
```

How the parmfile parameters and switches correspond to the SNP parameters in Section 3 can be inferred for the most part from their description in the parmfile. The most relevant are Lu $= L_u$, Lg $= L_g$, Lr $= L_r$, Lp $= L_p$, Kz $= K_z$, Iz $= I_z$, maxKz $=$ maxK_z, maxIz $=$ maxI_z, Kx $= K_x$, and Ix $= I_x$. We proceed now to the details.

The first concern is to describe the data so that it can be read in. This is done in the DATA DESCRIPTION block. Setting M and n is straightforward; M is the dimension of y_t and n is the total number of observations to be read in. The value of n can be smaller than the total number of observations, in which case those left over at the end will not be read. Drop is the number of observations at the beginning of the series to skip to produce initial lags; this should be larger than any value of L_u, $L_r + L_u$, or L_p envisaged in any fit. If drop is set too small, the missing lags are initialized to zero. If drop is 3 or less, the program terminates. The parameter n just above refers to the physical data and is not influenced by drop; obviously drop must be considerably less than n for running the program to be a sensible activity. The switch cond controls the conditioning information for plotting; one can condition either on the mean by putting it$=0$ or on the t-th observation in the data by putting it $= t$. The easiest way to condition on values other than these two options is to add them as observations to the end of the data set after estimation is finished, increase n, and set it$=$n. The switch reread has not been implemented yet; presently the data are read afresh for each line of control.dat. The string dsn specifies the name of the file containing the data. Lastly, one has fields. One must use care here because errors can cause the program to crash with unhelpful diagnostic messages, if any at all. The presumptions is that the data are arranged in a table with time t as the row index and the elements of y_t in the columns. The blank separated numbers here specify the fields (columns) of the data in the order in which they are to be assigned to the elements $y_{1t}, y_{2t}, \ldots, y_{Mt}$ of y_t. It does not hurt to have too many fields listed as we have done here. The disaster is when there are too few (less than M) or one of them is larger than the actual number of columns in the data set. A few of the first and last values of y_t read in are printed in detail.dat which should be checked to make sure the data was read correctly.

If the data are not arranged in the presumed form, or some preprocessing is needed, the object (i.e. class) that reads the data can be changed. It is class datread presented in file snpusr.h and coded in file snpusr.cpp. It is polymorphic via inheritance from class datread_base which is presented in file snp_base.h. One codes a substitute class that inherits from datread_base, adds its description to snpusr.h and defines it (codes it) in snpusr.cpp, and swaps names in the typedef that defines datread_type in snpusr.h. We shall give related examples later but it is hard to imagine an application where it would be easier to code an alternative to datread than to generate a new data set that satisfies the conventions using R, SAS, or Excel.

In the OPTIMIZATION DESCRIPTION block, SpotRate is a user chosen label. The version number 9.0 in the second line is obligatory; itmax0 and itmax1 are primary and secondary iteration limits that control the optimizer; toler is a convergence tolerance that is passed to the optimizer. If the switch print is set to 1, detailed output reporting the interpretation of the parmfile and progress of the computations is printed to a file detail.dat.

Task defines what is to be computed and what is to be written to the output file specified as the second string of the line of control.dat being processed. If task = 0, optimization is performed and a new parameter file is written to the output file that can be used to move from a sparse parameterization to a richer one (or conversely) or to use as an input parmfile for simulation, plotting, or moment computations. The other choices are: task = 1, residuals used for diagnostics are written to the output file; task = 2, the mean of the one-step-ahead conditional density is computed at each x_{t-1} in the sample and written to the output file; task = 3, the upper triangle of the variance-covariance matrix of the one-step-ahead conditional density is computed at each x_{t-1} in the sample and written to the output file; task = 4, plot data is written to the output file; and task = 5, a simulation is written to the output file that consists of the data up to drop, and simulations up to n + extra. The last, task = 6, points to an optional user written task; this is discussed in Section 5.

The parameter sfac determines the plotting increment; 3.0 is usually about right. The parameter ngrid is the number of plot points for a graphic. As distributed, coding task = 6 will generate a Gauss-Hermite quadrature rule that can be used for numerical integration over an SNP conditional density determined by the same conditioning set that the plot feature (task = 4) uses in which case ngrid determines the order of the Gauss-Hermite quadrature rule. iseed is the seed for a simulation.

When task = 0 an output parmfile contains descriptive statistics at the end that facilitate interpretation of results. This entails computation and inversion of the information matrix, which is not needed for any other purpose and can take considerable time. Setting kilse=1 stops this computation from being performed.

In the TRANSFORM DESCRIPTION block, the switch useold determines the initial normalization of the data. If useold = 1 and a TRANSFORM START VALUES block is present, the values for $\bar{y}$ and S found there are used as the initial location-scale transformation $\tilde{y}_t \to y_t$. If the block is not there, then no initial transformation is done. Transformation stabilizes computations without disturbing input and output conventions, so there is little reason to try to defeat it. Controlling it by means of the TRANSFORM START VALUES block is mainly only useful in connection with EMM or when the data being fed to SNP for some similar purpose is not that corresponding to the estimates in the parmfile. Setting diag = 1 puts the off diagonal elements of S to zero before normalization which can be an aid in interpreting results. Our example is univariate, so putting S to a diagonal is irrelevant. Setting spline to 1 selects the spline transformation $x_t \to \hat{x}_t$, 2 the logistic transformation, and 0 no transformation. The value inflec is σ_{tr} of Section 3. Centered and scaled data y_t within the interval $(-\sigma_{\mathrm{tr}}, \sigma_{\mathrm{tr}})$ are not affected by the transformation; refer to Figure 1.

The FUNCTION START VALUE DESCRIPTION blocks provide the information to in read values for θ if the corresponding FUNCTION START VALUES block is present or to set them according to predefined conventions if not. Here FUNCTION stands for one of POLYNOMIAL (afunc), MEAN FUNCTION (ufunc), or VARIANCE FUNCTION (rfunc). Other than for the initial VAR fit, these blocks are written to the parmfile by a previous run of SNP and there is no need to edit them. Doing so anyway will usually cause an error that terminates the program. What one does edit are the FUNCTION DESCRIPTION blocks. These complete the description of the model in terms of increments (positive values) or decrements (negative values) to the starting model. (Also, an intercept can be added or deleted from the location function ufunc and the type of one of the BEKK matrices in rfunc can be changed.) Increments and decrements can act to augment or reduce the model. The program imputes values for θ as specified by the increments and decrements from the values of θ read in (or imputed) as described above. For increments imputation is straightforward: old values remain the same and the incremental parameters are set to zero. For decrements, decisions are reasonable but of necessity arbitrary. Exactly what was done can be determined by examining the file detail.dat.

The parameter eps0 in the POLYNOMIAL START VALUE DESCRIPTION block can be set to a small positive value if the SNP fit is to be used in connection with EMM as discussed at the end of Subsection 3.1; although it does no harm to do so in general for eps0 set to about 1.0e-05 or less.

In the example, 10010000.in0, we set Lu=1 and everything else to zero. There are are no START VALUES blocks, so all elements of the vector θ are imputed. In fact, they all get set to zero except for the constant term of $\mathcal{P}(y,x)$ which is set to one.

Upon running the program one gets:

10010000.fit

```
PARMFILE HISTORY (optional)
#
# This parmfile was written by SNP Version 9.0 using the following line from
# control.dat, which was read as char*, char*, float, float, int, int
# ----------------------------------------------------------------------------
#   input_file  output_file      fnew      fold    nstart                jseed
# ------------ ------------ --------- --------- --------- --------------------
# 10010000.in0 10010000.fit 0.00e+000 0.00e+000         0               100542
# ----------------------------------------------------------------------------
# If fnew is negative, only the polynomial part of the model is perturbed.
# Similarly for fold.
#
OPTIMIZATION DESCRIPTION (required)
  SpotRate     Project name, pname, char*
```

```
       9.0     SNP version, defines format of this file, snpver, float
        15     Maximum number of primary iterations, itmax0, int
       385     Maximum number of secondary iterations, itmax1, int
 1.00e-008     Convergence tolerance, toler, float
         1     Write detailed output if print=1, int
         0     task, 0 fit, 1 res, 2 mu, 3 sig, 4 plt, 5 sim, 6 usr, int
         0     Increase simulation length by extra, int
 3.00e+000     Scale factor for plots, sfac, float
       457     Seed for simulations, iseed, int
        50     Number of plot grid points, ngrid, int
         0     Statistics not computed if kilse=1, int
DATA DESCRIPTION (required)
         1     Dimension of the time series, M, int
       834     Number of observations, n, int
        14     Provision for initial lags, must have 3<drop<n, int
         0     Condition set for plt is mean if cond=0, it-th obs if it, int
         0     Reread, do not use data from prior fit, if reread=1, int
dmark.dat      File name, any length, no embedded blanks, dsn, string
4              Read these white space separated fields, fields, intvec
TRANSFORM DESCRIPTION (required)
         0     Normalize using start values if useold=1 else compute, int
         0     Make variance matrix diagonal if diag=1, int
         0     Spline transform x if squash=1, logistic if squash=2, int
 4.00e+000     Inflection point of transform in normalized x, inflec, float
POLYNOMIAL START VALUE DESCRIPTION (required)
         0     Degree of the polynomial in z, Kz, int
         0     Degree of interactions in z, Iz, int
 0.00e+000     Zero or positive to get positive SNP for EMM, eps0, float
         1     Lags in polynomial part, Lp, int
         0     Max degree of z polynomial that depends on x, maxKz, int
         0     Max interaction of z polynomial that depends on x, maxIz, int
         0     Degree of the polynomial in x, Kx, int
         0     Degree of the interactions x, Ix, int
MEAN FUNCTION START VALUE DESCRIPTION (required)
         1     Lags in VAR part, Lu, int
         1     Intercept if icept=1, int
VARIANCE FUNCTION START VALUE DESCRIPTION (required)
         0     Lags in GARCH (autoregressive) part, may be zero, Lg, int
         s     Coded 's','d','f' for scalar, diagonal, full, Qtype, char
         0     Lags in ARCH (moving average) part, may be zero, Lr, int
         s     Coded 's','d','f' for scalar, diagonal, full, Ptype, char
         0     Lags in leverage effect of GARCH, may be zero, Lv, int
         s     Coded 's','d','f' for scalar, diagonal, full, Vtype, char
         0     Lags in additive level effect, may be zero, Lw, int
         s     Coded 's','d','f' for scalar, diagonal, full, Wtype, char
POLYNOMIAL DESCRIPTION (optional)
         0     Increment or decrement to Kz, int
         0     Increment or decrement to Iz, int
  0.00e+00     Increment or decrement to eps0, float
         0     Increment or decrement to Lp, int
         0     Increment or decrement to maxKz, int
         0     Increment or decrement to maxIz, int
         0     Increment or decrement to Kx, int
         0     Increment or decrement to Ix, int
MEAN FUNCTION DESCRIPTION (optional)
         0     Increment or decrement to Lu, int
         0     Increment or decrement to icept, int
VARIANCE FUNCTION DESCRIPTION (optional)
         0     Increment or decrement to GARCH lag Lg, int
         s     Coded 's','d','f' for scalar, diagonal, full, Qtype, char
         0     RIncrement or decrement to ARCH lag Lr, int
         s     Coded 's','d','f' for scalar, diagonal, full, Ptype, char
```

```
        0     Increment or decrement to leverage effect lag Lv, int
        s     Coded 's','d','f' for scalar, diagonal, full, Vtype, char
        0     Increment or decrement to level effect lag Lw, int
        s     Coded 's','d','f' for scalar, diagonal, full, Wtype, char
POLYNOMIAL START VALUES FOR A (optional)
  1.00000000000000000e+000    0
MEAN FUNCTION START VALUES FOR b0 (optional)
  5.08917192881666700e-004    1
MEAN FUNCTION START VALUES FOR B (optional)
  2.22361342057764730e-002    1
VARIANCE FUNCTION START VALUES FOR Rparms (optional)
  1.00252755384483170e+000    1
TRANSFORM START VALUES FOR mean (optional)
  5.51578971010593040e-002
TRANSFORM START VALUES FOR variance (optional)
  2.23657818199155180e+000
SUMMARY STATISTICS (optional)
Fit criteria:
  Length rho =           3
  Length theta =         4
  n - drop =           820
  -2 ln likelihood =        2331.19914594   2.33119914593695380e+003
  sn =           1.39760141   1.39760140643702260e+000
  aic =          1.40119853   1.40119852873918080e+000
  hq =           1.40445758   1.40445757663208150e+000
  bic =          1.40969895   1.40969894852759590e+000
Index          theta      std error    t-statistic    descriptor
    1        1.00000        0.00000        0.00000    A(1,1)  0 0
    2        0.00051        0.03559        0.01430    b0[1]
    3        0.02224        0.02697        0.82446    B(1,1)
    4        1.00253        0.01584       63.29936    R0[1]

For estimation of this SNP specification by MCMC, restrict the following
elements of theta to be positive:  4
```

Here we see the one exception to the remark that the FUNCTION START VALUES blocks should not be edited. Note the 1's to the right of the parameter values. To fix a value of the parameter at the value shown in the FUNCTION START VALUES block so that the optimizer will not change it, change that 1 to a 0. Also change the parameter value to what is required if necessary. Note also that the constant term of $\mathcal{P}(z|a_x)$ is fixed at 1.0. in the block labeled POLYNOMIAL START VALUES for A; don't ever change that line.

The remark about positivity restrictions at the bottom of the file is irrelevant for our purposes because the SNP package uses a BFGS hill climber. If simulated annealing or some other MCMC optimizer is used, these constraints may need to be imposed.

At this point, we have the wherewithal to make some progress. We shall now move upward to a $(L_u, L_g, L_r, L_p, K_z, I_z, K_x, I_x) = (1, 1, 1, 1, 4, 0, 0, 0)$ parameterization by copying the file 10010000.fit to the file 11114000.in0 and incrementing Kz by 4 in the POLYNOMIAL DESCRIPTION block of 11114000.in0 and Lg and Lr by 1 in the VARIANCE FUNCTION

DESCRIPTION block, leaving the rest of the parmfile alone, as follows:

11114000.in0

```
POLYNOMIAL DESCRIPTION (optional)
        4     Increment or decrement to Kz, int
        0     Increment or decrement to Iz, int
  0.00e+00    Increment or decrement to eps0, float
        0     Increment or decrement to Lp, int
        0     Increment or decrement to maxKz, int
        0     Increment or decrement to maxIz, int
        0     Increment or decrement to Kx, int
        0     Increment or decrement to Ix, int
VARIANCE FUNCTION DESCRIPTION (optional)
        1     Increment or decrement to Lg, int
        s     Coded 's','d','f' for scalar, diagonal, full, Qtype, char
        1     Increment or decrement to Lr, int
        s     Coded 's','d','f' for scalar, diagonal, full, Ptype, char
        0     Increment or decrement to Lv, int
        s     Coded 's','d','f' for scalar, diagonal, full, Vtype, char
        0     Increment or decrement to Lw, int
        s     Coded 's','d','f' for scalar, diagonal, full, Wtype, char
```

We edit control.dat to read:

control.dat

```
11114000.in0 11114000.f00    0.0e-0    0.0e-0       00              011677
11114000.in0 11114000.f01    1.0e-5    0.0e-5       25              011677
11114000.in0 11114000.f02    0.0e-5    1.0e-5       25              011677
11114000.in0 11114000.f03   -1.0e-5    1.0e-5       25              011677
11114000.in0 11114000.f04    1.0e-5   -1.0e-5       25              011677
11114000.in0 11114000.f05    1.0e-5    1.0e-5       25              011677
11114000.in0 11114000.f06    1.0e-4    0.0e-4       25              011677
11114000.in0 11114000.f07    0.0e-4    1.0e-4       25              011677
11114000.in0 11114000.f08   -1.0e-4    1.0e-4       25              011677
11114000.in0 11114000.f09    1.0e-4   -1.0e-4       25              011677
11114000.in0 11114000.f10    1.0e-4    1.0e-4       25              011677
11114000.in0 11114000.f11    1.0e-3    0.0e-3       25              011677
11114000.in0 11114000.f12    0.0e-3    1.0e-3       25              011677
11114000.in0 11114000.f13   -1.0e-3    1.0e-3       25              011677
11114000.in0 11114000.f14    1.0e-3   -1.0e-3       25              011677
11114000.in0 11114000.f15    1.0e-3    1.0e-3       25              011677
11114000.in0 11114000.f16    1.0e-2    0.0e-2       25              011677
11114000.in0 11114000.f17    0.0e-2    1.0e-2       25              011677
11114000.in0 11114000.f18   -1.0e-2    1.0e-2       25              011677
11114000.in0 11114000.f19    1.0e-2   -1.0e-2       25              011677
11114000.in0 11114000.f20    1.0e-2    1.0e-2       25              011677
11114000.in0 11114000.f21    1.0e-1    0.0e-1       25              011677
11114000.in0 11114000.f22    0.0e-1    1.0e-1       25              011677
11114000.in0 11114000.f23   -1.0e-1    1.0e-1       25              011677
11114000.in0 11114000.f24    1.0e-1   -1.0e-1       25              011677
11114000.in0 11114000.f25    1.0e-1    1.0e-1       25              011677
11114000.in0 11114000.f26    1.0e+0    0.0e+0       25              011677
11114000.in0 11114000.f27    0.0e+0    1.0e+0       25              011677
11114000.in0 11114000.f28   -1.0e+0    1.0e+0       25              011677
11114000.in0 11114000.f29    1.0e+0   -1.0e+0       25              011677
11114000.in0 11114000.f30    1.0e+0    1.0e+0       25              011677
```

We can now describe the file control.dat more completely. The file control.dat determines which parmfiles are read and how starting values are to be used. For each line in control.dat, the program reads from the parmfile designated by the filename in the first field of the line. The filename in the second field of the line determines the file to which results are written. Starting values of parameters that are zero are replaced by uniform[-1,1] random numbers times the value in third field, fnew. The starting values of the parameters that are not zero are multiplied by the quantity one plus a uniform[-1,1] random number times the forth field, fold. These start values are passed to the optimizer and iterated itmax0 times, unless iterations are terminated earlier by the tolerance check, toler. If fnew is negative then only the polynomial part of the model is perturbed. Similarly for fold. The fifth field, nstart, is the number of times this process is to be repeated for that line. The sixth field, jseed, is the seed for the random number generator. After nstart repetitions, the best result obtained is passed to the optimizer and iterated itmax1 times, unless terminated earlier by the tolerance check. The result of the final optimization is written to the output filename specified in the second field.

We could have progressed sequentially by first fitting a $(L_u, L_g, L_r, L_p, K_z, I_z, K_x, I_x) = (1, 1, 1, 1, 0, 0, 0, 0)$ model and then a $(L_u, L_g, L_r, L_p, K_z, I_z, K_x, I_x) = (1, 1, 1, 1, 4, 0, 0, 0)$ model as was done in constructing Table 2. There are risks to hanging at a local minimum when doing this because the GARCH parameters can hang at the values of the thin tailed (1,1,1,1,0,0,0,0) model. The problem is that both GARCH and $K_z > 0$ have the effect of thickening tails and the parameter values of a can trade off against those of Q and P. A simultaneous fit provides some measure of protection against this trade-off causing the optimizer to hang at a local minimum.

As computations progress, a line summarizing them is written to the file summary.dat. For this run we get

summary.dat

```
SNP Restart
-----------------------------------------------------------------------------
     pname   output_file      fitted_model   p opt iter   obj_func        bic
-----------------------------------------------------------------------------
  SpotRate  11114000.f00  11s1s0s0s1400000   9   0   10    1.34726   1.38355*
  SpotRate  11114000.f01  11s1s0s0s1400000   9   0   11    1.34726   1.38355*
  SpotRate  11114000.f02  11s1s0s0s1400000   9   0   17    1.32701   1.36330*
```

```
SpotRate  11114000.f03  11s1s0s0s1400000   9   0   17    1.32701    1.36330
SpotRate  11114000.f04  11s1s0s0s1400000   9   0   12    1.34726    1.38355
SpotRate  11114000.f05  11s1s0s0s1400000   9   0   17    1.32701    1.36330*
SpotRate  11114000.f06  11s1s0s0s1400000   9   0   11    1.34726    1.38355
SpotRate  11114000.f07  11s1s0s0s1400000   9   0   20    1.32701    1.36330*
SpotRate  11114000.f08  11s1s0s0s1400000   9   0   20    1.32701    1.36330
SpotRate  11114000.f09  11s1s0s0s1400000   9   0   12    1.34726    1.38355
SpotRate  11114000.f10  11s1s0s0s1400000   9   0   19    1.32701    1.36330
SpotRate  11114000.f11  11s1s0s0s1400000   9   0   12    1.34726    1.38355
SpotRate  11114000.f12  11s1s0s0s1400000   9   0   19    1.32701    1.36330*
SpotRate  11114000.f13  11s1s0s0s1400000   9   0   19    1.32701    1.36330
SpotRate  11114000.f14  11s1s0s0s1400000   9   1   12    1.34726    1.38355
SpotRate  11114000.f15  11s1s0s0s1400000   9   0   40    1.29426    1.33055*
SpotRate  11114000.f16  11s1s0s0s1400000   9   1   12    1.34726    1.38355
SpotRate  11114000.f17  11s1s0s0s1400000   9   0   35    1.29426    1.33055*
SpotRate  11114000.f18  11s1s0s0s1400000   9   0   35    1.29426    1.33055
SpotRate  11114000.f19  11s1s0s0s1400000   9   1   12    1.34726    1.38355
SpotRate  11114000.f20  11s1s0s0s1400000   9   0   36    1.29426    1.33055
SpotRate  11114000.f21  11s1s0s0s1400000   9   1   12    1.34726    1.38355
SpotRate  11114000.f22  11s1s0s0s1400000   9   0   28    1.29426    1.33055
SpotRate  11114000.f23  11s1s0s0s1400000   9   0   28    1.29426    1.33055
SpotRate  11114000.f24  11s1s0s0s1400000   9   0   12    1.34726    1.38355
SpotRate  11114000.f25  11s1s0s0s1400000   9   0   30    1.29426    1.33055
SpotRate  11114000.f26  11s1s0s0s1400000   9   0   14    1.34726    1.38355
SpotRate  11114000.f27  11s1s0s0s1400000   9   0   26    1.29426    1.33055*
SpotRate  11114000.f28  11s1s0s0s1400000   9   0   26    1.29426    1.33055
SpotRate  11114000.f29  11s1s0s0s1400000   9   0   24    1.36566    1.40195
SpotRate  11114000.f30  11s1s0s0s1400000   9   0   24    1.29426    1.33055
```

The meaning of most fields are adequately conveyed by the header. Those that need some explanation are the fields fitted_model, opt, iter, and the asterisks to the right of entries in the field BIC. The entry 11s1s0s0s1400000 means that the parameter settings were

$$(L_u, L_g, \text{Qtype}, L_r, \text{Ptype}, L_v, \text{Vtype}, L_w, \text{Wtype}, L_p, K_z, \max K_z, I_z, \max I_z, K_x, I_x)$$

$$= (1, 1, s, 1, s, 0, s, 0, s, 1, 4, 0, 0, 0, 0, 0).$$

The field labeled opt contains the termination code of the optimizer nlopt presented in libscl.h if two digits and linesrch in libscl.h if one. A return code of 0 means that the line search succeeded on a zero derivative condition and that the percentage change in toler was satisfied. A return code of 1 is similar except that the line search could not find a point better than the left end of the line. There are something on the order of 15 different other termination codes for which see linesrch.cpp and nolpt.cpp. Traps for NaN's and Inf's identify the most worrisome things that can go wrong. Termination codes that don't produce overly threatening warning messages are probably not a problem.

The iter field gives the sum of the number of the iterations of the best model from the optimizations of the first round fits (as limited by itmax0) and the iterations from the second round polish (as limited by itmax1). What one really wants are the number of function evaluations and the total over all iterations on random starts plus those of the second round polish of the best of them. This can be found in detail.dat for both iter and the number of function evaluations. The optimizer that we are using is new to us and seems to be very economical of iterations and function evaluations relative to those we have used in the past. It is beginning to seem that setting itmax0 = 15 is too large. One might be able to get away with itmax0 = 5 which would shorten run times by about half.

This leaves the asterisk in the BIC field. That's easy. The last line with an asterisk has the smallest BIC if all models in a control.dat have the same specification and the smallest value of obj_func if not. Thus we copy 11114000.f27 to 11114000.fit to save it and again to 11114010v.in0 to get a parmfile that we can edit to expand further. Here is the former:

11114000.fit

```
PARMFILE HISTORY (optional)
#
# This parmfile was written by SNP Version 9.0 using the following line from
# control.dat, which was read as char*, char*, float, float, int, int
# -----------------------------------------------------------------------------
#   input_file  output_file      fnew      fold    nstart                jseed
# ------------ ------------ --------- --------- --------- --------------------
# 11114000.in0 11114000.f27 0.00e+000 1.00e+000        25               454589
# -----------------------------------------------------------------------------
# If fnew is negative, only the polynomial part of the model is perturbed.
# Similarly for fold.
#
OPTIMIZATION DESCRIPTION (required)
  SpotRate     Project name, pname, char*
       9.0     SNP version, defines format of this file, snpver, float
        15     Maximum number of primary iterations, itmax0, int
       385     Maximum number of secondary iterations, itmax1, int
 1.00e-008     Convergence tolerance, toler, float
         1     Write detailed output if print=1, int
         0     task, 0 fit, 1 res, 2 mu, 3 sig, 4 plt, 5 sim, 6 usr, int
         0     Increase simulation length by extra, int
 3.00e+000     Scale factor for plots, sfac, float
       457     Seed for simulations, iseed, int
        50     Number of plot grid points, ngrid, int
         0     Statistics not computed if kilse=1, int
DATA DESCRIPTION (required)
         1     Dimension of the time series, M, int
       834     Number of observations, n, int
        14     Provision for initial lags, must have 3<drop<n, int
         0     Condition set for plt is mean if cond=0, it-th obs if it, int
         0     Reread, do not use data from prior fit, if reread=1, int
dmark.dat      File name, any length, no embedded blanks, dsn, string
4              Read these white space separated fields, fields, intvec
TRANSFORM DESCRIPTION (required)
         0     Normalize using start values if useold=1 else compute, int
         0     Make variance matrix diagonal if diag=1, int
         0     Spline transform x if squash=1, logistic if squash=2, int
```

```
 4.00e+000     Inflection point of transform in normalized x, inflec, float
POLYNOMIAL START VALUE DESCRIPTION (required)
        4      Degree of the polynomial in z, Kz, int
        0      Degree of interactions in z, Iz, int
 0.00e+000     Zero or positive to get positive SNP for EMM, eps0, float
        1      Lags in polynomial part, Lp, int
        0      Max degree of z polynomial that depends on x, maxKz, int
        0      Max interaction of z polynomial that depends on x, maxIz, int
        0      Degree of the polynomial in x, Kx, int
        0      Degree of the interactions x, Ix, int
MEAN FUNCTION START VALUE DESCRIPTION (required)
        1      Lags in VAR part, Lu, int
        1      Intercept if icept=1, int
VARIANCE FUNCTION START VALUE DESCRIPTION (required)
        1      Lags in GARCH (autoregressive) part, may be zero, Lg, int
        s      Coded 's','d','f' for scalar, diagonal, full, Qtype, char
        1      Lags in ARCH (moving average) part, may be zero, Lr, int
        s      Coded 's','d','f' for scalar, diagonal, full, Ptype, char
        0      Lags in leverage effect of GARCH, may be zero, Lv, int
        s      Coded 's','d','f' for scalar, diagonal, full, Vtype, char
        0      Lags in additive level effect, may be zero, Lw, int
        s      Coded 's','d','f' for scalar, diagonal, full, Wtype, char
POLYNOMIAL DESCRIPTION (optional)
        0      Increment or decrement to Kz, int
        0      Increment or decrement to Iz, int
  0.00e+00     Increment or decrement to eps0, float
        0      Increment or decrement to Lp, int
        0      Increment or decrement to maxKz, int
        0      Increment or decrement to maxIz, int
        0      Increment or decrement to Kx, int
        0      Increment or decrement to Ix, int
MEAN FUNCTION DESCRIPTION (optional)
        0      Increment or decrement to Lu, int
        0      Increment or decrement to icept, int
VARIANCE FUNCTION DESCRIPTION (optional)
        0      Increment or decrement to GARCH lag Lg, int
        s      Coded 's','d','f' for scalar, diagonal, full, Qtype, char
        0      Increment or decrement to ARCH lag Lr, int
        s      Coded 's','d','f' for scalar, diagonal, full, Ptype, char
        0      Increment or decrement to leverage effect lag Lv, int
        s      Coded 's','d','f' for scalar, diagonal, full, Vtype, char
        0      Increment or decrement to level effect lag Lw, int
        s      Coded 's','d','f' for scalar, diagonal, full, Wtype, char
POLYNOMIAL START VALUES FOR a0 (optional)
 -5.16076965515111090e-002    1
  4.29554878971911480e-002    1
  4.02774315405815190e-002    1
  1.16366622989738270e-001    1
POLYNOMIAL START VALUES FOR A (optional)
  1.00000000000000000e+000    0
MEAN FUNCTION START VALUES FOR b0 (optional)
  7.28194529718470680e-002    1
MEAN FUNCTION START VALUES FOR B (optional)
  5.83255326811739850e-002    1
VARIANCE FUNCTION START VALUES FOR Rparms (optional)
  1.59426597892471020e-001    1
 -3.78958421916034740e-001    1
 -8.98044691523147480e-001    1
TRANSFORM START VALUES FOR mean (optional)
  5.51578971010593040e-002
TRANSFORM START VALUES FOR variance (optional)
  2.23657818199155180e+000
```

```
SUMMARY STATISTICS (optional)
Fit criteria:
  Length rho =           9
  Length theta =        10
  n - drop =           820
  -2 ln likelihood =        2158.82095597   2.15882095596669840e+003
  sn =           1.29425717   1.29425716784574240e+000
  aic =          1.30504853   1.30504853475221720e+000
  hq =           1.31482568   1.31482567843091890e+000
  bic =          1.33054979   1.33054979411746220e+000
Index          theta      std error    t-statistic    descriptor
    1       -0.05161        0.03478       -1.48376    a0[1]   1
    2        0.04296        0.03224        1.33255    a0[2]   2
    3        0.04028        0.01866        2.15792    a0[3]   3
    4        0.11637        0.01830        6.35780    a0[4]   4
    5        1.00000        0.00000        0.00000    A(1,1)  0 0
    6        0.07282        0.05159        1.41142    b0[1]
    7        0.05833        0.03542        1.64655    B(1,1)
    8        0.15943        0.03705        4.30340    R0[1]
    9       -0.37896        0.03683      -10.28892    P(1,1)  s
   10       -0.89804        0.01891      -47.47874    Q(1,1)  s

For estimation of this SNP specification by MCMC, restrict the following
elements of theta to be positive:  8 9 10
```

One might note above that Q(1,1) and P(1,1) get squared in the BEKK formula so that their signs are irrelevant. Also note that the sum of their squares is less than one.

To move on further, we edit the POLYNOMIAL DESCRIPTION block of 11114010.in0 as follows:

```
POLYNOMIAL DESCRIPTION (optional)
         0     Increment or decrement to Kz, int
         0     Increment or decrement to Iz, int
  0.00e+00     Increment or decrement to eps0, float
         0     Increment or decrement to Lp, int
         4     Increment or decrement to maxKz, int
         0     Increment or decrement to maxIz, int
         1     Increment or decrement to Kx, int
         0     Increment or decrement to Ix, int
```

Note particularly the increment to maxKz. (We tried a specification as above with Lv=1 but the estimate of V(1,1) was 1.0e-5 with a large standard error.)

Running this specification we have:

```
SNP Restart
---------------------------------------------------------------------------
     pname   output_file      fitted_model   p opt iter   obj_func        bic
---------------------------------------------------------------------------
  SpotRate  11114010.f00  11s1s0s0s1440010  14   0   20    1.29277    1.34923*
  SpotRate  11114010.f01  11s1s0s0s1440010  14   0   25    1.29277    1.34923
  SpotRate  11114010.f02  11s1s0s0s1440010  14   0   25    1.29277    1.34923
  SpotRate  11114010.f03  11s1s0s0s1440010  14   0   25    1.29277    1.34923
  SpotRate  11114010.f04  11s1s0s0s1440010  14   0   24    1.29277    1.34923
  SpotRate  11114010.f05  11s1s0s0s1440010  14   0   25    1.29277    1.34923
```

```
  SpotRate  11114010.f06  11s1s0s0s1440010  14   0   25    1.29277    1.34923
  SpotRate  11114010.f07  11s1s0s0s1440010  14   0   25    1.29277    1.34923
  SpotRate  11114010.f08  11s1s0s0s1440010  14   0   24    1.29277    1.34923
  SpotRate  11114010.f09  11s1s0s0s1440010  14   0   24    1.29277    1.34923
  SpotRate  11114010.f10  11s1s0s0s1440010  14   0   25    1.29277    1.34923
  SpotRate  11114010.f11  11s1s0s0s1440010  14   0   25    1.29277    1.34923
  SpotRate  11114010.f12  11s1s0s0s1440010  14   0   24    1.29277    1.34923
  SpotRate  11114010.f13  11s1s0s0s1440010  14   0   24    1.29277    1.34923
  SpotRate  11114010.f14  11s1s0s0s1440010  14   0   25    1.29277    1.34923
  SpotRate  11114010.f15  11s1s0s0s1440010  14   0   25    1.29277    1.34923
  SpotRate  11114010.f16  11s1s0s0s1440010  14   0   24    1.29277    1.34923
  SpotRate  11114010.f17  11s1s0s0s1440010  14   0   22    1.29277    1.34923
  SpotRate  11114010.f18  11s1s0s0s1440010  14   0   25    1.29277    1.34923
  SpotRate  11114010.f19  11s1s0s0s1440010  14   0   22    1.29277    1.34923
  SpotRate  11114010.f20  11s1s0s0s1440010  14   0   25    1.29277    1.34923
  SpotRate  11114010.f21  11s1s0s0s1440010  14   0   23    1.29277    1.34923
  SpotRate  11114010.f22  11s1s0s0s1440010  14   0   21    1.29277    1.34923
  SpotRate  11114010.f23  11s1s0s0s1440010  14   0   22    1.29277    1.34923
  SpotRate  11114010.f24  11s1s0s0s1440010  14   0   26    1.29277    1.34923
  SpotRate  11114010.f25  11s1s0s0s1440010  14   0   29    1.29109    1.34755*
  SpotRate  11114010.f26  11s1s0s0s1440010  14   0   32    1.29277    1.34923
  SpotRate  11114010.f27  11s1s0s0s1440010  14   0   38    1.29283    1.34928
  SpotRate  11114010.f28  11s1s0s0s1440010  14   0   34    1.31779    1.37425
  SpotRate  11114010.f29  11s1s0s0s1440010  14   0   40    1.29277    1.34923
  SpotRate  11114010.f30  11s1s0s0s1440010  14   0   73    1.31399    1.37044
```

In this instance 11114010.f25 is the best fit. We copy 11114010.f25 to 11114010.fit.

We next edit the Schwarz preferred SNP-GARCH fit (see Subsection 3.2) 11114000.fit by putting task to 1, 2, 3, 4, 5 successively, saving the result as files 11114000.in1, 11114000.in2, 11114000.in3, 11114000.in4, 11114000.in5 respectively. As an example, here is the OPTIMIZATION DESCRIPTION block of 11114000.in4.

11114000.in4

```
OPTIMIZATION DESCRIPTION (required)
  SpotRate     Project name, pname, char*
       9.0     SNP version, defines format of this file, snpver, float
        15     Maximum number of primary iterations, itmax0, int
       385     Maximum number of secondary iterations, itmax1, int
 1.00e-008     Convergence tolerance, toler, float
         1     Write detailed output if print=1, int
         4     task, 0 fit, 1 res, 2 mu, 3 sig, 4 plt, 5 sim, 6 usr, int
         0     Increase simulation length by extra, int
 3.00e+000     Scale factor for plots, sfac, float
       457     Seed for simulations, iseed, int
        50     Number of plot grid points, ngrid, int
         0     Statistics not computed if kilse=1, int
```

We then edit control.dat to read as follows.

control.dat

```
11114000.in1 11114000.res     0.0e0     0.0e0         0              454589
11114000.in2 11114000.mu      0.0e0     0.0e0         0              454589
```

```
11114000.in3 11114000.sig     0.0e0     0.0e0         0              454589
11114000.in4 11114000.plt     0.0e0     0.0e0         0              454589
11114000.in5 11114000.sim     0.0e0     0.0e0         0              454589
```

Running SNP we get

summary.dat

```
SNP Restart
-------------------------------------------------------------------------------
     pname   output_file      fitted_model   p opt iter   obj_func        bic
-------------------------------------------------------------------------------
  SpotRate  11114000.res
  SpotRate   11114000.mu
  SpotRate  11114000.sig
  SpotRate  11114000.plt
  SpotRate  11114000.sim
```

The output file 11114000.sim contains a simulation of length n=834 from the fit using iseed=457 as the seed and the segment x_{t-1} of the series from 1 to drop=14 to start the simulation off. This simulation is plotted in the fourth panel of Figure 2. The units in .sim, as well as .plt, .sig, and .mu, are in the original units of the data; that is, in the same units as the raw data $\{\tilde{y}_t\}_{t=1}^{n}$.

To be more specific, the values in .mu are estimates of the conditional expectations $\mathcal{E}(\tilde{y}_t|\tilde{y}_{t-1},\ldots,\tilde{y}_{t-L})$ stored end-to-end. The values in .sig are vechs of estimates of the conditional variances $\mathrm{Var}(\tilde{y}_t|\tilde{y}_{t-1},\ldots,\tilde{y}_{t-L})$ stored end-to-end. The values in .res are estimates of the residuals $[\mathrm{Var}(\tilde{y}_t|\tilde{y}_{t-1},\ldots,\tilde{y}_{t-L})]^{-1/2}\,[\tilde{y}_t - \mathcal{E}(\tilde{y}_t|\tilde{y}_{t-1},\ldots,\tilde{y}_{t-L})]$ stored end-to-end. See the earlier remarks regarding the effect of the spline transformation because the bias in the values in the .mu, .sig, and .res files can be substantial without it for some specifications.

The output file 10314000.plt contains data with which to plot the one-step-ahead density $f_K(\tilde{y}_t|x_{t-1},\hat{\theta})$, conditional on the values for

$$x_{t-1} = (\tilde{y}_{t-1}, \tilde{y}_{t-2}, \cdots, \tilde{y}_{t-L})$$

specified by cond in the DATA DESCRIPTION block of the parmfile. Figure 4 is an example of such a plot. The density displays the typical shape for data from financial markets: peaked with fatter tails than the normal with a bit of asymmetry. In this case cond = 0 so the lags in x_{t-1} are all set to the unconditional mean of the data. These were the values used for Figure 4. To plot the density conditioned on a different x, set cond differently as discussed above.

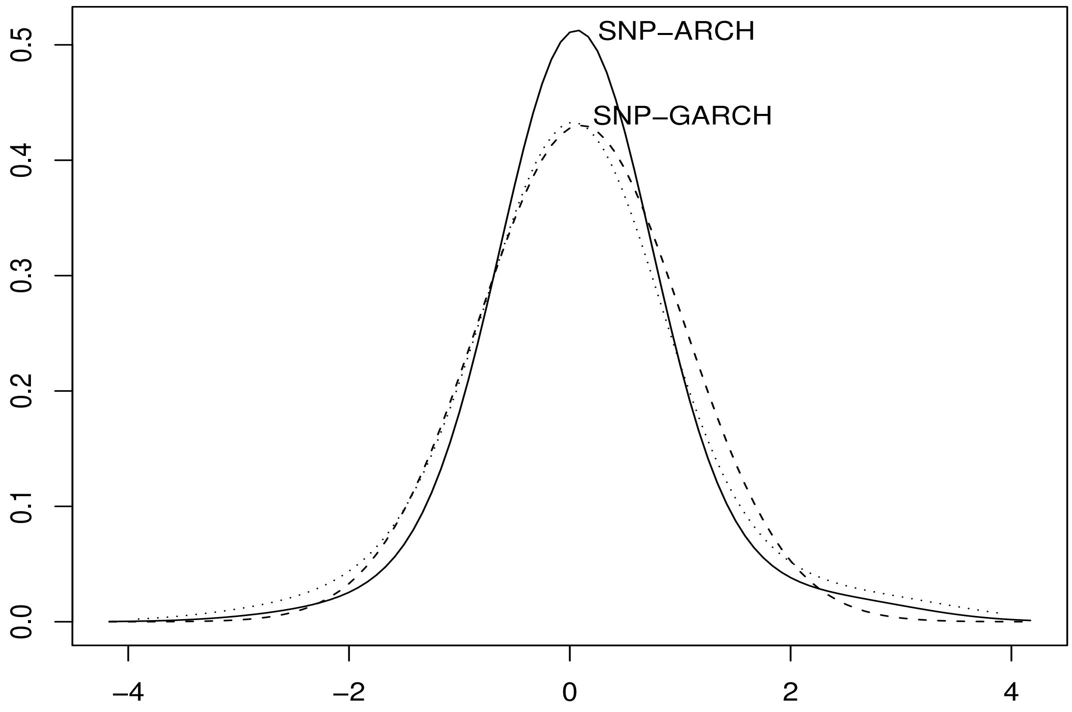

Figure 4. Conditional density. The data are weekly \$/DM spot exchange rates from 1975 to 1990, Friday's quote, expressed as percentage change from the previous week. The solid line labeled SNP-ARCH is a plot of an SNP fit with $(L_u, L_g, L_r, L_p, K_z, I_z, K_x, I_x) = (1, 0, 3, 1, 4, 0, 0, 0)$ and all lags set to the unconditional mean of the data; the dotted line is a normal with the same mean and variance. The dashed line labeled SNP-GARCH is a plot of an SNP fit with $(L_u, L_g, L_r, L_p, K_z, I_z, K_x, I_x) = (1, 1, 1, 1, 4, 0, 0, 0)$ and all lags set to the unconditional mean of the data.

A .plt file contains: first, $\hat{\mathcal{E}}(\tilde{y}_t|x_{t-1})$, M values; second, of $\widehat{\mathrm{Var}}(\tilde{y}_t|x_{t-1})$ stored columnwise, M*M values; third, the grid increment, M values; thereafter, $\tilde{y}_t$ and $f_K(\tilde{y}_t|x_{t-1}, \hat{\theta})$ written end to end, there are (M+1)*(2*ngrd+1)**M of these. Total file length is M+M*M+M+ (M+1)*(2*ngrd+1)**M. The reason for prepending $\hat{\mathcal{E}}(\tilde{y}_t|x_{t-1})$, $\widehat{\mathrm{Var}}(\tilde{y}_t|x_{t-1})$, and the grid increment is that one often wants to compute marginal distributions and compare the plot with the normal distribution at the same mean and variance as in Figure 4.

A .plt file can also be used for quadrature using a Riemann rule because the value of the density, the points at which is it is evaluated, and the increments between points are in the file. However, a Gauss-Hermite quadrature rule is far more efficient. An implementation is contained in the distribition and is discussed at the end of Section 5. Briefly its use and output are the same as for plotting (task=4) except that one codes task=6 and ngrid controls the order of the Gauss-Hermite quadrature rule. The output for the quadature rule (task=6) is the same as the plot output except that $\hat{\mathcal{E}}(\tilde{y}_t|x_{t-1})$, $\widehat{\mathrm{Var}}(\tilde{y}_t|x_{t-1})$, and the grid increment are not prepended. File length is (M+1)*ngrid**M.

Comments, which are lines that begin with a #, may be added to the end of the PARMFILE HISTORY block. One might, for instance, describe the data here. These comments are copied to output parmfiles. The first eleven lines that begin with a # are the property of the program. Don't touch them.

One last remark: As seen from Table 3, parameter estimates do not change much when fitting with and without the spline. Therefore, it is usually adequate to use the fit in hand as the input parameter file and to set the fnew, fold, and nstart parameters of control.dat to zero when changing the spline parameter or changing stran ($= \sigma_{\mathrm{tr}}$).

4.3 Running on a Parallel Machine

The parallel version of SNP, which is `snp_mpi`, is similar to the serial version, which is `snp`, but with the annoying quirk that path names must be absolute, which is caused by restrictions imposed by the LAM implementation of MPI for which the code was originally written. The way the absolute path name requirement is handled is to supply a header `pathname.h` that contains the absolute path name and builds it into the code at compile time. This header is generated automatically by the makefiles named `makefile.mpi.version` that are included with the distribution. For instance, `makefile.mpi.OpenMPI_1.4` is for a 48 core AMD box running CentOS 5. These makefiles assume that the build occurs in the same directory in which data, parmfiles, etc. are found. For our example, here is `pathname.h` which was generated automatically by the makefile:

```
#define PATHNAME "/home/arg/r/snp_develop/test_mpi"
```

The `control.dat` file is the same as for the serial version but with one caveat due to the fact that the host node only parcels out to the sub nodes lines of `control.dat` whose parmfile requests a fit (`task=0`), it does all other tasks itself. While the host node is doing a non-fit task, it cannot parcel out fits to sub nodes nor receive results from sub nodes.

This means that either the control.dat should only have lines in it whose input parmfiles request fits or that non-fit task lines should be evenly spaced within control.dat.

Running on a parallel machine requires initiation of MPI prior to execution. This is handled by shell scripts emm_mpi.version.sh included with the distribution: This is what emm_mpi.lam_7.0.sh looks like.

```
#! /bin/sh

# This shell script works for an 8 box cluster with 2 mono core CPUs
# per box running LAM Version 7.0.  The host node is named n0 and the
# subnodes are named n1, n2, n3, n4, n5, n6, n7.

echo n0  > lamhosts
echo n1 >> lamhosts
echo n2 >> lamhosts
echo n3 >> lamhosts
echo n4 >> lamhosts
echo n5 >> lamhosts
echo n6 >> lamhosts
echo n7 >> lamhosts

test -f snp_mpi.err  && mv -f snp_mpi.err  snp_mpi.err.bak
test -f snp_mpi.out  && mv -f snp_mpi.out  snp_mpi.out.bak

rm -f core core.*

lamboot -v lamhosts

RC=$?

case $RC in
  0) ;;
  1) exit 1;;
  esac

make -f makefile.mpi.lam_7.0 >snp_mpi.out 2>&1 && \
  mpirun -v -O -D -s h N N \
  ${PWD}/snp_mpi >>snp_mpi.out 2>snp_mpi.err

RC=$?

case $RC in
  0) exit 0 ;;
  esac
exit 1;
```

Also included with the distribution are shell scripts and makefiles for Version 7.1 of LAM and for Versions 1.4 and 2.1 of OpenMPI.

The results of a run are a set of files similar to those for the serial version. The two files that change are summary.dat and detail.dat. The changes are caused by the fact that results are printed when they are received from the sub nodes. This order is different than the ordering in control.dat. Also, in detail.dat, parmfiles are printed when read and results printed when received: there can be quite some distance between these two events. Extra labelling is provided to help sort things out.

5 Adapting the SNP Code

The options described in Section 4 do not cover every contingency. The program has been structured so that it is easy to modify to accomplish tasks that are not covered by the built-in options. In this section, we discuss how the code is modified to plot a conditional variance function, print statistics, and to implement Gauss-Hermite quadrature.

5.1 Plots of the Conditional Variance Function

Plots of the estimated conditional variance function

$$\begin{aligned}\widehat{\mathrm{Var}}(y|x) &= \int [y - \mathcal{E}(y|x)][y - \mathcal{E}(y|x)]' f_K(y|x, \hat{\theta})\, dy \\ \hat{\mathcal{E}}(y|x) &= \int y f_K(y|x, \hat{\theta})\, dy\end{aligned}$$

against the most recent lag y_{t-1} with all other lags put to their unconditional means are of interest in studying the "leverage effect" which is the tendency for variance to be higher subsequent to a down-tick than an up-tick in equities markets.

A difficulty with such plots is that they may be misleading because a history $x_{t-1} = (y_{t-1}, \ldots, y_{t-L})$ with each y_{t-i} set to the unconditional mean is not representative of any point in the data; it is too smooth. The conditional variance function $\widehat{\mathrm{Var}}(y|x)$ depends on the entire vector x_{t-1} and smoothness may alter the shape of the graph. One solution is to compute a plot for each point in the data and average them. Such a plot is shown in Figure 5.

The flexibility to perform such a computation is provided by having a base class, ancillary_base, which is presented in snp_base.h, and letting the classes that compute plots, simulations, etc. inherit from it. The objects used in snp.cpp for these computations are all references to classes of type ancillary_base. The antecedents of these references are determined by typedefs in snpusr.h.

The base class ancillary_base from snp_base.h is

snp_base.h

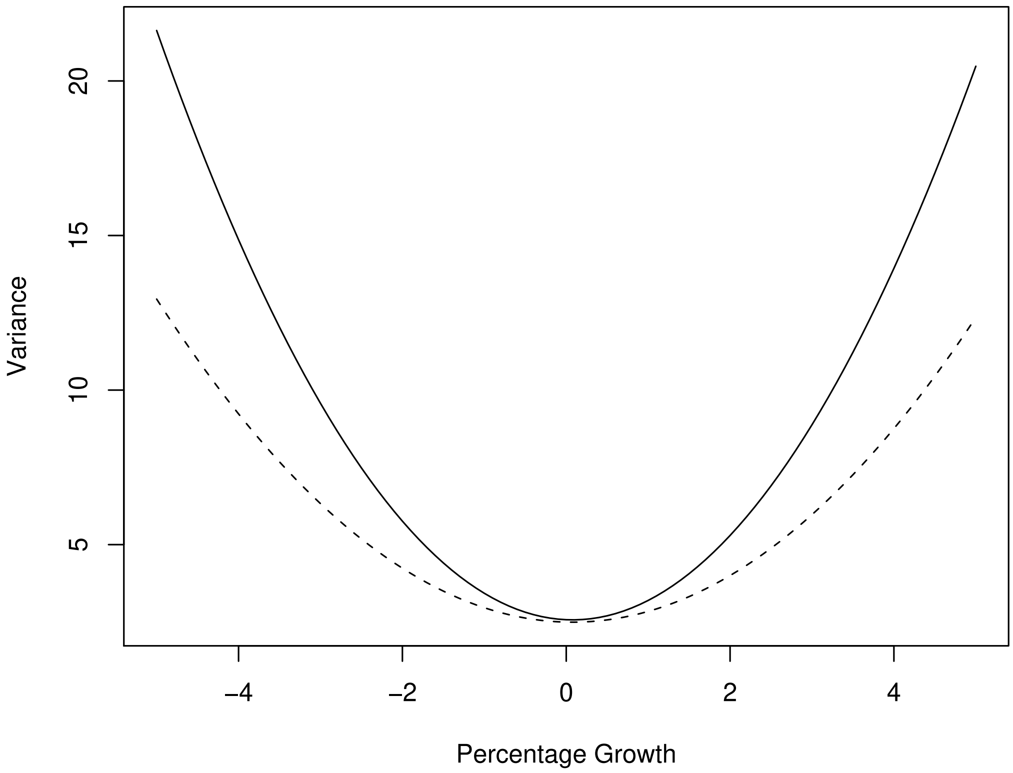

Figure 5. The conditional variance function. The data are weekly \$/DM spot exchange rates from 1975 to 1990, Friday's quote, expressed as percentage change from the previous week. Shown is the average over all $x_{t-1} = (y_{t-1} + \delta, \ldots, y_{t-L})$ in the data of the conditional variance $\widehat{\mathrm{Var}}(y_t | y_{t-1+\delta}, \ldots, y_{t-L})$ plotted against δ. The solid line corresponds to an SNP fit with $(L_u, L_g, L_r, L_p, K_z, I_z, K_x, I_x) = (1, 0, 3, 1, 4, 0, 0, 0, 0)$ and the dotted line to a fit with $(L_u, L_g, L_r, L_p, K_z, I_z, K_x, I_x) = (1, 1, 1, 1, 4, 0, 0, 0, 0)$.

```
class ancillary_base {
public:
  virtual void set_XY(const scl::realmat* x, const scl::realmat* y) = 0;
  virtual bool initialize(std::ostream* out_stream) = 0;
  virtual bool initialize(std::string out_filename) = 0;
  virtual bool calculate() = 0;
  virtual bool finalize() = 0;
  virtual ~ancillary_base() { }
};
```

The typedefs from snpusr.h (modified so that putting task = 6 in the input parmfile will point to the class leverage that we shall write) and the declaration of the class leverage that we intend to write are as follows:

snpusr.h

```
#include "snp_base.h"
#include "libsnp.h"

class datread;
class resmusig;
class plot;
class simulate;
class leverage;
class quadrature;
class rhostats;

typedef datread   datread_type;
typedef resmusig  residual_type;
typedef resmusig  mean_type;
typedef resmusig  variance_type;
typedef plot      plot_type;
typedef simulate  simulate_type;
typedef leverage  user_type;

class leverage : public ancillary_base {
private:
  optparms opm;
  datparms dpm;
  tranparms tpm;
  libsnp::snpden f;
  libsnp::afunc af;
  libsnp::ufunc uf;
  libsnp::rfunc rf;
  std::ostream& detail;
  const trnfrm* tr;
  const scl::realmat* X;
  const scl::realmat* Y;
  std::ostream* os;
public:
  leverage(optparms op, datparms dp, tranparms tp, libsnp::snpden fn,
       libsnp::afunc afn, libsnp::ufunc ufn, libsnp::rfunc rfn,
       libsnp::afunc afm, libsnp::ufunc ufm, libsnp::rfunc rfm,
       std::ostream& dos, const trnfrm* trn);
  void set_XY(const scl::realmat* x, const scl::realmat* y);
  bool initialize(std::ostream* out_stream);
  bool initialize(std::string out_filename) { return true; }
  bool calculate();
  bool finalize();
};
```

All classes in snpusr.h have the same form and the same form of constructor.

What we need to do now is to code the member function calculate to compute the conditional variance for a sequence of perturbed x_{t-1}, average them, and write them to os. These are straightforward modifications to the class resmusig in snpusr.cpp.

The only thing that is tricky is that, because the classes afunc, ufunc, and rfunc keep track of their own lags and recursions, their main member (an overloaded application operator) can only be called once per observation in a serial loop over the data. In general usage this requires that one must take care to retain results from the previous call to ufunc for input to rfunc. In our special usage, we intend to make multiple calls for delta added to x_{t-1}, which would wreck havoc with the recursions if not handled properly. The approach adopted in the code below is to take copies of the objects afunc, ufunc, and afunc, evaluate them at the perturbed x_{t-1} to get Σ_t, and then discard them. This works because taking copies will not disturb the state of the object copied. It is also a relatively cheap operation because the only actual work involved in the copy is copying the pieces of the parameter vector θ and x_t that each object contains.

snpusr.cpp

```
leverage::leverage(optparms op, datparms dp, tranparms tp, snpden fn,
  afunc afn, ufunc ufn, rfunc rfn,  afunc afm, ufunc ufm, rfunc rfm,
  ostream& dos, const trnfrm* trn)
: opm(op), dpm(dp), tpm(tp), f(fn), af(afn), uf(ufn), rf(rfn),
  detail(dos), tr(trn)
{ }

void leverage::set_XY(const realmat* x, const realmat* y)
{
  if (rows(*x) != rows(*y)) error("Error, leverage, row dim of x & y differ");
  if (cols(*x) != cols(*y)) error("Error, leverage, col dim of x & y differ");
  X = x;  Y = y;
}

bool leverage::initialize(ostream* out_stream)
{
  os = out_stream;
  return (*os).good();
}

bool leverage::calculate()
{
  if (Y==0||X==0) error("Error, leverage, data not initialized");
  if (dpm.M != rows(*Y) || dpm.M != rows(*X) || dpm.M != f.get_ly())
    error("Error, leverage, this should never happen");

  INTEGER ngrid = 50;
  realmat delta(1,2*ngrid+1);
  realmat average(f.get_lR(),2*ngrid+1,0.0);

  realmat dawa0,dawA;
  realmat duwb0, duwb0_lag;
  kronprd duwB, duwB_lag;
  realmat dRwb0,dRwB;
  realmat dRwRparms;

  realmat y(dpm.M,1);
  realmat x(dpm.M,1);
  realmat u(dpm.M,1);
  realmat y_lag(dpm.M,1);
  realmat x_lag(dpm.M,1);
```

```
realmat u_lag(dpm.M,1);

af.initialize_state();
uf.initialize_state();
rf.initialize_state();

for (INTEGER i=1; i<=dpm.M; ++i) {
  x[i] = (*X)(i,1);
}

u = uf(x,duwb0,duwB);

for (INTEGER t=3; t<=dpm.drop; ++t) {

  for (INTEGER i=1; i<=dpm.M; ++i) {
    y[i] = (*Y)(i,t);
    x[i] = (*X)(i,t);
    y_lag[i] = (*Y)(i,t-1);
    x_lag[i] = (*X)(i,t-1);
  }

  u_lag = u;
  duwb0_lag = duwb0;
  duwB_lag = duwB;

  u = uf(x_lag,duwb0,duwB);

  f.set_R(rf(x_lag,u_lag,x_lag,duwb0_lag,duwB_lag,dRwb0,dRwB,dRwRparms));
  f.set_a(af(x_lag,dawa0,dawA));
  f.set_u(u);

}

for (INTEGER t=dpm.drop+1; t<=dpm.n; ++t) {

  for (INTEGER i=1; i<=dpm.M; ++i) {
    y_lag[i] = y[i];
    x_lag[i] = x[i];
    y[i] = (*Y)(i,t);
    x[i] = (*X)(i,t);
  }

  u_lag = u;
  duwb0_lag = duwb0;
  duwB_lag = duwB;

  for (INTEGER grid=-ngrid; grid<=ngrid; ++grid) {

    snpden fd = f;
    afunc afd = af;
    ufunc ufd = uf;
    rfunc rfd = rf;

    INTEGER idx = grid + ngrid + 1;
    delta[idx] = 5.0*REAL(grid)/REAL(ngrid);

    realmat yd = y_lag;
    yd[1] += delta[idx];

    realmat xd = yd;

    if (tpm.squash == 1) {
```

```
        tr->spline(xd);
      } else if (tpm.squash == 2) {
        tr->logistic(xd);
      }

      fd.set_R(rfd(xd,u_lag,xd,duwb0_lag,duwB_lag,dRwb0,dRwB,dRwRparms));
      fd.set_a(afd(xd,dawa0,dawA));
      fd.set_u(ufd(xd,duwb0,duwB));

      realmat mu, sig;
      fd.musig(mu,sig);

      tr->unscale(sig);

      for (INTEGER j=1; j<=sig.get_cols(); ++j) {
        for (INTEGER i=1; i<=j; ++i) {
          INTEGER ij = (j*(j-1))/2 + i;
          average(ij,idx) += sig(i,j)/REAL(dpm.n-dpm.drop);
        }
      }
    }

    u = uf(x_lag,duwb0,duwB);

    f.set_R(rf(x_lag,u_lag,x_lag,duwb0_lag,duwB_lag,dRwb0,dRwB,dRwRparms));
    f.set_a(af(x_lag,dawa0,dawA));
    f.set_u(u);
  }

  for (INTEGER grid=-ngrid; grid<=ngrid; ++grid) {
    INTEGER idx = grid + ngrid + 1;
    (*os) << delta[idx] << ' ';
    for (INTEGER i=1; i<=f.get_lR(); ++i) (*os) << average(i,idx) << ' ';
    (*os) << '\n';
  }

  return (*os).good();
}

bool leverage::finalize()
{
  return (*os).good();
}
```

Note that in the constructor `afm, ufm, rfm` are unused. This will usually be the case. In the distributed code, they are only used by class `rhostats`, presented in `snpusr.h` and defined in `snpusr.cpp`. Also, `bool initialize(std::string out_filename)` is usually unused. It is available to provide a stem if files with various extensions need to be created; class `rhostats` uses this feature. As with class `leverage`, just described, and class `quadrature`, described below, `rhostats` is invoked by setting `type=6` and modifying the `user_type` line in `snpusr.h` as follows.

```
typedef rhostats  user_type;
```

What `rhostats` does is write $\hat{\rho}$, $n\hat{\mathcal{I}}$, $n\hat{\mathcal{J}}$ (the Hessian), ns_n etc. to files with extension `rho`, `infm`, etc. The format is appropriat for `vecread` in `libscl`; i.e. first line the number of rows, second line the number of columns, and the remaining lines the vec of the matrix.

The distribution also includes code to implement Gauss-Hermite quadrature. The use of the code is exaclty the same as producing a .plt file with type=4. The changes one makes relative to the foregoing description of user modifications are to change

```
typedef leverage  user_type;
```

to

```
typedef quadrature  user_type;
```

in snpusr.h, and recompile. Then set type=6 and ngrid=9 in the parmfile and run. Some experimentation will be required to get the correct order of the quadrature rule, which is determined by ngrid. In the file detail.dat are shown the relative errors in the computations of the integal of the density, of the mean, and of the variance. One can experiment with various values of ngrid until these relative errors are as small as possible.

The determination of the conditioning set is exactly the same as for .plt files. In the output file are first the abcissae and then the weight for each quadrature point, stored end-to-end. This is exactly the same as abcissae and density are stored in a .plt file. An expectation $\mathcal{E}[g(y)]$ with respect to the SNP density is approximated as

$$\mathcal{E}[g(y)] = \sum_{j=1}^{npts} g(\text{abcissa[j]}) * \text{weight[j]}$$

R code illustrating the reading and use of both the plot output file bivar.plt file and the Gauss-Hermite quadrature output file bivar.usr follows:

```
M <- 2

tmp  <- scan("bivar.plt")
lplt <- length(tmp) - (M+M*M+M)
npts <- lplt/(M+1)

mu   <- tmp[seq(from=1,to=M)]
sig  <- matrix(tmp[seq(from=M+1,to=M+M*M)],ncol=M,nrow=M,byrow=F)
yinc <- tmp[seq(from=M+M*M+1,to=M+M*M+M)]
plt  <- tmp[seq(from=M+M*M+M+1,to=M+M*M+M+lplt)]
plt  <- matrix(plt,nrow=M+1,ncol=npts,byrow=F)
```

```
sum <- 0
for (j in 1:npts) sum = sum + plt[3,j]*yinc[1]*yinc[2]
print(sum)

sum <- 0
for (j in 1:npts) {
  sum = sum + plt[3,j]*yinc[1]*yinc[2]*(exp(plt[1,j])+cos(plt[2,j]))
}
print(sum)

quad <- scan("bivar.usr")
npts <- length(quad)/(M+1)

quad  <- matrix(quad,nrow=M+1,ncol=npts,byrow=F)

sum <- 0
for (j in 1:npts) sum = sum + quad[3,j]
print(sum)

sum <- 0
for (j in 1:npts) {
  sum = sum + quad[3,j]*(exp(quad[1,j])+cos(quad[2,j]))
}
print(sum)
```

In each instance the first sum should agree with the computation in detail.dat if the .plt and .usr files have been read correctly.

6 References

Ahn, Dong-Hyun, Robert F. Dittmar, and A. Ronald Gallant (2002), "Quadratic Term Structure Models: Theory and Evidence," *The Review of Financial Studies* 15, 243–288.

Ahn, Dong-Hyun, Robert F. Dittmar, A. Ronald Gallant, and Bin Gao (2003), "Purebred or Hybrid?: Reproducing the Volatility in Term Structure Dynamics," *Journal of Econometrics* 116, 147–180.

Akaike, H. (1969), "Fitting Autoregressive Models for Prediction," *Annals of the Institute of Statistical Mathematics* 21, 243–247.

Aldrich, Eric M., and A. Ronald Gallant (2011), "Habit, Long-Run Risks, Pr*ospect*? A Statistical Inquiry," *Journal of Financial Econometrics* forthcoming.

Bansal, Ravi, A. Ronald Gallant, Robert Hussey, and George Tauchen (1995), "Nonparametric Estimation of Structural Models for High-Frequency Currency Market Data," *Journal of Econometrics* 66, 251–287.

Bollerslev, Tim (1986), "Generalized Autoregressive Conditional Heteroskedasticity," *Journal of Econometrics* 31, 307–327.

Chernov, Mikhail, A. Ronald Gallant, Eric Ghysels, and George Tauchen (2003), "Alternative Models for Stock Price Dynamics," *Journal of Econometrics* 116, 225–257 .

Coppejans, Mark, and A. Ronald Gallant (2002), "Cross-Validated SNP Density Estimates," *Journal of Econometrics* 110, 27–65.

Engle, Robert F. (1982), "Autoregressive Conditional Heteroskedasticity with Estimates of the Variance of United Kingdom Inflation," *Econometrica* 50, 987–1007.

Engle, Robert F., and Gloria Gonzales-Rivera (1991), "Semiparametric ARCH Models," *Journal of Business and Economic Statistics,* 9, 345–360.

Engle, R. F, and K. F. Kroner (1995), "Multivariate Simultaneous Generalized ARCH," *Econometric Theory* 11, 122–150.

Fenton, Victor M., and A. Ronald Gallant (1996), "Qualitative and Asymptotic Performance of SNP Density Estimators," *Journal of Econometrics* 74, 77–118.

Gallant, A. Ronald, Lars Peter Hansen, and George Tauchen (1990), "Using Conditional Moments of Asset Payoffs to Infer the Volatility of Intertemporal Marginal Rates of Substitution," *Journal of Econometrics* 45, 141–180.

Gallant, A. Ronald, David A. Hsieh, and George E. Tauchen (1991), "On Fitting a Recalcitrant Series: The Pound/Dollar Exchange Rate, 1974–83," in William A. Barnett, James Powell, George E. Tauchen, eds. *Nonparametric and Semiparametric Methods in Econometrics and Statistics, Proceedings of the Fifth International Symposium in Economic Theory and Econometrics,* Cambridge: Cambridge University Press, Chapter 8, 199–240.

Gallant, A. Ronald, David Hsieh, George Tauchen (1997), "Estimation of Stochastic Volatility Models with Diagnostics," *Journal of Econometrics* 81, 159–192.

Gallant, A. Ronald, Chien-Te Hsu, and George Tauchen (1999) "Using Daily Range Data to Calibrate Volatility Diffusions and Extract the Forward Integrated Variance" *The Review of Economics and Statistics* 81(4), 617–631.

Gallant, A. Ronald, and Jonathan R. Long (1997), "Estimating Stochastic Differential Equations Efficiently by Minimum Chi-Square," *Biometrika* 84, 125–141.

Gallant, A. Ronald, and Robert E. McCulloch (2009), "On the Determination of General Scientific Models," *Journal of the American Statistical Association* 104, 117–131

Gallant, A. Ronald, and Douglas W. Nychka (1987), "Seminonparametric Maximum Likelihood Estimation" *Econometrica* 55, 363–390.

Gallant, A. Ronald, Peter E. Rossi, and George Tauchen (1992), "Stock Prices and Volume," *The Review of Financial Studies* 5, 199–242.

Gallant, A. Ronald, Peter E. Rossi, and George Tauchen (1993), "Nonlinear Dynamic Structures," *Econometrica* 61, 871–907.

Gallant, A. Ronald, and George Tauchen (1989), "Seminonparametric Estimation of Conditionally Constrained Heterogeneous Processes: Asset Pricing Applications," *Econometrica* 57, 1091–1120.

Gallant, A. Ronald, and George Tauchen (1992), "A Nonparametric Approach to Nonlinear Time Series Analysis: Estimation and Simulation," in David Brillinger, Peter Caines, John Geweke, Emanuel Parzen, Murray Rosenblatt, and Murad S. Taqqu eds. *New Directions in Time Series Analysis, Part II.* New York: Springer-Verlag, 71-92.

Gallant, A. Ronald, and George Tauchen (1996), "Which Moments to Match?," *Econometric Theory* 12, 657–681.

Gallant, A. Ronald, and George Tauchen (1997), "Estimation of Continuous Time Models for Stock Returns and Interest Rates" *Macroeconomic Dynamics* 1, 135–168.

Gallant, A. Ronald, and George Tauchen (1998), "Reprojecting Partially Observed Systems with Application to Interest Rate Diffusions," *Journal of the American Statistical Association* 93, 10-24.

Gallant, A. Ronald, and George Tauchen (1999), "The Relative Efficiency of Method of Moments Estimators," *Journal of Econometrics* 92, 149–172.

Gallant, A. Ronald, and George Tauchen (1999), "The Relative Efficiency of Method of Moments Estimators," *Journal of Econometrics* 92, 149–172.

Gallant, A. Ronald, and George Tauchen (2004), "Simulated Score Methods and Indirect Inference for Continuous-time Models," in Yacine Aït-Sahalia and Lars Peter Hansen, eds. (2004), *Handbook of Financial Econometrics,* Elsevier/North-Holland, Amsterdam, forthcoming.

Hannan, E. J. (1987), "Rational Transfer Function Approximation," *Statistical Science* 2, 1029–1054.

Potscher, B. M. (1989), "Model Selection Under Nonstationarity: Autoregressive Models and Stochastic Linear Regression Models," *Annals of Statistics* 17, 1257–1274.

Robinson, P. M. (1983), "Nonparametric Estimators for Time Series," *Journal of Time Series Analysis* 4, 185–207.

Schwarz, G. (1978), "Estimating the Dimension of a Model," *Annals of Statistics* 6, 461–464.

Tauchen, George (1985), "Diagnostic Testing and Evaluation of Maximum Likelihood Models," *Journal of Econometrics* 30, 415–443.

Tauchen, George (1997), "New Minimum Chi-Square Methods in Empirical Finance," in: D. Kreps and K. Wallis, eds., Advances in Econometrics, Seventh World Congress, Cambridge University Press, Cambridge UK, 279–317.

Tauchen, George, and Robert Hussey (1991), "Quadrature-Based Methods for Obtaining Approximate Solutions to Nonlinear Asset Pricing Models." *Econometrica* 59, 371–396.