GSM: A Program for Determining General Scientific Models

Version 1.5

User's Guide [1]

A. Ronald Gallant
Penn State University
Department of Economics
Durham NC 27708-0120 USA

Robert E. McCulloch
University of Texas
Graduate School of Business
Austin TX 78712-1175 USA

January 2005
Last revised December 2013

---

[1] Research supported by the National Science Foundation. The code and this guide are available at http://www.aronaldg.org.

Abstract

GSM implements a methodology proposed by Gallant and McCulloch (2009) for the statistical analysis of models derived from scientific considerations that exhibit five characteristics: (1) a likelihood is not available; (2) the model can be simulated; (3) prior information is available; (3) a portion of the prior information is expressed in terms of functionals of the model that are not easily converted into an analytic prior on model parameters but can be computed from a simulation of the model; and (5) a parametric statistical model for the data, determined without reference to the scientific model, is either known from the literature or can be determined. The latter is nearly always the case because richly parameterized statistical models can be accommodated.

Their proposal is a computationally intensive Bayesian MCMC modeling strategy for estimation and inference together with methods for assessing model adequacy. An important adjunct to the method is that the implied map from the parameters of the scientific model to the parameters of the statistical model and to functionals of both the scientific and statistical models becomes available. This map is a powerful tool for eliciting the properties of the scientific model and understanding the reasons for model success or failure.

GSM, coded in C++, is available at http://www.aronaldg.org The code is provided at no charge for research purposes without warranty. Two versions are offered: a serial version and a parallel version.

The purpose of this Guide is to review the underlying methodology and explain and illustrate the use of the program. After explaining the methodology, use of the program is illustrated with a stochastic volatility model that is determined using both the serial and parallel versions of the program. The 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 GSM method.

Contents

1 Introduction

Models derived from scientific considerations often exhibit four characteristics: (1) a likelihood is not available because, for instance, the state vector is only partially observed, the model's output is continuous but observed discretely, or the model contains latent variables; (2) the model can be simulated; (3) prior information is available; (4) a portion of the prior information is expressed in terms of functionals of the model that cannot be converted into an analytic prior on model parameters, for example, a prior on the expectation of the solution of a system of nonlinear equations involving model variables and parameters;. Examples of such models are the SEIR model from epidemiology for which the state variable is the proportion of a population that is susceptible, exposed, infected, and recovered from a disease whereas the data are from case reports that report only those infected (Olsen and Schaffer, 1990); continuous and discrete time stochastic volatility models of speculative markets from finance (Ghysels, Harvey, and Renault, 1995); general equilibrium models from economics (Aldrich and Gallant, 2011); and compartment models from pharmacokinetics (Mallet, Mentré, Steimer, and Lokiec, 1988). Aldrich and Gallant (2011) is an application of GSM; they compare three general equilibrium asset pricing models: habit, long run risks, and prospect theory.

GSM is designed for estimation and inference for models that at least exhibit characteristics (1) and (2) and for which, in addition, it is the case that (5) an adequate statistical model for simulations from the scientific model is available. Because richly parameterized statistical models are admissible in this connection, an adequate statistical model can nearly always be found.

In theory the statistical model should be an adequate model for simulations from the scientific model. However, Gallant and McCulloch (2009) argue that in practice one might well prefer that the statistical model be an accurate model of the data even if doing so causes it to fail to reflect some features of the scientific model. Also, if the maximum likelihood of the statistical model is easy to compute, the run times are considerably reduced and accuracy is improved. This, also, should be a consideration in selecting a statistical model.

Given (5), we can construct a map from the parameters of the scientific model to those of the statistical model such that a point in the parameter space of the scientific model and its image under the map both correspond to the same data generating process. Typically the parameters of the statistical model will live in a higher dimensional space than that of the scientific model. The scientific model may therefore be viewed as a prior on the statistical model that has support entirely on the manifold that is the image of the map. Scientific prior information will then generate preferences on the manifold. The GSM methodology allows the scientific prior information to be expressed either directly on the parameters of the scientific model or on functionals of the scientific model that can be evaluated via simulation.

The discovery of the mapping from the parameters of the scientific model to those of the statistical model, which is an intermediate step of the methods proposed here, is often itself of scientific interest. For instance, the statistical model must, perforce, be expressed entirely in terms of observables whereas scientific models often contain unobservables. Having a mapping from the subset of the parameters that control the unobservable features of the scientific model to the parameters of a statistical model consisting entirely of observables can be extremely helpful in understanding the observable consequences of changes in a model's unobservable internal structure. The utility of this approach can be extended by using the same methods to find the map from the parameters of the scientific model to functionals of both the scientific and statistical models.

A Bayesian approach suggests itself for problems that exhibit the five characteristics just listed because the methodology gracefully accepts prior information into the analysis and, for dynamic models, does not require growth conditions on model output or data that are often counter factual. Moreover, the estimates of parameter uncertainty are credible. That is, the asymptotics on which frequentist methods rely are often grossly inaccurate for the class of problems considered here.

The implementation relies on modern object oriented programming methods, modern data structures, and a discretization at a critical point in the computations. Bringing these elements to bear on the problem seems to be both novel to this work and essential to success.

2 The GSM Method

We shall use the notational conventions of time series analysis because most models for which GSM is called for are dynamic. This is in no way essential because the results apply equally well to other data structures with a few obvious changes to notation.

3 Scientific and Statistical Models

Let the transition density of the scientific model be denoted as $p(y_t|x_{t-1},\theta),\ \theta \in \Theta$, where $x_{t-1} = (y_{t-1},\dots,y_{t-L})$ if Markovian and $x_{t-1} = (y_{t-1},\dots,y_1)$ if not. We assume that there is no direct information about $p(\cdot|\cdot,\theta)$. All that we can do is simulate data from $p(\cdot|\cdot,\theta)$ for given θ. If the model produces ergodic output, then a single long simulation for each setting of θ suffices for our purposes. If not, then many independently simulated replicates of the data are used.

Because we do not have access to $p(\cdot|\cdot,\theta)$, there is no direct way to compute the likelihood. Our approach is to find a parametric family of distributions that is capable of representing the process $\{y_t\}$:

ASSUMPTION 1 We assume that there is a transition density $f(y_t|x_{t-1},\eta),\ \eta \in H$, and that there is a one-to-one map $g : \theta \mapsto \eta$ such that

$$p(y_t|x_{t-1},\theta) = f(y_t|x_{t-1},g(\theta)) \qquad \theta \in \Theta \tag{1}$$

and that the form of $f(\cdot|\cdot,\eta)$ is known.

When we need a likelihood based on the unknown $p(\cdot|\cdot,\theta)$, we substitute $f(\cdot|\cdot,g(\theta))$. The model $f(\cdot|\cdot,\eta)$ is a statistical description of the observed data that we call the statistical model. Often this model will be known from the literature. In other cases it must be determined as part of the analysis. As richly parameterized models are permitted, success in finding an acceptable statistical model can be anticipated. It is to be emphasized that we only use the statistical model to fit large simulations from the scientific model (Section 4) or when augmented by a strong prior dictated by the scientific model (Section 5) so that the fact that the data may be too sparse to support it is not a consideration.

When Assumption 1 is satisfied the likelihood is exactly that implied by the scientific model. When Assumption 1 is violated the likelihood is different from that implied by the scientific model. To use Poirier's (1988) terminology, when Assumption 1 holds one is looking at the world through the window implied by the scientific model. When Assumption 1 is violated one is looking at the world through a different window.

One might deliberately choose to violate Assumption 1. For example, if satisfaction of (1) leads to a statistical model $f(y|x, \theta)$ with characteristics markedly different from what is known about the distribution of the data, one might deliberately opt for a simplification that does not exhibit these characteristics. The issues that arise in this connection are discussed in Subsection 4.4.

A key motivation for implementing a Bayesian approach to the problem is the importance of using prior information. This can be critical when data are sparse. When data are sparse, prior information can be used to fill in model features about which the data says little but the literature says much thereby enabling extraction of features about which the data are informative.

The scientific model is built using subject matter knowledge. Thus, we expect that real prior information is available. This prior information may be expressible either in terms of elements of θ or in terms of characteristics ψ of the process.

$$\Psi : p(\cdot|\cdot, \theta) \mapsto \psi \tag{2}$$

that is computable from a simulation. Thus, ψ is a function of θ through the composition $\theta \mapsto p(\cdot|\cdot, \theta) \mapsto \psi$. GSM captures both of these types of information in our prior $\pi(\theta)$ through the construction

$$\pi(\theta) \propto h(\theta, \psi(\theta)). \tag{3}$$

Note that because we will be using the Metropolis-Hastings algorithm to compute the posterior, we only need a function proportional to the prior. We shall also discretize θ on a finite grid so that any positive h will be integrable.

We may not want to impose the belief that the scientific model holds exactly. We can capture this idea by recasting the problem so that η of the statistical model is viewed as the parameter of interest and constructing a prior that expresses a preference for η that are close to the manifold

$$\mathcal{M} = \{\eta \in H \, : \, \eta = g(\theta), \, \theta \in \Theta\} \, , \tag{4}$$

where Θ is the parameter space of the scientific model and H is the parameter space of the statistical model. Our prior construction uses a single parameter that we call κ to control prior beliefs about how close η should be to the manifold (Section 5). The smaller κ is, the more prior weight is placed on η close to the manifold. We assess the scientific model by seeing if the marginal posterior distributions of interpretable features of the statistical model are sensitive to the choice of κ. If changing our prior so as to support η farther from the manifold results in location shifts of the posteriors that are appreciable from a practical point of view, then we conclude that the evidence in the likelihood is against the restriction corresponding to the scientific model.

We illustrate the ideas in Figure 1. The scientific model is $p(y|\theta) = n(y; \theta, \theta^2)$, and the statistical model is $f(y|\eta) = n(y; \eta_1, \eta_2)$, where $n(y; \mu, \sigma^2)$ denotes the normal density with mean μ and variance σ^2. The mapping of the parameter θ of the scientific model to the parameters (η_1, η_2) of the statistical model is $g : \theta \mapsto (\theta, \theta^2)$. In each panel of Figure 1 the actors in our piece are displayed as follows: (i) the curve depicts the manifold (4), (ii) the dotted contours are those of the likelihood of the statistical model, (iii) the shading depicts the prior on η corresponding to a choice of κ, (iv) the dots are draws from the posterior for the parameter θ of the scientific model mapped into $\mathcal{M} \in H$ using the map $\eta = g(\theta)$ and then jittered (without the jittering, all the draws would be on the manifold), and (v) the solid contours represent the posterior of η given the prior (iii). We will call this the tinker toy example hereafter. It is included in the GSM distribution.

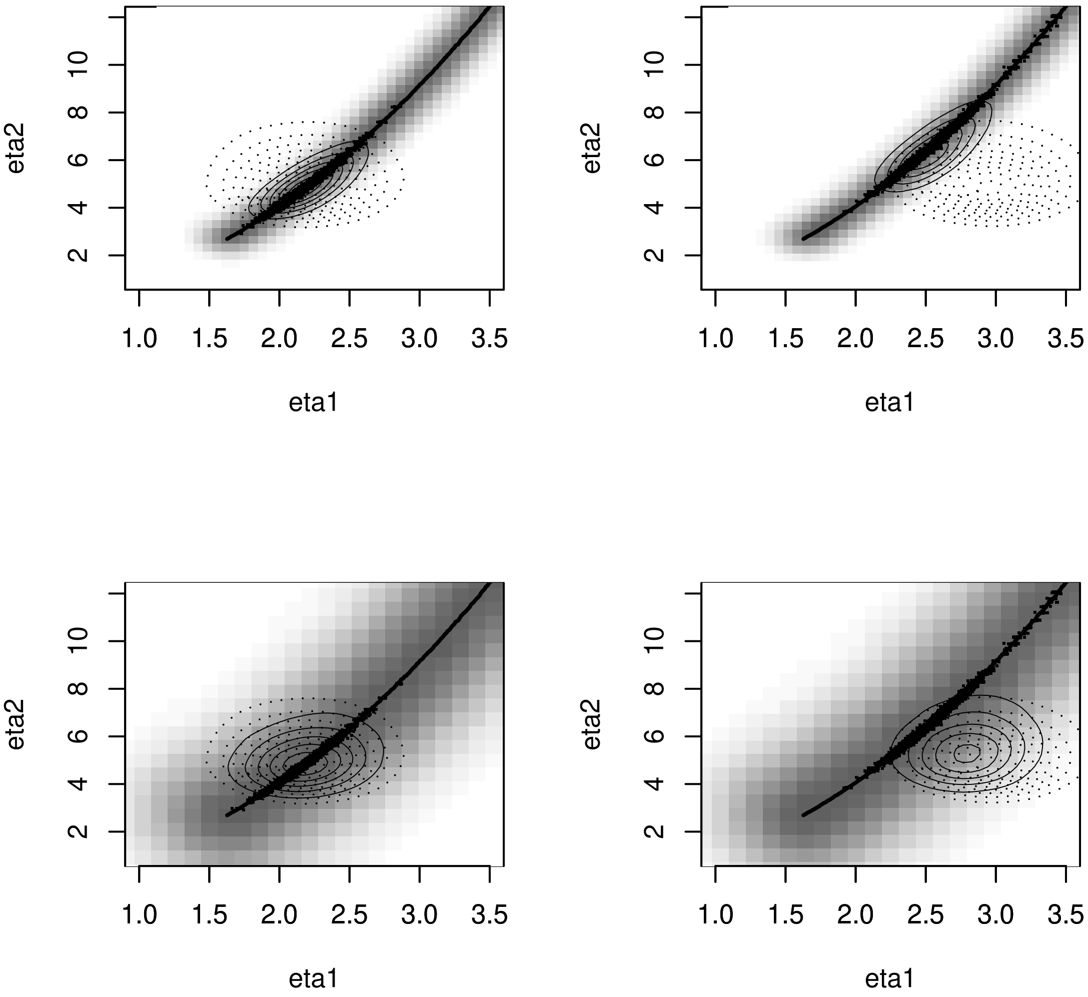

Figure 1. Priors and posteriors for the statistical model, tinker toy example. The the dotted lines are contours of the likelihood of the statistical model $f(y|x, \eta)$ of the tinker toy example. The line is the prior on η determined by the implied map $\eta = g(\theta)$ from the parameters θ of scientific model $p(y|x, \theta)$ to the parameters η of the statistical model. In the left panels the scientific model is true, in the right it is false. The thickness of the line is proportional to the posterior of η. The prior $\pi(\eta)$ can be relaxed as indicated by the shading. The lower panels are more relaxed than the upper. The solid contours show the posterior under the relaxed prior. Relaxation causes the contours to enlarge in all cases. When the scientific model is false, the posterior shifts in search of the likelihood.

The likelihood in the two left panels was obtained by simulating 50 observations from the scientific model with $\theta = 2$, or equivalently, ($\eta_1 = 2$, $\eta_2 = 4$). The likelihood in the two right panels was obtained by simulating 50 observations from the statistical model with $\eta = (2.8, 4)$. The prior off the manifold corresponds to a small value of κ in the top two panels, and a larger value in the bottom two. Our method for assessment of the scientific model is based on the observation that when the data support the scientific model, increasing κ may cause the posterior to become more spread out, but it will not dramatically shift location (panels (1,1) and (2,1)). Conversely, when the data does not support the scientific model, increasing κ will result in a shift of the posterior (panels (1,2) and (2,2)).

Of course, in high dimensional problems, we cannot simply look at the contours of the η posterior. Instead we must examine low dimensional marginals of interest. This can be illustrated for the tinker toy example by considering the marginal posterior of the coefficient of variation $\Upsilon : f(\cdot|\cdot, \eta) \mapsto \upsilon = \frac{\eta_1}{\sqrt{\eta_2}}$. Under the tinker toy scientific model the coefficient of variation is one with probability one. The four panels of Figure 2 correspond to the four panels of Figure 1. In Figure 2 the dotted curves are the densities of the prior marginal of υ while the solid curves are the posterior densities. In the left two panels we see that as the prior is relaxed (κ increases) the posterior spreads out but does not shift away from the true value which is one (the value consistent with the scientific model). In the right two panels, the posterior shifts as κ increases.

At this point, the main conceptual ideas that we shall propose have been set forth. The devil is in the details, to which we now proceed. The reader who would rather see how GSM works before bothering with details can skip to Subsection 6.1.

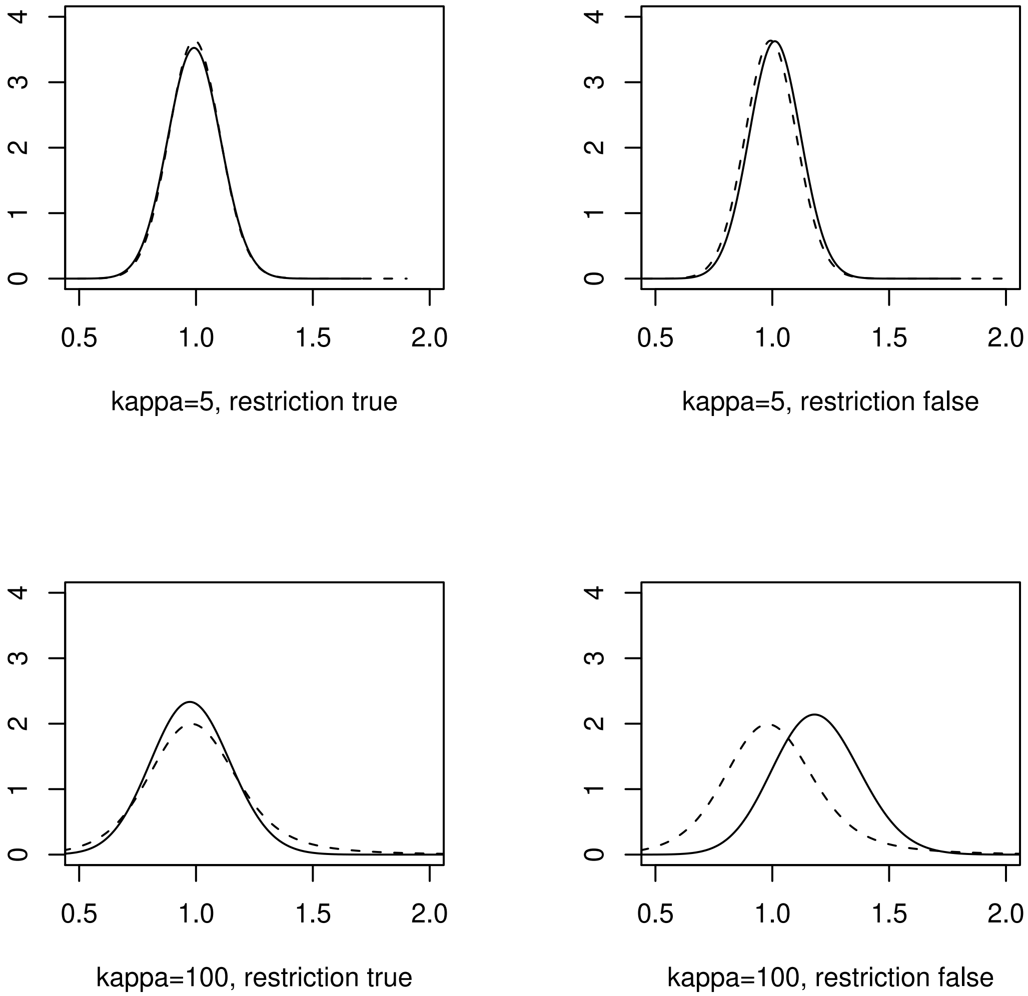

Figure 2. Priors and posteriors for a functional of the statistical model, tinker toy example. The posterior of the coefficient of variation for the tinker toy example is the solid line; the dashed line is prior. In the left panels the scientific model is true, in the right it is false. The prior is more relaxed in the lower panels than it is in the upper panels. The panels correspond to those of Figure 1

4 Bayesian Estimation of Scientific Models

We have two cases to consider. In the first case, we assume the scientific model is true and seek inference for θ. Here the statistical model is just a tool for computing the likelihood. In the second case we work in the context of the statistical model and η is the parameter. In this case, the scientific model is a source of prior information. We will consider the first case in this section and the second in Section 5.

4.1 A Metropolis Algorithm for θ

We use the Metropolis algorithm to compute the posterior distribution of θ. The Metropolis algorithm is an iterative scheme generating a sequence of θ values according to a Markov chain whose stationary distribution is the posterior. To implement it, we must require a likelihood, a prior, and transition density in θ called the proposal density.

Let $\mathcal{L}(\theta)$ denote the likelihood assuming that (1) holds. To compute it we use

$$\mathcal{L}(\theta) = \prod_{t=1}^{n} f(y_t \,|\, x_{t-1}, g(\theta)),$$

where $\{y_t, x_{t-1}\}$ denotes the observed data and n the sample size. Let $\pi(\theta)$ denote the prior distribution on θ. As discussed in Section 3, in order to compute this prior $\pi(\theta)$ we may need the value ψ taken on by the functionals Ψ given by (2). Let q denote our proposal density. For a given θ, $q(\theta, \theta^*)$ defines a distribution of potential new values θ^*.

Given a current θ^o and the corresponding $\eta^o = g(\theta^o)$, we obtain the next pair (θ', η') as follows:

1. Draw θ^* according to $q(\theta^o, \theta^*)$.
2. Draw $\{\hat{y}_t, \hat{x}_{t-1}\}_{t=1}^{N}$ according to $p(y_t|x_{t-1}, \theta^*)$.
3. Compute $\eta^* = g(\theta^*)$ and ψ^* from the simulation $\{\hat{y}_t, \hat{x}_{t-1}\}_{t=1}^{N}$.
4. Let $\alpha = \min\left(1, \frac{\mathcal{L}(\theta^*)\,\pi(\theta^*)\,q(\theta^*,\theta^o)}{\mathcal{L}(\theta^o)\,\pi(\theta^o)\,q(\theta^o,\theta^*)}\right)$.
5. With probability α, $(\theta', \eta') = (\theta^*, \eta^*)$, otherwise $(\theta', \eta') = (\theta^o, \eta^o)$.

Steps 1, 4, and 5 are just the standard Metropolis algorithm. Steps 2 and 3 are essential features of our approach. If the proposed θ in Step 1 violates a support condition that can be checked without running Step 2, one skips Step 2 because α in Step 4 will be zero.

4.2 Choice of θ Proposal

Step 1 of the Metropolis algorithm in Section 4.1 requires a proposal density q for θ. In choosing q we need to take into consideration that to compute the likelihood at a proposed θ the scientific model must be simulated. For a sophisticated scientific model, this simulation may involve significant computation. A nonlinear optimizer, a nonlinear equation solver, or some other routine that needs starting values might be called in the coarse of these computations. This motivates us to consider proposing small changes in θ so that computational results from the old θ may be used in doing the computations for the proposed θ. In particular, if θ is not changed too much, results from the previous computation can be used as starting values for the new one. The cost of this strategy is in dependence in the Markov chain.

We start by discretizing θ because, as seen later, discretization permits significant improvements in computational efficiency. For the i^{th} component of θ we choose $a_i < b_i$, and s_i. We then let θ_i take on the values $a_i + js_i$ where j ranges from 1 to g_i which is equal to the integer part of $(b_i - a_i)/s_i$. Thus, θ_i takes values between a_i and b_i on a grid of mesh s_i. To propose a new θ we first randomly choose a component to change, with each component having the same chance of being chosen. If the i^{th} component is chosen, there is some j such that the current $\theta_i = a_i + js_i$. We choose a set of distributions $q_i(j,k)$ on $\{1, 2, \ldots, g_i\}$ where i is the θ component, j is the current grid position of that component, and k denotes the random new grid position to be drawn. We draw $k \sim q_i(j, \cdot)$ and let θ^* be obtained from θ by changing the i^{th} component from $a_i + js_i$ to $a_i + ks_i$. For $q_i(j,k)$ use

$$q_i(j,k) \propto \begin{cases} \exp(-\frac{1}{2\sigma_i^2}(k-j)^2) & k \neq j \\ 0 & \text{else} \end{cases}$$

The choice of σ_i determines the number of s_i that we tend to move. We assign 0 probability to proposing that we stay put as there is no point in proposing that we go to where we are.

The choice of a_i and b_i is not critical; a_i and b_i can be set so that the intervals $(a_i,\ b_i)$ cover the support of the posterior by a wide margin without noticeably degrading the performance of the Metropolis algorithm. The choice of s_i is crucial. We will move away from the starting value in integer multiples of s_i. The combination of the choice of s_i and σ_i determines the size of the changes that q proposes. The choice of s_i determines the accuracy of our inference. When we choose s_i we are saying that, as a practical matter, we only need to know θ_i in terms of s_i units. Two θ's that differ in component i by less than s_i are virtually the same as a practical matter. Because computation is expensive, we should not waste resources by determining θ on a finer scale than we actually care about.

4.3 Computing the Map

In Step 3 of the Metropolis algorithm in Section 4.1 we need to uncover the map $g : \theta \mapsto \eta$ that satisfies (1) from a simulation $\{\hat{y}_t, \hat{x}_{t-1}\}_{t=1}^{N}$ of $p(y_t|x_{t-1}, \theta)$. At an intuitive level what we propose is simple: We choose N so large that the simulated data $\{\hat{y}_t, \hat{x}_{t-1}\}_{t=1}^{N}$ gives us complete information about its distribution for given θ. We find the corresponding η by maximizing the likelihood of the simulated data under the statistical model $f(\cdot|\cdot, \eta)$. That is, we find the η which gives the same kind of data under $f(y_t|x_{t-1}, \eta)$ as did θ under $p(y_t|x_{t-1}, \theta)$. At a formal level, we are finding the η that puts the Kullback-Liebler divergence $d(f, p) = \iint [\log p(y|x, \theta) - \log f(y|x, \eta)]\ p(y|x, \theta)\, dy\, p(x|\theta)\, dx$ to zero by minimizing $d(f, p)$ with respect to η and are noting that $\iint \log p(y|x, \theta)\, p(y|x, \theta)\, dy\, p(x|\theta)\, dx$ does not have to be computed to solve this minimization problem. We approximate the integral that does have to be computed in the usual way: $\iint \log f(y|x, \eta)\, p(y|x, \theta)\, dy\, p(x|\theta)\, dx \approx \frac{1}{N}\sum_{t=1}^{N} \log f(\hat{y}_t|\hat{x}_{t-1}, \eta)$. (Or by $\frac{1}{R}\sum_{r=1}^{R} \frac{1}{n}\sum_{t=1}^{n} \log f(\hat{y}_{t,r}|\hat{x}_{t-1,r}, \eta)$ if not ergodic. We assume ergodicity hereafter; if not, the requisite modifications are obvious.) Thus, upon dropping the division by N, the map is computed as

$$g : \theta \mapsto \underset{\eta}{\operatorname{argmax}} \sum_{t=1}^{N} \log f(\hat{y}_t \mid \hat{x}_{t-1}, \eta).$$

Because the $f(\cdot|\cdot, \eta)$ family is generally chosen to be flexible and high dimensional, this likelihood can be complicated. However, the simulated data set is large and η^o should be a good starting value in the search for η^*. This assumes that θ^* is not too different from θ as discussed in Section 4.2. In order to keep our analytical requirements to a minimum, we would like our method only to require the computation of the objective $\mathcal{L}(\eta) = \prod_{t=1}^{N} f(\hat{y}_t|\hat{x}_{t-1}, \eta)$.

Given these considerations, to find the mle we run a Markov chain for η using the simulated data. Because our goal is to find the mle and the sample size is large, we use a flat prior on η when running this chain. With the large sample size, the Markov chain will quickly move from the η to values close to η^*. We use a normal random walk Metropolis within Gibbs approach. That is, we first subdivide the η vector into subvectors. In the manner of a Gibbs sampler, we cycle through the subvectors one at a time. For subvector η_i, we use the normal proposal $q(\eta_i, \eta_i^*) \sim n(\eta_i, \Sigma_i)$ in a standard random walk Metropolis algorithm. Effectively, this is a simulated annealing optimization algorithm where the simulation size N is the temperature parameter because N is what controls the peakedness of the likelihood. A side benefit is that the chain for η also provides the scaling for the model assessment strategy proposed in Section 5.

We choose a fixed number of steps to run the η chain, and keep the visited η which has the highest likelihood under the simulated data. In our experience, it is relatively straightforward to choose (i) a simulation sample size which is large enough to ensure that the map is adequately recovered by the mle and (ii) a number of steps to iterate the Markov chain in η that will ensure the η chain has finished moving away from the starting value of η.

This is a computationally costly part of our overall procedure. Because the simulation sample size N is large, each computation of the likelihood for the η chain can take a long time. Nonetheless, we have found that, because of the large N, this part of the procedure is remarkably stable, even though the statistical model may actually be difficult to estimate on data samples of the size n that we actually observe.

The first reason for placing θ on a grid is that a significant reduction in computational time can be achieved. With θ on a grid, it takes only a modest amount of memory to store all previously computed values of η, $\mathcal{L}(\eta)$, $\pi(\theta)$, etc. in a binary tree indexed by θ. When θ is revisited, both the fact that it is a revisit and the information required for Step 4 of the Metropolis algorithm for θ can be quickly obtained by traversing the tree. The two costliest Steps 2 and 3 are thereby eliminated. By storing previous results in a tree and looking them up, the θ chain runs faster as it becomes longer.

The second reason for putting θ on a grid is that it reduces the accuracy to which the mle has to be computed, which also reduces computational time significantly. In fact, one might say that putting θ on a grid is what makes the GSM software work at all, becuase it is next to impossible to compute the mle to sufficient accuracy to allow θ to be on a continuum.

Analytic derivatives of the likelihood of some statistical models are not hard to obtain. When there are support conditions, these can often be enforced by adding a smooth penalty function to the likelihood that is zero when the support conditions are satisfied and sufficiently positive otherwise. For this case, the GSM package includes a derivative based hill climber to complete the computation of the mle after a short run of the η chain. The hill climber is a BFGS quasi-Newton method that makes more use of derivative information in the line search part of the algorithm than most BFGS implementations thereby making it economical of function evaluations and well suited to this application. The design of GSM is modular: One is not required to supply methods for the statistical model that compute the derivatives and the penalty if the hill climber is not going to be used. Experience so far indicates that there is a limit to the hill climber's effectiveness. It can shorten the η chain but not eliminate it. The BFGS quasi-Newton iterations, if started from the previously BFGS-computed η, tend to stick, which, in turn, causes the η and θ chains to stick. Therefore, in connection with the BFGS hill climber, the role of the η chain becomes that of supplying a starting value and it must be made sufficiently long to do so effectively.

Figure 3 displays the results of ten runs of an η chain. Every run is clearly visible in the figure as a segment of 200 iterations. In the notation of Section 4.1, each segment displays the results of a Markov chain in η that is started at η^o, uses the data simulated from the scientific model at θ^*, and uses the likelihood of the statistical model coupled with a flat prior on η. On the vertical axis, the log-likelihood from the statistical model is plotted. We can see the likelihood quickly increase as the η value moves toward the mle. The segments level off at different likelihoods because they represent the likelihoods of different simulated data sets. Because of the large size of each simulated data set, $N = 50,000$ in this instance, the posterior is very tight around the mle and the chain quickly moves to a new level.

Figure 4 repeats the computation with 9 iterations of the Markov chain in η followed by a BFGS polish. The same final point is computed, but run times are much faster. This

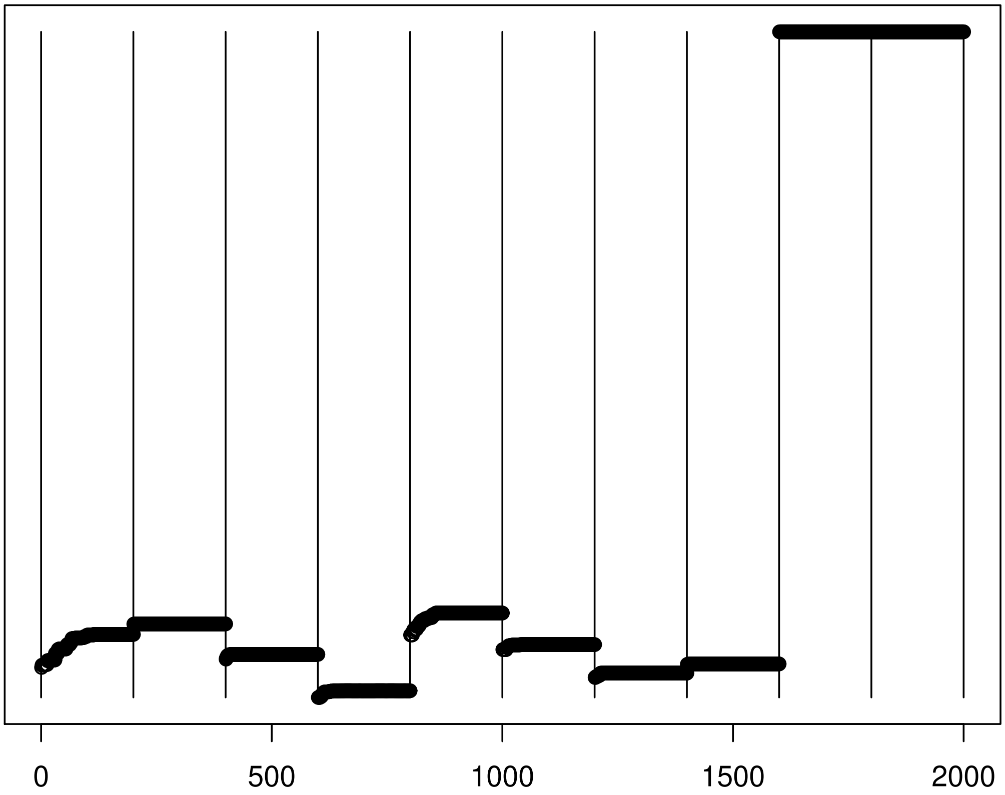

Figure 3. **η Chain for the tinker toy model.** Ten successive runs of the η chain. Each run is 200 iterations. The log-likelihood of the simulated data set is plotted on the vertical axis. Vertical bars mark where θ changes. Jumps are because $\{\hat{y}_t\}_{t=1}^N$ changes at each vertical bar.

example is so simple that we can get away with 9 MCMC iterations. This will not be true in general. Specifically, 9 iterations are too few for the example in Section 9.

The second reason for putting θ on a grid is that it reduces the accuracy to which the mle has to be computed, which also reduces computational time significantly. In fact, one might say that putting θ on a grid is what makes the GSM software work at all, becuase it is next to impossible to compute the mle to sufficient accuracy to allow θ to be on a continuum.

4.4 Identification and Map Recovery

The scientific model and the statistical model must work in concert with one another. In this subsection we discuss issues arising from the relationship between the two models.

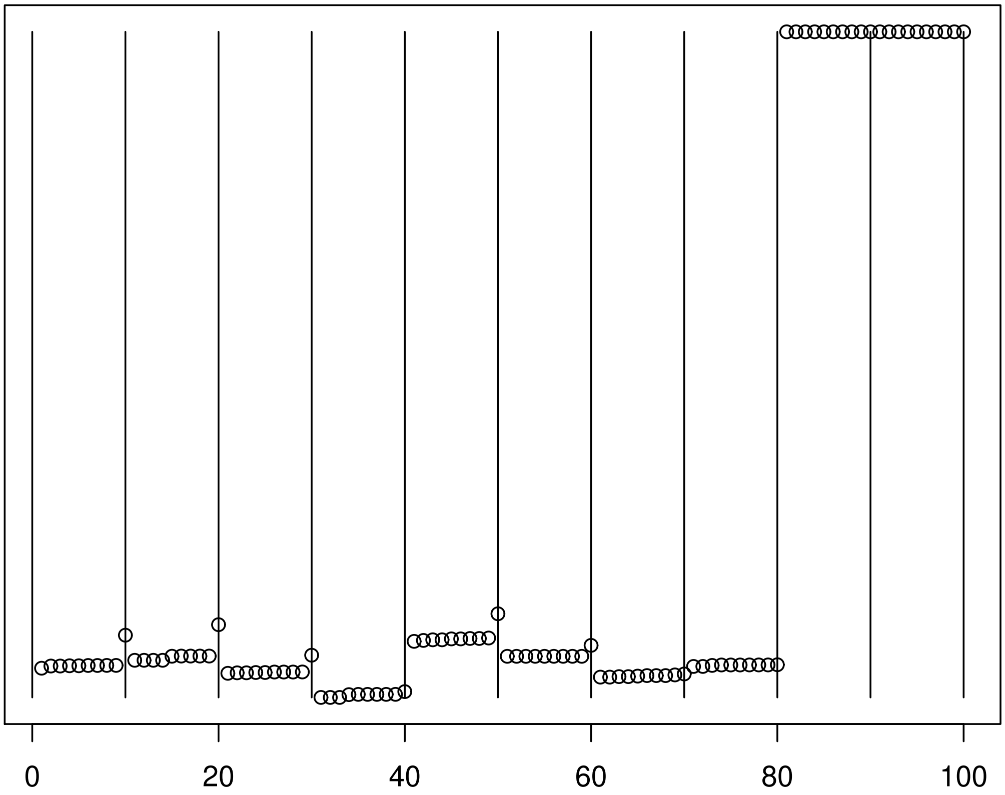

Figure 4. **η Chain for the tinker toy model.** Ten successive runs of the η chain. Each run is 9 iterations of the MCMC chain followed by a BFGS polish. The jump at the end of each chain shows the effect of the polish. The log-likelihood of the simulated data set is plotted on the vertical axis. Vertical bars mark where θ changes. Jumps in the level of the chains at the vertical bars are because $\{\hat{y}_t\}_{t=1}^N$ changes at each vertical bar.

The map $g(\theta) = \eta$ should not be one-to-many if Assumption 1 is to be satisfied. This is equivalent to stating that the statistical model should be identified by simulations from the scientific model. Identification of the statistical model by simulations from the scientific model entails some obvious conditions such as that the support of the statistical model should include the support of the scientific model. A violation of the support condition can occur if the scientific model has fewer random shocks than the dimension of y thereby causing the support of the scientific model to be a lower dimensional submanifold of the support of the statistical model. This can happen inadvertently if a proposed θ implies a singular variance matrix somewhere within the scientific model. However, singularity is easy to check.

Other violations can be more subtle. An example is the statistical model $y = \eta_1 \exp(\eta_2 x) + \eta_3 \exp(\eta_4 x) + e$ with simulated data from a scientific model that is actually $y = \theta_1 \exp(\theta_2 x) + e$ in disguise. In this case, either $\eta_3 = 0$ and η_4 is not identified or $\eta_2 = \eta_4$ and only the sum $\eta_1 + \eta_3$ is identified. One can usually detect this situation by checking the η chain described in Subsection 4.1 for a unit root. One way to do this is to find the eigen vector ℓ of the variance matrix of the chain $\{\eta_t\}_{t=1}^N$ with largest eigen value and examine the sequence of inner products $\{\ell'\eta_t\}_{t=1}^N$ for a unit root.

Lack of identification of the statistical model does not matter in the computation of the θ chain described in Subsection 4.1 as long as the likelihood is actually maximized at the computed η. The implied map $g(\theta) = \eta$ will be one-to-many but often considerations similar to estimability in less than full rank linear models come into play so that the features of the statistical model that are of interest in an application have the same value regardless of which maximizing value of η is chosen. These considerations are discussed in Gallant (1987). The methods proposed in Section 5 would have to be modified if η is not identified.

An omnibus check for identification failure and nearly anything else that can go wrong with the algorithm of Subsection 4.1, such as poor start values or not running the chain long enough to compute η^* to sufficient accuracy, is to run a regression of all computed η^* on low degree polynomials in the corresponding values of θ. If the R^2 are high one has some assurance that the statistical model is identified and that accuracy is adequate.

Another requirement of Assumption 1 is that (1) should hold. To check directly whether (1) holds for a specific value of θ one may (a) fit the statistical model to a simulation of the scientific model at that value of θ, (b) simulate from the fitted statistical model, and (c) check to see if the empirical distributions of the two simulations match. This can serve as a partial empirical check if (1) cannot be established analytically.

The investigator may be motivated to use a statistical model that is known from experience to be in accord with the data. This can make the uncovered map more interpretable and easier to compute and facilitate the analysis in other ways as well. This could, however, lead to a violation of (1). The likelihood is then based on the statistical model closest in Kullback-Liebler divergence to the scientific model rather than the scientific model itself.

Elaborations of the statistical model in order to satisfy (1) beyond those needed to fit the data may not have much effect on inference for θ or the functionals of $p(y_t|x_{t-1},\theta)$ of interest.

Deleting a feature that is both in the data and in the scientific model is another matter. This sort of deletion can change results rather dramatically.

5 Inference Off the Manifold: Model Assessment

In this section $\eta \in H$ becomes the parameter of interest. The scientific model $p(y|x,\theta)$ may be viewed as a sharp prior that restricts η to lie on the manifold $\mathcal{M} \subset H$. What one would like to do is see how results change as this prior is relaxed. However, once we have moved off the manifold $\mathcal{M}$ we can no longer view results from the perspective of the scientific model $p(y|x,\theta)$ and must view them from the perspective of the statistical model $f(y|x,\eta)$ because the scientific model loses meaning off the manifold. Therefore, seeing how results change must be taken to mean seeing how the marginal posterior distribution of a parameter or functional of the statistical model changes. Denote the vector of functionals of the statistical model of interest by

$$\Upsilon : f(\cdot|\cdot,\eta) \mapsto \upsilon. \tag{5}$$

For convenience, if an element of η is of interest, we make it an element of υ.

We assume that we have a discrete set of points on the manifold $\{\eta_j \in \mathcal{M} : j = 1,\dots,G\}$. The analysis of Section 4 generates a discrete set of points $\{\theta_j \in \Theta : j,\dots,G\}$ at which the map g has been evaluated; putting $\eta_j = g(\theta_j)$ provides such a set of points. Relaxation of the prior will be formulated in terms of a weighted distance of η from the manifold $\mathcal{M}$. We can cheaply compute the distance from η to the manifold as

$$d(\eta,\mathcal{M}) = \min_{j=1,\dots,G} \, (\eta-\eta_j)' \, A_j \, (\eta-\eta_j)\,, \tag{6}$$

where A_j are scaling matrices. Let $\hat{j}$ denote the index j at which the minimum occurs, let $\hat{J}$ denote the map $\hat{J} : \eta \mapsto \hat{j}$, and let $\hat{h}(\eta) = \eta_{\hat{J}(\eta)}$.

The prior we propose is the product of preferences along the manifold, preferences about how close the statistical model is to the manifold, and general preferences about η

$$\pi_\kappa(\eta) \propto w_1[\hat{h}(\eta)] \, \exp\left(-\frac{d(\eta,\mathcal{M})}{2\kappa}\right) \, w_3(\eta), \tag{7}$$

where $w_1(\eta)$ and $w_3(\eta)$ are suitably chosen positive functions and we assume the middle $w_2(\eta, \kappa)$ term assures integrability. The three terms in the product (7) correspond to our three kinds of preferences regarding η. The prior becomes more diffuse and the scientific model less influential as the scale factor κ increases. Note that none of the individual terms is thought of as being a prior in its own right, each is just a part of the overall construction. To check that (7) results in a reasonable prior, draws of η may be simulated from the prior and prior marginals of interest checked.

The functions $w_1(\eta)$ and $w_3(\eta)$ in (7) may be related to the preferences along $\mathcal{M}$ that we had for θ of the scientific model, but there is no logical necessity that this be the case. If we desire to use roughly the same kind of preferences along $\mathcal{M}$ for η as we used for θ, we can use $\eta_j = g(\theta_j)$ to generate our points on the manifold, put $w_1[\hat{h}(\eta)] \propto \pi(\theta_{\hat{\jmath}(\eta)})$, where $\pi(\theta)$ is given by (3), and put $w_3(\eta) = 1$. Note that $\pi(\theta_{\hat{\jmath}(\eta)})$ is a composite function that depends only on η that is easily retrieved from our stored map. With these choices, if η_1 and η_2 both have the same distance from the manifold, then $\pi_\kappa(\eta_1)/\pi_\kappa(\eta_2) = \pi(\theta_{\hat{\jmath}(\eta_1)})/\pi(\theta_{\hat{\jmath}(\eta_2)})$. Recall that one of our motivating considerations was the problem of sparse data and that this problem is overcome by the introduction of prior information. A small κ may imply sufficient prior information for inference. With large κ, additional prior information may be needed in the form of a choice of $w_3(\eta)$.

The computation of the posterior distribution of η using the statistical model $f(y|x, \eta)$ and prior $\pi_\kappa(\eta)$ can be accomplished by a routine application of the Metropolis algorithm because $\pi_\kappa(\eta)$ is easily computable and an analytic expression for $f(y|x, \eta)$ is available.

Our proposal is that the scientific model be assessed by plotting a suitable measure of the location and scale of the posterior distribution of υ against κ or, better, sequential density plots. What one expects to see, for a well fitting scientific model, is that the location measure does not move by a scientifically meaningful amount as κ increases, which indicates that the model fits, and that the scale measure increases, which indicates that the scientific model has empirical content.

For the scaling matrices A_j, we put $A_1 = \ldots = A_G = \Sigma_\eta^{-1}$ in (6) where Σ_η is computed as follows: Initialize to zero. Whenever the Metropolis-Hastings chain for computing the mle of η at Step 3 of the Metropolis algorithm in Subsection 4.1 must be run, update

$$\Sigma_\eta \leftarrow \Sigma_\eta + (\eta_1 - \eta_2)(\eta_1 - \eta_2)' \tag{8}$$

where η_1 is a point on the chain immediately after transients have died out and η_2 is the last point on the chain. This method of scaling the distance measure is reasonable because it puts η on the scale of the posterior: Distance is being measured in units of standard deviation.

The distance function (6) can be made invariant to a linear reparameterization of the statistical model and first order invariant to a nonlinear transformation by letting A_j be the inverse of the covariance of the draws η_t^j from the chain used to obtain $\eta_j = g(\theta_j)$ at Step 3 of the Metropolis algorithm. The scaling proposed in the paragraph above will achieve approximate invariance if the Σ_{η_j} are relatively homogeneous.

One should note that using posterior draws from the scientific model to compute the image $\mathcal{M}$ of the map $g(\cdot)$ does make use of the data to determine the prior $\pi_\kappa(\eta)$. We do not regard this as a problem because the θ draws will contain enough extreme values to make sure that the extent of $\mathcal{M}$ is large enough. If one is particularly worried about this, one could run the θ chain with a smaller amount of data to make sure that $\mathcal{M}$ is over explored.

6 Building and Running GSM

6.1 Availability

C++ code and this Guide as a PostScript or PDF file are at `http://www.aronaldg.org`.

6.2 Building and Running GSM

The GSM code will on Linux machines and on Macs that have Xcode installed. On a Windows machine it will run under either Cygwin from `http:/www.cygwin.com` or MinGW from `http://www.mingw.org`.

Download `gsm.tar` from `http://www.aronaldg.org`. On a Unix machine use `tar -xf gsm.tar` to expand the tar archive into a directory that will be named `gsm`. 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:

- gsmman
- gsmrun
- gsmsrc
- lib
  - libscl
  - libsmm
  - libsnp
- snpman
- snprun
- snpsrc
- svfx
- tt1d
- tt2d

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

First the three libraries libscl, libsnp, and libgsm must be built, in that order. Change directory to `lib/libscl/gpp` and type `make`. Building libsnp and libgsm is similar.

To run the SNP example that comes with the distribution, within the directory `snprun` copy `makefile.gpp` to `makefile`, type `make` and then `./snp`. Similarly for GSM, within `gsmrun` copy `makefile.gpp` to `makefile`, type `make` and then `./gsm sv.ctrl.000.dat`.

7 The Example

We will illustrate the ideas in the remainder of the *Guide* using the GSM program to estimate a stochastic volatility model. The main advantage of this example is that the model and code are both simple and easily understood. Of course this simplicity means that there are other ways of estimating the stochastic volatility model that are less computationally intensive than GSM.

The data set consists of 834 observations on the daily US dollar to German mark exchange rate over the years 1975 to 1990 expressed as a percentage change from the previous week. This is the same series used in both the *SNP User's Guide* and the *EMM User's Guide*.

7.1 The Scientific Model: A Stochastic Volatility Model

Let y_t denote the percent change in the price of a security. The stochastic volatility model in the form used by Gallant, Hsieh, and Tauchen (1997) with a slight modification to produce a leverage effect (correlation between return innovations and volatility innovations) is

$$\begin{aligned} y_t - a_0 &= a_1(y_{t-1} - a_0) + \exp(v_t)\, u_{1t} & (9) \\ v_t - b_0 &= b_1(v_{t-1} - b_0) + u_{2t} & (10) \\ u_{1t} &= z_{1t} & (11) \\ u_{2t} &= s\left(r\, z_{1t} + \sqrt{1-r^2}\, z_{2t}\right) & (12) \end{aligned}$$

where z_{1t}, z_{2t} are iid Gaussian random variables. The parameter vector is

$$\theta = (a_0, a_1, b_0, b_1, s, r)$$

Early references are Clark (1973) and Tauchen and Pitts (1983). More recent references are Gallant, Hsieh, and Tauchen (1991, 1997), Andersen (1994), and Durham (2003). See Shephard (2004) for more background and references.

There is controversy regarding the timing convention in equation (9) and the references above are not in agreement. The alternative timing convention is

$$y_t - a_0 = a_1(y_{t-1} - a_0) + \exp(v_{t-1})\, u_{1t} \qquad (13)$$

which is consistent with an Euler discretization of the continuous time stochastic volatility model. See Yu (2005) for more details but be aware that his specifications do not include an autoregressive term to account for the well known slight predictability in daily returns so that his empirical results may not be relevant.

Reasonable starting values are essential to success with GSM. Fitting y_t to $a_0 + a_1(y_{t-1} - a_0)$ by least squares, one finds that $a_0 = 0.055934$, $a_1 = 0.02224$, and the standard deviation of the residuals is $1.49930 = \exp(0.404998)$. For these data, we expect little leverage and set $r = 0$ to start and impose a normal prior that states $P(|r - 0.0| < 0.05) = 0.95$ to reflect this view. Setting $b_0 = 0$ and using $\mathcal{E}e^X = \exp(\mu + \sigma^2/2)$ for normally distributed X, we have $s = 0.9$. For data from financial markets, the volatility is usually quite persistent. Therefore we set $b_1 = 0.95$ and impose a normal prior that implies $P(|b_1 - 0.95| < 0.1) = 0.95$.

7.2 The Statistical Model: An SNP Time Series Model

The GSM package as distributed presumes application to stationary time series data for which the SNP model of Gallant and Nychka (1987) as modified for time series applications by Gallant and Tauchen (1992) is an ideal choice of a statistical model. The best discussion of the SNP model for our purposes is that of the *SNP User's Guide* (Gallant and Tauchen, 2004). One is by no means restricted to this choice and two tinker toy cross sectional examples that were used in Gallant and McCulloch (2008) are included with the package for illustration. But the SNP model is difficult to code in any generality so it is included in the package to spare users the trouble. The SNP constructor uses an SNP parmfile as its input. Therefore first step is to estimate the SNP model using the SNP package to get that parmfile.

Instead of working through a full SNP specification search, which is described in detail for this example in the *SNP User's Guide*, we show how to implement the GSM estimator using a fit found there, namely that described by the parmfile `11114000.fit`.

The settings for that fit are $L_u = 1$, $L_g = 1$, $L_r = 1$, $L_p = 1$, $K_z = 4$, $K_x = 0$. These settings define an AR(1) model for $\{y_t\}$ with a GARCH(1,1) conditional scale function and a time homogeneous nonparametric innovation density with fat tails accommodated via $K_z = 4$. The dependence on the past is through the linear location function and GARCH scale function. This model is optimal under the Schwarz criterion. Even though labeled as optional in the SNP parmfile, the location and scale information in the blocks labeled `TRANSFORM START VALUES FOR mean` and `TRANSFORM START VALUES FOR variance` are essential and must be present. In fact, it is best to just use a parmfile generated by SNP and not edit it at all. When running the parallel version of the code, `iprint` must be set to zero. For the uniprocessor version it can be one; which will produce useful diagnostic information in a file named `detail.dat`. In the parallel version, at best that information would be splattered to standard output by each process in random order; at worst it would cause a crash. For use with GSM one would often use a more richly parameterized model than the Schwarz preferred model as discussed in Section 1. Here it might have been better to use a parmfile with leverage (`Lv`>0). Here is the SNP parmfile:

```
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   2.1588209559666984 0e+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
```

The simulations from the scientific model must actually identify the parameters of the statistical model. This can be a problem when trying to tune chains. For example, if the parameters a_1 and b_1 of the stochastic volatility model are set to zero then, in a practical sense, the `11114000.fit` is not identified and the subchain computes the likelihood so inaccurately as to make it nearly impossible to get chains properly tuned.

One approach to remedy this problem is to have a good idea what the parameter values of the scientific model ought to be. Another is to use the EMM package, distributed at the same ftp site as GSM, to get starting values and tune the chains for both the scientific model and the statistical model. With GSM one is trying to tune two chains simultaneously. Because they interact, this can be difficult. Using EMM, one can focus on one chain at a time, which is much easier. In the EMM distribution is a model labeled `self` that fits an SNP model to itself using EMM. This can be used to tune the chain for the statistical model. The coding required to use the EMM package is nearly identical to that for the GSM package, including the use of prior information, so EMM can be used to get an initial tune of the chain for the scientific model also.

8 Using the GSM Package

The structure for the example distributed with the distribution is presumed to be as above: All user-supplied C++ classes, and all other code, excepting library code, resides in the directories (folders), `snpsrc` and `gsmsrc` for SNP and GSM respectively. Input and output files for SNP are in directory `snprun` and those for GSM are in `gsmrun`. The specific code to implement the stochastic volatility scientific model and the SNP statistical model is in directory `svfx` as files `gsmusr.h` and `gsmusr.h`.

8.1 User Supplied Class Declaration

The user supplies a class that represents the scientific model, which here we shall call `sv_sci_mod`. This code, as just mentioned, is in directory `svfx`. The declaration for the class is in file `gsmusr.h`, the code implementing it is in file `gsmusr.cpp`. The functionality that `sv_sci_mod` must provide is dictated by inheritance from class `sci_mod_base` declared in `libgsm/libgsm_base.h`. Here is the relevant portion of `libgsm/libgsm_base.h`

```
namespace libgsm {

  struct den_val {
    bool positive;
    REAL log_den;
         den_val() : positive(false), log_den(-REAL_MAX) { }
         den_val(bool p, REAL l) : positive(p), log_den(l) { }
  };

  class sci_mod_base {
  public:
    virtual INTEGER len_parm() = 0;
    virtual INTEGER len_func() = 0;
    virtual void    get_parm(scl::realmat& parm) = 0;
    virtual void    set_parm(const scl::realmat& parm) = 0;
    virtual bool    support(const scl::realmat& parm) = 0;
    virtual den_val prior(const scl::realmat& parm,
      const scl::realmat& func) = 0;
    virtual bool    gen_sim(scl::realmat& sim,
      scl::realmat& func) = 0;
    virtual ~sci_mod_base() { }
  };
}
```

and here is the corresponding `gsmusr.h`

```
#include "libgsm.h"
```

```
namespace gsm {

  class sv_sci_mod;

  typedef sv_sci_mod   sci_mod_type;

  class sv_sci_mod : public libgsm::sci_mod_base {
  private:
    scl::realmat theta;
    INTEGER slen;
    INTEGER spin;
    const INTEGER ltheta;
    const INTEGER lfunc;
  public:
    sv_sci_mod
      (const scl::realmat* dat_ptr, const std::vector<std::string>& pfvec,
       const std::vector<std::string>& alvec, std::ostream& detail);
    INTEGER len_parm() {return ltheta;}
    INTEGER len_func() {return lfunc;}
    void get_parm(scl::realmat& parm) { parm = theta; }
    void set_parm(const scl::realmat& parm) { theta = parm; }
    bool support(const scl::realmat& parm);
    libgsm::den_val prior(const scl::realmat& parm,
      const scl::realmat& func);
    bool gen_sim(scl::realmat& sim, scl::realmat& func);
  };
}
```

Class `sv_sci_mod` gets bound to program `gsm` via the statement

```
typedef sv_sci_mod sci_mod_type;
```

as shown.

The types `REAL`, `INTEGER`, and `INT_32BIT` are defined by `typedef`'s in `scltypes.h` which gets included with `libscl.h`. On most machines these are `double`, `int`, and `int`, respectively. Class `realmat` is presented in `realmat.h` which gets included with `libscl.h`. This is a fairly complete matrix class that supports most linear algebra related to statistical applications including equation solving, inversion, and singular value decomposition. In general there is much in `libscl` that will aid the user in writing a `sci_mod`, including a nonlinear equation solver and a nonlinear optimizer.

As discussed in Section 1, the idea behind `func` is that there is more information about a simulation that one needs to know about a scientific model besides the value of `theta`. A moment of a latent variable is an obvious example. Hereafter, `theta` will usually be called `sci_parm` and `func` will be called `sci_func`, the reason being that the statistical model has corresponding quantities `stat_parm` and `stat_func` and we want to be certain that it is clear which we mean. The MCMC chains for the posterior of these quantities are written to files by program `gsm` as does much else as described later.

Often `sci_func` is computed to know if the simulation from the scientific model is useable. For instance, in the asset pricing model analyzed in Gallant and McCulloch (2008) the real rate is a latent variable whose average over the simulation must be kept near 1%. If `sci_parm` generates a real rate that is not near 1%, then that `sci_parm` should be rejected by the Metropolis-Hastings algorithm `mcmc` in `libgsm`. To determine usability, `sci_func` gets passed by `mcmc` to member `prior` of `sci_mod` which makes the decision and returns the verdict as a `den_val`. If `den_val.positive` is `false`, then the `sci_parm` that generated the simulation is rejected by the Metropolis-Hastings algorithm. If `den_val.positive` is `true`, then `den_val.log_den` is added to the log objective function before the Metropolis-Hastings accept/reject decision is made; i.e. acts as an informative prior.

Member `support` of `sci_mod` plays a similar role: it returns `false` if if `sci_parm` is to be rejected. The difference between `support` and `prior` is that `support` is called before the simulation and `prior` after. The intent is to save the cost of an unnecessary simulation if `sci_parm` violates support conditions that can be cheaply determined. Be warned: method `set_parm` is called before `support`. The reason is that for some models, such as SNP, calling `set_parm` changes the state of various objects and support conditions are determined by checking the resultant state rather than directly checking the parameters. This calling sequence may require bullet-proofing of the code in `set_parm`.

As discussed in Section 1, exactly the same considerations apply to members `support` and `prior` of `stat_mod`, although for the SNP statistical model distributed with the GSM package there are only a few support conditions that impose some sign conventions. These are handled automatically and do not require user intervention. In general this is the case: Whatever is said for `sci_mod` also applies to `stat_mod`. The two exceptions are that the `stat_mod` proposal does not put `stat_parm` on a grid and therefore does not need the information to do so and that `stat_mod` has an additional prior termed π_κ in Section 1 and called `assess_prior` in the code that is used for model assessment.

The `sci_mod` constructor gets passed a pointer `dat_ptr` to the data, two `std::vector`s of `std::string` named `pfvec` and `alvec` and a `std::ostream` named `detail` to which details of the construction can be written. However, when the parallel version of the code is used, the constructor should not write anything. The data is passed because some simulators may want to use the data for initial lags. The contents of `pfvec` and `alvec` are entirely controlled by the user through the parmfile that is described immediately below; `pfvec` contains the contents of a file whose filename is specified as a line of the parmfile and `alvec` contains whatever additional lines the user chooses to add to the parmfile.

For the stochastic volatility model that we are using for illustration, `pfvec` is absent and `alvec` contains two lines of substance that determine the length N of the simulation returned by `gen_sim` and how many initial simulations to discard in order to dissipate transients. The other two lines of `alvec` are a header and trailer that are passed to but are to be ignored by the constructor. On the other hand, the SNP `stat_mod` gets nearly all the information needed for construction from `pfvec`, which is actually a parameter file produced by SNP.

The functionality of the member functions of `sci_mod` are straightforward. Member `gen_sim` is the most important. It returns the simulation in a `realmat` named `sim` of row dimension the same as the data, namely M, and column dimension N and also the functional of the scientific model computed from the simulation as a `realmat` that contains the vector `func` of dimension `sci_mod.len_func()` by 1.

If one uses a statistical model other than the SNP statistical model, then entirely analogous code is substituted for class `snp_stat_mod` in the files `gsmusr.h` and `gsmusr.cpp`. There are examples in directories `tt1d` and `tt2d` of the distribution. In the distribution, `svfx` and `tt1d` have code for use of the BFGS polish but `tt2d` does not.

Before illustrating the code for class `sv_sci_mod` we need to discuss the parameter file upon which part of it depends.

8.2 The Input Parameter File

The GSM input parameter file contains several blocks of control information. An example, taken from some debugging runs, follows. It was obtained using the EMM package. The parameter settings in the blocks for the SNP statistical model are from the EMM parmfile in directory `self` of the EMM distribution and for the scientific model they are from a parmfile in directory `svfx` of the EMM distribution.

```
PARMFILE HISTORY (optional)
#
# This parmfile was written by GSM Version 1.5 using the following line from
# control.dat, which was read as char*, char*
# ---------------------------------------------------------------------------
#        parmfile.in0                 tst
# ---------------------------------------------------------------------------
#
#  stat_mod parameters
#
#  eta 1 is  a0[1]  (degree 1)
#  eta 2 is  a0[2]  (degree 2)
#  eta 3 is  a0[3]  (degree 3)
#  eta 4 is  a0[4]  (degree 4)
#  eta 5 is  b0[1]
#  eta 6 is B(1,1)
#  eta 7 is  R0[1]  (support and penalty impose positivity on R0)
#  eta 8 is P(1,1)  (support and penalty impose positivity on P)
#  eta 9 is Q(1,1)  (support and penalty impose positivity on Q)
#
#
#  sci_mod parameters
#
#  theta 1 is a0  mean of level process
#  theta 2 is a1  autoregressive parameter of level process
#  theta 3 is b0  mean of volatility process
#  theta 4 is b1  autoregressive parameter of volatility process
#  theta 5 is s   scale of volatility process variance
#  theta 5 is r   correlation beteen level and volatility processes
#
# ---------------------------------------------------------------------------
#
ESTIMATION DESCRIPTION (required)
        svfx   Project name, pname, char*
         1.5   GSM version, defines format of this file, gsmver, float
           1   Write detailed output if print=1, int
           0   Prior draws in sci_mod chain if sci_draw_from_prior=1, int
           1   Run sci_mod chain if run_sci_chain=1, int
           0   Prior draws in stat_mod chain if stat_draw_from_prior=1, int
           1   Run stat_mod chain (i.e. assess_chain) if run_stat_chain=1, int
         1.0   Value of kappa for assess prior, kappa, float
DATA DESCRIPTION (required)
           1   Dimension of the data, M, int
         834   Number of observations, n, int
dmark.dat      File name, any length, no embedded blanks, dsn, string
4              Read these white space separated fields, fields, intvec
STAT_MOD DESCRIPTION (required)
           9   Number of parameters, len_stat_parm, int
           2   Number of functionals, len_stat_func, int
STAT_MOD PARMFILE (required) (constructor sees as vector<string> pfvec, alvec)
11114000.fit   File name, code __none__ if none, stat_parmfile, string
#begin additional lines
         100   Number of observations in simulated data, lsim (=N), int
          10   Initial simulations to eliminate transients, spin, int
#end additional lines
STAT_MOD MCMC DESCRIPTION (required) (describes assess chains)
      740726   Seed for stat_mod MCMC simulations, stat_seed, int
         100   Number stat_mod MCMC simulations per file, len_stat_chain, int
           1   Number of extra MCMC simulation files, num_stat_files, int
         1.0   Rescale proposal scaling by this value, stat_sclfac, float
STAT_MOD PARAMETER START VALUES (required)
   1.10674395197598967e-02    1    a0[1]   1
```

```
  -3.91003511770859902e-02    1    a0[2]   2
   4.33068128029127008e-04    1    a0[3]   3
   1.14430261931776966e-01    1    a0[4]   4
   1.01291480786568121e-03    1    b0[1]
   9.99252079756945966e-02    1    B(1,1)
   2.11708991450607092e-01    1    R0[1]      Postitivity imposed
   5.53673061392817134e-01    1    P(1,1)  s  Posiitivity imposed
   8.41406889779417799e-01    1    Q(1,1)  s  Posiitivity imposed
STAT_MOD PROPOSAL SCALING (required)
   6.95600000000000038e-03    a0[1]   1
   6.44799999999999974e-03    a0[2]   2
   3.73200000000000007e-03    a0[3]   3
   3.66000000000000006e-03    a0[4]   4
   1.03180000000000008e-02    b0[1]
   7.08400000000000002e-03    B(1,1)
   7.40999999999999992e-03    R0[1]
   7.36600000000000102e-03    P(1,1)  s
   3.78400000000000004e-03    Q(1,1)  s
STAT_MOD PROPOSAL GROUPING (optional) (frequencies are relative)
 0.2     1    5
   1   1.0 -0.6
   5  -0.6  1.0
 0.1    2
   2  1.0
 0.1    3
   3  1.0
 0.1    4
   4  1.0
 0.1    6
   6  1.0
 0.2    7    9
   7  1.0 -0.5
   9 -0.5  1.0
 0.1    8
   8  1.0
SCI_MOD DESCRIPTION (required)
           6   Number of parameters, len_sci_parm, int
           8   Number of functionals, len_sci_func, int
SCI_MOD PARMFILE (required) (constructor sees as vector<string> pfvec, alvec)
__none__       File name, code __none__ if none, sci_parmfile, string
#begin additional lines
       50000   Number of observations in simulated data, lsim (=N), int
         100   Initial simulations to eliminate transients, spin, int
#end additional lines
SCI_MOD MCMC DESCRIPTION (required)
      740726   Seed for sci_mod MCMC simulations, sci_seed, int
          10   Number sci_mod MCMC simulations per file, len_sci_chain, int
           1   Number of extra MCMC simulation files, num_sci_files, int
           0   Use analytic expression to compute mle if analytic_mle=1, int
          15   Length of sub chain to compute mle, len_sub_chain, int
           0   Number of extra sub chains, num_sub_chains, int
          15   Number of quasi-Newton iterates, num_polish_iter, int
    1.00e-09   Tolerance for quasi-Newton, polish_toler, float
         1.0   Rescale proposal scaling by this value, sci_sclfac, float
         2.0   Rescale parameter increments by this value, sci_incfac, float
SCI_MOD PARAMETER START VALUES (required)
   7.78808593750000000e-02    1  a0
   9.74121093750000000e-02    1  a1
   1.21093750000000000e-01    1  b0
   9.23828125000000000e-01    1  b1
   2.28515625000000000e-01    1  s
   1.56250000000000000e-02    1  r
```

```
SCI_MOD PROPOSAL SCALING (required)
   1.95312500000000000e-03   a0
   1.95312500000000000e-03   a1
   3.90625000000000000e-03   b0
   3.90625000000000000e-03   b1
   3.90625000000000000e-03   s
   3.90625000000000000e-03   r
SCI_PARAMETER INCREMENTS (required) (must be (fractional) powers of two)
   4.88281250000000000e-04   a0
   4.88281250000000000e-04   a1
   9.76562500000000000e-04   b0
   9.76562500000000000e-04   b1
   9.76562500000000000e-04   s
   9.76562500000000000e-04   r
SCI_MOD PROPOSAL GROUPING (optional) (frequencies are relative)
 0.1    1
   1  1.0
 0.1    2
   2  1.0
 0.2     3     4     5
   3   1.0   0.7  -0.7
   4   0.7   1.0  -0.8
   5  -0.7  -0.8   1.0
 0.1    6
   6  1.0
```

A description of each block of the input file follows.

8.2.1 PARMFILE HISTORY

This block is optional. It is written by program `gsm` to the output parmfiles `parmfile.fit` and `parmfile.end` at the end of every run. It consists of seven lines, each beginning with a #, that should be left alone. After these seven lines, the user can add additional lines that begin with a # and these will get copied from the input parmfile to the output parmfile. In the displayed parmfile are additional lines that describe the parameters of the statistical and scientific models.

Each run of `gsm` produces output parmfiles `parmfile.fit` and `parmfile.end` that may be used to restart computations: `parmfile.fit` restarts at the posterior mode; `parmfile.end` starts at the end of the chain that produced it. If `run_stat_chain`=1, then an additional parmfile `parmfile.alt` is produced that will restart `stat_mod`'s `assess_chain` at its posterior mode.

GSM is controlled by the contents of a file `control.dat`, each line of which produces an output parmfile. That line is placed in the sixth line of the PARMFILE HISTORY block as shown. Each line gives the name of the parmfile to be read and the prefix that is to be prepended to all output files. Thus, the name of the new parmfile would actually be `sv.parmfile.fit`. We shall omit this prefix in the discussion that follows.

8.2.2 ESTIMATION DESCRIPTION

In the block labeled `ESTIMATION DESCRIPTION`, there are parameters that govern the computations:

pname: Project name. Chosen by the user for identification purposes.

gsmver: Version of the GSM program. Necessary for backward compatibility because parmfile formats can change between versions.

print: If `print`=1, then voluminous debugging information is written to file detail.dat. To completely suppress printing, the control variable `print` in the SNP parmfile should be set to 0 also. For the parallel version of the code, both must be set to 0 to prevent at best a mess and at worst a crash.

sci_draw_from_prior: When `sci_draw_from_prior`=1 the output files `sci_parm.000.dat`, … contain draws from the prior rather than the posterior. This is useful for determining where a prior actually puts its mass. Also, some algorithms for computing posterior odds require this information. One must set `sci_chain`=1 when `sci_draw_from_prior`=1. It is usually best to leave `sci_mod` tuning parameters at the correct values for the case `draw_from_prior`=0 when simulations from `sci_mod` require nonlinear root finding, optimization, or equation solving in order to keep these computations from being destablized by larger proposal scaling.

sci_chain: The MCMC chain for the scientific model is run when `sci_chain`=1. The output are files named `sci_parm.000.dat` containing the MCMC posterior draws of the parameters of the scientific model, `sci_func.000.dat` containing the values of `sci_func` that correspond, `stat_parm.000.dat` containing the values of `stat_parm` that correspond (i.e. $\eta = g(\theta)$ where $g(\cdot)$ is the implied map, η is `stat_parm` and θ is `sci_parm`), etc. Two files are of special importance, `implied_map.new` and `assess_sigma.new`, which we next discuss.

stat_draw_from_prior: When `draw_from_prior`=1 the output files `as_st_parm.000.dat`, … contain draws from the prior rather than the posterior. One must set `stat_chain`=1 when `stat_draw_from_prior`=1. It is usually best to leave `sci_mod` tuning parameters at the correct values for the case `draw_from_prior`=0 when simulations from `sci_mod` require nonlinear root finding, optimization, or equation solving in order to keep these computations from being destablized by larger proposal scaling.

stat_chain: MCMC chain for the statistical model run with the `assess_prior` (i.e. π_κ) imposed is run when `stat_chain`=1. The assess prior cannot be computed unless the implied map $g(\cdot)$ and scale matrix Σ described in Section 1 are known. Thus, we must either have `sci_chain`=1 or have copied `implied_map.new` and `assess_sigma.new` from a previous run to the files `implied_map.dat` and `assess_sigma.dat` respectively if $g(\cdot)$ and Σ are to be known to `gsm`. Here it is important to be aware of prefixes. The output `.new` files will have the prefix prepended and the input `.dat` files must have that prefix as well; e.g. `sv.implied_map.dat`. The output files for the assess chain are labeled `as_st_parm.dat`, `as_st_func.dat`, etc.

kappa: The parameter `kappa` is the value to be used for `assess_prior`; i.e. the subscript κ of the prior π_κ.

8.2.3 DATA DESCRIPTION

In the block labeled `DATA DESCRIPTION` are parameters that specify the dimension of the data, the number of observations, and govern reading of the data. The data is presumed to be stored in a file containing rows that have values separated by blanks containing the data for each observation y_t and perhaps additional values such as dates or the index t. There should be one line for each $t = 1, \ldots, n$. (The presence of the line terminating character is important as the C++ function `getline` does the reading.)

M: The dimension of the vector y_t.

n: The number of observations to be read. The value can be smaller than the number of observations in the file in which case those at the end will not be read.

dsn: The name of the file from which the data is to be read.

fields : Lastly, one has `fields`. One must use care here because errors can cause the program to crash with misleading diagnostic messages, if any at all. As just mentioned, 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 because only the first M are read. 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 the file `detail.dat` which should be checked to make sure the data was read correctly. Fields can be specified as a single digit or as a range. Thus, one can enter either "1 2 3 5" or "1:3 5". (At time of writing, SNP does permit fields to be entered as a range.)

8.2.4 STAT_MOD DESCRIPTION

The STAT_MOD DESCRIPTION block is straightforward, it gives the dimensions of the parameters of the statistical model.

len_stat_parm : The dimension of `stat_parm`, which is the parameter vector of the statistical model that is denoted as η in Section 1.

len_stat_func: The dimension of `stat_func`, which is the vector of functionals of the statistical model that are computed from a simulation of the statistical model; `stat_func` is denoted as Ψ in Section 1.

8.2.5 STAT_MOD PARMFILE

The vectors `stat_pfvec` and `stat_alvec` of type `vector<string>` that are passed to the `stat_mod` constructor are defined in the STAT_MOD PARMFILE block. Note that the size of the simulation used to compute `stat_func` is not set by the GSM program but rather is determined by the user supplied `stat_mod` constructor. If the user wants the simulation size to be a value specified in the parmfile, then that value goes in this block as in our sample parmfile.

stat_parmfile: This is the name of a file containing lines of the user's choosing. For our sample parmfile this file is `11114000.fit` which is the output parmfile written by SNP. This file is read and passed to the `stat_mod` constructor as the `std::vector` of `std::string` `stat_pfvec`. If there is no `stat_parmfile` then code `__none__` as the filename.

#begin additional lines, #end additional lines: Lines between these two markers are read and passed to the `stat_mod` constructor as `stat_alvec` of type `vector<string>`. The two marker lines are passed as well so that the first user line is `stat_alvec[1]` and not `stat_alvec[0]`.

8.2.6 STAT_MOD MCMC DESCRIPTION

The STAT_MOD MCMC DESCRIPTION block mostly controls the MCMC simulations for model assessment. For this chain, draws are from the statistical model with the `stat_mod_prior` and the `assess_prior` imposed. This contrasts with the subchain used to compute the mle for the implied map in which case neither prior is imposed.

stat_seed: Seed for simulations. This seed affects only the chains for model assessment. The seed for the subchain is set by the GSM program and not is under the user's control.

len_stat_chain: The MCMC chain is broken up into pieces and written to files with names `as_st_parm_000.dat`, `as_st_parm_001.dat`, etc.; `len_stat_chain` determines the number of draws per file.

num_stat_files: Determines how many files in addition to `as_st_parm_000.dat` are generated. Total length of the MCMC chain is $R =$ `len_stat_chain`*(`num_stat_files`+1). Many other files are produced to describe the chain such as `as_st_func_000.dat` and `as_stat_logl_000.dat`. These individual chains can be concatenated in serial order to produce one long chain. Concatenation will usually but not always produce the same chain as would have been generated with `len_stat_chain`=R and `num_stat_files`=0 because a recomputation of the `stat_mod` likelihood may occur at the boundaries.

stat_sclfac: Rescales the proposal standard deviations that are set in the STAT_PROPOSAL SCALING block without changing relative values. This setting affects both the chain for model assessment and the subchain for computing the implied map. When writing `parmfile.end`, `parmfile.fit`, and `parmfile.alt`, the values in the new STAT_PROPOSAL SCALING block are rescaled and and the new stat_sclfac is set to 1.0.

8.2.7 STAT_MOD PARAMETER START VALUES

The STAT_MOD PARAMETER START VALUES block specifies the first value for both the model assess chain and the subchain for the implied map. The simulation it generates must satisfy the support conditions; i.e. `stat_mod::gen_sim` must return true, `stat_mod::support` must return true, and `stat_mod::prior` must return a value of `libgsm::dev_val.positive` = true for this initial value of `stat_parm` (i.e. η). The numbers to the right, 0 or 1, determine whether that element is held fixed or is active. If 0, then the proposal never moves that element of `stat_parm`. One can add annotation to the right as we have done in the example. As many characters of this annotation as will fit on a line are retained in `parmfile.end`, `parmfile.fit`, and `parmfile.alt`.

8.2.8 STAT_MOD PROPOSAL SCALING

These are the standard deviations of the proposal. They should be roughly proportional to the standard errors of an estimate of `stat_parm` if such is known. If not, they can be determined from `gsm` output as discussed in Section 9. Annotation can be added to the right as in the example.

8.2.9 STAT_MOD PROPOSAL GROUPING

Statistical model chains use a group move proposal which defaults to a single move proposal when the optional STAT_MOD PROPOSAL GROUPING block is missing from the parmfile. When the STAT_MOD PROPOSAL GROUPING block is missing, the proposal randomly selects an element of `stat_parm` to move and then draws from a normal; i.e. a move-one-at-a-time random walk. When the STAT_MOD PROPOSAL GROUPING block is present the proposal randomly selects one of the groups defined therein to move and draws from a user specified multivariate normal.

Each sub-block of the STAT_MOD PROPOSAL GROUPING block is a matrix, which defines a group. The number in the (1,1) position is the relative frequency with which that group is to be sampled. Continuing down the first column are the indexes of the variables in the group; continuing along the first row are these same indexes. Filling in the rest of the matrix is the correlation matrix for this group. Every index of `stat_parm` must be in some group and may be in in only one group.

For example, the sub-block

0.2	1	5
1	1.0	-0.6
5	-0.6	1.0

states that elements 1 and 5 of `stat_parm` are to be moved together with a relative frequency 0.2 and a correlation of -0.6.

The values in the STAT_MOD PROPOSAL SCALING block, scaled by `stat_sclfac`, are used as standard deviations to convert the correlation matrices to variance matrices. If some variables are fixed by coding 0's to the right of the values in the STAT_MOD PROPOSAL SCALING block, then `gsm` will pull them out of their listed groups and put them in a separate group that is moved with relative frequency 0.0. This is done automatically without user intervention. The details of the constructed proposal groups are written to file `detail.dat`.

The file `detail.dat` contains a listing of the groups and scaling that actually get used and should be examined to be sure that all the blocks that collectively determine the group move proposal were correctly interpreted.

8.2.10 SCI_MOD DESCRIPTION

The SCI_MOD DESCRIPTION block is analogous to the STAT_MOD DESCRIPTION block described in Subsection 8.2.4.

8.2.11 SCI_MOD PARMFILE

The SCI_MOD PARMFILE block is analogous to the STAT_MOD PARMFILE block described in Subsection 8.2.5.

8.2.12 SCI_MOD MCMC DESCRIPTION

The SCI_MOD MCMC DESCRIPTION block is analogous to the STAT_MOD MCMC DESCRIPTION block described in Subsection 8.2.5 with these exceptions:

analytic_mle: Explicit formulas or fast algorithms to compute the maximum likelihood esitimator are available for some statistical models. These formulas can be coded and used instead of MCMC subchains and quasi-Newton iterations to compute the maximum likelihood estimator. In this case one sets `analytic_mle = 1`. An example is `tt1d` included with the distribution. This option is not available for the SNP statistical model.

len_sub_chain: The subchain is the statistical model chain that is run within iterations of the scientific model chain to determine the implied map; `len_sub_chain` determines the number of draws for this computation.

num_sub_chains: When positive, the subchain is run num_sub_chains extra times. Each time the subchain is restarted at the mode of the previous subchain. This option is useful when num_polish_iter is positive. If it is used, one would reduce len_sub_chain correspondingly; i.e., if num_sub_chains is increased from 0 to 2, then set len_sub_chain to half its previous value.

num_polish_iter: If derivatives are coded for `stat_mod`, then a derivative based hill climber can be used to aid in the computation of the `stat_mod` mle. If derivatives are not coded, set `no_polish_iter`=0. If derivatives are coded, then `no_polish_iter` determines the maximum number of BFGS quasi-Newton iterations to be run using the last value of the MCMC subchain as the start. This is the value used as the likelihood. However, the next MCMC subchain starts from last computed value of the MCMC subchain, not the value computed by the BFGS hill climber.

polish_toler: The tolerance for the BFGS quasi-Newton iterations. Iterations stop when `num_polish_iter` is exceeded, the tolerence check succeeds, or the algorithm fails. If the hill climber fails to produce an improvement, the largest value in the subchain is used.

sci_incfac: To increase speed, the chain is cached. This cache can be thought of as a `std::vector` of `stuct` of type `libgsm::sci_val` for input and output purposes. Its internal representation is as a `std::map` containing `stuct` of type `libgsm::map_val` indexed by `sci_parm` of type `realmat`. The difference between a `map_val` and a `sci_val` is that a `sci_val` contains `sci_parm` and a `map_val` does not. A `sci_val` has all information required to make an MCMC draw from the scientific model. After every run, a new cache named `implied_map.new` is produced which the user should copy to `implied_map.dat` if the run was successful. To generate the cache, `sci_parm`, which is θ in Section 1, is put on grid as determined by the SCI_MOD PARAMETER INCREMENTS block described below. The variable `sci_incfac` allows one to make this grid coarser or finer without changing the relative increments; `sci_incfac` should be a power of two, e.g. 8 or 0.125.

8.2.13 SCI_MOD PARAMETER START VALUES

The SCI_MOD PARAMETER START VALUES block is analogous to the STAT_MOD PARAMETER START VALUES block described in Subsection 8.2.7.

8.2.14 SCI_MOD PROPOSAL SCALING

The SCI_MOD PROPOSAL SCALING block is analogous to the STAT_MOD PROPOSAL SCALING block described in Subsection 8.2.8.

8.2.15 SCI_PARAMETER INCREMENTS

The block labeled SCI_PARAMETER INCREMENTS determines the grid for caching. These increments should be determined by scientific relevance and be as coarse as possible. They should also be either integer or fractional powers of two. Here are some fractional powers of two.

```
5.00000000000000000e-01   =   0.50000000000000000000000
2.50000000000000000e-01   =   0.25000000000000000000000
1.25000000000000000e-01   =   0.12500000000000000000000
6.25000000000000000e-02   =   0.06250000000000000000000
3.12500000000000000e-02   =   0.03125000000000000000000
1.56250000000000000e-02   =   0.01562500000000000000000
7.81250000000000000e-03   =   0.00781250000000000000000
3.90625000000000000e-03   =   0.00390625000000000000000
1.95312500000000000e-03   =   0.00195312500000000000000
9.76562500000000000e-04   =   0.00097656250000000000000
4.88281250000000000e-04   =   0.00048828125000000000000
2.44140625000000000e-04   =   0.00024414062500000000000
1.22070312500000000e-04   =   0.00012207031250000000000
6.10351562500000000e-05   =   0.00006103515625000000000
3.05175781250000000e-05   =   0.00003051757812500000000
1.52587890625000000e-05   =   0.00001525878906250000000
7.62939453125000000e-06   =   0.00000762939453125000000
3.81469726562500000e-06   =   0.00000381469726562500000
1.90734863281250000e-06   =   0.00000190734863281250000
9.53674316406250000e-07   =   0.00000095367431640625000
4.76837158203125000e-07   =   0.00000047683715820312500
2.38418579101562500e-07   =   0.00000023841857910156250
1.19209289550781250e-07   =   0.00000011920928955078125
```

Increments can be no coarser than the values in SCI_MOD PROPOSAL SCALING if the proposal is to distribute mass on the grid in a reasonble way. Setting increments the same as the values in SCI_MOD PROPOSAL SCALING will permit moves of at most two increments up or down; setting it half will permit moves of at most four increments up or down. The implied maximum move in terms of grid increments is printed in the file `detail.dat` to help guide the choice. Annotation to the right of the increments is copied into `parmfile.fit`, `parmfile.end`, and `parmfile.alt`.

8.2.16 SCI_MOD PROPOSAL GROUPING

The SCI_MOD PROPOSAL GROUPING block is analogous to the STAT_MOD PROPOSAL GROUPING block described in Subsection 8.2.9.

There is one important difference: Storage requirements increase exponentially with group size because the support of the proposal density is a grid. This can become a problem when a large number of parameter values are fixed by coding a 0 to the right of their start value in the SCI_MOD PARAMETER START VALUES block because GSM will put them all in one group and then terminate with an error message rather than allow itself to consume excessive memory.

To fix the problem, one can adjust proposal scalings downward and/or parameter increments upward. Alternatively, code a 1 to the right and add a SCI_MOD PROPOSAL GROUPING block where all parameters are in groups small enough not to consume excessive space; e.g. put each parameter in its own group so all groups have size one. To hold the parameters in a group fixed, put the move frequency of that group to zero.

8.3 User Supplied Class Definition

Having now described the GSM parmfile, we can return to a discussion of the user supplied code for the scientific model in `gsmusr.h` and `gsmusr.cpp`.

Looking at the header `gsmusr.h` in Subsection 8.1, we see that there are four member functions that remain to be written: the constructor, the member that does the simulation, the member that checks that a proposed θ satisfies the model's support conditions, and the prior. Specifically, code must be supplied in `gsmusr.cpp` to implement these lines from `gsmusr.h`

```
sv_sci_mod
  (const scl::realmat* dat_ptr, const std::vector<std::string>& pfvec,
   const std::vector<std::string>& alvec, std::ostream& detail);
bool support(const scl::realmat& parm);
libgsm::den_val prior(const scl::realmat& parm, const scl::realmat& func);
bool gen_sim(scl::realmat& sim, scl::realmat& func);
```

Everything else has been coded in the header `gsmusr.h`.

The job of the constructor is to initialize the private members of usrmod by parsing the two vectors of strings from the SCI_MOD PARMFILE block of the parmfile that get passed to the constructor as the vectors of strings `pfvec` and `alvec`. For the stochastic volatility model we do not need the data for anything, `pfvec` is empty, and we will not write anything to the output stream `detail`. All we need deal with is `alvec`. Here is the constructor which parses `alvec`.

```
gsm::sv_sci_mod::sv_sci_mod
  (const realmat* dat_ptr, const vector<string>& pfvec,
   const vector<string>& alvec, ostream& detail)
: ltheta(6), lfunc(8)
{
  vector<string>::const_iterator pfv_ptr = alvec.begin();
  slen = atoi((++pfv_ptr)->substr(0,12).c_str());
  spin = atoi((++pfv_ptr)->substr(0,12).c_str());
}
```

We will compute four stats (funcs) for the generated data, min, max, mean, and standard deviation, and the same four for the latent volatility factor; a total of eight. Otherwise the code that implements `gen_sim` below is a straightforward implementation of equations (9) through (12) of Subsection 7.1 with precaution taken to force class `mcmc` of `mcmc` to reject the simulation when the `exp` function overflows by returning false when that happens. The documentation for the matrix class `realmat` is in its header `realmat.h` which is in the `libscl` distribution. Here is the code.

```
#include <cerrno>
#include "libgsm.h"
#include "gsm.h"
#include "gsmusr.h"

using namespace scl;
using namespace libgsm;
using namespace gsm;
using namespace std;

bool gsm::sv_sci_mod::gen_sim(realmat& sim, realmat& func)
{
  INT_32BIT seed = 740726;

  sim.resize(1,slen);
  realmat latent(1,slen);

  REAL a0 = theta[1];
  REAL a1 = theta[2];
  REAL b0 = theta[3];
  REAL b1 = theta[4];
  REAL s  = theta[5];
  REAL r  = theta[6];
  REAL rr2 = sqrt(1.0 - pow(r,2));

  REAL vlag = 0.0;
  REAL ylag = 0.0;
  errno = 0;

  for (INTEGER t=1; t<=spin; ++t) {
    REAL z1 = unsk(seed);
    REAL z2 = unsk(seed);
    REAL u1 = z1;
    REAL u2 = s*(r*z1 + rr2*z2);
    REAL v = b0 + b1*(vlag - b0) + u2;
    REAL y = a0 + a1*(ylag - a0) + u1*exp(v); // or u1*exp(vlag)
    vlag = v;                                 // see User's Guide
    ylag = y;
  }

  if (errno == ERANGE) return false;

  REAL ymin = REAL_MAX;
  REAL ymax = -REAL_MAX;
  REAL ymean = 0.0;
  REAL ysdev = 0.0;
  REAL vmin = REAL_MAX;
  REAL vmax = -REAL_MAX;
  REAL vmean = 0.0;
  REAL vsdev = 0.0;

  for (INTEGER t=1; t<=slen; ++t) {
    REAL z1 = unsk(seed);
    REAL z2 = unsk(seed);
    REAL u1 = z1;
    REAL u2 = s*(r*z1 + rr2*z2);
    REAL v = b0 + b1*(vlag - b0) + u2;
    REAL y = a0 + a1*(ylag - a0) + u1*exp(v); // or u1*exp(vlag)
    vlag = v;                                 // see User's Guide
    ylag = y;
    sim[t] = y;
    latent[t] = v;
    ymin = y < ymin ? y : ymin;
    ymax = y > ymax ? y : ymax;
    ymean += y;
    ysdev += pow(y,2);
    vmin = v < vmin ? v : vmin;
    vmax = v > vmax ? v : vmax;
    vmean += v;
    vsdev += pow(v,2);
  }

  if (errno == ERANGE) return false;

  if (lfunc != 8) error("Error, gsmusr, wrong size for func");
  if (func.size() != lfunc) func.resize(lfunc,1);

  ymean = ymean/REAL(slen);
  ysdev = sqrt( (ysdev - REAL(slen)*pow(ymean,2))/REAL(slen) );

  vmean = vmean/REAL(slen);
  vsdev = sqrt( (vsdev - REAL(slen)*pow(vmean,2))/REAL(slen) );

  func[1] = ymin;
```

```
  func[2] = ymax;
  func[3] = ymean;
  func[4] = ysdev;
  func[5] = vmin;
  func[6] = vmax;
  func[7] = vmean;
  func[8] = vsdev;

  return true;
}
```

We have four support conditions to check. The absolute value of r, which is `theta[6]`, must be less than one; s, which is `theta[5]`, must be positive; and the autoregressive parameters a_1 and b_1, which are `theta[2]` and `theta[4]` respectively, must be less than one. Here is the code that checks them.

```
bool gsm::sv_sci_mod::support(const realmat& parm)
{
  if (parm[5] <= 0.0) return false;
  if (fabs(parm[6]) >= 1.0) return false;
  if (fabs(parm[2]) >= 1.0) return false;
  if (fabs(parm[4]) >= 1.0) return false;
  return true;
}
```

A word of warning here. The model parameters are set before the support condition is checked. This is because SNP cannot check support until the parameters are set. Therefore the code that sets parameters will have to be bullet-proofed if trying to set invalid parameters causes problems. The support condition will be called immediately after the parameters are set, so one can assume no other method will be called before support is checked.

We have no reason to reject a completed simulation other than `exp` overflow, which we have already checked for during the simulation itself, so the prior need only concern itself with r and b_1. Here it is.

```
den_val gsm::sv_sci_mod::prior(const realmat& parm, const realmat& func)
{
  const REAL minus_log_root_two_pi = -9.1893853320467278e-01;

  den_val sum(true,0.0);

  REAL r = parm[6];
  REAL rmu = 0.0;
  REAL rsd = 0.05/1.96;
  REAL rz = (r - rmu)/rsd;
  REAL rld = minus_log_root_two_pi - log(rsd) - 0.5*pow(rz,2);

  sum += den_val(true,rld);
  REAL b1 = parm[4];
  REAL b1mu = 0.95;
  REAL b1sd = 0.1/1.96;
  REAL b1z = (b1 - b1mu)/b1sd;
  REAL b1ld = minus_log_root_two_pi - log(b1sd) - 0.5*pow(b1z,2);

  sum += den_val(true,b1ld);

  return sum;
}
```

9 Running the Example

In this section we illustrate the use of the package with a sequence of sample runs.

To keep a record of our work, we will separately name each control file which means we will have to use a command line argument when we execute `gsm`. The control file for the first run is `sv.ctrl.000.dat`. In our example, this has one line, which is

```
sv.parm.000.in0  sv
```

Here, `sv.parm.000.in0` is the name of the input parameter file and `sv` is the prefix to be added to all output files. The control file can have additional similar lines which allow different parmfiles for the same or different projects to be run at one time. The prefix must be different for every line or results will be overwritten.

For each run, we increment: `sv.ctrl.001.dat`, `sv.parm.001.in0`, `sv.ctrl.002.dat`, `sv.parm.002.in0`, and so on.

9.1 Tuning the Subchain

The initial parameter file `sv.parm.000.in0` is

```
ESTIMATION DESCRIPTION (required)
        svfx   Project name, pname, char*
         1.5   GSM version, defines format of this file, gsmver, float
           1   Write detailed output if print=1, int
           0   Prior draws in sci_mod chain if sci_draw_from_prior=1, int
           1   Run sci_mod chain if run_sci_chain=1, int
           0   Prior draws in stat_mod chain if stat_draw_from_prior=1, int
           0   Run stat_mod chain (i.e. assess_chain) if run_stat_chain=1, int
         1.0   Value of kappa for assess prior, kappa, float
DATA DESCRIPTION (required)
           1   Dimension of the data, M, int
         834   Number of observations, n, int
dmark.dat      File name, any length, no embedded blanks, dsn, string
```

```
4              Read these white space separated fields, fields, intvec
STAT_MOD DESCRIPTION (required)
           9   Number of parameters, len_stat_parm, int
           2   Number of functionals, len_stat_func, int
STAT_MOD PARMFILE (required) (constructor sees as vector<string> pfvec, alvec)
11114000.fit   File name, code __none__ if none, stat_parmfile, string
#begin additional lines
         100   Number of observations in simulated data, lsim (=N), int
          10   Initial simulations to eliminate transients, spin (=N0), int
#end additional lines
STAT_MOD MCMC DESCRIPTION (required) (describes assess chains)
      740726   Seed for stat_mod MCMC simulations, stat_seed, int
           2   Number stat_mod MCMC simulations per file, len_stat_chain, int
           0   Number of extra MCMC simulation files, num_stat_files, int
       0.125   Rescale proposal scaling by this value, stat_sclfac, float
STAT_MOD PARAMETER START VALUES (required)
 -0.05161    1    a0[1]   1
  0.04295    1    a0[2]   2
  0.04028    1    a0[3]   3
  0.11637    1    a0[4]   4
  0.07282    1    b0[1]
  0.05833    1    B(1,1)
  0.15943    1    R0[1]
  0.37897    1    P(1,1)  s
  0.89804    1    Q(1,1)  s
STAT_MOD PROPOSAL SCALING (required)
  0.03478      a0[1]   1
  0.03224      a0[2]   2
  0.01867      a0[3]   3
  0.01830      a0[4]   4
  0.05159      b0[1]
  0.03542      B(1,1)
  0.03705      R0[1]
  0.03683      P(1,1)  s
  0.01892      Q(1,1)  s
SCI_MOD DESCRIPTION (required)
           6   Number of parameters, len_sci_parm, int
           8   Number of functionals, len_sci_func, int
SCI_MOD PARMFILE (required) (constructor sees as vector<string> pfvec, alvec)
__none__       File name, code __none__ if none, sci_parmfile, string
#begin additional lines
       50000   Number of observations in simulated data, lsim (=N), int
        1000   Initial simulations to eliminate transients, spin, (=N0) int
#end additional lines
SCI_MOD MCMC DESCRIPTION (required)
      740726   Seed for sci_mod MCMC simulations, sci_seed, int
          25   Number sci_mod MCMC simulations per file, len_sci_chain, int
           1   Number of extra MCMC simulation files, num_sci_files, int
           0   Use analytic expression to compute mle if analytic_mle=1, int
         100   Length of sub chain to compute mle, len_sub_chain, int
           0   Number of extra sub chains, num_sub_chains, int
           0   Number of quasi-Newton iterates, num_polish_iter, int
    1.00e-10   Tolerance for quasi-Newton, polish_toler, float
         1.0   Rescale proposal scaling by this value, sci_sclfac, float
         1.0   Rescale parameter increments by this value, sci_incfac, float
SCI_MOD PARAMETER START VALUES (required)
   0.0546875  1   a0
   0.0234375  1   a1
   0.0        1   b0
   0.9453125  0   b1
   0.8984375  1   s
   0.0        0   r
```

```
SCI_MOD PROPOSAL SCALING (required)
   0.001953125    a0
   0.001953125    a1
   0.001953125    b0
   0.001953125    b1
   0.001953125    s
   0.001953125    r
SCI_PARAMETER INCREMENTS (required) (must be (fractional) powers of two)
   0.000244140625    a0
   0.000244140625    a1
   0.000244140625    b0
   0.000244140625    b1
   0.000244140625    s
   0.000244140625    r
```

This file was described in detail in Subsection 8.2 and there is little more about it that needs to be said here except where the numbers came from.

The parameter values and scale for the statistical model come directly from the SNP parmfile shown in Subsection 7.2. Because that model was fit with $n = 834$ and our simulation from the scientific model has $N = 50000$, we rescale by $0.125 \doteq \sqrt{n/N}$.

The parameter start values for the scientific model are set to a (fractional) power of two close to the values in Subsection 7.1. Initially we will fix b_1 and r at a (fractional) power of two near the means of their priors, which are 0.95 and 0.0, respectively, by coding 0 to their right instead of 1.

The reason that all parameter values were changed from the values in Subsection 7.1 to a nearby (fractional) power of two is that only values on the grid will be proposed which has the effect of moving a parameter value that is not on the grid regardless of the 0's or 1's coded to the right of them. If we do not start at a power of two and have a 0 coded, then that value will actually get moved from its parmfile value to the nearest (fractional) power of two at every draw. Those that have a 1 to the right will suffer the same fate until a draw is accepted. None of this does any real harm but it does make accept/reject counts look strange although, strange as they may appear, they are, technically, correct.

The parameter increments are set to a power of two near 0.0002 and the scaling for the proposal is a power of two about ten times that. This is just a guess and for the moment it does not matter much because we are only trying to tune the η chain for the statistical model.

Another way to get starting values, as was mentioned in Section 7, would be to fit the model using the EMM package. Fitting the SNP model to itself using EMM provides start values and tuning information for the subchain (i.e. the η chain). Fitting the scientific model by EMM using the SNP statistical model as the score generator provides reasonable starting values for the scientific model and tuning parameters for the scientific model MCMC chain (i.e. the θ chain). That is how the parmfile shown in Section 8.2 was obtained.

We now execute `gsm` by typing

```
gsm sv.ctrl.000.dat
```

We get two warning messages.

```
Warning, gsm, could not read implied map, will compute afresh
Warning, gsm, could not read assess sigma, will compute afresh
```

These warning messages are because the files `sv.implied_map.dat` and `assess_sigma.dat` were not present. For the next few runs we will not be using results from previous runs so we will keep getting these messages. When we get closer to getting the θ and η chains tuned we will copy `sv.implied_map.new` and `assess_sigma.new` to `sv.implied_map.dat` and `assess_sigma.dat`, respectively. If tuning parameters do not change too much, these files will be accepted and used. If not, they will be rejected with warning messages.

The result is the following set of files. There would be more if `run_stat_chain`=1; summary files are not listed.

```
sv.assess_sigma.new          sv.sci_sub_logl.000.dat
sv.detail.dat                sv.sci_sub_logl.001.dat
sv.implied_map.new           sv.sci_sub_parm.000.dat
sv.parmfile.000.end          sv.sci_sub_parm.001.dat
sv.parmfile.001.end          sv.sci_sub_rej.000.dat
sv.parmfile.end              sv.sci_sub_rej.001.dat
sv.parmfile.fit              sv.stat_func.000.dat
sv.sci_func.000.dat          sv.stat_func.001.dat
sv.sci_func.001.dat          sv.stat_logl.000.dat
sv.sci_mode.000.dat          sv.stat_logl.001.dat
sv.sci_mode.001.dat          sv.stat_parm.000.dat
sv.sci_parm.000.dat          sv.stat_parm.001.dat
sv.sci_parm.001.dat          sv.stat_prior.000.dat
sv.sci_prior.000.dat         sv.stat_prior.001.dat
sv.sci_prior.001.dat         sv.stat_sub_logl.000.dat
sv.sci_rej.000.dat           sv.stat_sub_logl.001.dat
sv.sci_rej.001.dat
```

The contents of these files all relate to the MCMC chain for the scientific model (i.e. the θ chain). The log posterior is the sum of the values in the files `sv.sci_prior.000.dat`, `sv.stat_logl.000.dat`, and `sv.stat_prior.000.dat`. These files have logical length `len_sci_chain` and their values are aligned with the parameter values in `sv.sci_parm.000.dat`, which has logical dim*ension* `len_sci_parm` rows and `len_sci_chain` columns. Prepended to each file are two integers giving the row and column dimensions as discussd below.

It is important to understand the differences among the `logl` files. The values in `sv.stat_logl.000.dat` are for the statistical model evaluated at $\eta = g(\theta)$ and at the data. The values in `sv.stat_sub_logl.000.dat` are for the statistical model evaluated at $\eta = g(\theta)$ and at the simulation from the scientific model at θ . (`sv.stat_sub_logl.000.dat` is new to Version 1.4.) The contents of `sv.sci_sub_logl.000.dat` are discussed below. The critical file is `sv.stat_logl.000.dat` because it is the one used to compute the posterior. The others are to provide help in tuning MCMC chains.

The first two entries in each of these files are the row and column numbers. They are written by `vecwrite` and can be read by `vecread` from library `libscl`; `vecread` and `vecwrite` are presented in `realmat.h` and defined in `realmat.cpp`. They follow the same formatting conventions as Matlab®. The files `sv.sci_func.000.dat` and `sv.stat_func.000.dat` are similarly formatted and aligned. If the compiler is correctly written and the machine is 32 bit and adheres to IEEE standards, then reading these files with `vecread` will produce the same internal binary representation of the `realmat` that wrote them. The formatting of `sv.implied_map.new` is determined by `operator>>` of struct `sci_val` of `libgsm`, which is presented in `libgsm_base.h` and defined in `libgsm_base.cpp`. What is written by `operator>>` can be read using `operator<<`. The file `sv.sci_mode.000.dat` is that entry in the implied map that corresponds to the mode. Everything for the additional files of the form `...001...` is the same. Files can be meaningfully concatenated; e.g. `sv.sci_parm.dat`=`sv.sci_parm.000.dat`+`sv.sci_parm.001.dat`. Often files of the form `...000...` are discarded to dissipate transients. The parmfiles with extensions `.fit` and `.end` and files of the form `...sub...` and `...rej...` are used to tune the MCMC chains as discussed next.

For initial tuning, information of most interest is the rejection rate of the sci chain (i.e. the θ chain) in `sv.sci_rej.000.dat`, the likelihoods of the first 25 subchains (i.e. η chains) in `sv.sci_sub_logl.000.dat`, the first 25 subchains in `sv.sci_sub_parm.000.dat`, the rejection rates of the first 25 subchains `sv.sci_sub_rej.000.dat`, and this portion of `sv.detail.dat`:

```
*****************************************************************
*                                                               *
*         Implied maximum move of grid_group_move proposal      *
*         formula is INTEGER(2.0*sci_scale[i]/sci_incr[i])      *
*                                                               *
*****************************************************************

                            Row 1   16
                            Row 2   16
                            Row 3   16
                            Row 4   16
                            Row 5   16
                            Row 6   16
```

The above states that we have set the proposal scaling and the proposal increments for the scientific model such that an element of θ can move at most 16 increments to the right or left.

The file `sv.sci_rej.000.dat` looks like this:

```
                Col  1      Col  2      Col  3      Col  4

     Row  1    1.00000     0.20000     5.00000     5.00000
     Row  2    1.00000     0.32000     8.00000     8.00000
     Row  3    1.00000     0.12000     3.00000     3.00000
     Row  4        0.0         0.0         0.0         0.0
     Row  5    1.00000     0.36000     9.00000     9.00000
     Row  6        0.0         0.0         0.0         0.0
     Row  7    1.00000     1.00000    25.00000    25.00000
```

The fourth column gives the number of moves for each parameter with the last row being the total. The third column gives the number of rejections. The second column is the proportion that each parameter was moved; i.e. the elements of the fourth column divided by the last element of the fourth column. The first column gives the rejection rates, by parameter and in total; i.e. the elements of the third column divided by the corresponding elements of the fourth column. In this instance we have rejected every move.

The later columns of the file `sv.sci_sub_rej.000.dat` look like this:

```
              Col 13      Col 14      Col 15      Col 16      Col 17      Col 18

 Row  1      0.33333     0.36364     0.57143     0.57143     0.30769     0.58824
 Row  2      0.80000     0.52632     0.42857     0.36364     0.42857     0.25000
 Row  3      0.36364     0.33333    0.083333     0.50000         0.0     0.25000
 Row  4      0.63636     0.58333     0.42857     0.55556     0.50000     0.66667
 Row  5      0.36364     0.28571     0.41667     0.44444     0.66667     0.33333
 Row  6      0.50000     0.60000     0.54545     0.38462     0.50000     0.54545
 Row  7      0.44444     0.41667    0.090909     0.50000     0.69231     0.50000
 Row  8      0.35714     0.36364     0.44444     0.30000     0.22222     0.44444
 Row  9      0.50000     0.72727     0.62500     0.50000     0.50000     0.71429
 Row 10      0.46000     0.47000     0.40000     0.46000     0.41000     0.50000
```

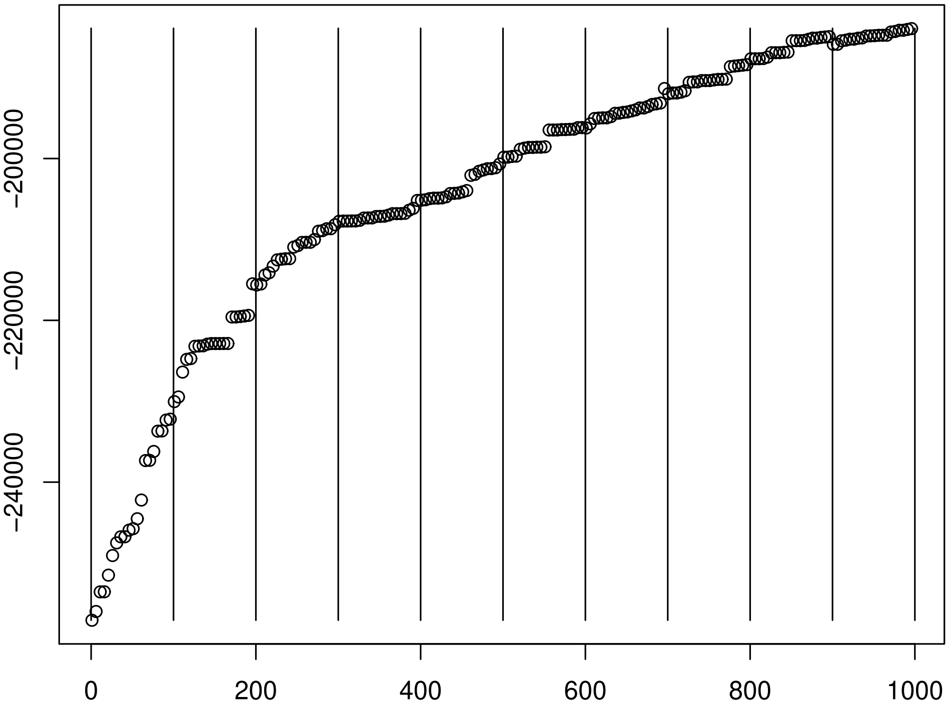

Figure 5. Subchain from Parameter File sv.parm.000.in0. Ten successive runs of the η subchain. Each run is 100 iterations of which every fifth is plotted. The values of log-likelihood of the simulated data set are on the vertical axis. Vertical bars mark where θ changes. Jumps are because $\{\hat{y}_t\}_{t=1}^N$ changes at each vertical bar.

These are the first columns of the rejection matrix for the subchain; there are 25 columns in total with zero right padding when the actual number of valid entries is less than 25. The file `sv.sci_sub_logl.000.dat` contains the information needed to produce Figure 5. It too has 25 columns with right padding when the actual is less.

Comparing Figure 5 to Figure 3 we see that we are far from having a correctly tuned η chain. The most probable reason that the chain for the scientific model (i.e. the θ chain) does not move is that we have a horrid start value for the subchain (i.e. the η chain) and hence do not correctly compute $\hat{\eta} = \underset{1 \le i \le 100}{\operatorname{argmax}} \sum_{t=1}^N \log f(\hat{y}_t | \hat{x}_{t-1}, \eta_i)$. The evidence that we are not computing $\hat{\eta}$ accurately is the steady rise of the log likelihood seen Figure 5. What can happen is that the computed $\hat{\eta}$ of the first subchain happens to have the largest value of the likelihood evaluated at the data, which is the value $\sum_{t=1}^{n} \log f(\tilde{y}_t|\tilde{x}_{t-1}, \hat{\eta})$. This large, erroneous value can block all subsequent moves. This seems to be the problem here although determining the exact cause is not important at this early stage while our focus is on the subchain.

In Version 1.1 of GSM a change was made to try and eliminate this particular cause of sticky behavior: If the θ chain sticks at a θ for a number of times in succession, the $\hat{\eta}$ corresponding to θ is recomputed. The number of times the chain is allowed to stick before this recomputation occurs is twice the length of θ. Also, in Version 1.1 a BFGS polish was added as discussed in Subsection 9.5.

To determine the cause of a sticky θ chain, or any other mystifying behavior, change `undef` in the line `#undef SCI_MCMC_DEBUG` in `mcmc.cpp` in directory `lib/libgsm/src` to `define`, recompile, and run with output redirection; e.g. `gsm sv.ctrl.000.dat>gsm.out`. Similarly, one can examine the moves of the η chain by changing `#undef STAT_MCMC_DEBUG`.

Version 1.2 added the capability to draw from the prior of the scientific model; Version 1.3 added the same for the statistical model.

The changes between Versions 1.3 and 1.4 are as follows. The chains `sci_sub_logl`, `sci_sub_parm`, and `sci_sub_reject` are computed differently. Now, the subchains in these files are those actually run. Previous versions of GSM excluded the first. Also now, if a polish succeeds, the value computed by the polish replaces the end value of a `_sub_` chain. Previously versions appended. If a polish fails, the replacement is from the mode of the chain. If there is no polish, no replacement occurs. The ability to run the subchain more than once was added; see Subsection 9.5 for more details.

Although parmfiles have been updated to Version 1.4 syntax, all results in the Guide were actually produced by Versions 1.0, excepting that the results in Subsection 9.5 are from Version 1.1.

We next copy `sv.parmfile.end` to `sv.parm.001.in0` and set `len_sci_chain` to 100, and run again. As discussed in Section 8, `sv.parmfile.fit` and `sv.parmfile.end` are for restarting the chain. As noted earlier, the difference between `sv.parmfile.end` and `sv.parmfile.fit` is that the `.end` parmfile has its parameter values set to those at the end of the chain whereas the `.fit` parmfile has its values set to those of the mode of the posterior.

The parmfile `sv.parmfile.000.end` is for restarting at the end of `sv.sci_parm.000.dat`; similarly for `sv.sci_parm.001.end`, but, because it is the last, `sv.sci_parm.001.end` and `sv.sci_parm.end` are the same. However, we have not yet correctly computed the mode of the posterior so there is no point to using the `.fit` parmfile. But the `.end` parmfile allows us to begin where we left off, which is useful.

What happens is much the same as we have just seen. The subchain is not yet burned in. We will copy `sv.parmfile.end` to `sv.parm.002.in0` and run again. What we need is a plot that looks like Figure 3. What we have at the end of run 001 (not shown) looks like Figure 5.

On termination of run 002, it looks like burn-in may have been achieved. A plot of `sv.sub_logl_001.dat` (not shown) looks like Figure 3. We will run one more time, but with b_1 and r free. We still have a 100% rejection rate for the scientific model chain which means that the increment is set wrong. Setting the increment is a delicate process. Too small and inaccuracies in computing the likelihood become large relative to the increment in the likelihood which can cause problems, among which can be a sticky chain. Too large and the chain sticks for sure because the proposed move is too far from the mode of the posterior. We will try a reduction in the increment to see what happens. If worse comes to worse we can increase the simulations size N of the scientific model and/or increase `len_sub_chain` but for now we will hold off because the entire cost of the computations depends on those two numbers and we want to keep them as small as we can get away with.

We copy `sv.parmfile.end` to `sv.parm.003.in0`, set the zeros to the right of b_1 and r to ones; set `sci_sclfac` to 0.25, set `sci_incfac` to 0.5, and run again.

On termination of run 003, the files `sv.sci_rej.001.dat` and `sv.sci_sub_rej.001.dat` are as follows:

```
                Col   1       Col   2       Col   3       Col   4

Row  1        0.60000       0.15000       9.00000      15.00000
Row  2        0.57143       0.14000       8.00000      14.00000
Row  3        0.77778       0.18000      14.00000      18.00000
Row  4        0.61905       0.21000      13.00000      21.00000
Row  5        0.68750       0.16000      11.00000      16.00000
Row  6        0.87500       0.16000      14.00000      16.00000
Row  7        0.69000       1.00000      69.00000       100.000

Col 19        Col 20        Col 21        Col 22        Col 23        Col 24
```

```
Row  1      0.50000      0.55556      0.71429      0.50000      0.53333      0.38462
Row  2      0.60000      0.72727      0.35714      0.45455      0.90000      0.58333
Row  3      0.30000      0.12500      0.41667      0.25000      0.18182      0.29412
Row  4      0.23077      0.27273      0.11111      0.42857      0.33333      1.00000
Row  5      1.00000      1.00000      1.00000      0.92857      1.00000      1.00000
Row  6      0.23077      0.23077      0.35294      0.22222      0.20000      0.27273
Row  7      0.77778      0.88889      0.90000      1.00000      0.92308      1.00000
Row  8          0.0      0.22222          0.0      0.25000     0.090909          0.0
Row  9      0.60000      0.37500      0.61538      0.84615      0.27273      0.62500
Row 10      0.46000      0.54000      0.51000      0.55000      0.48000      0.60000
```

It is also of interest to examine the moves made by the parameters of the subchain. This information is in `sv.sub_parm_001.dat` which contain the first 25 subchains of the second half of the run. These, together with `sv.sub_logl_001.dat` are plotted in Figure 6. The figure and rejection rates suggest that the scale for the proposal for η_5 and η_7 should be reduced and that for η_8 should be increased. Because a plot of `sv.sub_logl_001.dat` (not shown, but see bottom panel of Figure 6) looks reasonable, our present guess is that the increment should be left alone for the moment. But more information would help, so we will now set about getting it.

We copy `sv.parmfile.end` to `sv.parm.004.in0`. In the STAT_MOD PROPOSAL SCALING block, we halve the proposal scale factors for η_5, and η_7, which are the location parameters of the mean function and the the variance function of the SNP statistical model, respectively, and double the scale of η_8, which is the MA part of the GARCH variance function. We will also decrease `len_sci_chain` to 50, increase `num_sci_files` to 9, and run again. The reason for breaking the chain up into smaller pieces is to generate a large number of observations from the files `sv.sub_parm_00x.dat` in order to get more information on the subchains. This will also enable us to compute correlations to configure a group move proposal for the subchain.

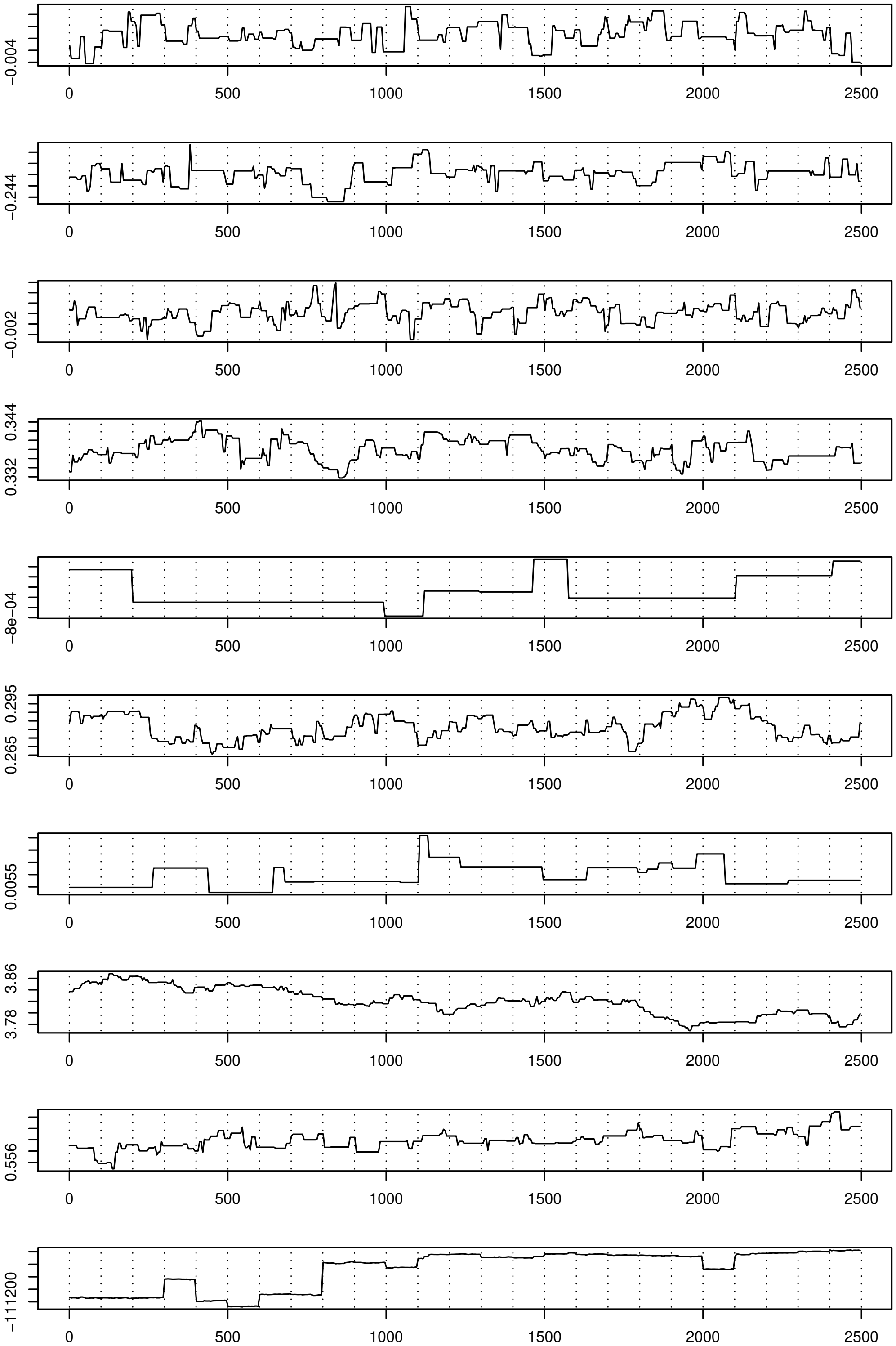

Figure 6. Subchain from Parameter File sv.parm.003.in0. Two sets of ten successive runs of the η subchain. Each run is 100 iterations of which every point is plotted. The values of η_i are shown in the first nine panels and the log-likelihood of the simulated data set in the last. The dotted lines mark where θ changes.

Here are the rejection rates at the end of run004 from file `sv.sci_sub_rej.009.dat`:

```
            Col 19      Col 20      Col 21      Col 22      Col 23      Col 24

 Row  1    0.45455     0.77778     0.81250     0.66667     0.58333     0.60000
 Row  2    0.26667     0.45455     0.60000     0.60000     0.50000     0.50000
 Row  3    0.35714     0.53846     0.38462         0.0     0.40000     0.14286
 Row  4    0.57143     0.20000     0.12500         0.0     0.33333     0.10000
 Row  5    0.86667     0.87500     0.90000     0.92857     0.90000     0.80000
 Row  6    0.46154    0.090909         0.0     0.20000     0.27273     0.45455
 Row  7    0.77778     0.90909     0.61538     0.80000     0.71429     0.76471
 Row  8        0.0    0.090909     0.16667     0.50000     0.10000     0.20000
 Row  9    0.41667     0.50000     0.37500     0.50000     0.58333     0.53846
 Row 10    0.49000     0.48000     0.47000     0.51000     0.48000     0.45000
```

We have some improvement, but not as much as one might have hoped. We copy `sv.parmfile.end` to `sv.parm.005.in0`. For the subchain, we will divide the proposal scale factor for η_5 by four and η_7 by two because their rejection rate is still much higher than others. A plot of the subchain (not shown) suggests that the scale for η_5 needs more reduction than the scale for η_7. Figure 7, which plots every move of the parameters of the scientific model, suggests that we may be able to get away with increasing the increment because it looks like nearly every accepted move was larger than the smallest possible increment. There is certainly an incentive for doing so: the larger the increment, the faster the chain will run because proposed moves will be found in the implied map more frequently. We set `sci_incfac` to 2.0.

Here are the overall rejection rates at the end of run005:

```
theta_1   0.6851154      eta_1   0.5396179
theta_2   0.6636797      eta_2   0.535367
theta_3   0.6054365      eta_3   0.272621
theta_4   0.6315368      eta_4   0.2483388
theta_5   0.571847       eta_5   0.7391988
theta_6   0.5449495      eta_6   0.2325578
                         eta_7   0.6191235
                         eta_8   0.2210103
                         eta_9   0.530551
theta     0.63           eta     0.437
```

Inspection of a plot every move of the parameters of the scientific model (not shown), suggests that we may be still able to get away with increasing the increment because, once again, it looks like nearly every accepted move was larger than the smallest possible increment.

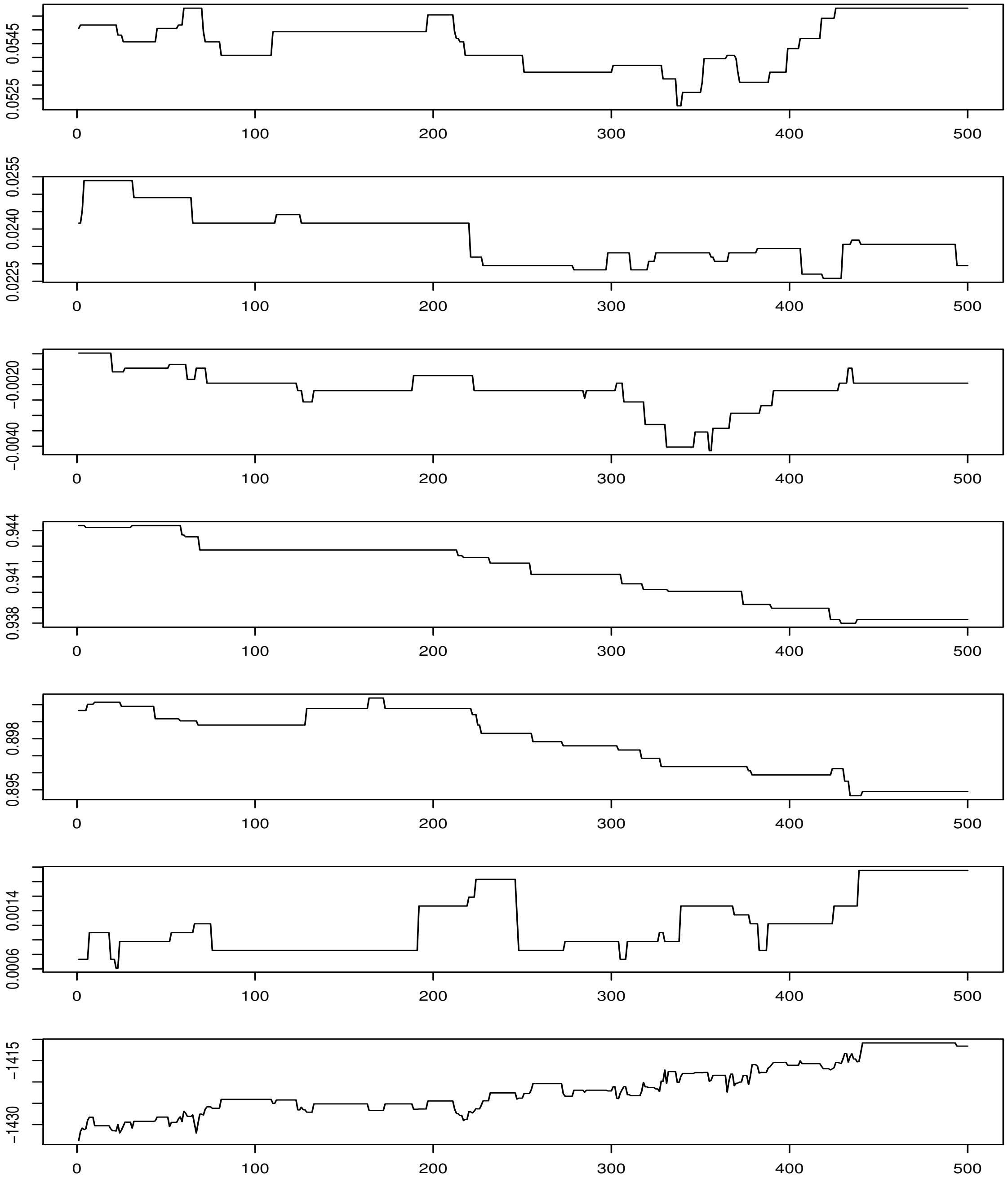

Figure 7. MCMC Chain from Parameter File sv.parm004.in0. The panels are draws of the elements of $\theta = (a_0,\, a_1,\, b_0,\, b_1,\, \mathrm{s},\, r)$ in order from top to bottom; the last panel is proportional to the log posterior density evaluated at θ. Every fifth point is plotted. $R = 500$.

We copy `sv.parmfile.end` to `sv.parm.006.in0` and make the following adjustments: We set `sci_incfac` to 2.0. We reduce the scale for η_1, η_2, and η_9 by 10%; we reduce the scale of η_5 and η_7 by 50%; and we increase the scale for η_3, η_4, η_6, and η_8 by 50%. To compensate for the general increase, we set `stat_sclfac` to 0.5.

The correlations among the parameters of the subchain from run005 are as follows with those less than 0.6 in absolute value put to zero:

```
   V1 V2 V3 V4 V5 V6 V7         V8         V9
V1  1  0  0  0  0  0  0  0.0000000  0.0000000
V2  0  1  0  0  0  0  0  0.0000000  0.0000000
V3  0  0  1  0  0  0  0  0.0000000  0.0000000
V4  0  0  0  1  0  0  0  0.0000000  0.0000000
V5  0  0  0  0  1  0  0  0.0000000  0.0000000
V6  0  0  0  0  0  1  0  0.0000000  0.0000000
V7  0  0  0  0  0  0  1  0.0000000  0.0000000
V8  0  0  0  0  0  0  0  1.0000000 -0.6579042
V9  0  0  0  0  0  0  0 -0.6579042  1.0000000
```

The relationship between η_8 and η_9 is linear as is seen from the scatter plots shown in Figure 8. To introduce a group move strategy that reflects these correlations, we insert the following block into `sv.parm.006.in0`:

```
STAT_MOD PROPOSAL GROUPING (optional) (frequencies are relative)
 0.1    1
   1  1.0
 0.1    2
   2  1.0
 0.1    3
   3  1.0
 0.1    4
   4  1.0
 0.1    5
   5  1.0
 0.1    6
   6  1.0
 0.1    7
   7  1.0
 0.2      8      9
   8    1.0  -0.66
   9  -0.66    1.0
```

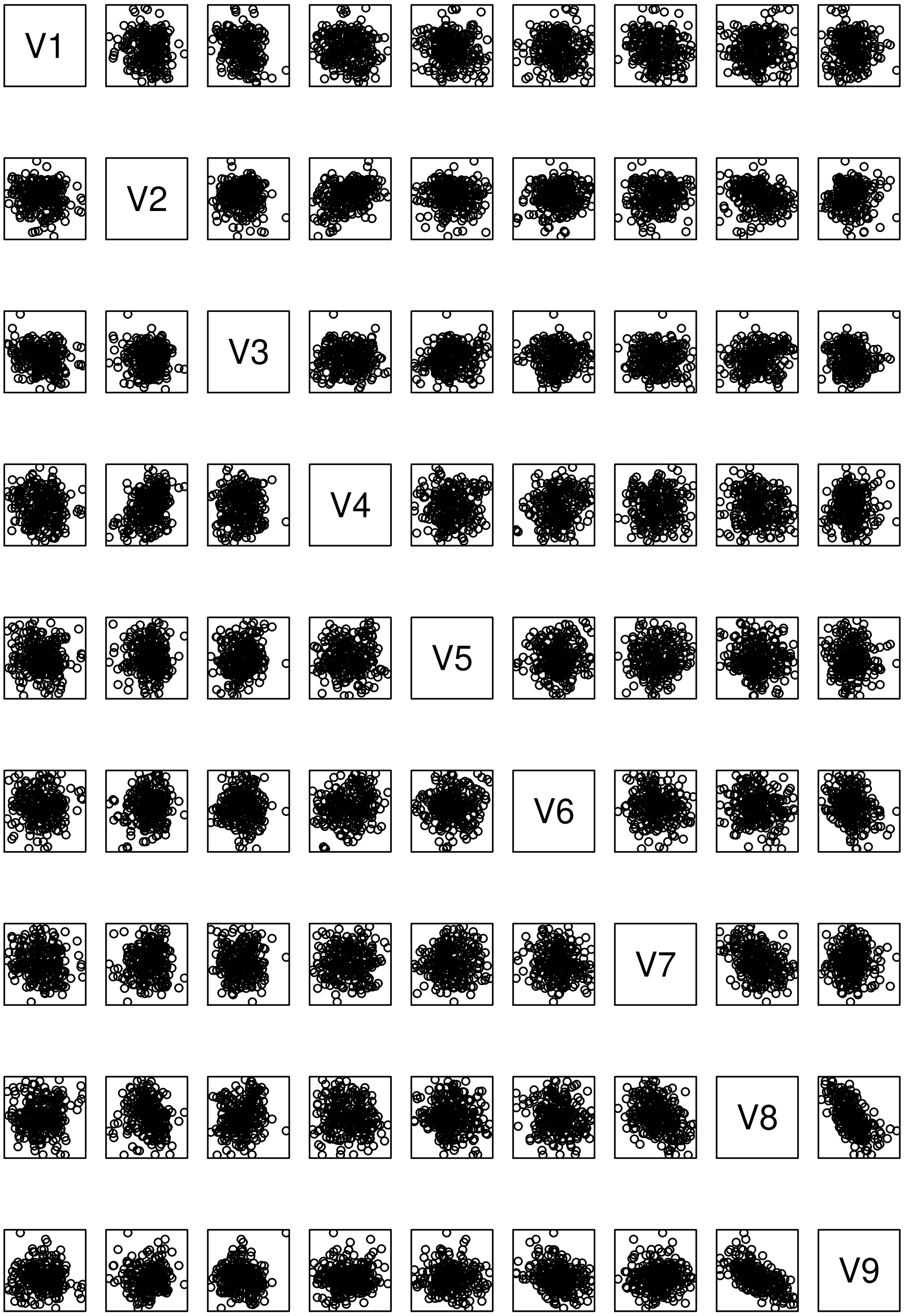

Figure 8. Scatter Plots of Chain from Parameter File sv.parm.005.in0. Every 100th point is plotted. $R = 25000$.

The run looks good. Here are the rejection rates.

```
theta_1     0.5899856       eta_1     0.2909092
theta_2     0.7010823       eta_2     0.2862135
theta_3     0.6625397       eta_3     0.2372093
theta_4     0.6037121       eta_4     0.1894127
theta_5     0.6615296       eta_5     0.3408927
theta_6     0.6066162       eta_6     0.1811702
                            eta_7     0.2227835
                            eta_8     0.2939598
                            eta_9     0.2939598
theta       0.642           eta       0.25724
```

Plots of `sv.sci_sub_logl.dat` (not shown) look like Figure 3. We are finished tuning the subchain.

9.2 Tuning the Sci_Mod Chain

We now have to tune the chain for the scientific model. Also we must keep running the chain until the posterior quits climbing and it looks like the mode has been reached.

We copy `sv.ctrl.parmfile.end` to `parm.007.in0`, leave all settings the same except that we increase `len_sci_chain` to 200. At the present settings of the increment and scale, we are only allowing two increment moves at most. We repeat with `parm.008.in0`, copying `sv.implied_map.new` to `sv.implied_map.dat`.

Then, having processors to spare, we copy `sv.ctrl.parmfile.end` to `parm.009.in0` and `sv.implied_map.new` to `sv.implied_map.dat` into four separate directories and try four simultaneous runs with `(sci_sclfac, sci_incfac)`=(1,1), (2,1), (2,2), and (4,2). The run that performed best in terms of moving uphill the most was `(sci_sclfac, sci_incfac)` = (4,2). At the conclusion of the run, here are the moves that were made:

```
            -------------   increment   --------------

            -4   -3   -2   -1    0    1    2    3    4    reject

theta_1      5   18   35   42  165   26   28   15    7      0.49
theta_2      3   16   17   26  163   46   26   17    6      0.51
theta_3      7   14   28   41  168   24   32   12    5      0.50
theta_4      1    8   14   35  180   42   34   18    9      0.51
theta_5      4    7   15   26  166   45   35   22    7      0.50
theta_6      6   17   23   40  175   36   25    9    8      0.52

total       26   80  132  210 1017  219  180   93   42      0.51
```

These moves are plotted in Figure 9.

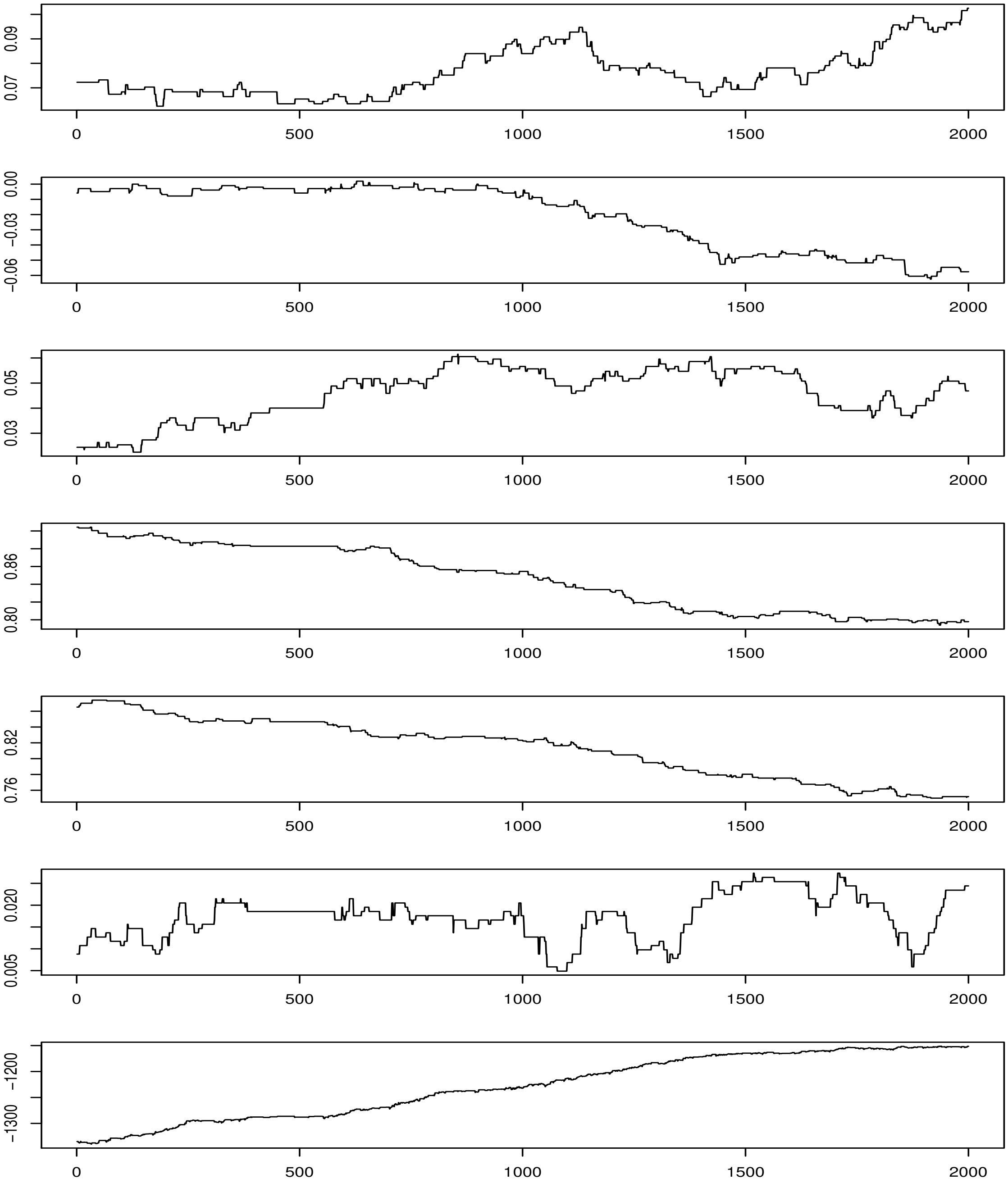

Figure 9. MCMC Chain from Parameter File sv.parm009.in0. The panels are draws of the elements of $\theta = (a_0, a_1, b_0, b_1, s, r)$ in order from top to bottom; the last panel is proportional to the log posterior density evaluated at θ. Every move is plotted. $R = 2000$.

We repeat, copying `sv.ctrl.parmfile.end` to `parm.010.in0` and `sv.implied_map.new` to `sv.implied_map.dat` four times and try four simultaneous runs with `(sci_sclfac, sci_incfac)` = (1,1), (1,2), (2,2), and (4,2).

The results are much the same: the best run has `(sci_sclfac, sci_incfac)` = (4,2) as before. The overall rejection rate is 0.285 and we are still climbing.

We repeat, copying `sv.ctrl.parmfile.end` to `parm.011.in0` six times and try six simultaneous runs with `(sci_sclfac, sci_incfac)` = (1,1), (1,2), (2,2), (2,4), (4,2), (4,4). We will not copy `sv.implied_map.new` to `sv.implied_map.dat`, because, due to the hill climbing, it is mostly filled with excess baggage.

All runs appear to have found the mode. Of them, four merit consideration: (1,2), (2,2), (4,2), and (4,4) with overall rejection rates of 0.25, 0.44, 0.54, and 0.70 respectively. The other runs either have too high a rejection rate or show a tendency to stick at one value of θ for long periods. The runs (2,2), (4,2), and (4,4) stick some also with (2,2) sticking the worst. This is probably just chance; given a long enough run sticking would probably be worse for (4,2) and (4,4). Being risk averse, we will select the file `parm.011.new` from the (1,2) run as our final tuning of the chain for the scientific model. We now move to a parallel machine.

9.3 Running on a Parallel Machine

The parallel version of GSM, which is `gsm_mpi`, is similar to the serial version, which is `gsm`, but with some quirks caused by restrictions imposed by the LAM implementation of MPI for which the code was written. These are that path names must be absolute, that command line parameters should not be used, and that subnodes cannot print anything.

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 makefile `makefile.mpi` included with the distribution. It assumes that the build occurs in the same directory in which data, parmfiles, etc. are found.

The command line requirement is met by always using the file `control.dat` rather than entering a file name on the command line. Also, for the parallel version, only the first line of `control.dat` is read and processed.

The no print requirement is handled by always coding `print` = 0 on the SNP parmfile. If this is not done, at best an unintelligible mess will be printed to standard output, at worst the program will crash.

For our example, here is `pathname.h` which was generated automatically by the makefile:

```
#define PATHNAME "/home/arg/r/gsm_guide/svfx_mpi"
```

This is `control.dat`:

```
sv.parm.012.in0  sv
```

and here follows `sv.parm.012.in0`, which was obtained by copying `sv.parmfile.fit` from `run011` with (`sci_sclfac, sci_incfac`) = (1,2) and editing the relative file names `dmark.dat` and `11114000.fit` to absolute file names. This last step is not necessary. Relative file names in the parmfile will work.

```
PARMFILE HISTORY (optional)
#
# This parmfile was written by GSM Version 1.5 using the following line from
# control.dat, which was read as char*, char*
# ---------------------------------------------------------------------------
#     sv.parm.011.in0                  sv
# ---------------------------------------------------------------------------
#
ESTIMATION DESCRIPTION (required)
        svfx   Project name, pname, char*
         1.5   GSM version, defines format of this file, gsmver, float
           1   Write detailed output if print=1, int
           0   Prior draws in sci_mod chain if sci_draw_from_prior=1, int
           1   Run sci_mod chain if run_sci_chain=1, int
           0   Prior draws in stat_mod chain if stat_draw_from_prior=1, int
           0   Run stat_mod chain (i.e. assess_chain) if run_stat_chain=1, int
         1.0   Value of kappa for assess prior, kappa, float
DATA DESCRIPTION (required)
           1   Dimension of the data, M, int
         834   Number of observations, n, int
/home/arg/r/gsm_guide/svfx_mpi/dmark.dat
4              Read these white space separated fields, fields, intvec
STAT_MOD DESCRIPTION (required)
           9   Number of parameters, len_stat_parm, int
           2   Number of functionals, len_stat_func, int
STAT_MOD PARMFILE (required) (constructor sees as vector<string> pfvec, alvec)
/home/arg/r/gsm_guide/svfx_mpi/11114000.fit
#begin additional lines
         100   Number of observations in simulated data, lsim (=N), int
          10   Initial simulations to eliminate transients, spin (=N0), int
#end additional lines
STAT_MOD MCMC DESCRIPTION (required) (describes assess chains)
      740726   Seed for stat_mod MCMC simulations, stat_seed, int
           2   Number stat_mod MCMC simulations per file, len_stat_chain, int
           0   Number of extra MCMC simulation files, num_stat_files, int
         1.0   Rescale proposal scaling by this value, stat_sclfac, float
STAT_MOD PARAMETER START VALUES (required)
```

```
   9.43702329096275487e-03    1    a0[1]   1
  -4.47167172729224585e-02    1    a0[2]   2
   4.22254530258357394e-03    1    a0[3]   3
   1.28819308482844674e-01    1    a0[4]   4
   2.93581996731074046e-02    1    b0[1]
   3.44346668251972353e-04    1    B(1,1)
   3.19200429080221559e-01    1    R0[1]
   6.09862061919064113e-01    1    P(1,1)  s
   8.03802290575046374e-01    1    Q(1,1)  s
STAT_MOD PROPOSAL SCALING (required)
   1.95637499999999978e-03      a0[1]   1
   1.81350000000000001e-03      a0[2]   2
   1.75031249999999992e-03      a0[3]   3
   1.71562500000000014e-03      a0[4]   4
   2.00000000000000037e-04      b0[1]
   3.32062499999999979e-03      B(1,1)
   3.75000000000000008e-04      R0[1]
   6.90825000000000033e-03      P(1,1)  s
   1.06425000000000009e-03      Q(1,1)  s
STAT_MOD PROPOSAL GROUPING (optional) (frequencies are relative)
 0.1    1
   1  1.0
 0.1    2
   2  1.0
 0.1    3
   3  1.0
 0.1    4
   4  1.0
 0.1    5
   5  1.0
 0.1    6
   6  1.0
 0.1    7
   7  1.0
 0.2      8      9
   8    1.0  -0.66
   9  -0.66    1.0
SCI_MOD DESCRIPTION (required)
           6   Number of parameters, len_sci_parm, int
           8   Number of functionals, len_sci_func, int
SCI_MOD PARMFILE (required) (constructor sees as vector<string> pfvec, alvec)
__none__       File name, code __none__ if none, sci_parmfile, string
#begin additional lines
       50000   Number of observations in simulated data, lsim (=N), int
        1000   Initial simulations to eliminate transients, spin, (=N0) int
#end additional lines
SCI_MOD MCMC DESCRIPTION (required)
      740726   Seed for sci_mod MCMC simulations, sci_seed, int
        1000   Number sci_mod MCMC simulations per file, len_sci_chain, int
           9   Number of extra MCMC simulation files, num_sci_files, int
           0   Use analytic expression to compute mle if analytic_mle=1, int
         100   Length of sub chain to compute mle, len_sub_chain, int
           0   Number of extra sub chains, num_sub_chains, int
           0   Number of quasi-Newton iterates, num_polish_iter, int
    1.00e-10   Tolerance for quasi-Newton, polish_toler, float
         1.0   Rescale proposal scaling by this value, sci_sclfac, float
         1.0   Rescale parameter increments by this value, sci_incfac, float
SCI_MOD PARAMETER START VALUES (required)
   1.05468750000000000e-01    1   a0
  -7.81250000000000000e-03    1   a1
   3.32031250000000000e-01    1   b0
   9.02343750000000000e-01    1   b1
   2.65625000000000000e-01    1   s
   1.95312500000000000e-02    1   r
SCI_MOD PROPOSAL SCALING (required)
   7.81250000000000000e-03    a0
   7.81250000000000000e-03    a1
   7.81250000000000000e-03    b0
   7.81250000000000000e-03    b1
   7.81250000000000000e-03    s
   7.81250000000000000e-03    r
SCI_PARAMETER INCREMENTS (required) (must be (fractional) powers of two)
   3.90625000000000000e-03    a0
   3.90625000000000000e-03    a1
   3.90625000000000000e-03    b0
   3.90625000000000000e-03    b1
   3.90625000000000000e-03    s
   3.90625000000000000e-03    r
```

Running on a parallel machine requires initiation of MPI prior to execution. This is handled by a shell script `emm_mpi.lam_7.0.sh` included with the distribution:

```
#! /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 gsm_mpi.err  && mv -f gsm_mpi.err  gsm_mpi.err.bak
test -f gsm_mpi.out  && mv -f gsm_mpi.out  gsm_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 >gsm_mpi.out 2>&1 && \
  mpirun -v -O -D -s h N N \
  ${PWD}/gsm_mpi >>gsm_mpi.out 2>gsm_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 Version 2.1 of OpenMPI.

The results of a run are a set of files similar to those for the serial version. The main omission is that files for tuning the subchain are not written. Instead of files being named like `sv.sci_sub_parm.001.dat` they are named like `sv.sci_sub_parm.015.001`. This would be `ifile`=001 produced by processor 15. (There are 16 processors numbered 0 to 15 because each of the eight nodes has two processors.) Files from one processor can be meaningfully concatenated. Files from different processors start at the same value of θ (which should be the mode of the posterior to prevent initial transients) but use a different seed. The files `sv.implied_map.new` and `sv.assess_sigma.new` are jointly produced by all processors. The implied maps of each processor are passed to the others as the run progresses so that a point visited by any processor is made available to the others. If two processors visit the same point, that point with the larger mle evaluated at the simulation is the one that is retained in the map.

Our chains look good. Here are the rejection rates over all processors

```
theta_1    0.18
theta_2    0.19
theta_3    0.18
theta_4    0.22
theta_5    0.21
theta_6    0.22

total      0.20
```

Figure 10 shows the moves of the elements of θ for the first processor; the others look about the same.

Putting θ on a grid serves two purposes: (1) It reduces the accuracy to which the subchain must compute the mle because points are separated. (2) It speeds computations because a previous point on the chain is likely to be repeated. Our risk aversion hurt us with respect to (2) because we only had about a 5% reuse rate. In retrospect, we probably should have been more aggressive and set `sci_incfac` to a larger value.

Figure 11 shows the posterior densities for the elements of θ over all processors.

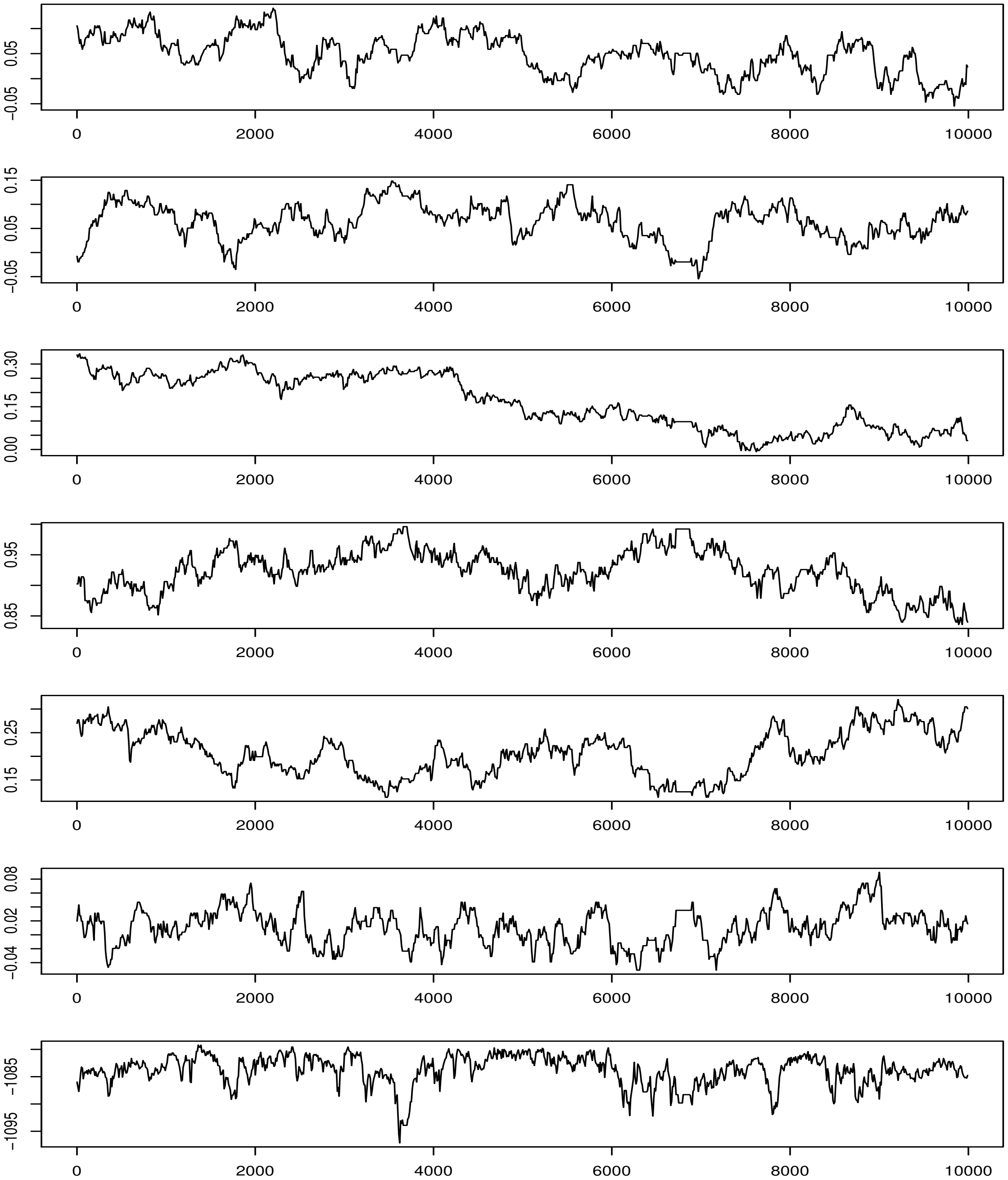

Figure 10. MCMC Chain from Processor 1 using Parameter File sv.parm012.in0. The panels are draws of the elements of $\theta = (a_0,\, a_1,\, b_0,\, b_1,\, \mathrm{s},\, r)$ in order from top to bottom; the last panel is proportional to the log posterior density evaluated at θ. Every tenth move is plotted. $R = 10,000$.

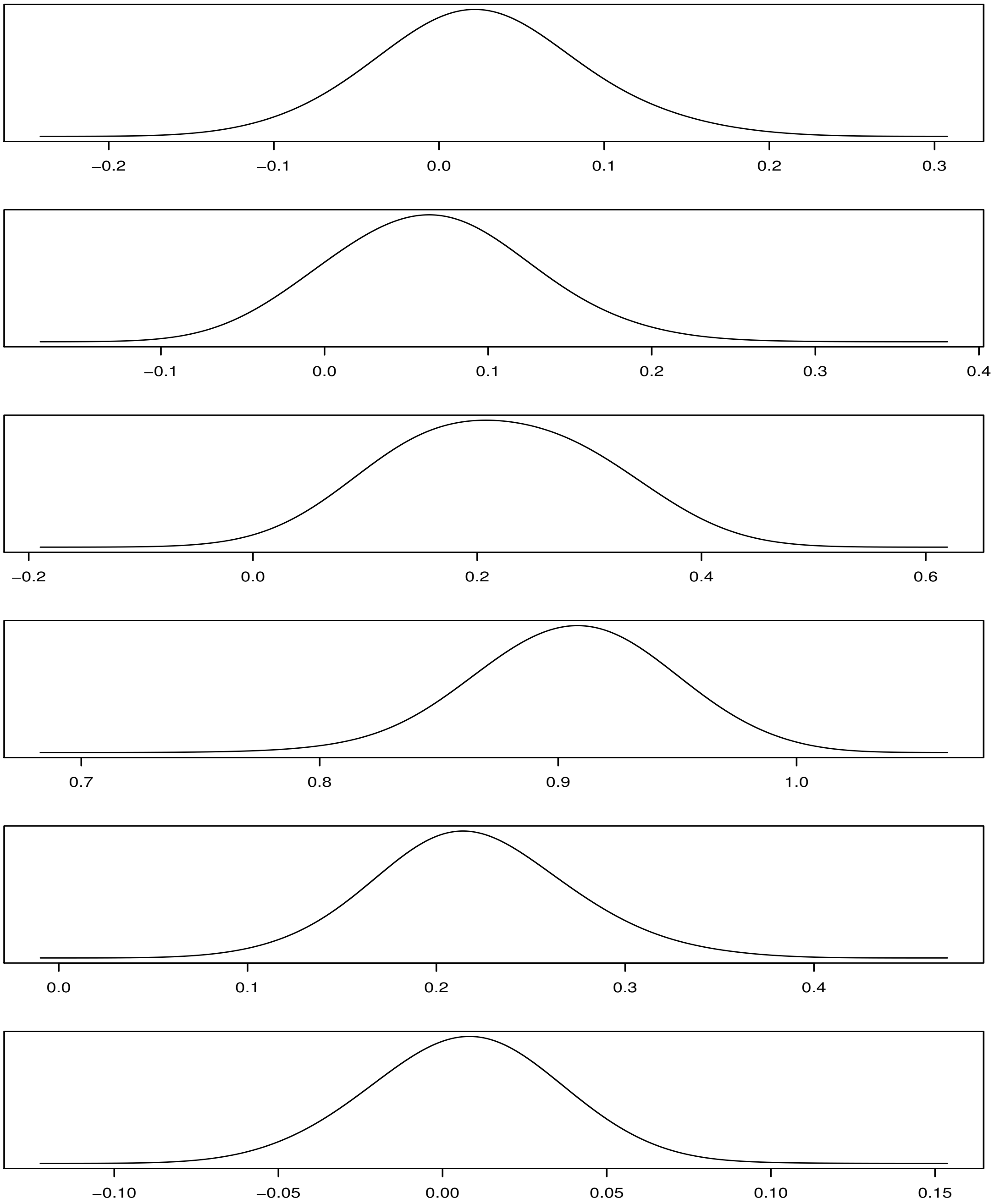

Figure 11. Posterior Densities from all Processors using Parameter File sv.parm012.in0. The panels are posterior densities for $\theta = (a_0,\, b_0,\, b_1,\, c_1,\, d_1,\, \text{s},\, \text{r})$ in order from top to bottom. Every twenty-fifth move is used. $R = 150,000$.

9.4 Model Assessment

Model assessment is straightforward and fast. The serial version of gsm is adequate although the parallel version can perform model assessment. We copy `sv.parm.012.in0` to `sv.parm.013.in0` and modify the blocks that control the model assessment chain as follows:

```
ESTIMATION DESCRIPTION (required)
        svfx   Project name, pname, char*
         1.5   GSM version, defines format of this file, gsmver, float
           1   Write detailed output if print=1, int
           0   Prior draws in sci_mod chain if sci_draw_from_prior=1, int
           0   Run sci_mod chain if run_sci_chain=1, int
           0   Prior draws in stat_mod chain if stat_draw_from_prior=1, int
           1   Run stat_mod chain (i.e. assess_chain) if run_stat_chain=1, int
         1.0   Value of kappa for assess prior, kappa, float

STAT_MOD MCMC DESCRIPTION (required) (describes assess chains)
      740726   Seed for stat_mod MCMC simulations, stat_seed, int
       10000   Number stat_mod MCMC simulations per file, len_stat_chain, int
          15   Number of MCMC simulation files, num_stat_files, int
         4.0   Rescale proposal scaling by this value, stat_sclfac, float
```

It is also necessary to retune the statistical model chain. The rescaling `stat_sclfac` $= 4$ above compensates for the fact that we are now running the chain on $n = 834$ observations rather than the $N = 50000$ for the subchain runs. But we also have to do a bit more tinkering using techniques similar to those described in Subsection 9.1. The parmfile `parmfile.alt` can be used to restart runs from the mode of chain that wrote it. Some additional guidance is provided by the fact that an unfettered chain should give roughly the same results as for the mle in `11114000.fit`. We retuned with `kappa` set to 10. The retuning is shown in the following two blocks.

```
STAT_MOD PROPOSAL SCALING (required)
   3.90000000000000000e-03      a0[1]   1
   3.60000000000000000e-03      a0[2]   2
   3.50000000000000000e-03      a0[3]   3
   3.40000000000000000e-03      a0[4]   4
   3.20000000000000000e-03      b0[1]
   6.64000000000000000e-03      B(1,1)
   3.00000000000000000e-03      R0[1]
   4.50000000000000000e-03      P(1,1)  s
   2.00000000000000000e-03      Q(1,1)  s

STAT_MOD PROPOSAL GROUPING (optional) (frequencies are relative)
 0.2           1            5
   1   1.0000000   -0.7211482
   5  -0.7211482    1.0000000
 0.1    2
   2  1.0
 0.1    3
   3  1.0
```

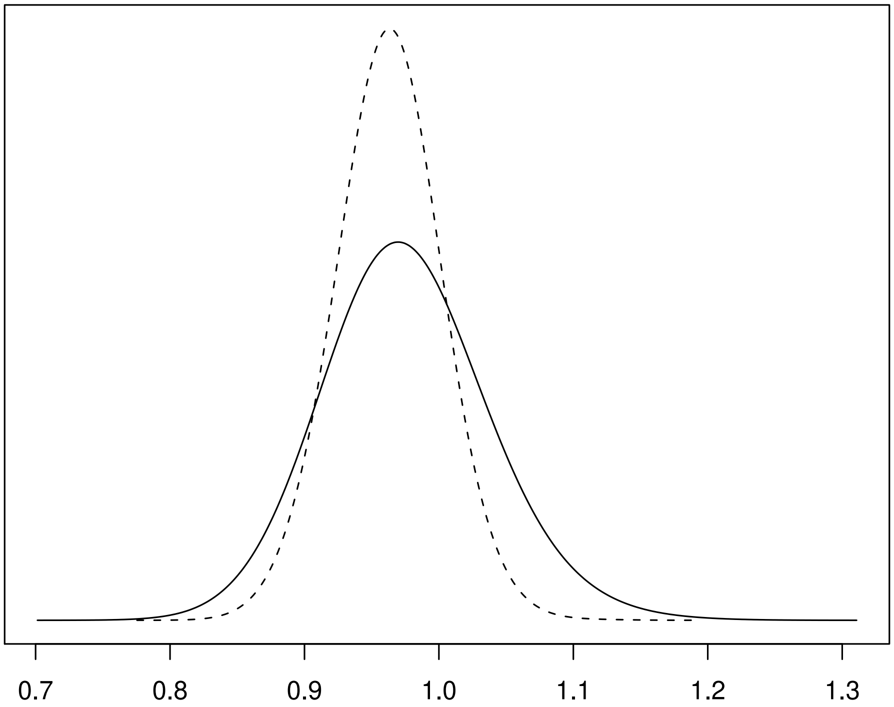

Figure 12. Persistence. The solid line is a plot of the posterior density of $\psi = \eta_8^2 + \eta_9^2$, which is a measure of volatility persistence when the stochastic volatility scientific model is imposed on the SNP statistical model. The dotted line is the posterior density of ψ when the prior π_κ with $\kappa = 1$ is imposed on the statistical model.

```
0.1    4
  4  1.0
0.1    6
  6  1.0
0.3          7          8          9
  7   1.0000000   0.7418488  -0.9137840
  8   0.7418488   1.0000000  -0.8653144
  9  -0.9137840  -0.8653144   1.0000000
```

When assessing results we will discard `sv.as_st_parm.000.dat` to dissipate transients because we are not starting this chain at the posterior mode. This will leave us with a concatenated chain `sv.as_st_parm.000.dat` $+ \ldots +$ `sv.as_st_parm.015.dat` of length $R = 150000$. We also copy the files `sv.implied_map.new` and `sv.assess_sigma.new` from the parallel run to `sv.implied_map.dat` and `sv.assess_sigma.dat`.

First we will look at $\psi = \eta_8^2 + \eta_9^2$ which is a measure of volatility persistence. Figures 12

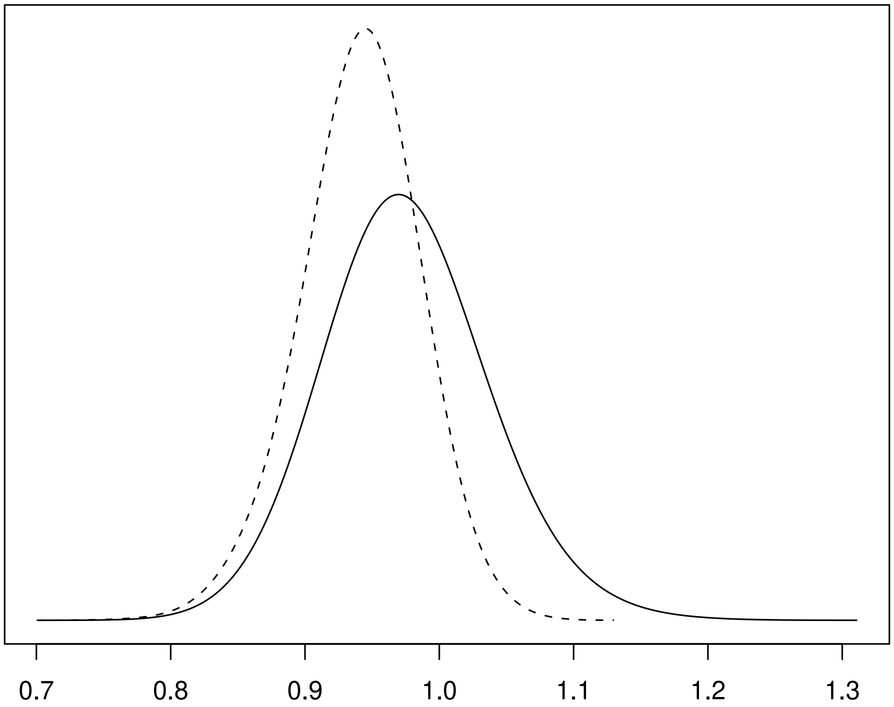

Figure 13. Persistence. The solid line is a plot of the posterior density of $\psi = \eta_8^2 + \eta_9^2$, which is a measure of volatility persistence when the stochastic volatility scientific model is imposed on the SNP statistical model. The dotted line is the posterior density of ψ when the prior π_κ with $\kappa = 100$ is imposed on the statistical model.

and 13 plot this measure for $\kappa = 1$ and 100, respectively. In Figure 12 we see the effect of imposing π_κ when it binds: the location of ψ aligns. In Figure 13 π_κ has negligible effect and the location shifts to the posterior under the SNP statistical model, which is considerably lower.

Figures 14 and 15 plot the posterior densities of the parameters of the SNP statistical model under the priors π_κ for $\kappa = 1$ and 100, respectively. The main effect of imposing the stochastic volatility scientific model on the SNP statistical model is to force symmetry on the SNP conditional density by making η_1 and η_3, which are the linear and cubic terms of the polynomial part of the SNP conditional density, nearly zero. The other effect is to thin the tails by making η_2, which is the quadratic term, negative and η_4, which is the quartic part, less positive. The shift in η_5, which is the location parameter of the conditional density, is a side effect. If the polynomial terms that control skewness, (η_1, η_3), shift, then η_5 must shift to compensate.

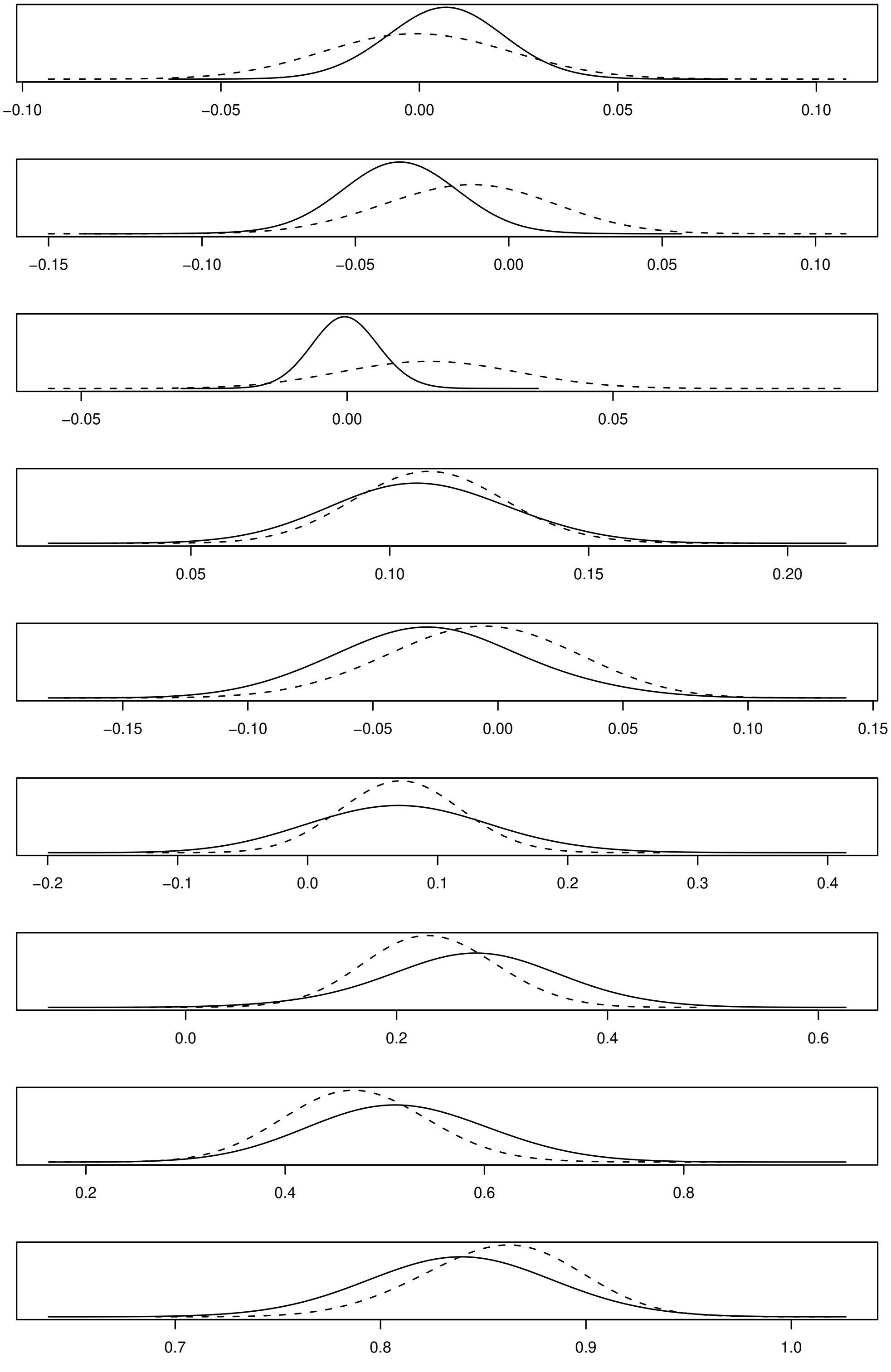

Figure 14. Parameters of the Statistical Model. The solid line is a plot of the posterior densities of the parameters η of the SNP statistical model when the stochastic volatility scientific model is imposed. The dotted lines are the posterior densities when the prior π_κ with $\kappa = 1$ is imposed.

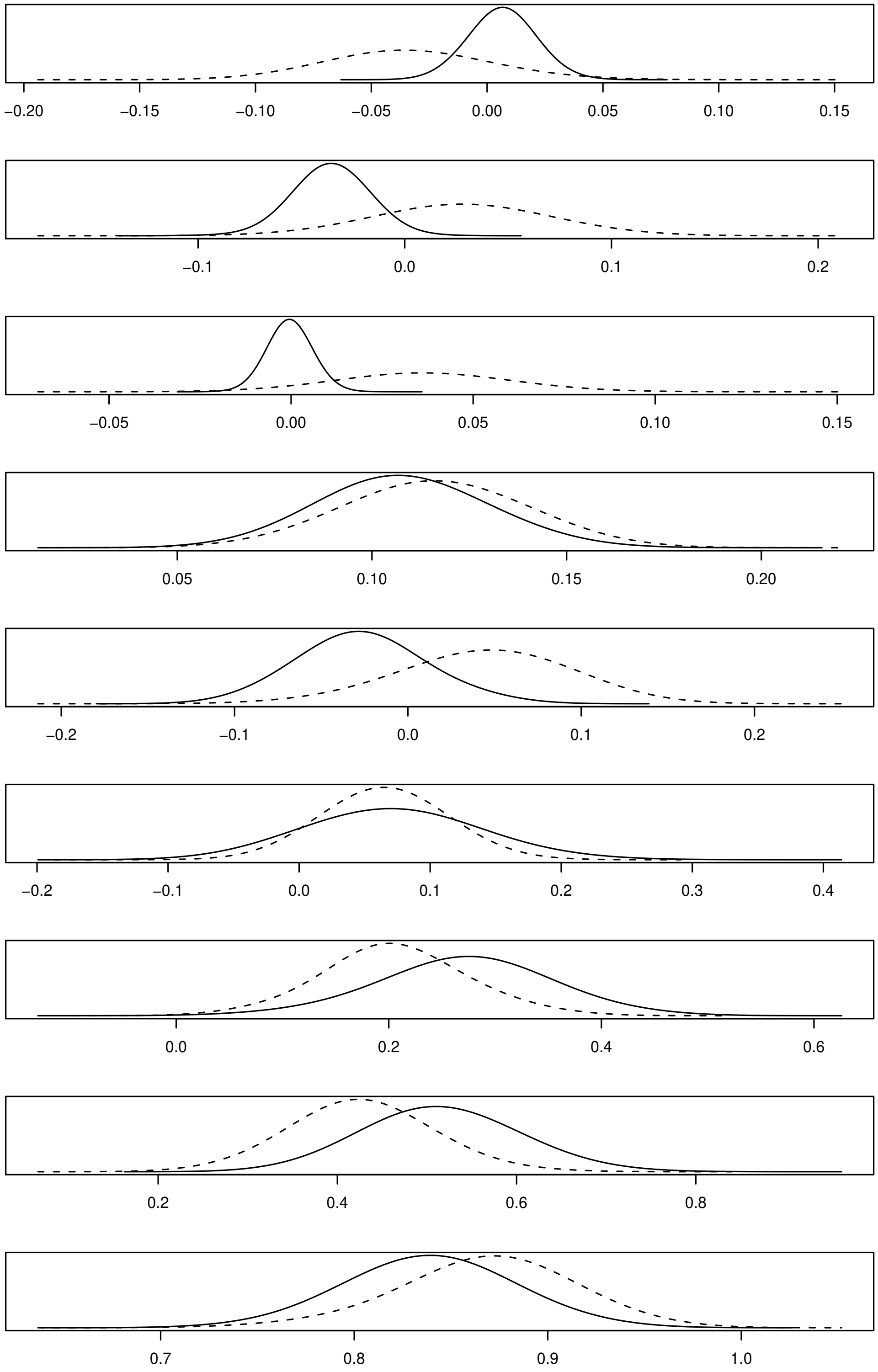

Figure 15. Parameters of the Statistical Model. The solid line is a plot of the posterior densities of the parameters η of the SNP statistical model when the stochastic volatility scientific model is imposed. The dotted lines are the posterior densities when the prior π_κ with $\kappa = 100$ is imposed.

9.5 Polishing the Subchain

Version 1.1 of GSM added the capability to add a BFGS quasi-Newton polish to the end of the subchain. Here we copy `sv.parm.012.in0`, from Subsection 9.3 above, to `sv.parm.014.in0` and make changes to the two blocks affected by the change from Version 1.0 to Version 1.1 so that they read as follows:

```
ESTIMATION DESCRIPTION (required)
        svfx   Project name, pname, char*
         1.5   GSM version, defines format of this file, gsmver, float
           1   Write detailed output if print=1, int
           0   Prior draws in sci_mod chain if sci_draw_from_prior=1, int
           1   Run sci_mod chain if run_sci_chain=1, int
           0   Prior draws in stat_mod chain if sci_draw_from_prior=1, int
           0   Run stat_mod chain (i.e. assess_chain) if run_stat_chain=1, int
         1.0   Value of kappa for assess prior, kappa, float

SCI_MOD MCMC DESCRIPTION (required)
      740726   Seed for sci_mod MCMC simulations, sci_seed, int
        1000   Number sci_mod MCMC simulations per file, len_sci_chain, int
           9   Number of extra MCMC simulation files, num_sci_files, int
   0   Use analytic expression to compute mle if analytic_mle=1, int
          49   Length of sub chain to compute mle, len_sub_chain, int
           0   Number of extra sub chains, num_sub_chains, int
          11   Number of quasi-Newton iterates, num_polish_iter, int
    1.00e-05   Tolerance for quasi-Newton, polish_toler, float
         1.0   Rescale proposal scaling by this value, sci_sclfac, float
         1.0   Rescale parameter increments by this value, sci_incfac, float
```

Reducing the subchain from 100 to 49 and adding a BFGS polish reduced run time by about 1/3. Recall that the BFGS polish uses the subchain to get its starting values so there is a limit to how small one can make it. With this example, `len_sub_chain` $= 49$ seems to be about as small as one can go without causing the θ chain to stick excessively. Running the parallel version of the code with this choice, all results are effectively the same as those shown in Subsection 9.3.

Runs with `len_sub_chain` < 49 allow b_1 to drift to one increment from 1.0 and then stick. This seems to be because the accuracy of the computation of

$$\hat{\eta} = \underset{\eta}{\operatorname{argmax}} \sum_{t=1}^{N} \log f(\hat{y}_t | \hat{x}_{t-1}, \eta_i)$$

by the combined subchain and polish deteriorates as b_1 moves toward 1.0 in a perverse way that causes

$$\sum_{t=1}^{n} \log f(\tilde{y}_t | \tilde{x}_{t-1}, \hat{\eta})$$

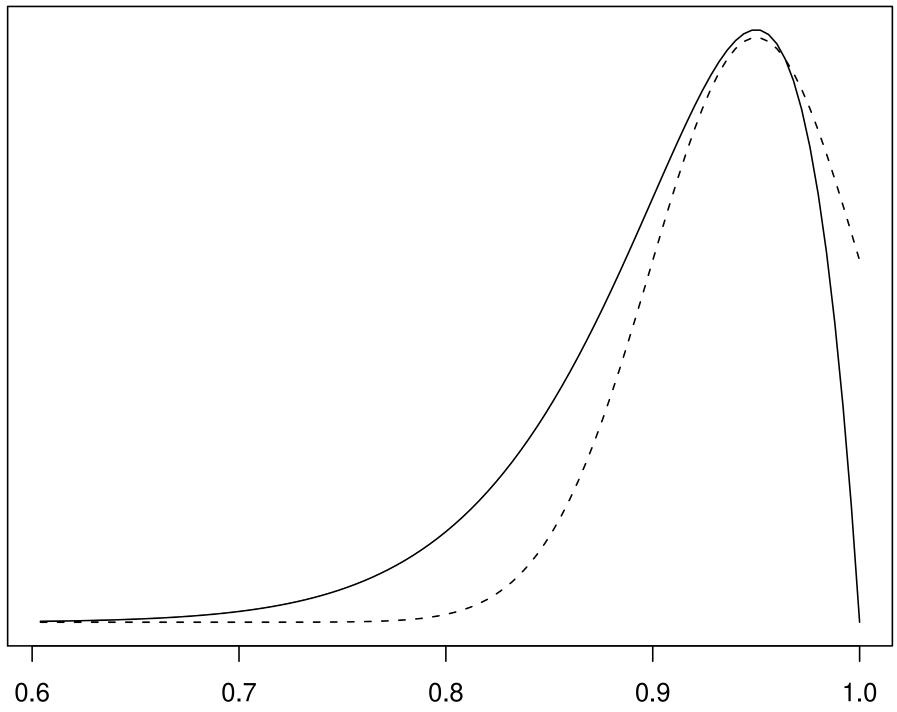

Figure 16. Normal vs Beta Prior for b_1. The solid line is a plot of the beta density with with $\alpha = 20$ and a mode of $m = 0.95$, which implies $\beta = (\alpha - 1 - ma + 2m)/m$. The dashed line is the normal density with $\mu = m$ and $\sigma = 0.1/1.96$.

to be over estimated. This, in turn, causes the chain to stick at the over estimated value. The feature that was added in Version 1.1 that restarts the subchain when this happens gets it unstuck eventually, but it appears that the root cause of the problem is the normal prior on b_1. Figure 16 plots the (truncated) normal prior that we have used thus far and an alternative beta prior that downweights points near 1.0 but is a reasonable substitute for the normal prior over the rest of the support.

We will substitute the beta prior and rerun. In the distributed code, the beta or normal prior can be toggled with the compiler directive `USE_BETA_PRIOR` in `gsmusr.cpp`. This change helped a little, but not enough to warrant a setting lower than `len_sub_chain` = 49. The prior would have to be more aggressive to achieve further reduction.

Rather than polishing once at the end of the subchain, one might try running the subchain, polishing, running, polishing, etc. To do this, e.g., three times, set `num_sub_chains` to 3. Because `len_sub_chain` is already small, one probably would not want to reduce it further.

10 References

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

Andersen, Torben G. (1994), "Stochastic Autoregressive Volatility: A Framework for Volatility Modeling," *Mathematical Finance* 4, 75-102.

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

Clark, P. K. (1973), "A Subordinated Stochastic Process Model with Finite Variance for Speculative Prices," *Econometrica* 41, 135–56.

Durham, Garland (2003), "Monte Carlo Methods for Estimating, Smoothing, and Filtering One- and Two-Factor Stochastic Volatility Models," Manuscript, Leeds School of Business, University of Colorado, Boulder, Colorado 80309-0419, USA.

Gallant, A. Ronald (1987), "Identification and Consistency in Seminonparametric Regression," in Bewley, Truman F., ed. (1987), *Advances in Econometrics Fifth World Congress, Volume 1*, Cambridge University Press, New York, 145–170.

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, and Robert E. McCulloch (2009), "On the Determination of General Scientific Models with Application to Asset Pricing," *Journal of the American Statistical Association*, forthcoming.

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

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 (2004), *SNP User's Guide*, Manuscript, Fuqua School of Business, Duke University, Durham NC 27708-0120, USA.

Ghysels, E., A. Harvey, and E. Renault (1995), "Stochastic Volatility," in G. S. Maddala, ed., *Handbook of Statistics, Vol. 14, Statistical Methods in Finance,* North Holland, Amsterdam.

Leon, S., Tsiatis, A. A., and Davidian, M. (2003) "Semiparametric Estimation of Treatment Effect in a Pretest-Posttest Study," *Biometrics* 59, 1046-1055.

Mallet, A., F. Mentré, J-L. Steimer, and F. Lokiec (1988), "Nonparametric Maximum Likelihood Estimation for Population Pharmacokinetics, with Application to Cyclosporine," *Journal of Pharmacokinetics and Biopharmaceutics* 16, 311–327.

Newton, Michael A. and Adrian E. Raftery (1994), "Approximate Bayesian Inference with the Weighted Likelihood Bootstrap." *Journal of the Royal Statistical Society (B)* 56, 3-48.

Olsen, L. F., and W. M. Schaffer (1990), "Chaos Versus Noisy Periodicity: Alternative Hypotheses for Childhood Epidemics," *Science* 249, 499–504.

Poirier, Dale J., (1988) "Frequentist and Subjectivist Perspectives on the Problems of Model Building in Economics," *Journal of Economic Perspectives* 2, 121–144.

Shephard, N. (2004), *Stochastic Volatility: Selected Readings* Oxford University Press.

Song, X., Davidian, M. , and Tsiatis, A. A. (2002), “A Semiparametric Likelihood Approach to Joint Modeling of Longitudinal and Time-to-Event Data,” *Biometrics* 58, 742-753.

Yu, Jun, (2005), “On Leverage in a Stochastic Volatility Model,” *Journal of Econometrics* 127, 165–178.