Online Detection of Markov Switching Models

Winston Chong
winston@utstat.toronto.edu

Nikita Reymer
reymer@utstat.toronto.edu

April 30, 2012

1 Introduction

We are interested in modeling time series using Markov switching models, which are commonly applied in financial econometrics to model returns with non-stationary data. We deviate slightly from our original proposal of Bayesian change point detection, but still incorporate the detection of structural breaks using online methodology. We emphasize the usefulness of online detection, especially in real life applications, such as in process control, where data streams in continuously over time, and re-computation of parameters does not require re-estimation over the entire time series.

We consider a basic HMM-GARCH(1,1) model, which allows for two unobserved possible states: low and high volatility. We apply the methods used for probabilistic inference from Hidden Markov Models (HMMs) and Monte Carlo techniques to derive filtering densities for the hidden states. In comparison to regular maximum likelihood estimation, this eliminates the need to consider up to 2^T possible paths, where T is the length of the time series. The nonlinear form of the GARCH model prevents us from using conventional filtering methods (e.g. the Kalman filter), hence each update step will include a Monte Carlo draw (the 'particles'), which will be propagated forward through time. This process is known as Sequential Monte Carlo (SMC). We discuss standard SMC procedures such as Importance Sampling and Resampling (IS and ISR). Later on we investigate and apply the Auxiliary Particle Filter (APF) as an improvement to regular ISR. In addition to state filtering, we also face the problem of parameter estimation, which will require an additional filtering step. Specifically, we use the kernel smoothing

approach to approximate the posterior density of parameters to propose new values. Parameters to be estimated include model parameters from the GARCH model, and the Markov chain transition probabilities for the states.

We will test our algorithm on a data set generated by an HMM-GARCH(1,1) time series, with states determined by a Markov Chain. This will give a good sense of the performance of the algorithm without needing to be concerned about model misspecification. We base our report mainly on the paper by He and Maheu ('Real Time Detection of Structural Breaks in GARCH Models') and the paper by Liu and West ('Combined parameter and state estimation in simulation-based filtering'), which gives a more detailed overview on the use of the APF for parameter estimation.

The report will be organized as follows: Section 2 will provide an overview of the methodology, Section 3 will contain discussion over different variations of the algorithm that may help reduce the variance of the Monte Carlo estimates, and Section 4 will provide a summary of our conclusions. This will be followed by an appendix, which will include all relevant code and mathematical derivations.

2 Methods

2.1 GARCH

A GARCH (Generalized Auto Regressive Conditional Heteroskedastic) process is a common model used in time series analysis for analyzing stochastic volatility. Specifically, a GARCH(1,1) model is the following:

$$y_t = \sigma_t \epsilon_t \quad \text{with} \quad \epsilon_t \sim N(0,1) \tag{1}$$

$$\sigma_t^2 = w + a y_{t-1}^2 + b \sigma_{t-1}^2 \tag{2}$$

where $w, a, b > 0$ and $a + b < 1$. σ_t^2 is often referred to as the innovation at time t. We hope to use this model to analyze the asset returns data y_t, and in particular the evolution of an unobserved volatility process. The model is able to capture the

persistence of volatility in the returns since each new innovation of the GARCH process is highly dependent on the previous one. However, the GARCH model does a poor job at modeling abrupt changes in volatility. Various modifications have been suggested to correct for this, one modification that is very popular in the current literature is the incorporation of structural breaks within the GARCH process. Instead of assuming that the observed time series is governed by a single process, we allow it to be composed of multiple different ones. Our goal is to estimate the times at which the change from one process to another occurs as well as to conduct parameter estimation for each of the component processes. In general, we assume that each new observation at time t depends on a hidden and unobserved state s_t. We model the evolution of states with a discrete time Markov Chain. For our case, we only consider two states: state one which would represent low volatility and state two would represent high volatility. This can of course be generalized to more than two states, though the maximum number of states must be defined, a particular weakness of this model.

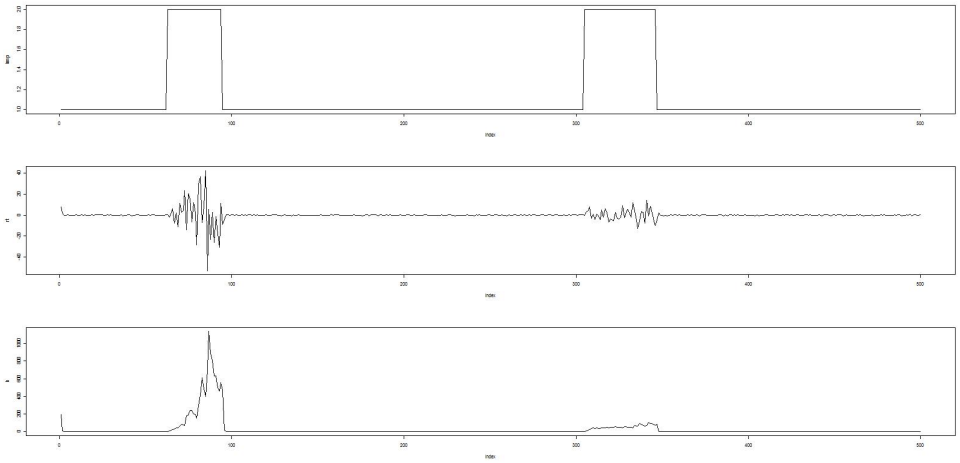

Figure 1: Simulated Data: State, Returns, Innovations

2.2 Sequential Monte Carlo

If we assume that model parameters are known and set our goal to estimate the posterior distribution $p(x_{t+1}|D_{t+1})$, given our HMM model assumption and by applying Bayes'

Theorem we can set up the following recursive relationship:

$$p(x_{t+1}|D_{t+1}) \propto p(y_{t+1}|x_{t+1})p(x_{t+1}|D_t) \tag{3}$$

$$p(x_{t+1}|D_t) = \int p(x_{t+1}|x_t)p(x_t|D_t)dx_t \tag{4}$$

Where $D_t = \{y_1, \ldots, y_t\}$ represents all observations up to time t, y_t are observations at time t, x_t are the hidden states, $p(y_{t+1}|x_{t+1})$ is the likelihood of observing y_t and $p(x_{t+1}|x_t)$ is the transition density for the states. A prediction is made for the next time step using all previous data, and when a new data point arrives, a correction step (or the update step) generates the new filtering density. For the case when the model is linear and Gaussian (with the appropriate conjugate priors), we use the usual Kalman filter. Otherwise, filtering becomes problematic, particularly when it comes to the prediction step, which requires the computation of an integral over all possible previous states. In some cases, this can be computed exactly. For example, in a finite state space model, this integral becomes a sum and is easily computed. However, even when exact inference is possible, it can become very impractical, with computational costs usually increasing linearly with time. The goal of SMC is not only to approximate these probabilities, but also keep computational costs constant at each update step.

To be more specific, we wish to calculate the posterior distribution of hidden states given previous data. Typically, we want to find $p(x_{1:t}|D_t)$. In our case, where we will only need $p(x_t|D_t)$, we can easily obtain it from the former via marginalization. A possibility would be to approximate this using MCMC. However, this is computationally expensive and would require a large number of iterations for each time step. It also turns out to be completely unnecessary. Instead, we can use an Importance Sampler (IS). With IS, our main concern is finding a proposal distribution from which we can easily generate samples. The requirement conditions for q are very mild and essentially only require q to have the same support as p. Ideally, $q(x_{1:t}|D_t)$ should be as close to $p(x_{1:t}|D_t)$ as possible. Supposing we do have such a distribution, $q(x_t|D_t)$, then we can rewrite our posterior of

interest as the following:

$$
\begin{aligned}
p(x_{1:t}|D_t) &= \frac{w(x_{1:t}, D_t)q(x_{1:t}|D_t)}{p(D_t)} && (5)\\
w(x_{1:t}, D_t) &= \frac{p(x_{1:t}, D_t)}{q(x_{1:t}|D_t)} && (6)\\
&= \frac{p(D_t)P(x_{1:t}|D_t)}{q(x_{1:t}|D_t)} && (7)\\
p(D_t) &= \int w(x_{1:t}, D_t)q(x_{1:t}|D_t)dx_{1:t} && (8)
\end{aligned}
$$

$q(x_{1:t}|D_t)$ is approximated by generating samples distributed accordingly, and $p(D_t)$ is done similarly (it is preferable that both are calculated using the same sample from $q(x_{1:t}|D_t)$ to reduce variability). Our importance sampler allows us to use samples generated from another distribution, with appropriate reweighting, to approximate our actual distribution of interest.

At each time step, we must generate a new set of samples x_t from $p(x_t|D_t)$. One major benefit of Monte Carlo sampling is that if we have a sample from a joint distribution, say from $p(x_{1:t}|D_t)$, we automatically get a sample from a marginal distribution $p(x_t|D_t)$ just by retaining the $\{x_t\}$. The proposal distribution for the joint parameter density can be factored as follows using Bayes' Theorem:

$$
\begin{aligned}
q(x_{1:t}|D_t) &\propto q(x_{1:t-1}|D_{t-1})q(x_t|y_t, x_{t-1}) && (9)\\
&\propto q(x_1|y_1)\prod_{k=2}^{t} q(x_k|y_k, x_{k-1}) && (10)
\end{aligned}
$$

Therefore at time t, to get a joint sample $x_{1:t}$, we can use our previous sample $\{x_{1:t-1}^{(i)}\}_{i=1}^{N}$, and update it with a new component $\{x_{t+1}^{(i)}\}_{i=1}^{N}$ generated from $q(x_t|y_t, x_{t-1})$. We also update weights in the following way:

$$
\begin{aligned}
w(x_{1:t}, D_t) &= \frac{p(x_{1:t}, D_t)}{q(x_{1:t}|D_t)} && (11)\\
&= \frac{p(x_{1:t-1}, D_{t-1})}{q(x_{1:t-1}|D_{t-1})}\frac{f(x_t|x_{t-1})p(y_t|x_t)}{q(x_t|y_t, x_{t-1})} && (12)\\
&= w(x_{1:t-1}, D_{t-1})\frac{f(x_t|x_{t-1})p(y_t|x_t)}{q(x_t|y_t, x_{t-1})} && (13)
\end{aligned}
$$

where f models the transition between latent variables, and p models our observations conditional on observing the hidden variables i.e. the likelihood. This is sequential importance sampling (SIS). Specific to our model, where we are interested in sampling from the posterior $p(x_{t+1}|y_{1:t+1})$, that is given the arrival of the new observation y_{t+1} at time $t+1$ we are interested in updating the latent variable, x_{t+1} which the observation came from. At this point in time, we assume we know the values of the parameters that are associated with the computation of the likelihood. Given a previous Monte Carlo sample, also known as the particle set at time t, $\{x_t^{(k)}\}_{k=1}^N$ and corresponding weights $\{w_t^{(k)}\}_{k=1}^N$ which was generated using Importance Sampling we can approximate the quantity of interest by:

$$p(x_{t+1}|D_{t+1}) \propto p(y_{t+1}|x_{t+1}) \sum_{k=1}^{N} w_t^{(k)} p(x_{t+1}|x_t^{(k)}) \tag{14}$$

To approximate this density with Monte Carlo techniques, we generate samples from $p(x_{t+1}|x_t^{(k)})$ using prior information about transition probabilities and update the associated weights of each sample using its current weight $w_t^{(k)}$ and the likelihood of observation y_{t+1} coming from state $x_{t+1} \sim p(x_{t+1}|x_t^{(k)})$. In this scenario, the new samples for x_{t+1} depend only on the prior distribution of x_t. In the context of the HMM model, given a set of transition probabilities, we need to predict which states all the particles will traverse to, and then evaluate how accurate our prediction was based on the observed likelihood $p(y_{t+1}|x_{t+1}^{(k)})$.

Again there are complications with a selection of the importance distribution, which we will ignore since SIS is not the main focus of the paper. The other primary concern is degeneracy. As the particles are propagated, weight becomes concentrated on only a few selected particles, while particles which are uninformative, still continue to be updated. One common solution is through resampling. Our Monte Carlo sample is a discrete approximation to our target density and each particle in the sample carries a weight. Thus we can just sample with replacement from this discrete distribution. By resampling the particles at each time step, we keep only meaningful ones. Those with large weights continue to be updated, while those with low weights are discarded. The intuition behind

resampling is that a particle with low weight at time t, will tend to still have low weight at time $t+1$ and will not be useful in making inferences. With the resampling, particles are reweighted uniformly. As well, from prior information, transition probabilities from one state to a different one are quite small, so the algorithm may generate many incorrect predictions by either concentrating all the weight on uninformative particles or by missing a change in state. This prediction process also makes the algorithm very sensitive to parameter values. For example, if the parameters associated with each state are fairly close to each other in value, states essentially look the same, thus the algorithm assigns small weights to particles that indicate a change in state.

Alternatively, we can incorporate other information which will put more mass on relevant particles, this should improve how our algorithm performs. This is the motivation for auxiliary particle filters (APF), developed by Pitt and Sheppard (1999).

2.3 Auxiliary Particle Filter

At this point we still assume we have some estimate of our parameters at time t. Instead of blindly guessing what the update state x_{t+1} will be, we select the most promising particles to update. The idea is to use some form of the likelihood $p(y_{t+1}|x_{t+1})$ in the particle selection process. We proceed as before, and set $\mu_{(t+1)}^{(k)}$ as an estimate of the new state x_{t+1} - for our case, the mode of our transition density $p(x_{t+1}|x_t^{(k)})$ - for each particle $k = 1, \ldots, N$. The quality of our estimate is evaluated by computing weight:

$$g_{t+1}^{(k)} \propto w_t^{(k)} p(y_{t+1}|\mu_{t+1}^{(k)}) \tag{15}$$

The set $\{g_{t+1}^{(k)}\}_{k=1}^{N}$ are called the auxiliary weights. Generally, a large $g_{t+1}^{(k)}$ serves as a good indicator as to whether the path the particle is about to undertake is promising and in accordance to the true underlying process. To perform particle selection, we first normalize the auxiliary weights and sample with replacement a set (of size K) of indices $\{j\}$ with from the set of integers $\{1, \ldots, N\}$ with probabilities proportional to $g_{t+1}^{(j)}$ where N is the size of the particle set at time t. The algorithm allows the particle

set size to change at every time point if necessary, but usually the particle set size is kept constant, with $K = N$. The generated set of indices $\{j\}$ represents the set of most likely paths the process will take given the arrival of the new datum y_{t+1}. Now we draw the new values $x_{t+1}^{(j)} \sim p(x_{t+1}|x_t^{(j)})$ based on our auxiliary indexes and compute the new associated weights:

$$w_{t+1} \propto \frac{p(y_{t+1}|x_{t+1}^{(j)})}{p(y_{t+1}|\mu_{t+1}^{(j)})} \tag{16}$$

Furthermore, we can proceed with the standard SMC resampling procedure and resample the particle set according to the new weights w_{t+1} and obtain an equally weighted final sample.

2.4 Parameter Filtering

In this section, we extend state filtering procedure to a general joint state and parameter filtering problem. First we need to extend our particle set of states to include parameter values. Thus at time t we will have a joint importance sample of the state and parameter values for the model $\{x_t^{(j)}, \theta_t^{(j)}\}_{j=1}^N$ with associated weights $\{w_t^{(j)}\}_{j=1}^N$. At this stage our goal is to approximate a joint posterior $p(x_{t+1}, \theta|D_t)$ instead of just a posterior for the state. We can decompose $p(x_t, \theta|y_{1:t})$ into the following three factors:

$$p(x_{t+1}, \theta|D_{t+1}) \propto p(y_{t+1}|x_{t+1}, \theta)p(x_{t+1}|\theta, D_t)p(\theta|D_t) \tag{17}$$

which is just the product of marginal likelihood given the state and parameters; prediction of hidden states given past data and parameter values; and the posterior density for the parameters given the data. Note that if we know the parameter values, the above equation simplifies just to the state filtering problem as discussed previously. Now the challenge lies in approximating the density for $p(\theta|D_t)$. We again follow the approach of Liu and West (2001) and implement a non-parametric, Kernel Smoothing method for approximation of $p(\theta|D_t)$. If we assume we have solved the problem until time t, that is at time t we have Monte Carlo samples $\theta_t^{(j)}$ and associated weights $\{w_t^{(j)}\}_{j=1}^N$ from the posterior distribution

of parameters. It is important to remember that θ is fixed and t indicates only the time step from which our samples come from. We approximate the parameter density by a smooth kernel density as follows:

$$p(\theta|D_t) \approx \sum_{j=1}^{N} w_t^{(j)} N(\theta|m_t^{(j)}, b^2 V_t) \tag{18}$$

where $N(\theta|m, V)$ is a multivariate normal distribution with mean vector m and covariance matrix V. Kernel smoothing approximates the posterior density for θ by a mixture of multivariate normal distributions with the weights coming from the particle weights. b^2 is the kernel shrinkage parameter and is chosen to be between 0 and 1, and usually decrease slowly as the particle set size increases so that all the parameter estimates are concentrated closer to the mean. V_t is the Monte Carlo sample variance and is computed by:

$$V_t = \sum_{i=1}^{N} w_t^{(i)} (\theta_t^{(i)} - \overline{\theta}_t)(\theta_t^{(i)} - \overline{\theta}_t)^T \tag{19}$$

and $\overline{\theta}_t$ is the Monte Carlo sample mean given by:

$$\overline{\theta}_t = \sum_{i=1}^{N} w_t^{(i)} \theta_t^{(i)} \tag{20}$$

If we take $m_t^{(j)}$ to be just $\theta_t^{(i)}$ then each Gaussian component is centered around the existing sample and the overall mixture will be an over-dispersed mixture of Gaussians with variance $(1 + b^2)V_t$. As time progresses, the variance grows larger, resulting in a very noisy and inconsistent parameter estimates. To resolve this issue of degeneracy, Liu and West (2001) use a kernel shrinkage method. They take $m_t^{(j)} = a\theta_t^{(i)} + (1-a)\overline{\theta}_t$ where $a = \sqrt{1 - b^2}$. The intuition behind this mean value is to make the centres of each of the Normal components to be closer to each other so that the resulting distribution will have thinner tails and the desired variance V_t. Maheu and He (2010) use a discount factor $\delta \in (0, 1)$ to control the shrinkage of the kernel means as $b^2 = 1 - [(3\delta - 1)/2\delta]^2$ and $a = \sqrt{1 - b^2}$. Now, conditional on the sample values drawn from the above mixture, we

can apply the regular APF filter and perform state inference.

2.5 A General Algorithm

Given a set of particles and weights $\{\theta_t^{(i)}, s_t^{(i)}, w_t^{(i)}\}_{i=1}^N \sim p(s_t, \theta, |D_t)$ (where N is the size of the particle set), the following outlines the algorithm for the APF:

1. For $i = 1, 2, \ldots, N$ let r_{t+1} be the mode of $p(s_{t+1}|s_t^i, \theta_t^{(i)})$
2. Compute the auxiliary weights $g_{t+1}^{(i)} \propto p(y_{t+1}|D_t, m_{r,t}^k) w_t^{(i)}$, with $m_{r,t}^k = a\theta_{r,t}^{(i)} + (1 - a)\bar{\theta}_{r,t}$ and $\bar{\theta}_{r,t} = \sum_{i=1}^N w_t^{(i)} \theta_{r,t}^{(i)}$ and draw a sample from $1, \ldots, N$ with the corresponding auxillary weights, call this sampled index k
3. Sample a new parameter vector from the k^{th} normal component of the kernel function, $\theta_{t+1}^{(k)} \sim N(m_{r,t}^k, h^2 V_t)$ where $V_t = \sum_{i=1}^N w_t^{(i)} (\theta_t^{(i)} - \bar{\theta}_t)(\theta_t^{(i)} - \bar{\theta}_t)'$ and $h^2 = 1 - (\frac{3\delta - 1}{2\delta})^2$
4. Sample a new state value s_{t+1}^k $p(s_{t+1}^k | s_t^k, \theta_{t+1}^{(k)})$
5. Evaluate the new weight, $w_{t+1}^{(k)} \propto \frac{p(y_{t+1}|x_{t+1}^{(k)}), \theta_{t+1}^{(k)}}{p(y_{t+1}|r_{t+1}^{(k)}, m_t^{(k)})}$
6. Repeat (2)-(5), possibly using stratified sampling, to generate a new particle set.

3 Simulation Results

We model the data using a GARCH(1,1), using two sets of parameters for the low volatility state and high volatility state. We reparameterize all the variables so that they can take on all real values. We initialize w_1 using a Gamma(1,0.5), a_1 with Beta(4,1), b_1 with Beta(4,1)*(1-a_1) (because of the constraint), w_2 using Gamma(1,0.5), a_2 with Beta(4,1) and b_2 with Beta(4,1)*(1-a_2). The first set will represent parameters under the low volatility regimen. We reparameterize w, a and b as follows:

$$w* = \log(w) \tag{21}$$

$$a* = \log\left(\frac{a}{1-a}\right) \tag{22}$$

$$b* \;=\; \log\left(\frac{b}{1-a-b}\right) \tag{23}$$

We initialize the logit transformed p_{11} and p_{22} (the probability of staying in the same state) using a N(8,1), i.e. the probability of not changing states will be very close to 1. This is a sensible initialization: otherwise, particles will tend to change states too frequently. We initialize all states to be state 1 (the low volatility state) and the first innovation to be the average of the unconditional variances of state 1. For our analysis, we will be running the algorithm for 150 time steps, varying the number of particles as needed, with each initially weighted equally.

We considered various initializations for the transition probabilities and the δ, which acts like a tuning parameter for the kernel density. As well, we tried different sampling schemes, such as stratified vs. non-stratified to reduce variance. In general, we find that stratified sampling produces much more stable estimates (i.e. tighter confidence intervals), hence for the rest of our analysis, we only consider outputs generated by this sampling method. We will elaborate further on the use of stratified sampling in a later section. To analyze the performance of the APF algorithm we track the evolution of the various quantities over time as the algorithm progresses:

3.1 Parameter Evolution

We are interesting in monitoring the evolution of the Monte Carlo estimates of the parameters as time proceeds. It seems plausible that as time progresses, particles should 'gain more information' due to the APF procedure that we apply, and that estimates should converge to their true value. We expect that estimates at least become less variable (i.e. confidence intervals narrow as time progresses). We also hope that the two parameter sets are distinguishable, without using additional prior information (e.g. both a_1 and a_2 have the same prior and are initialized identically). Like traditional importance sampling, the calculation of standard errors in some closed form method is not obvious. Instead, we resort to repeated runs. We limit ourselves to 100 repeated runs, each run consisting of 5000 particles over 150 time steps, with $\delta = 0.75$. We plot these Monte Carlo estimates

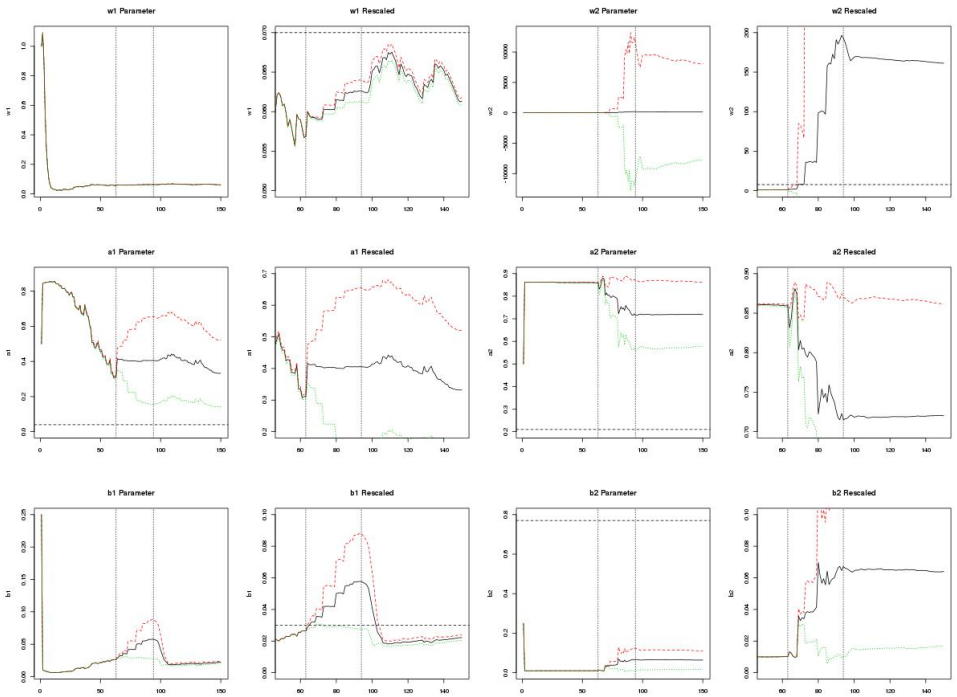

Figure 2: Monte Carlo estimates of parameters with a 95 percent CI. $\delta = 0.75$, dashed horizontal lines represent the true value, dotted verticle lines represent switch points. Due to the variability of w_2 parameter, CI bounds extend into the negative region, even though the parameter only takes positive values.

(averaged over all 100 runs) and their associated confidence intervals (Figure 2). We find that the parameters do in fact deviate, though some more than others (in particular w_1 vs w_2, where the latter value becomes highly variable). We also find that though parameter values were initialized to very low values, there was a 'correction' almost immediately. Parameter sets behave very oddly. In general, estimates seem to become more variable after the first change point, and stablize slightly after the reversion to the less volatile state. The stability of the w_1 parameter (the intercept) shouldn't be too surprising when the other parameters are capturing the volatility. The b_1 parameter becomes variable once the state changes, but recovers fairly quickly once the state reverts. a_1 however does not seem to stabilize after reverting to the more stable state. Again, this is not surprising, as a_1 is the coefficient for the response varible y_t, an observed value, while b_1 is the coefficient of the innovation, which is estimated (and tends to be a very stable estimate). State 2 parameters actually behave quite well. On average, these parameters only update under state 2, and almost immediately stop changing upon the switch to state 1.

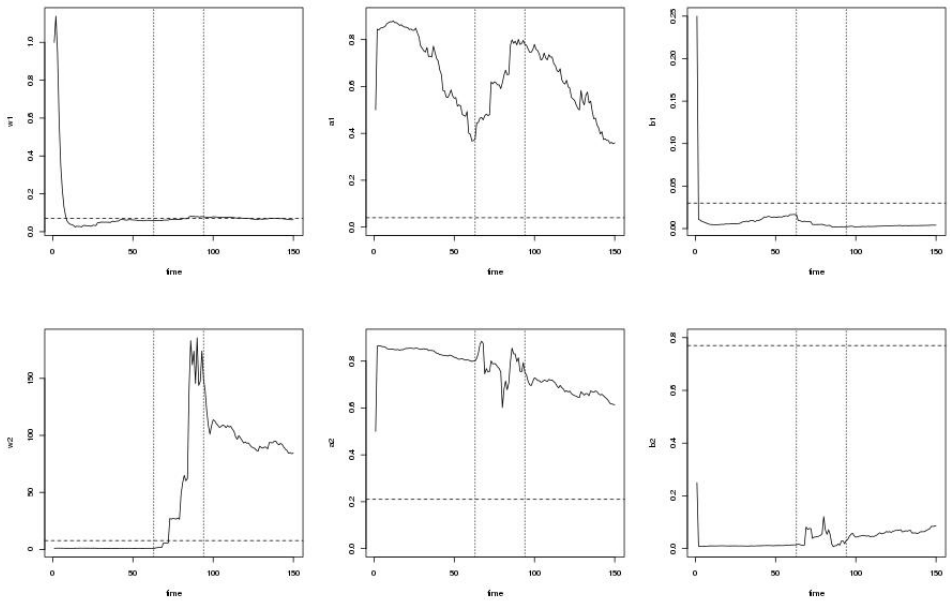

Figure 3: $\delta = 0.25$

For this reason, it is possible to suspect that state 1 parameters are more accurate than state 2 parameters, since the chain spends most time in state 1. This only seems slightly true. Overall, the confidence intervals tend to miss the true value of the parameter, and that this Kernel approximation seems slightly questionable. However, we must not also discount the fact that reparameterizations were needed due to the constraints involved. Our choice of δ was not a completely a precise one. We considered $\delta = 0.25, 0.50, 0.75$ and did five test runs for each setting and very roughly selected the δ which generated the most consistent parameter estimates. By the Kernel shrinkage method used to correct the Kernel variance, higher δ values will generate a smaller variance (hence more stable values), but the mean of the Kernel will be weighted more towards the individual particle (θ_t) instead of the Monte Carlo estimate ($\bar{\theta}_t$) (less stable). With $\delta = 0.75$, the stablizing effect of the reduced Kernel variance is more dominant. Maheu and He chose $\delta = 0.99$ due to his significantly larger parameter set, while we find this choice of δ too restrictive, preventing the paramters from exploring much of the space. We graph the parameter changes for $\delta = 0.25$ (Figure 3) and $\delta = 0.50$ (Figure 4)

In general, we find that often the confidence interval will miss the true parameter value. Sometimes it will actually miss it significantly even by factors of 10 or more.

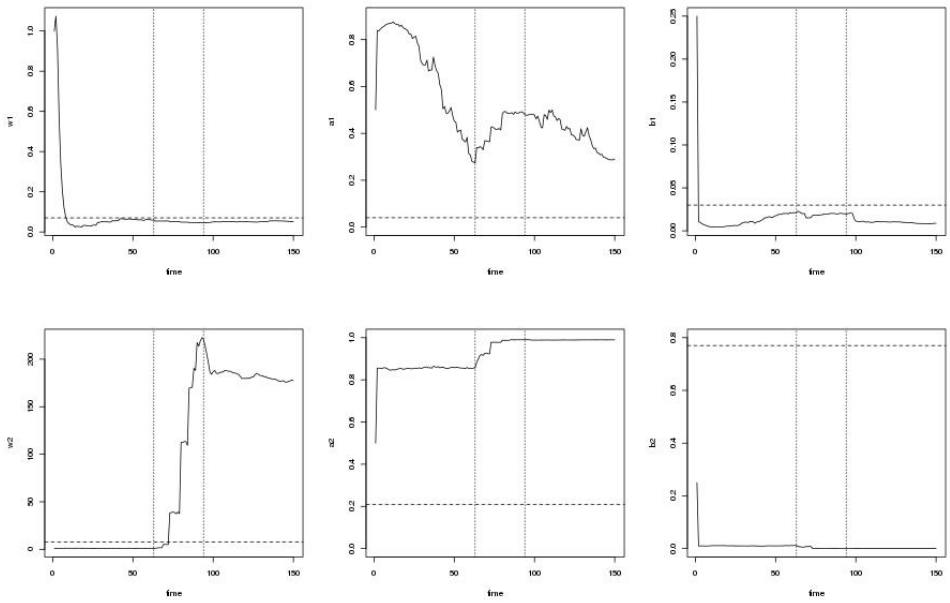

Figure 4: $\delta = 0.50$

Though we do find that state 2 parameter values will tend to have larger values than state 1 parameter values, and in particular, the model is able to capture the huge w_2 value that we chose for data generation. Again, we find for high values of delta, the parameters tend to be move very little over the course of the algorithm run. With lower values, we find that w_2 values will often overshoot significantly. Runs appear to be highly variable in parameter estimation given a fixed configuration but very consistent in detection of break points.

3.2 Particle Weights

With each particle is associated a weight value, which corrects for our samples being generated by another distribution. If the algorithm performs properly, more weight should be allocated towards those that carry a lot of 'information,' in our case, the state in which the particle is in. Optimally, we would like it that when the process is in a low volatility state, particles that represent state one carry a majority of the weight, or alternatively, the particles in state 2 should carry as little weight as possible. For our algorith, we are using two different sets of weights: auxilliary weights which we use to select which particles to update; and the second set of weights which is the actual importance weight.

	TRUE	Run 1	Run 2	Run 3	Run 4	Run 5	Average (100 runs)
w_1	0.07	0.050	0.043	0.059	0.031	0.072	0.061 (0.061,0.062)
a_1	0.04	0.062	0.236	0.040	0.466	0.013	0.333 (0.144,0.522)
b_1	0.03	0.001	0.076	0.012	0.021	0.011	0.022 (0.020,0.024)
w_2	7.75	147.304	116.711	145.432	151.547	233.960	161.396 (-7777,8100)
a_2	0.21	0.983	0.737	0.835	0.941	0.580	0.720 (0.579,0.862)
b_2	0.77	0.000	0.045	0.012	0.000	0.000	0.064 (0.017,0.111)

Table 1: Parameter estimates for five runs. 5000 particles, 150 time steps, $\delta = 0.75$

Figure 5 shows the evolution of both the auxilliary weights and importance weights in comparison to the true state the process is in. What is immediately noticeable is that both weight sets closely match up with those of the actual hidden states, though the weight continues to be high for a couple time steps after reverting to the low volatility state (this we will discuss later), and that generally, the algorithm performs correctly. It is actually quite critical that stratified sampling be used to select which particles to update (i.e. stratify sample from the auxilliary weights). When not used, the algorithm can completely fail - either inconsistently detecting state two by continually reverting states or being trapped in state two (and unable to detect state one).(Figure 5) Though sometimes the algorithm doesn't perform too poorly, for example, when the process is in state one, at least 50 percent of the weight will be in state one. The problem is mainly due to relevant particles not being propagated forward due to random chance.

3.3 Monitoring States

State prediction is a critical part of this analysis. Whereas many papers focus entirely on state prediction, assuming parameters are known, we have the additional complication of estimating these parameters and using them to predict states. We include two measurements of the predicted state, a weighted mean and a mode (selected by choosing the

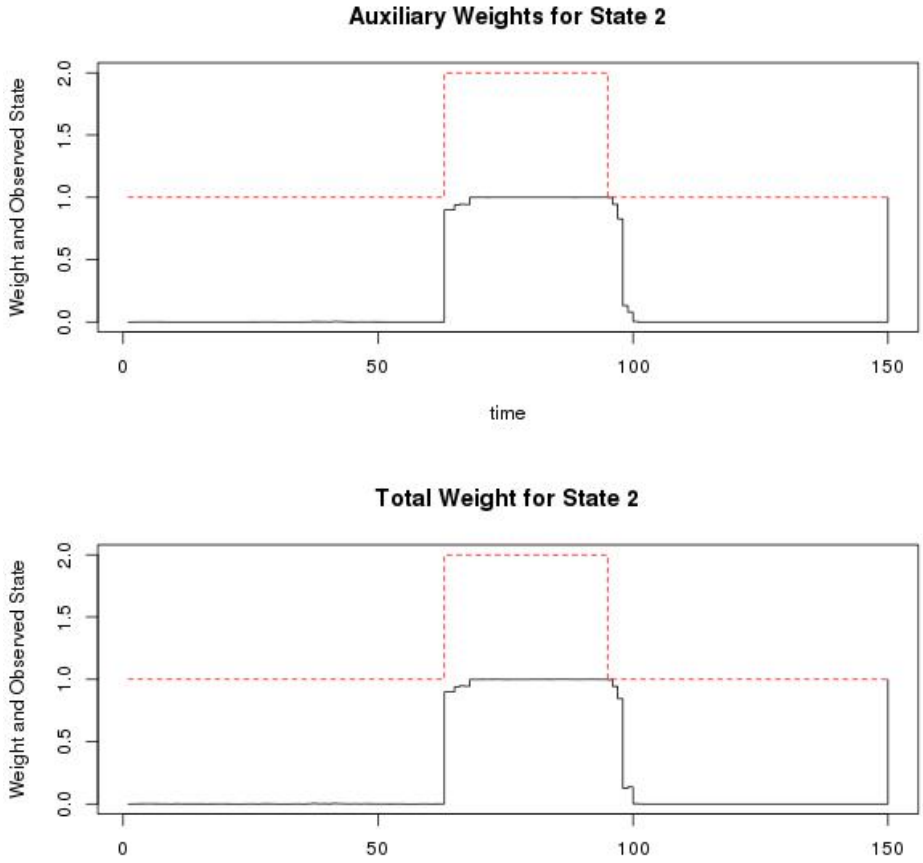

Figure 5: $\delta = 0.75$, stratified sampling

state which had most weight) (Figure 7). Again, it is clear that the algorithm performs well during both low volatility regimen, and high volatility regimen. We also find that the mean plot is essentially the same as the mode plot, i.e. either all weight is in state 1, or all weight is in state 2, where there is little indecision between the two states. We find that this is extremely sensitive to our prior for transition probabilities (Figure 8). A N(2,1) prior generates transition probabilities on average of around 0.88, which is still fairly high, but results in the chain wanting to move too frequently. We lose the power to predict the first regimen of state 1 (the chain appears to be unable to choose either state, based on the mean plot), and in some cases, when the algorithm detects state 2, it stays in state 2 even when the regimen has switched (which was not a problem for our previous setting) (Figure 9).

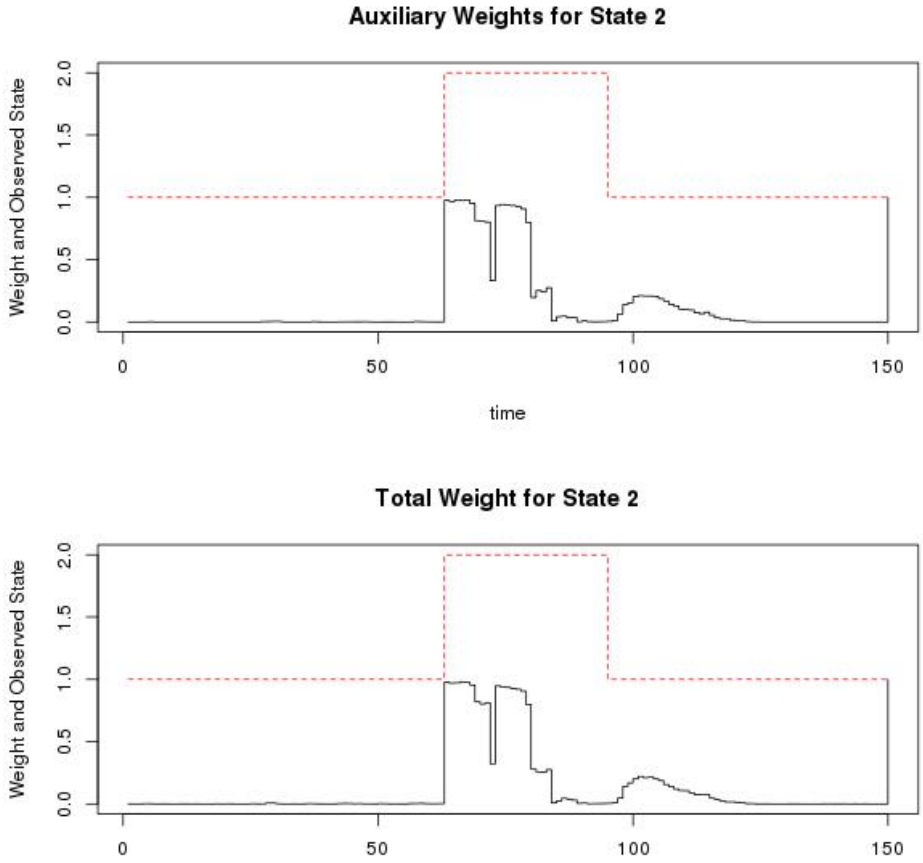

Figure 6: $\delta = 0.75$, simple random sampling

3.4 An Extended Run - 10,000 Particles over 400 Time Points

In this section we analyze the effect of increasing the observed length of the time series as well as increasing the particle set size. Intuitively, increasing the observation time allows particles to collect more information about the actual process and hence improve parameter estimation. Increasing the particle set is equivalent to increasing the Monte Carlo sample size and is necessary with the increased time series length since each particle represents a possible path our chain can undertake. Thus having more particles allows us to better explore the joint state and parameter space. Using a longer time series allows us to test the long term behavior of the algorithm, for example its ability to detect multiple change points. Note that during the run our process was in state 2 twice and had 4 change points in total. Our APF was able to detect all 4 change points with high precision over all the test runs. Note that we have some lag in detecting the transition

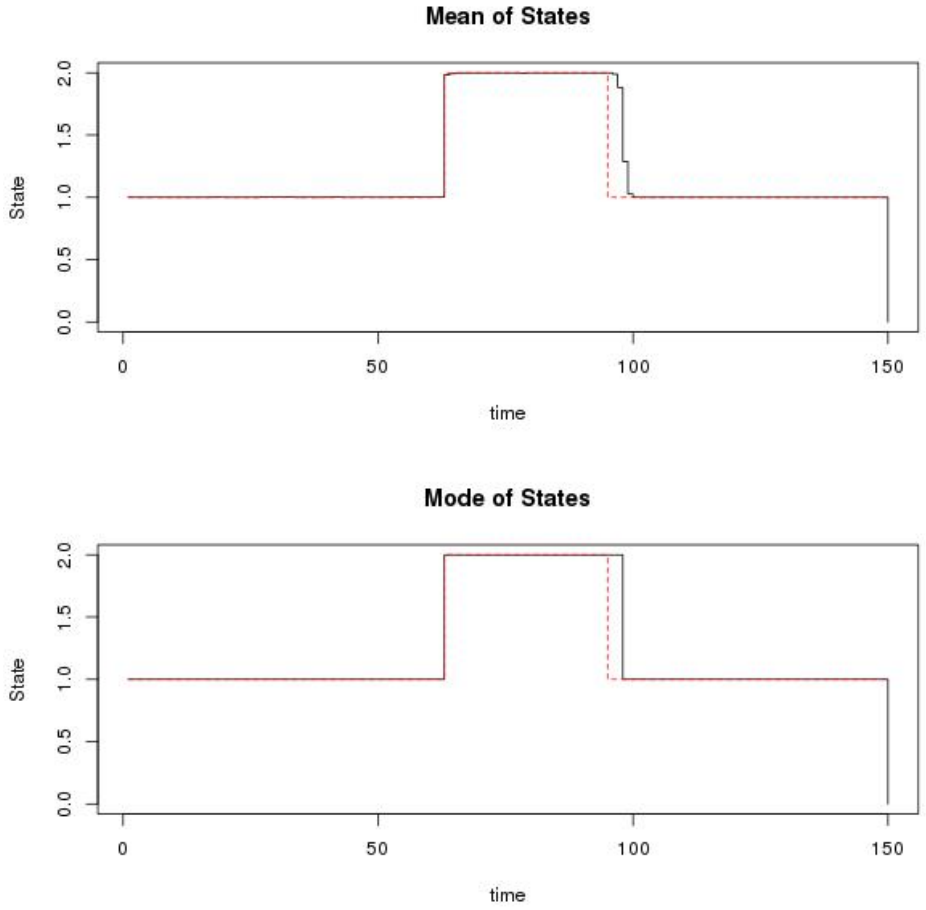

Figure 7: $\delta = 0.75$, prior for transitions is N(8,1), stratified sampling

from state two back to state 1 but this is purely due to data. Since next step innovation is highly dependent on the previous step, it takes around 3 or 4 steps for the transition to be visible in actual returns data. Algorithm also showed robustness against getting stuck in any one state. Surprisingly though parameter estimation is very inconsistent over each run. The only more or less consistent were the estimates for w_1, the constant term for process in state one and they usually were around 0.07 however the estimates for w_2 were ranging from 35 to 84 and very far away from the true value. One reason for it could be that w_2 and w_1 have less restrictions on them. The only requirement for them is that they must be positive whereas a and b parameters must be positive and $a + b < 1$. Also the full likelihood is highly multi-modal with a lot of local maxima and since the new innovation depends on 3 parameters there are multiple combinations of those parameters that will result in a similar likelihood value.

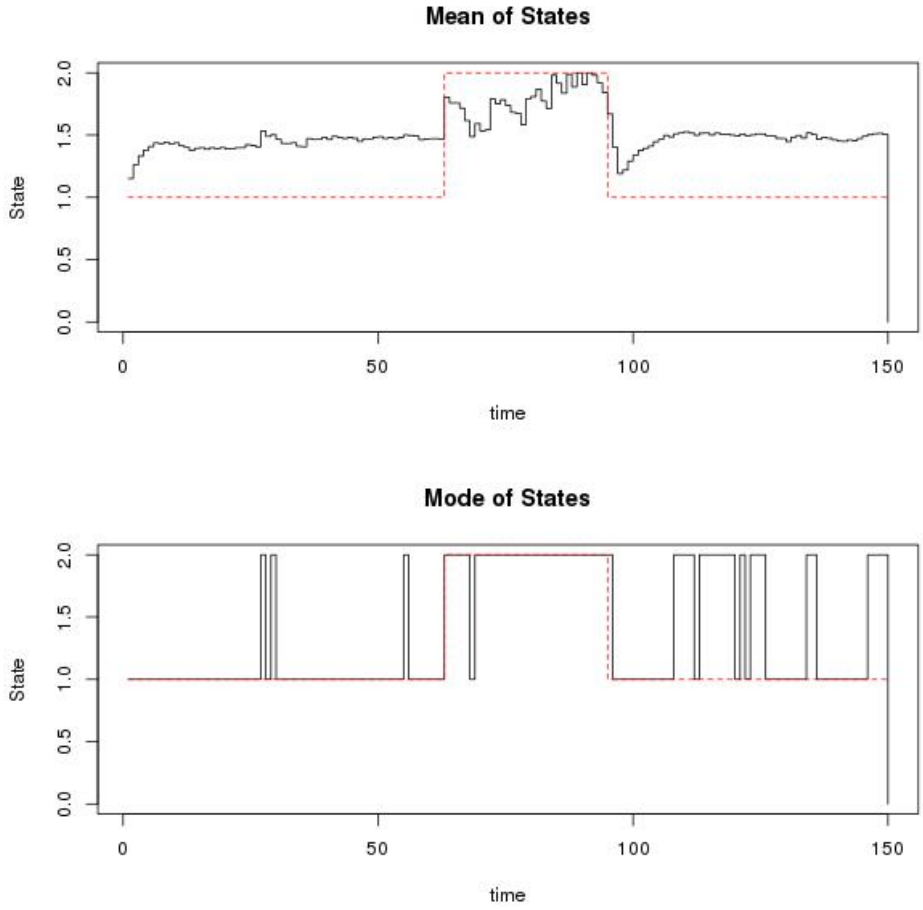

Figure 8: $\delta = 0.75$, prior for transitions is N(2,1), stratified sampling

4 Conclusion

We find that the use of SMC with the APF enhancement is able to accurately predict the occurence of switch points in the GARCH process. This algorithm is highly sensitive to priors used to model the transition probabilities, and in order to produce stable results, requires priors that favor no changes. Parameter estimation using the kernel smoothing method is not as successful. They are generally very inconsistent, with the true value often lying outside the confidence interval, and the extended runs with increased particle set does not seem to yield better results. A tuning parameter closer to one seems to increase precision but not accuracy. Further work may include investigating the effectiveness of the algorithm on a misspecified model and considering multiple states, some of which may be more similar to each other, instead of being very distinct.

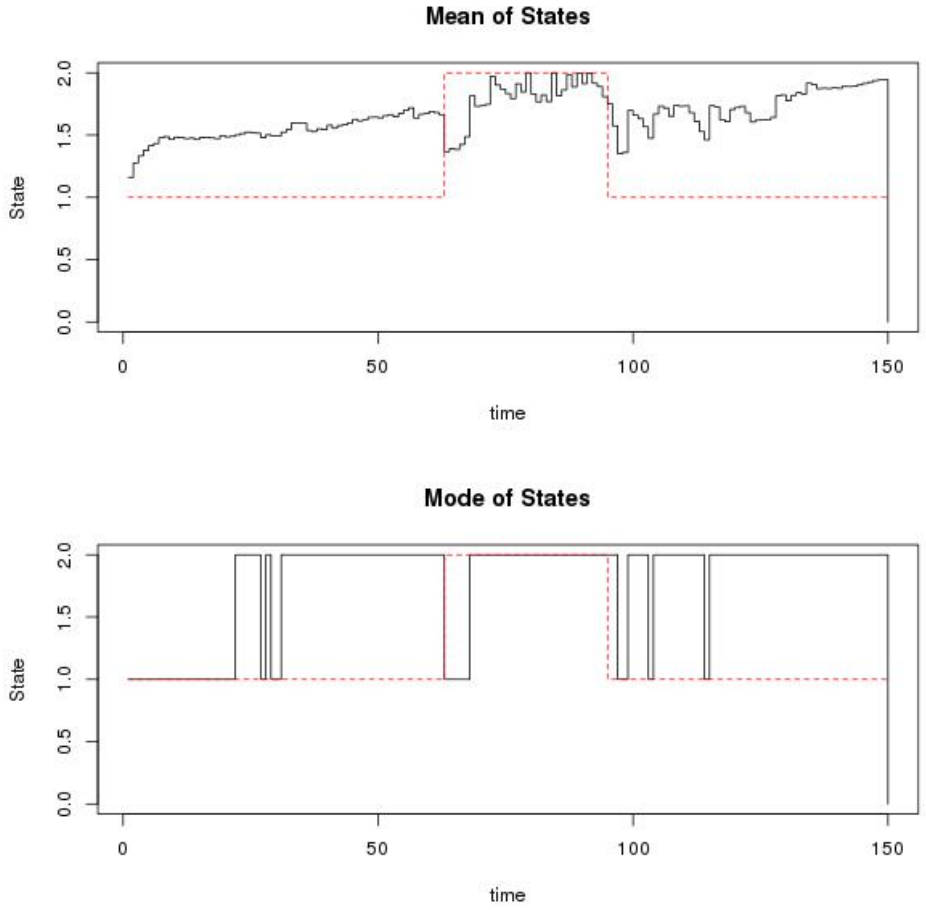

Figure 9: $\delta = 0.75$, prior for transitions is N(2,1), stratified sampling

5 References

1. Doucet, A. 2007. Introduction to Sequential Monte Carlo Methods. [lecture] Machine Learning Summer School 2007 - Tuebingin

2. Godsill, S. 2009. Sequential Monte Carlo Methods. [lecture] Machine Learning Summer School 2009 - Cambridge

3. He, Z., and J.M. Maheu 2009. Real Time Detection of Structural Breaks in GARCH Models. RCEA Working Papers series.

4. Liu, J., and M. West 2001. Combined parameter and state estimation in simulation-based filtering, in Sequential Monte Carlo Methods in Practice, ed. by A. Doucet, N. de Freitas, and N. Gordon. Springer-Verlag.

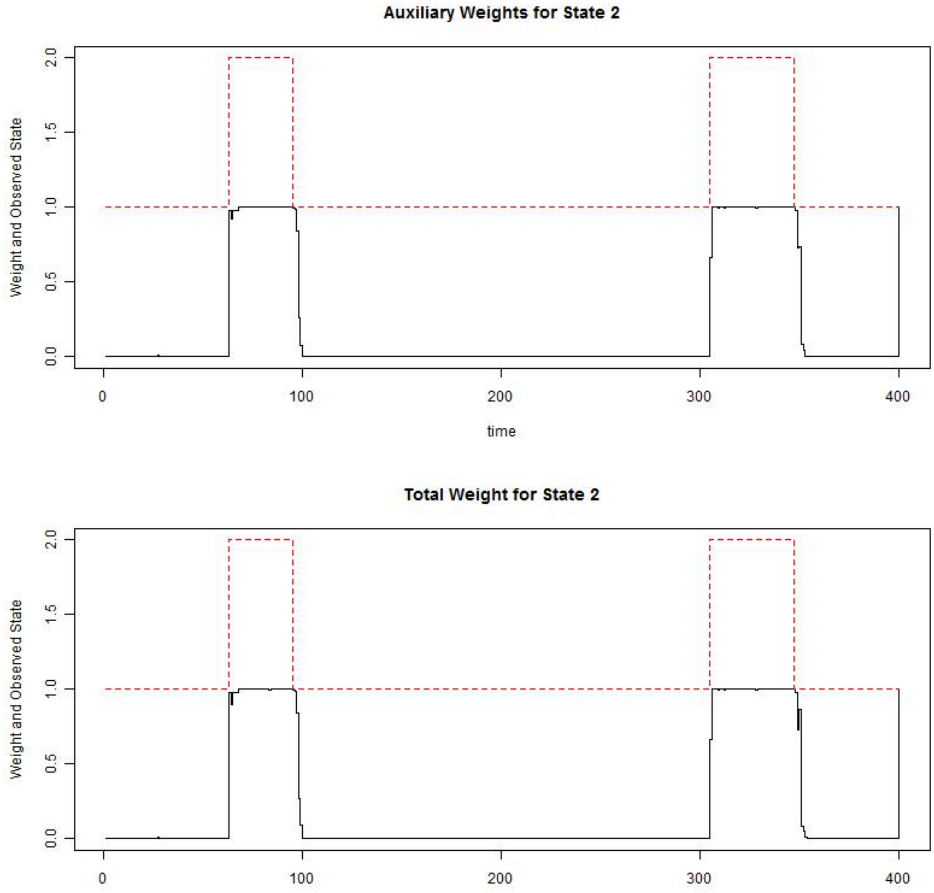

Figure 10: $\delta = 0.75$, prior for transitions is N(8,1), stratified sampling

5. Pitt, M. and N. Sheppard 1999. Filtering via simulation: Auxiliary particle filters, Journal of the American Statistical Association 94:590-599.

	TRUE	Run 1	Run 2	Run 3	Run 4	Run 5
w_1	0.07	0.073	0.072	0.066	0.068	0.067
a_1	0.04	0.090	0.040	0.152	0.103	0.085
b_1	0.03	0.037	0.013	0.014	0.008	0.077
w_2	7.75	69.063	35.596	83.745	61.954	217.527
a_2	0.21	0.405	0.608	0.655	0.760	0.630
b_2	0.77	0.041	0.139	0.030	0.031	0.049

Table 2: Parameter estimates over an extended run. 10,000 particles, 400 time steps, $\delta = 0.75$