PHYS 6006 Final Report

Speeding up Stochastic Volatility Simulations for Financial Markets

Daniel J. Rose
27568326

University of Southampton
Andreas Jüttner

April 27, 2019

Abstract

We take a Stochastic Volatility model, and apply Bayesian inference in order to create a Hybrid Monte Carlo algorithm capable of extracting the model parameters. We generate artificial financial returns data with known parameters, and by applying our algorithm confirm that it can correctly extract these parameters to within one or two standard deviations. We compare our results with existing studies to verify our model, and then implement two successful improvements to our algorithm. We find an optimum setting utilising both improvements which maximises the decorrelation of the parameters, resulting in a shorter required run time for our algorithm.

Contents

List of Abbreviations

HMC Hybrid Monte Carlo (also known as Hamiltonian Monte Carlo)

LAHMC Look Ahead Hybrid Monte Carlo

MCMC Markov Chain Monte Carlo

SV Stochastic Volatility

1 Introduction

Financial institutions are responsible for carefully managing large amounts of assets from different sources, and constantly face the risks coming from many unpredictable future events such as financial crises [1]. To minimise these risks companies calculate financial derivatives (security values that are reliant upon, or derived from, an underlying asset or group of assets), which are used to form contracts between multiple investing parties [2]. For such assets, the measure of the dispersion of returns for a given security is known as the volatility [3]. Traditional volatility models such as 'Black-Scholes options' assume that volatility is constant over the derivative lifetime, and is unaffected by price changes of the underlying security. While these models are still important for modern financial theory [4], in reality volatility is actually a stochastic process (randomly determined and not able to be predicted precisely [5]) which is not well approximated by constant volatility models. By assuming that the volatility of the underlying price fluctuates randomly with trading activity [6], it should be possible to calculate the derivatives more accurately than with simpler models as mentioned before.

Therefore in this project we will be using a Stochastic Volatility (SV) model to simulate security prices. The initial aims are to generate artificial data for the SV model, and then create an algorithm capable of extracting the parameters used in the SV data generation. To achieve this we will be following the same general method as detailed in the 2009 paper by Tetsuya Takaishi [7]. Takaishi's work will be used as reference and for comparison with the results obtained whilst building a working simulation. The type of algorithm chosen is known as a Hybrid Monte Carlo (HMC), which will extract the model parameters from artificial or real world stock data by implementing molecular dynamics based Bayesian inference (discussed in sections 3 and 6.2). Depending on the length of the time-series this method can require a large amount of processing time, and so it will be useful for the simulation code to be as efficient as possible. The ultimate goal of this project is to compare the results from our implementation with existing studies, and then attempt to improve our own simulation algorithm (section 8).

2 Stochastic Volatility Model

We will be using the following equations to generate our SV model, as given by Takaishi [7] and [8, 9]. The time-series data y_t is given by:

$$y_t = \sigma_t \epsilon_t = \exp(h_t/2)\epsilon_t \tag{1}$$

where σ_t represents the volatility and h_t is the associated volatility variable (related by $h_t = \ln \sigma_t^2$). ϵ_t is an error term taken from a standard normal distribution $N(0, 1)$, and so the volatility is effectively the width of the time-series distribution. The volatility variables are then given by the equation:

$$h_t = \mu + \phi(h_{t-1} - \mu) + \eta_t \tag{2}$$

where η_t is an error term taken from the normal distribution $N(0, \sigma_\eta^2)$. The model parameters to be determined are μ, ϕ and σ_η^2, and it is also assumed that $|\phi| < 1$ (discussed in section 5.3). In order to determine the parameter expectation values we will apply Bayesian inference (section 3) with the aim of fitting a probability model to our data, the distribution of which is detailed in section 3.1.

2.1 Validation

First however it is important to validate that our SV model is correct, and in order to do this we can work out the expectation value and the variance of $\ln \sigma_t^2$ analytically to be[1]:

$$\langle \ln \sigma_t^2 \rangle = \langle h_t \rangle = \mu \tag{3}$$

$$\mathrm{Var}(\ln \sigma_t^2) = \mathrm{Var}(h_t) = \frac{\sigma_\eta^2}{1 - \phi^2} \tag{4}$$

Our SV model was implemented in Python using equations (1) and (2), with chosen parameters $(\mu, \phi, \sigma_\eta^2) = (-1, 0.97, 0.05)$. For these parameters we expect $< h_t >= -1$ and $\mathrm{Var}(h_t) = 0.84602$ (to 5 significant figures) using equations (3) and (4) respectively. The function `numpy.random.normal` was chosen to generate the normally distributed random numbers, and the length of the time-series was chosen to be $110,000$ while validating the model. We have discarded the first $10,000$ samples to ensure that the model has stabilised before calculating the mean and variance. The respective values were obtained as -1.0006 and 0.84101, again by using standard Python functions (`numpy.mean` and `numpy.var`). These values acquired from the generated h_t correspond closely to the analytically calculated values within three significant figures for the first result, and a single significant figure for the latter, confirming that the model is functioning correctly for this length of time-series.

2.1.1 Error from the SV Model

To determine the statistical error arising from the length of the time-series, we plot the difference between the analytically calculated values and our experimental

[1]Full derivations of equations (3) and (4) are given in appendices A.1 and A.2 respectively.

results (the mean and variance of h_t), against the sampled length of the time-series N. Due to their stochastic nature these results for a single generated SV model contain many fluctuations and random elements which make for a difficult analysis. However by generating 30 different instances of the SV model using the same parameters, and then taking the mean values calculated from each sample length across all 30 models, we minimise these issues and can see the overall trend. These averaged results are plotted in Figure 1, which shows that for equation (3) the statistical error in the model reduces at a rate of -0.50331 ± 0.00137 with respect to N, and similarly for equation (4) at a rate of -0.48200 ± 0.00103. While these values are not quite within what would normally be considered acceptable error boundaries (of -0.5), in this case this is accounted for by the stochastic nature of our model, and so these results suggest that the statistical error in the mean and variance of our SV model function is actually $\propto 1/\sqrt{N}$. This result is

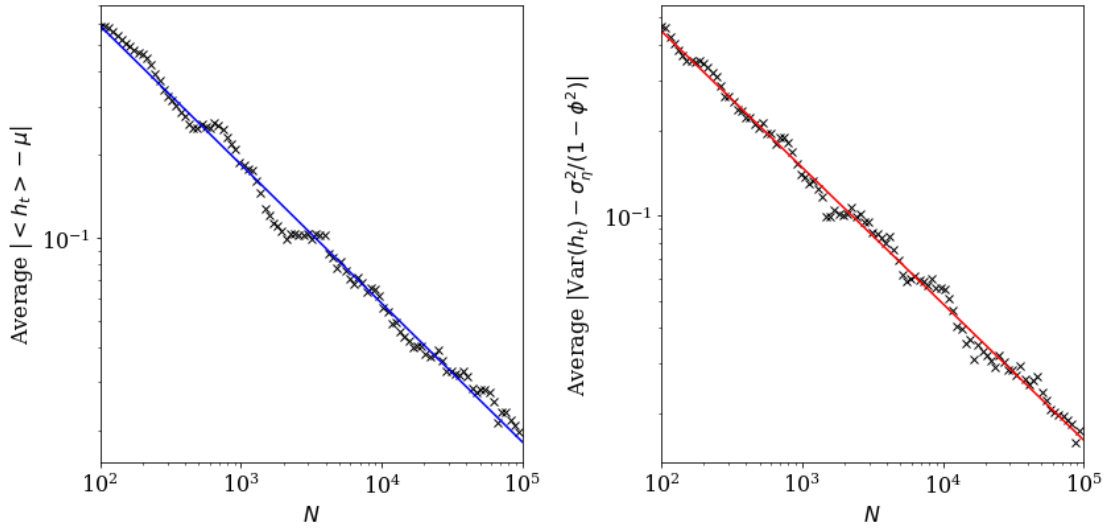

Figure 1: Plots showing the averaged results of 30 SV models generated with $(\mu, \phi, \sigma_\eta^2) = (-1, 0.97, 0.05)$, where each result for a different value of N is the average of the results from each of the 30 models. The reduction in statistical errors for equations (3) and (4) are shown in the left and right plots respectively.

important for later on as we will be able to save processing time by decreasing the length of our time-series, for example reducing N by a factor of 20 (5% of the maximum length used here) will increase the statistical error by less than a factor of 5. As the SV model we are using has an inherent randomness we do not require it to represent the input parameters to an extremely high degree of accuracy, so this increase in error is acceptable given the computational gain achieved as a result.

3 Bayesian Inference

Bayesian inference is the process of fitting a probability model to a set of data, and summarizing the result through a probability distribution on the model parameters and unobserved quantities such as predictions for new observations [10]. From Bayes' theorem we have the posterior probability, or the probability of the parameters $\theta = (\mu, \phi, \sigma_\eta^2)$ given the data y as:

$$p(\theta|y) = \frac{p(\theta)p(y|\theta)}{p(y)} \tag{5}$$

where $p(\theta)$ is prior probability (of θ before y), $p(y|\theta)$ is the likelihood function (probability of y given θ) and $p(y) = \int p(y|\theta)p(\theta)d\theta$ is the marginal likelihood or model evidence [11]. For this project we will need to compute the posterior probability $p(\theta|y)$ using the above equation. As only the model-dependant likelihood function $p(y|\theta)$ depends on the data y, we can determine expectation values of the model parameters as:

$$\langle f(\theta) \rangle = \int f(\theta)p(\theta|y)d\theta \tag{6}$$

3.1 Likelihood Function for the SV Model

We then define the likelihood function for our SV model to be [7–9]:

$$p(y|\theta) = \int \prod_{t=1}^{N} p(y_t|h_t)p(h_1|\theta) \prod_{t=2}^{N} p(h_t|h_{t-1}, \theta)dh_1 dh_2 ... dh_N \tag{7}$$

where $\theta = (\mu, \phi, \sigma_\eta^2)$. The first term represents the data distribution and is defined:

$$p(y_t|h_t) = \frac{1}{\sqrt{2\pi\sigma_t^2}} \exp\left(-\frac{y_t^2}{2\sigma_t^2}\right) \tag{8}$$

And the probability densities for the volatility distributions are defined as:

$$p(h_1|\theta) = \sqrt{\frac{1-\phi^2}{2\pi\sigma_\eta^2}} \exp\left(-\frac{(h_1-\mu)^2}{2\sigma_\eta^2/(1-\phi^2)}\right) \tag{9}$$

$$p(h_t|h_{t-1}, \theta) = \frac{1}{\sqrt{2\pi\sigma_\eta^2}} \exp\left(-\frac{(h_t - \mu - \phi(h_{t-1}-\mu))^2}{2\sigma_\eta^2}\right) \tag{10}$$

The likelihood function as given in equation (7) is a highly multidimensional integral over all the volatility variables. As it is impractical to solve such a complicated

integral analytically [9], we use the Markov Chain Monte Carlo (MCMC) method (discussed in section 4) to obtain the maximum likelihood numerically. To apply this method we will then need to obtain the probability distribution for each parameter by extracting the corresponding terms from equation (5). Equation (7) is the likelihood function $p(y|\theta)$, and $p(\theta)$ is the prior (parameter) probability for which we assume $p(\sigma_\eta^2) = (\sigma_\eta^2)^{-1}$ and for the others $p(\mu) = p(\phi) = constant$ (making this term irrelevant when extracting the dependant terms) [7]. The marginal likelihood is just a normalisation constant which is also not important here, but will be needed later to check that our method for randomly drawing each parameter corresponds with the intended probability distribution (section 5).

4 Markov Chain Monte Carlo

The integral of a function $f(x)$ from a to b can be approximated as the mean of $f(x_n)$, where the values of x_n are distributed according to $\rho(x_n) = 1/(b-a)$ [12]:

$$\langle f(x) \rangle = \frac{1}{b-a} \int_a^b f(x)dx = \lim_{N\to\infty} \frac{1}{N} \sum_{n=1}^{N} f(x_n) \tag{11}$$

Here N is the sampled length of the data, and we have already shown in section 2.1 that the statistical error in the mean and variance of our SV model function reduces $\propto 1/\sqrt{N}$. If however the integral is over a function and a probability distribution $\int_a^b f(x)\rho(x)dx$, then the distribution $\rho(x)$ will suppress or enhance certain values of x. To take this into account when sampling the integral, we make use of a technique known as importance sampling. This is where samples are drawn from a known distribution (which ideally is similar to the desired distribution), and then accepted or rejected using a probability which accounts for the differences between the chosen and desired distributions [13, 14]. Hence we now write the expectation value as:

$$\langle f(x) \rangle = \frac{\int_a^b f(x)\rho(x)dx}{\int_a^b \rho(x)dx} \tag{12}$$

where the denominator is the normalisation of the probability density $\rho(x)$. Using equations (6) and (12), we can write the expectation values of the parameters as:

$$\langle f(\theta) \rangle = \frac{1}{N} \sum_{n=1}^{N} (\theta_n) \tag{13}$$

where θ_n represents a series element generated according to the probability distribution $P(\theta) = p(\theta|y)$. As we will require use of distributions for which it is impossible to directly draw random numbers, sections 4.1 and 4.2 will now explain how to proceed in these cases (via importance sampling).

4.1 Markov Chains

A Markov chain is a stochastic model, defined as a sequence of random events for which the probability of each event depends only on itself and the previous event, and not its position in the overall chain [12, 15]. Starting from some arbitrary configuration x_0, consider the following Markov chain sequence evolving through some multidimensional phase space [16]:

$$x_0 \to x_1 \to x_2 \to ... \to x_n \to x_{n+1} \to ... \to x_N \tag{14}$$

Changing from configuration x_n to x_{n+1} is called an update, which has the conditional transition probability $T(x_{n+1}|x_n)$. All the transition probabilities follow:

$$0 \geq T(x_{n+1}|x_n) \geq 1 \qquad \text{and} \qquad \sum_{x_{n+1}} T(x_{n+1}|x_n) = 1 \tag{15}$$

The latter of these conditions ensures that the total probability of getting from a configuration x_n to any x_{n+1} configuration is 1 (normalisation). As the algorithm evolves the probability of any transition $x_n \to x_{n+1}$ must be the same as the opposite transition $x_{n+1} \to x_n$, which prevents any sinks or sources occurring in the phase space[2]. Mathematically this can be written as a balance relation:

$$P(x_{n+1}) = \sum_x T(x_{n+1}|x_n)P(x_n) \equiv \sum_x T(x_n|x_{n+1})P(x_{n+1}) \tag{16}$$

which says that the probability $P(x_{n+1})$ of being in state x_{n+1} must equal the sum of the transitional probabilities from x_n to x_{n+1}, multiplied by the probability of being in original state $P(x_n)$. And furthermore by our definition this must be exactly equivalent to the sum of the transitional probabilities from x_{n+1} to x_n, multiplied by the probability of being in state $P(x_{n+1})$. Equation (16) also tells us that so long as $T(x_{n+1}|x_n)$ is positive, every point in phase space will eventually be visited. In other words all possible configurations will be appropriately sampled by our Markov chain, a property which is known as strong ergodicity [17]. Finally a sufficient solution to the above relation is the following detailed balance condition [18]:

$$T(x_{n+1}|x_n)P(x_n) = T(x_n|x_{n+1})P(x_{n+1}) \tag{17}$$

4.2 Metropolis-Hastings Algorithm

The following steps describe a transition from state x_n to x_{n+1} in equation (14):

[2]Although if the opposite transition were to occur the final state would still be labelled x_{n+2} with respect to the Markov chain, even though in this instance $x_{n+2} = x_n$.

1. Choose a new configuration x_{n+1} according to some guess transition probability $T_0(x_{n+1}|x_n)$

2. Accept x_{n+1} with the following probability:

$$T_A(x_{n+1}|x_n) = \min\left[\frac{T_0(x_n|x_{n+1})P(x_{n+1})}{T_0(x_{n+1}|x_n)P(x_n)}, 1\right] \tag{18}$$

   If x_{n+1} is not accepted, then measure x_n instead.

Repeated application of this algorithm defines a Markov chain from section 4.1 [19, 20]. To this end we can show that this algorithm fulfils the detailed balance condition in equation (17) via the accept/reject step:

$$\begin{aligned} T(x_{n+1}|x_n)P(x_n) &= T_0(x_{n+1}|x_n)T_A(x_{n+1}|x_n)P(x_n) \\ &= T_0(x_{n+1}|x_n)\min\left[\frac{T_0(x_n|x_{n+1})P(x_{n+1})}{T_0(x_{n+1}|x_n)P(x_n)}, 1\right]P(x_n) \\ &= \min\left[T_0(x_n|x_{n+1})P(x_{n+1}), T_0(x_{n+1}|x_n)P(x_n)\right] \\ &= T(x_n|x_{n+1})P(x_{n+1}) \end{aligned} \tag{19}$$

where we have gone from the third to fourth line using equation (17), and knowing that as probabilities all T and P are positive.

5 Parameter Probability Distributions

5.1 Distribution for σ_η^2

Extracting terms from equation (7) which depend on σ_η^2, and then multiplying by the prior probability $p(\sigma_\eta^2)$ according to equation (5) we obtain the distribution[3]:

$$P(\sigma_\eta^2) = (\sigma_\eta^2)^{-\frac{N}{2}-1}\exp\left(-\frac{A}{\sigma_\eta^2}\right) \tag{20}$$

where N is the total number of the volatility variables h_t, and A is defined as:

$$A = \frac{1}{2}[(1-\phi^2)(h_1-\mu)^2 + \sum_{t=2}^{N}(h_t - \mu - \phi(h_{t-1}-\mu))^2] \tag{21}$$

Equation (20) is equivalent to an inverse gamma distribution, and hence σ_η^2 can be simply drawn using a standard `scipy` function with appropriate choice of parameters. The function `scipy.stats.invgamma.rvs` was used with parameters

[3]The full derivation of equation (20) is given in appendix A.3.

`shape` $= N/2$ and `scale` $= A$. Later on the input values for N and A will be calculated from the SV model, but for now just to verify that our implementation matches equation (20) we set $N = 20$, $A = 2$, and draw $1,000,000$ distributed numbers. The results are displayed as a histogram in Figure 2, and equation (20) is plotted as a line on top. From this it can be clearly seen that the distribution of drawn numbers matches the σ_η^2 distribution as desired. Note that the histogram has been normalised by including the optional argument `density=True` when using `matplotlib.pyplot.hist`, and the line for equation (20) has been multiplied by the normalisation factor $\frac{A^{N/2}}{\Gamma(N/2)}$ where $\Gamma(x) = (x-1)!$ is the gamma function of x [21]. Code efficiency for equation (21) was also improved by changing the imple-

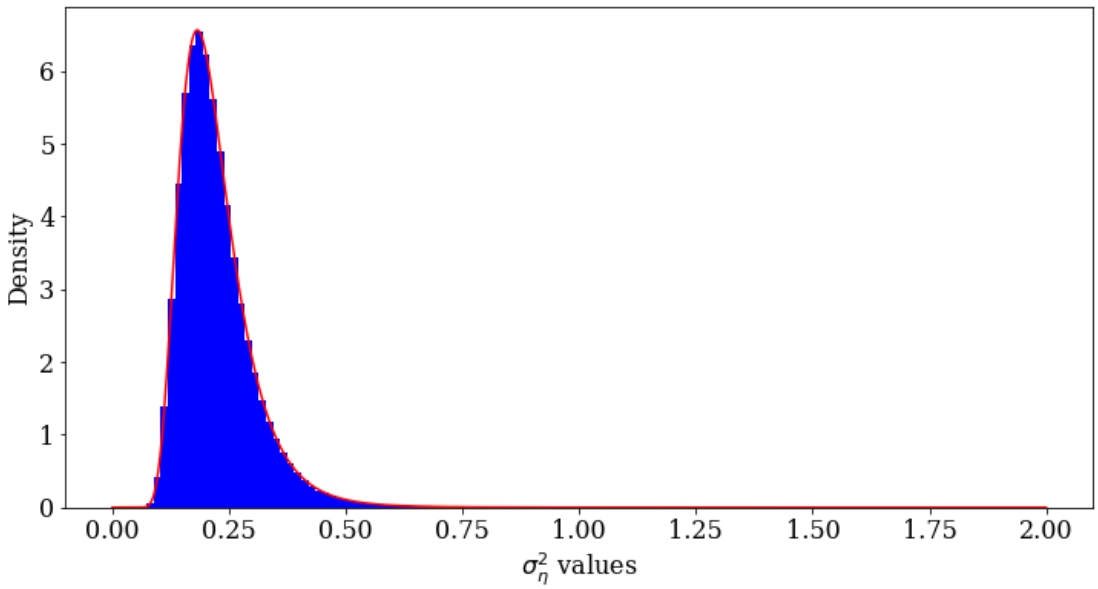

Figure 2: Histogram (100 bins) of numbers drawn from $P(\sigma_\eta^2)$, equation (20).

mentation of the sum from a `for` loop to `numpy.sum`. It proved slightly difficult to quantify this improvement because the code completed so much faster than a `for` loop, but by measuring the time taken for repeated iterations of `numpy.sum` and comparing this to a single completion of the `for` loop we estimate that `numpy.sum` is over $80,000$ times faster in this instance. This improvement however was not noticeable when producing the histogram in Figure 2, as the `for` loop only took a couple of seconds to complete and was required only once. However this change will prove extremely beneficial for the full simulation algorithm, as the equation (21) sum and several others will each be required to complete many thousands of times. To this end the same change was also made later for equations (24), (26) and (27).

5.2 Distribution for μ

Extracting terms from equation (7) which depend on μ, we obtain the distribution[4]:

$$P(\mu) = \exp\left(-\frac{B}{2\sigma_\eta^2}(\mu - \frac{C}{B})^2\right) \tag{22}$$

where B and C are defined as:

$$B = (1-\phi^2) + (N-1)(1-\phi)^2 \tag{23}$$

$$C = (1-\phi^2)h_1 + (1-\phi)\sum_{t=2}^{N}(h_t - \phi h_{t-1}) \tag{24}$$

Equation (22) is equivalent to a normal distribution, so μ can also be drawn using a standard NumPy routine. `numpy.random.normal` was used with parameters `loc` $= C/B$ and `scale` $= \sqrt{\sigma_\eta^2/B}$, and while validating the implementation $B = 2$, $C = 10$, and $1,000,000$ numbers were drawn. The results are displayed as a histogram in Figure 3 and equation (22) is plotted as a line. Again the distribution of drawn numbers matches the μ distribution as desired, noting that equation (22) is multiplied by the normalisation factor $\sqrt{B/2\pi\sigma_\eta^2}$ [22].

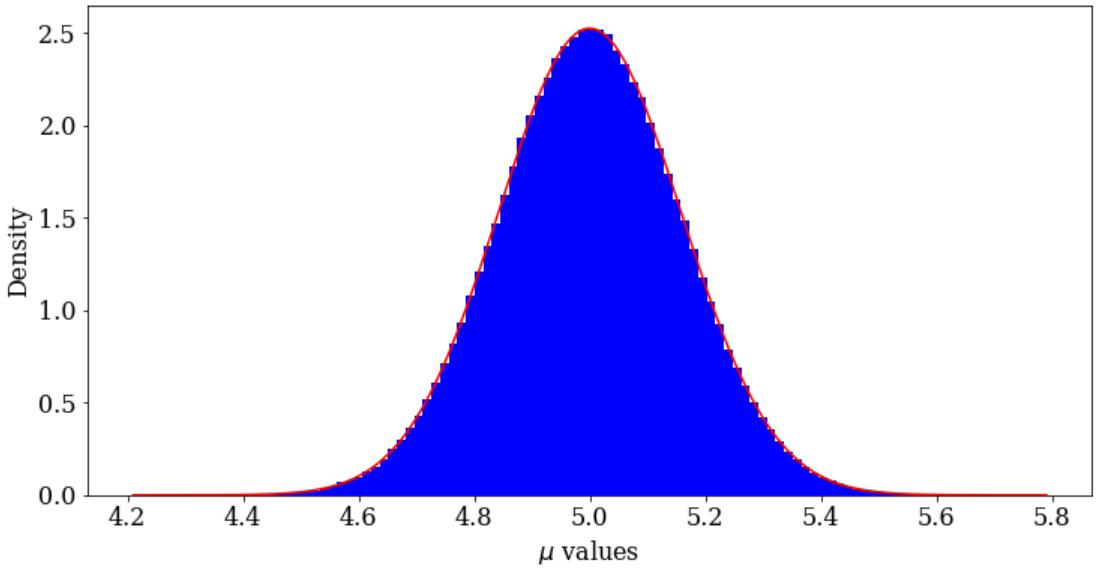

Figure 3: Histogram (100 bins) of numbers drawn from $P(\mu)$, equation (22).

[4]The full derivation of equation (22) is given in appendix A.4.

5.3 Distribution for ϕ

Extracting terms from equation (7) which depend on ϕ, we obtain the distribution[5]:

$$P(\phi) = \sqrt{1-\phi^2}\exp\left(-\frac{D}{2\sigma_\eta^2}(\phi - \frac{E}{D})^2\right) \tag{25}$$

where D and E are defined as:

$$D = -(h_1 - \mu)^2 + \sum_{t=2}^{N}(h_{t-1} - \mu)^2 \tag{26}$$

$$E = \sum_{t=2}^{N}(h_t - \mu)(h_{t-1} - \mu) \tag{27}$$

Equation (25) partly resembles a normal distribution like in section 5.2, however

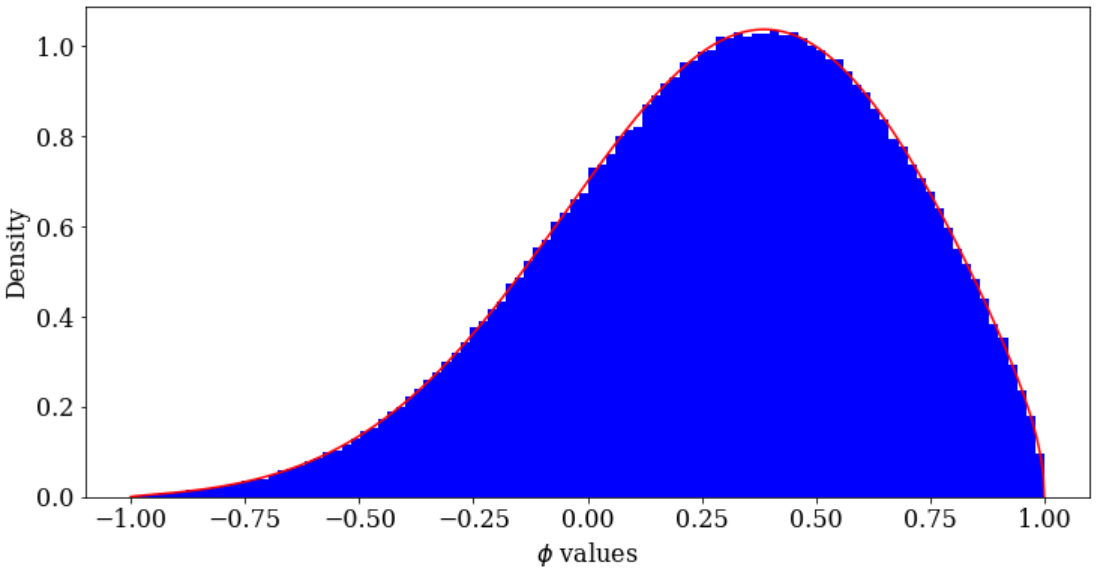

Figure 4: Histogram (100 bins) of numbers drawn from $P(\phi)$, equation (25).

now there is also a $\sqrt{1-\phi^2}$ term in front. As this is not a standard distribution there is no existing Python function we can use, and instead to draw values of ϕ we use the Metropolis method as described in section 4. To utilise this algorithm we first define $P(\phi) = P_1(\phi)P_2(\phi)$, where:

$$P_1(\phi) = \sqrt{1-\phi^2} \tag{28}$$

[5]The full derivation of equation (25) is given in appendix A.5.

$$P_2(\phi) = \exp\left(-\frac{D}{2\sigma_\eta^2}(\phi - \frac{E}{D})^2\right) \tag{29}$$

Since equation (29) alone is equivalent to a normal distribution, ϕ can be drawn from it in the same way that μ is drawn from equation (22). If ϕ_{new} is drawn from $P_2(\phi)$, then in order to obtain the correct distribution $P(\phi)$, the value of ϕ_{new} is accepted with the following probability[6] [7]:

$$P^{\phi}_{metro}(\phi_{new}, \phi) = \min\left[\sqrt{\frac{1-\phi_{new}^2}{1-\phi^2}}, 1\right] \tag{30}$$

ϕ in equation (30) refers to the previous value in the Markov chain, and it is also clear that we must restrict $|\phi| < 1$ so as to avoid a negative value in the square root. In practice, the Metropolis-Hastings algorithm from section 4.2 is implemented in the following way:

1. Choose some random starting ϕ_0 (remembering $\phi \in [-1, 1]$)
2. Draw a ϕ_n using the distribution $P_2(\phi)$, equation (29)
3. Compute $P^{\phi}_{metro}(\phi_n, \phi_{n-1})$ using equation (30)
4. Draw a uniformly distributed random number $u \in [0, 1]$ and:
   (a) accept ϕ_n if $u \leq P^{\phi}_{metro}(\phi_n, \phi_{n-1})$
   (b) reject ϕ_n if $u > P^{\phi}_{metro}(\phi_n, \phi_{n-1})$, and set $\phi_n = \phi_{n-1}$

Steps $2 \rightarrow 4$ are then repeated until a sufficient number of values have been obtained so as to represent the desired distribution. `numpy.random.normal` was used with parameters `loc` $= E/D$ and `scale` $= \sqrt{\sigma_\eta^2/D}$, and while validating the implementation $D = 0.2$, $E = 0.1$, and $1,000,000$ numbers were drawn. The results are displayed as a histogram in Figure 4 and equation (25) is plotted as a line. Again the distribution of drawn numbers matches the ϕ distribution as desired, noting that the $P(\phi)$ line has been divided by $\lambda = \int_1^1 P(\phi)$ in order for it to be normalised (λ integrated numerically using `scipy.integrate.quad`).

6 Updating the Volatility Variables

6.1 Distribution for h_t

Along with each iteration of the parameters, the volatility variables h_t also have to be updated as it is with respect to them by which we are integrating in equation

[6] The full derivation of equation (30) is given in appendix A.6.

(7). Extracting terms which depend on h_t, we obtain the distribution[7]:

$$P(h_t) \equiv P(h_1, h_2, ...h_N) = \tag{31}$$
$$\exp\left[-\sum_{t=1}^{N}\left(\frac{h_t}{2}+\frac{y_t^2}{2}e^{-h_t}\right)-\frac{(h_1-\mu)^2}{2\sigma_\eta^2/(1-\phi^2)}-\sum_{t=2}^{N}\frac{(h_t-\mu-\phi(h_{t-1}-\mu))^2}{2\sigma_\eta^2}\right]$$

This is a difficult probability distribution to attempt to draw numbers from, so we will again be using a Markov chain method. While it would be possible to update the variables locally using a Metropolis algorithm (similar to section 5.3), here we choose to use a HMC algorithm which updates h_t globally and therefore greatly increases the processing efficiency [7, 8, 23, 24].

6.2 Hybrid Monte Carlo

The HMC algorithm was originally developed for MCMC simulations of lattice Quantum Chromo Dynamics, with the basic idea combining molecular dynamics and a Metropolis accept/reject step [25]. We define our Hamiltonian:

$$H(p,x) = \frac{1}{2}p^2 - \ln f(x) \tag{32}$$

where p represents the conjugate momentum variables to x. By then setting $f(x) = P(h_t)$ from equation (31) we obtain the following Hamiltonian $H(p_t, h_t)$:

$$H = \sum_{t=1}^{N}\left(\frac{1}{2}p_t^2+\frac{h_t}{2}+\frac{y_t^2}{2}e^{-h_t}\right)+\frac{(h_1-\mu)^2}{2\sigma_\eta^2/(1-\phi^2)}+\sum_{t=2}^{N}\frac{(h_t-\mu-\phi(h_{t-1}-\mu))^2}{2\sigma_\eta^2} \tag{33}$$

New values of p_t and h_t are then drawn by integrating the equations of motion[7]:

$$\frac{dh_t}{dt} = \frac{\partial H}{dp_t} = p_t \tag{34}$$

$$\frac{dp_t}{dt} = -\frac{\partial H}{dh_t} = \tag{35}$$
$$\begin{cases} \frac{y_1^2}{2}e^{-h_1} - \frac{1}{2} - \frac{1}{\sigma_\eta^2}(h_1 - \mu - \phi(h_{t-2} - \mu)) & \text{for } t = 1 \\ \frac{y_t^2}{2}e^{-h_t} - \frac{1}{2} - \frac{1}{\sigma_\eta^2}(h_t(1+\phi^2) - \phi(h_{t-1} + h_{t+1}) - \mu(\phi-1)^2) & \text{for } 1 < t < N \\ \frac{y_N^2}{2}e^{-h_N} - \frac{1}{2} - \frac{1}{\sigma_\eta^2}(h_N - \mu - \phi(h_{N-1} - \mu)) & \text{for } t = N \end{cases}$$

Realistically the integrals in equations (34) and (35) cannot be solved analytically, so therefore we require a numerical integrator which must both conserve energy (the Hamiltonian) and be reversible in time [7, 25]. We have chosen a 2nd order leapfrog integrator for these reasons, as discussed in section 6.3.

[7]Full derivations of equations (31) and (35) are given in appendices A.7 and A.8 respectively.

6.3 Leapfrog Integrator

The integration is performed in n steps and we wish for the total trajectory length to be 1 [25], so the number of steps $n = 1/\Delta t$ where Δt is the step size. First an initial choice must be made for the arrays x_0 (the volatility variables h_t) and p_0 (the momenta p_t). Then an initial x half-step is made:

$$x_{\Delta t/2} = x_0 + p_0 \frac{\Delta t}{2} \tag{36}$$

p and x then take turns moving a whole step, with this process looping $n-1$ times (hence the name leapfrog)[8]:

$$\begin{aligned} p_{t+\Delta t} &= p_t + \frac{dp_t}{dt}\Delta t \\ x_{(t+\Delta t/2)+\Delta t} &= x_{t+\Delta t/2} + p_{t+\Delta t}\Delta t \end{aligned} \tag{37}$$

Finally p moves one more step and x one half-step to finish the trajectory:

$$\begin{aligned} p_n &= p_{n-1} + \frac{dp_{n-1}}{dt}\Delta t \\ x_n &= x_{n-\Delta t/2} + p_n \frac{\Delta t}{2} \end{aligned} \tag{38}$$

From this we obtain new values x_n and p_n which we globally accept or reject using a Metropolis step with the following probability [7]:

$$P^H_{metro} = \min\left[\exp(-\Delta H), 1\right] \tag{39}$$

where ΔH is the energy difference given by $\Delta H = |H(p_n, h_n) - H(p_0, h_0)|$, using $H(p_t, h_t)$ from equation (33). In order to validate the implementation of our integrator, we first check that it is time reversible. By starting with all values of $x_0 = -0.6$ and p_0 (with the same length as x_0) as normally distributed random numbers, we integrate forward $n = 100$ steps ($\Delta t = 0.01$). Now taking the obtained values for x_n and p_n we integrate back by setting $\Delta t = -0.01$, and find that the values obtained at the end of this integration all correctly match the initial values used for x_0 and p_0 to at least 6 decimal places. Furthermore for a 2nd order integrator as the step size Δt decreases, $|\Delta H|$ should converge to the second order. This result is plotted in Figure 5, showing that our algorithm is converging correctly with a convergence rate of 1.99973 ± 0.00014.

[8]Leapfrog method detailed here is slightly more efficient than that described on page 7 of Takaishi's paper [7], as the half-steps are only done at the beginning and the end of the process (instead of twice for every x step) so less overall computations are required.

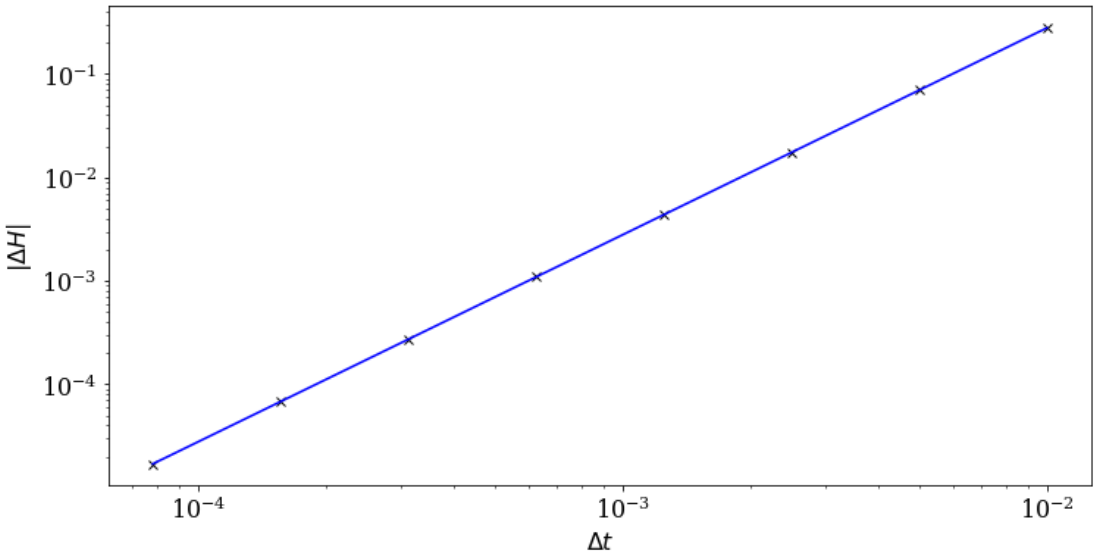

Figure 5: Plot showing the convergence of $|\Delta H|$ for our leapfrog integrator.

6.4 Full Algorithm Overview

The algorithm to draw new values of the parameters θ and volatility variables h_t will function as the following loop (initially starting with random values for θ and h_t^{old} array):

1. Draw a new μ value according to $P(\mu)$, equation (22)
2. Draw a new ϕ value according to $P(\phi)$, equation (25)
3. Draw a new σ_η^2 value according to $P(\sigma_\eta^2)$, equation (20)
4. Generate p_t^{old} ($N(0,1)$ normal distribution) for the same length as h_t^{old}
5. Calculate $H_{old} = H(p_t^{old}, h_t^{old}, \mu, \phi, \sigma_\eta^2)$ using equation (33)
6. Run the leapfrog integrator on (h_t^{old}, p_t^{old}) for n steps to obtain (h_t^{new}, p_t^{new})
7. Calculate $H_{new} = H(p_t^{new}, h_t^{new}, \mu, \phi, \sigma_\eta^2)$ using equation (33)
8. Calculate $\Delta H = H_{new} - H_{old}$
9. Draw a uniformly distributed random number $u \in [0,1]$ and:
   (a) accept h_t^{new} if $u \leq P_{metro}^H$ using equation (39)
   (b) reject h_t^{new} if $u > P_{metro}^H$, and set $h_t^{new} = h_t^{old}$ (meaning that h_t^{old} for the next loop will be the same array as for this loop)

The loop is run as many times as needed, and after each iteration we add the values of μ, ϕ, σ_η^2 and h_t to separate arrays (an array of arrays for h_t). After the program has finished the mean and standard deviation of each parameter can be extracted from the relevant array, these values should match the input parameters for our SV model test data if the algorithm functions correctly.

7 Algorithm Implementation

In order to test our implementation of section 6, we require SV model test data with known parameters. Using the same parameters $(\mu, \phi, \sigma_\eta^2) = (-1, 0.97, 0.05)$ as in section 2.1, we generate a time-series of length 5000 (6000, discarding the first 1000 to allow the model to stabilise) using equations (1) and (2). This artificial time-series will be our focus for section 7.2, and is shown explicitly in Figure 6.

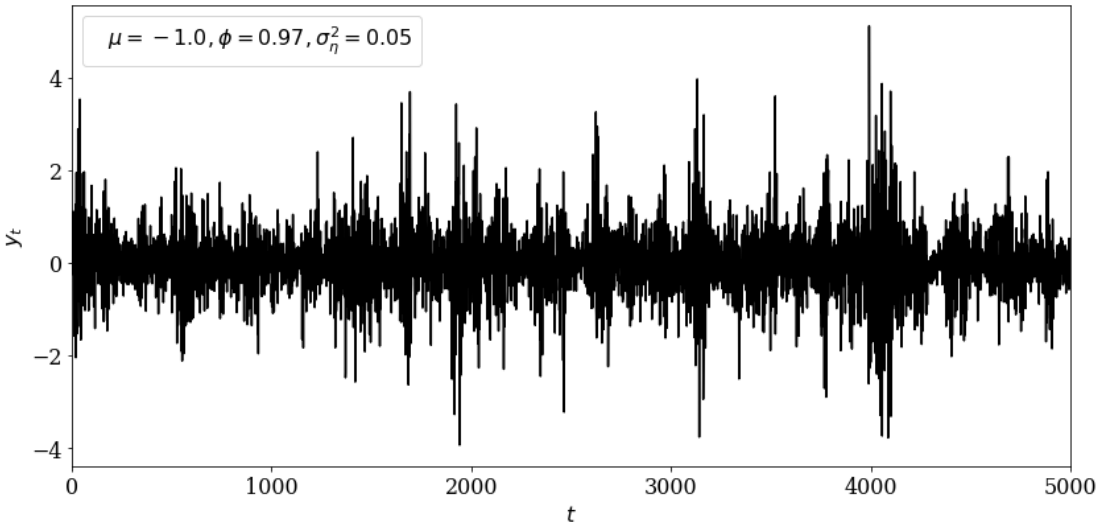

Figure 6: Plot showing the generated SV time-series we will mainly be using.

7.1 Parameter Impact on the SV Model

Firstly however we wish to check that the parameters impact the SV model in the way we expect from section 2. We also generate 6 other SV models changing one of either μ, ϕ or σ_η^2 each time, with the first 2000 elements of each time-series plotted in Figure 7. Note that we have seeded NumPy's random number generator (via `numpy.random.seed`) using the same seed value for each SV model, so that the random numbers used each time will be identical [26] and therefore the results more easily compared. Comparing the two plots in the first row of

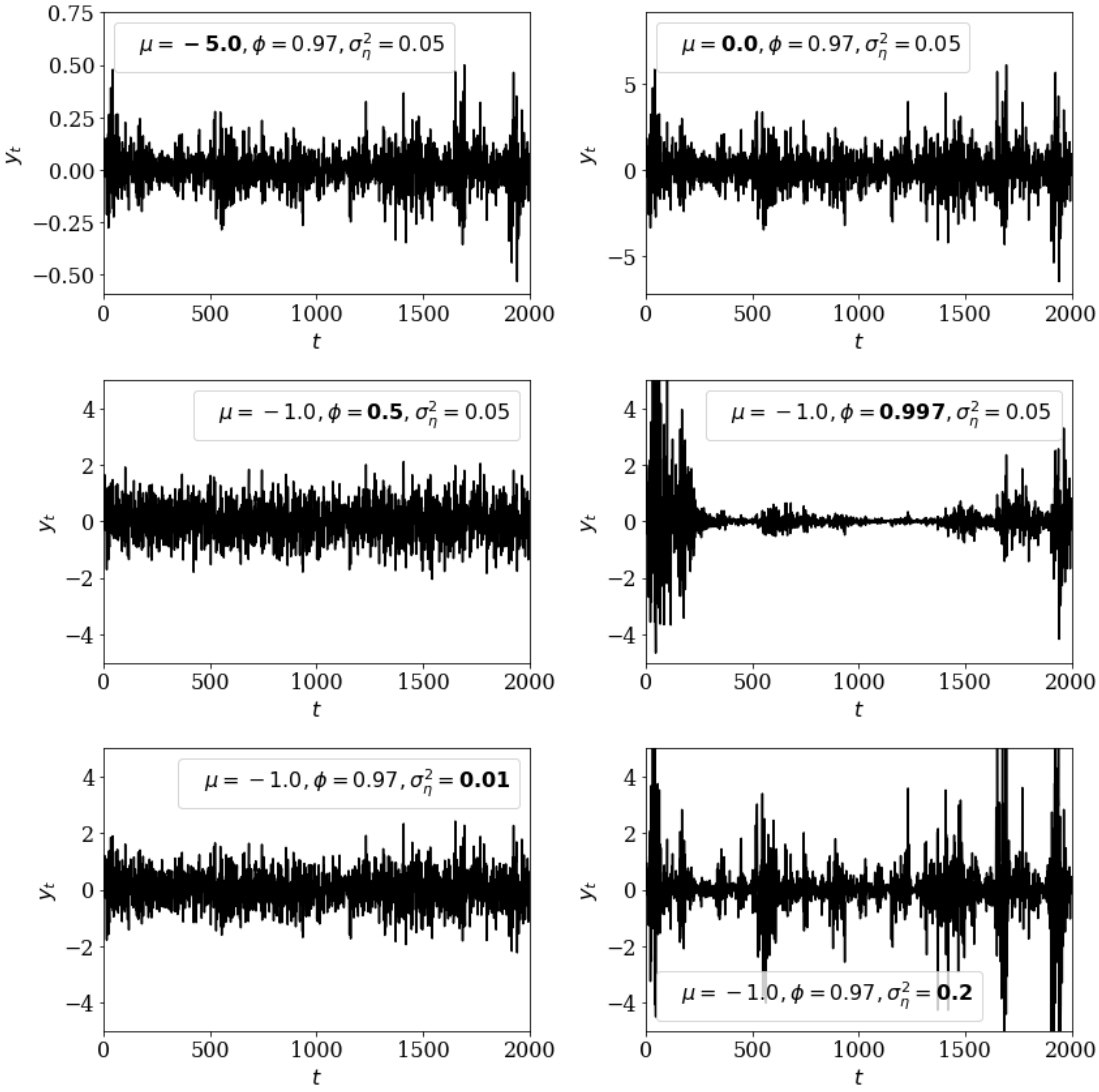

Figure 7: Plots showing SV model time-series with different parameters (varying μ in the first row, ϕ in the second row and σ_η^2 in the third). Note that the y-axis scaling for the first row plots has not been fixed so that the shape can be seen.

Figure 7 to the first 2000 time-series elements of Figure 6, we clearly see that the input value of μ does not affect the shape of the time-series produced as all three plots look identical. Instead looking at the scaling of y_t (on the y-axis) we see that μ instead affects the overall amplitude of the time-series, with the ratios of y_t between different t values remaining unchanged. This agrees with what we expect from equations (1) and (2), because $y_t \propto e^{h_t}$ and $h_t \propto \mu$, so y_t is simply multiplied by the scaling factor e^μ. Similarly $h_t \propto \eta_t$, where η_t is drawn from the normal distribution $N(0, \sigma_\eta^2)$. As the same order of random numbers will be drawn due to

the seed, varying the input value of σ_η^2 should simply affect the amplitude of these drawn numbers. Again this agrees with the plots in the third row of Figure 7 where we can see that increasing σ_η^2 causes the features in the distribution to occur in the same places but be exaggerated when compared to Figure 6, and the opposite to happen when σ_η^2 is decreased and the features lost to the general noise of the model. Finally the parameter ϕ affects the impact of the value of h_{t-1} on the next variable h_t, with $h_t \propto \phi h_{t-1}$. With a smaller value of ϕ we would expect features of the time-series to be lost as each variable is less affected by the previous, and therefore each new value more randomly determined. This can be seen from the plots in the second row of Figure 7, where the features of the time-series are lost when ϕ is decreased. Furthermore the opposite occurs when ϕ is increased very close to 1 with much of the random noise decreased compared to Figure 6, and the features that are present are enlarged both in amplitude and duration. In section 7.5 we also use these same SV models produced for Figure 7 to see how our HMC algorithm handles a variety of SV input parameters.

7.2 Building and Tuning the HMC

We now proceed with building the algorithm laid out in section 6.4. In order to tune our implementation we apply it to the full (length 5000) time-series generated using $(\mu, \phi, \sigma_\eta^2) = (-1, 0.97, 0.05)$, as displayed in Figure 6. The initial guesses are set to $h_t = -0.6$ (for the whole array), $\mu = 0$, ϕ as a uniformly distributed random number $\in [-1, 1]$, and then using these values the initial σ_η^2 is drawn from equation (20). We require the algorithm to be as efficient as possible, and during

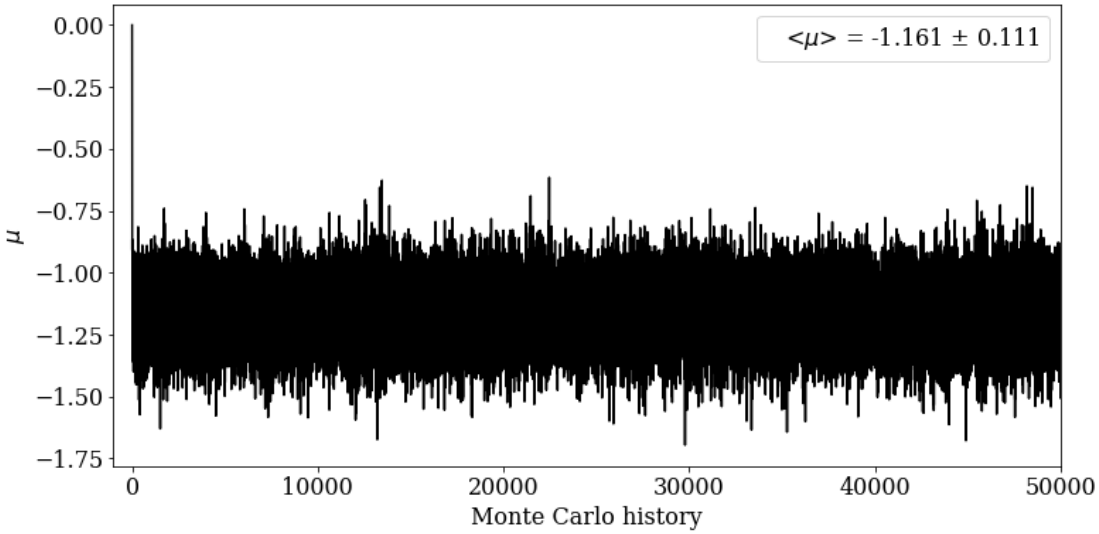

Figure 8: Plot showing parameter μ as the algorithm progresses.

step 9 of the algorithm loop the ideal P^H_{metro} acceptance rate for good performance is around 80-90% [27]. This is a trade-off between autocorrelation time (due to the discretisation size in the leapfrog and discussed further in section 7.3) and computing cost (at a fixed trajectory length). Here we achieved an acceptance rate of 90.4% by tuning the leapfrog step size to $n = 50$ (see section 6.3). We then run the HMC for $60,000$ iterations, discarding the first $10,000$ as the thermalisation or burn-in process. Figure 8 shows the Monte Carlo history of μ, and we have also extracted[9] $\langle\mu\rangle = -1.161 \pm 0.111$ where the error is one standard deviation on the array. This value is within two standard deviations of the value used to generate the artificial data presumably due to the noise present in Figure 8, and so we determine the HMC has extracted this parameter to an acceptable degree of accuracy. Figures 9 and 10 display ϕ and σ^2_η respectively, and we have also

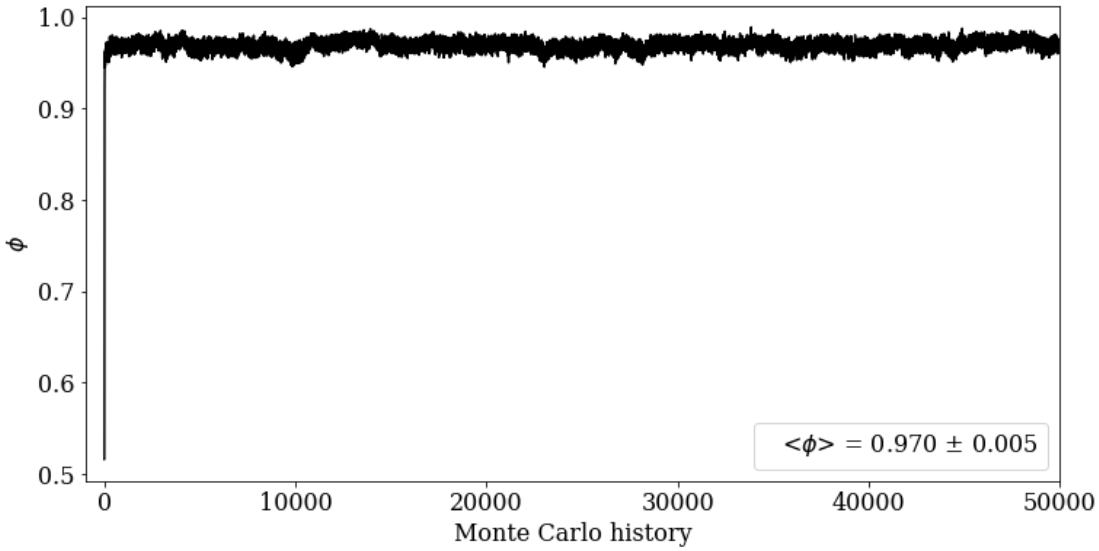

Figure 9: Plot showing parameter ϕ as the algorithm progresses.

extracted[9] $\langle\phi\rangle = 0.970 \pm 0.005$ and $\langle\sigma^2_\eta\rangle = 0.052 \pm 0.007$. These values are both within a standard deviation of their SV model counterparts, and so we determine the HMC has correctly extracted these parameters too. Figure 11 shows the Monte Carlo history of h_{100}, with this specific volatility variable being the one chosen by Takaishi. We also choose h_{100} as our representative volatility variable in section 7.4 using the same reasoning (Figure 12), which allows for a direct comparison. We observe that the amplitude and overall shape of the path taken in Figure 11 is equivalent to the similar plot on page 9 of Takaishi's paper [7]. This initial run of the HMC with $60,000$ iterations (on an SV time-series of length 5000), took six and a half minutes to complete. This is a reasonable amount of time for a

[9]We also ignore the first $10,000$ values when calculating the error and standard deviation.

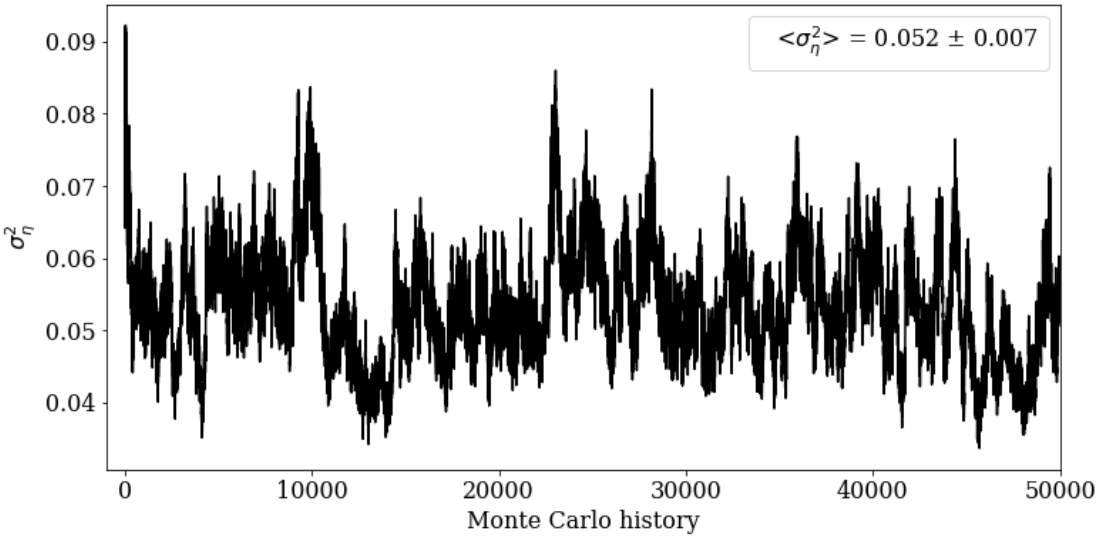

Figure 10: Plot showing parameter σ_η^2 as the algorithm progresses.

single run, and our HMC has correctly extracted the parameters used in the SV data generation to within acceptable error margins. Therefore we conclude that our tuning has been successful, and choose $60,000$ iterations as the standard run length of our HMC (for sections 7.4, 7.5 and 8). The next step is to establish a quantitative analysis of our algorithm via autocorrelation in section 7.3, before proceeding with reproducing more of the results in Takaishi's paper (section 7.4).

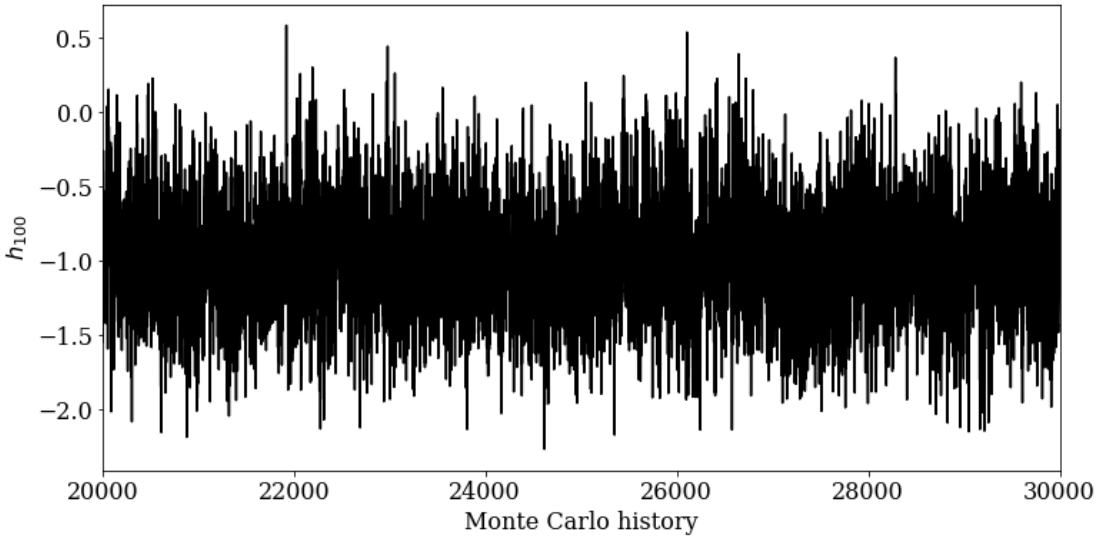

Figure 11: Plot showing volatility variable h_{100} as the algorithm progresses, in the window from $20,000$ to $30,000$ of Monte Carlo time.

7.3 Autocorrelation

Autocorrelation refers to the degree of similarity between a time-series, and a lagged version of itself over successive time intervals [28]. For this project it is important that both our parameters $\theta = (\mu, \phi, \sigma_\eta^2)$ and the volatility variables h_t thoroughly explore the Markov chain's phase space (section 4.1) [29, 30]. To ensure this happens we require each iteration of θ and h_t to decorrelate as quickly as possible from their previous respective values as the HMC progresses. We define our autocorrelation function $ACF(t)$ for a specific time interval as [7, 31, 32]:

$$ACF(t) = \frac{\frac{1}{n}\sum_{i=1}^{n}[X_i - \langle X\rangle][X_{i+t} - \langle X\rangle]}{\sigma_X^2} \tag{40}$$

where n is the total length of the array X, and $\langle X\rangle$ and σ_X^2 are the mean and variance of X respectively. To quantify the speed at which a parameter or variable decorrelates, we also define the autocorrelation time τ as[10] [7, 33, 34]:

$$\tau = \frac{1}{2} + \sum_{t=1}^{\infty} ACF(t) \tag{41}$$

This is effectively the time it takes for the subject in question to become completely decorrelated from previous iterations. For correlated data the actual variance of the mean is a factor of 2τ larger than the value obtained naively using $\sigma_X^2 = \langle X^2\rangle - \langle X\rangle^2$, as is true for uncorrelated data [35, 36]. Therefore to obtain the true error on the parameters extracted by our HMC we will use[10]:

$$\sigma_{true} = \sigma_{naive}\sqrt{2\tau} \tag{42}$$

It is also important to determine τ for each parameter to ensure that our HMC runs for long enough as to appropriately sample any structures found in their fluctuations. We suggest that our HMC needs to run for roughly 100 times longer than the largest parameter autocorrelation time [37] (so when running the HMC for $60,000$ iterations such as in section 7.2, we require τ for each parameter to be no larger than around 600). It is worth mentioning that it is possible there will be other unseen observables within our data with much larger values of τ. While these may be of significance when applying a HMC to a more complicated system [25], for the scale of this project and the SV model we are applying it to, we ignore this possibility.

7.3.1 Jackknife Resampling

When implementing equation (41) it is impractical to literally sum t to ∞ (or in this case the full number of HMC iterations). Once the parameters have fully

[10] The full derivations of equations (41) and (42) are given in appendix A.9.

decorrelated, contributions from $ACF(t)$ at larger values of t will be extremely small, and so it becomes inefficient to continue the computations. Therefore we choose to end the summation once t becomes large enough that the lower limit of the error bar for a value of $ACF(t)$ is less than zero (for example this occurs between $t = 25$ and 75 for the volatility variables in Figure 12, remembering that the y-axis has a logarithmic scale). At this point we assume any subsequent correlations in the data are due to random fluctuations in the HMC, and can be ignored. To calculate the error on $ACF(t)$, we use jackknife resampling [38–40]. This is a method of variance estimation which works by systematically leaving out each measurement from a set of data (of length n), calculating an estimate for each sub-sample (each of length $n-1$), and then taking the mean value of these estimates. By now writing equation (40) as $ACF(t) = \Gamma(t)/\sigma_X^2$, we define our unnormalised autocorrelation function as:

$$\Gamma(t) = \frac{1}{n}\sum_{i=1}^{n} \Gamma_t^i \tag{43}$$

where $\Gamma_t^i = [X_i - \langle X\rangle][X_{i+t} - \langle X\rangle]$ is the i-th $ACF(t)$ bin for a specific time interval t from equation (40). Using the same notation we then define a jackknife bin:

$$\Gamma^k(t) = \frac{1}{n-1}\sum_{i=1, i\neq k}^{n} \Gamma_t^i \tag{44}$$

Finally then the variance of $\Gamma(t)$ is given by:

$$\sigma^2_{\Gamma(t)} = \frac{n-1}{n}\sum_{k=1}^{n}(\Gamma^k(t) - \Gamma(t))^2 \tag{45}$$

We obtain the standard deviation of $\Gamma(t)$ by taking the square root of equation (45), and obtain the error on $ACF(t)$ as $\sigma_{\Gamma(t)}/\sigma_X^2$ (where we have also divided by the variance σ_X^2 as we did in equation (40), to normalise our results).

7.4 Reproducing Takaishi's Results

Before reproducing the bulk of Takaishi's measurements, we first verify that we can take only one volatility variable as representative, as he does on page 10 of his paper [7]. Figure 12 shows the autocorrelation function $ACF(t)$ for volatility variables h_{50}, h_{100}, h_{500} and h_{1000} from the HMC, with the errors calculated using equation (45). We observe that all four functions have similar correlation behaviour, which implies that this is the case for all the volatility variables. Therefore for subsequent analysis we focus on only one variable h_{100}, as it is representative of all h_t. We

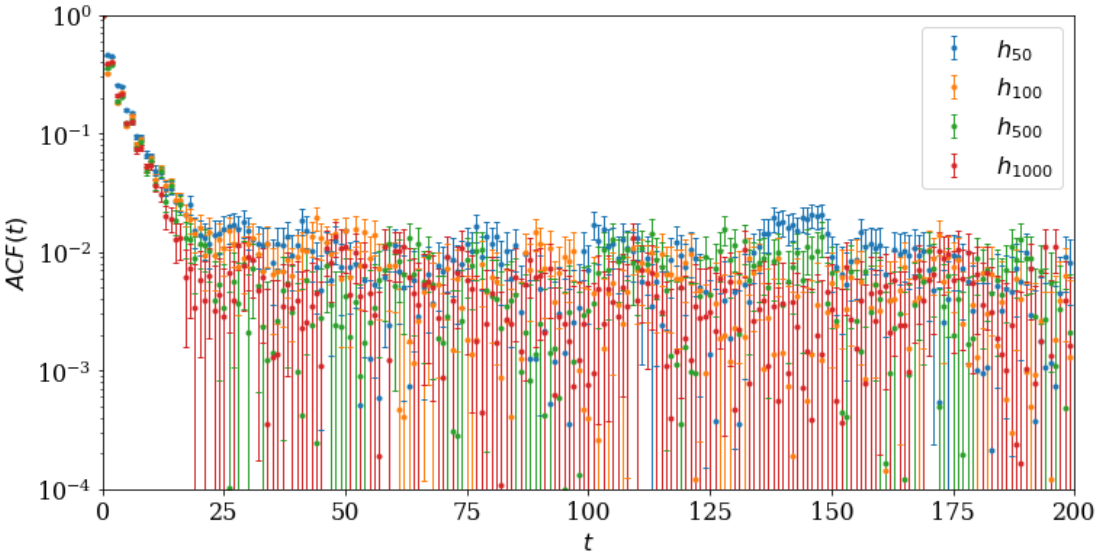

Figure 12: Autocorrelation function for several volatility variables from the HMC in section 7.2, with errors calculated via jackknife resampling using equation (45).

now proceed to generate an SV model of length 5000 using parameters $(\mu, \phi, \sigma_\eta^2) = (-1, 0.97, 0.05)$, and then prepare 3 data sets using the first $N = 1000$ elements, the first $N = 2000$ and then all $N = 5000$. We then run the HMC on each of these data sets for $60,000$ iterations, and discard the first $10,000$ as before. These parameters and data sets match those used by Takaishi, although he chooses to run the HMC for $200,000$ iterations. We feel this length of HMC is unnecessary to determine the parameters to a reasonable degree of accuracy, based on our preliminary testing in section 7.2. Takaishi also compares the performance of his HMC to that of a more basic Metropolis algorithm[11], and determines that the HMC algorithm samples μ and h_t more effectively (and ϕ and σ_η^2 equivalently) to the Metropolis. Therefore once we have corroborated his results we choose to move onto testing our HMC with other parameters (in section 7.5), instead of spending more time running the HMC for longer or comparing it to simpler algorithms.

In his results table (on page 11 of [7]) Takaishi also includes 'statistical error' values, on both the parameter autocorrelation times and the extracted values of the parameters themselves. He mentions that these errors are calculated using the jackknife method, but does not go into any more detail about how exactly

[11]Takaishi's Metropolis algorithm randomly updates the volatility variables using $h_t^{new} = h_t^{old} + \delta(r - 0.5)$, where r is a uniform random number $\in [0, 1]$ and δ is a parameter to tune the acceptance. The new proposal h_t^{new} is then accepted with probability $P_{metro} = \min\left[1, P(h_t^{new})/P(h_t^{old})\right]$, where $P(h_t)$ is given by equation (31) [7].

this technique has been applied. It would seem logical that the error on the autocorrelation time is obtained through some combination of the errors on values of $ACF(t)$ from the sum in equation (41). How Takaishi has chosen to do this however (and for how long he continues the summation for τ as we discuss in section 7.3.1), we are unaware. Therefore we choose to obtain our error on 2τ in Table 1 by summing the errors on each $ACF(t)$ used in equation (41), adding 0.5 and then multiplying by 2. For Takaishi's statistical error on the parameters themselves we were also unsure about his calculation method. After spending a considerable amount of time manipulating the data in his results table we believe he determines this error using equation (42), but then also divides by the square root of the total number of HMC iterations (or uses exactly equation (52) in appendix A.9). As a result these error values are heavily dominated by the length of the HMC, and reduce as the total iterations are increased. This would make sense as the HMC should be determining the parameters more and more accurately, as it is run for longer. However as discussed in section 2.1.1, the SV models we are using do not themselves represent the input parameters to an extremely high degree of accuracy. Therefore it seems inefficient to run the HMC for longer in order to minimise this error, as it may actually be extracting slightly the wrong parameters anyway. For this reason we choose to not calculate this error for our data, and consider the current length of our HMC sufficient for our project.

The results from our HMC (and related error values) for the three data sets are shown in Table 1. Note that the results for the $N = 5000$ data set are the same ones as given in section 7.2. For all three data sets we observe that the HMC has correctly extracted the parameters used in the SV data generation, to within one standard deviation for ϕ and σ_η^2, and two for μ. However as the length of the data set increases, the actual sizes of these standard deviations decrease for all three parameters. This matches what we would expect from section 2.1.1, where we determined that the error in the SV model itself reduces $\propto 1/\sqrt{N}$ (where N is the SV length). Conversely there appear to be no significant differences between the autocorrelation times for each data set. We find that the autocorrelation times for μ and h_t are small as desired, whereas they are large for ϕ and σ_η^2. These observations are consistent with those made by Takaishi (on page 11 of [7]) using data sets of the same length, corroborating his findings. The only major discrepancy comes from the size of the errors on Takaishi's autocorrelation times compared to ours, but as discussed above we are unaware how these have been obtained and therefore are likely to not be using the same method (these errors only give an estimate of the true autocorrelation time and are not used in further calculations, so this should not be greatly significant). Due to the similar autocorrelation times between data sets we now decide to use SV models of length $N = 2000$ as standard

(in sections 7.5 and 8), and consider the slight increase in error acceptable for the gain in processing speed.

		μ	ϕ	σ_η^2	$h_{100}[\Delta H\%]$
	Input	-1	0.97	0.05	
$N = 1000$	HMC	-1.384	0.964	0.048	[95.2%]
	σ	0.234	0.014	0.018	
	2τ	1.5(1.1)	513.2(30.0)	894.4(32.4)	4.9(1.4)
	σ_{true}	0.283	0.322	0.534	
$N = 2000$	HMC	-1.229	0.961	0.061	[94.7%]
	σ	0.153	0.010	0.014	
	2τ	1.6(1.1)	433.9(15.9)	784.3(16.2)	4.1(1.1)
	σ_{true}	0.191	0.209	0.400	
$N = 5000$	HMC	-1.161	0.970	0.052	[90.4%]
	σ	0.111	0.005	0.007	
	2τ	1.4(1.1)	423.3(20.7)	862.1(21.7)	5.8(1.5)
	σ_{true}	0.131	0.104	0.214	

Table 1: Parameters extracted by the HMC for various lengths of the same SV model generated using $(\mu, \phi, \sigma_\eta^2) = (-1, 0.97, 0.05)$. σ is one standard deviation on the parameters, τ is the autocorrelation time calculated using equation (41), and σ_{true} is the true parameter error using equation (42). For 2τ the values in parentheses represent the error, calculated as explained above. The ΔH acceptances for each HMC are shown in square brackets to the right of the extracted parameters.

7.5 Further Testing the HMC

We now move onto testing our HMC with different parameters, in order to determine its stability and find any shortcomings of our implementation. We run the HMC on various SV models generated with different input parameters to the previously used $(\mu, \phi, \sigma_\eta^2) = (-1, 0.97, 0.05)$. We choose to change only one parameter from these values each time to make for a clearer analysis, with the results displayed in Table 2. From preliminary runs we discovered that our HMC fails if we attempt to use it on an SV model generated with either too large a value of μ, too small a value of σ_η^2, or a negative value of ϕ. We found that this was due to a memory overflow error arising when calculating the exponent in equation (20), which would sometimes become too large for the system we were using to process. As a result some of the input parameters for the various SV models used here were altered to avoid this occurring, and the same alterations were also made in section 7.1 (Figure 7) for consistency. From the first two rows of Table 2 we see that the HMC performs equivalently for a different value of μ, compared to when using the

	μ	ϕ	σ_η^2	$h_{100}[\Delta H\%]$
Input	-5	0.97	0.05	
HMC	-5.230	0.962	0.060	[94.5%]
σ	0.153	0.009	0.013	
2τ	1.5(1.1)	350.4(14.9)	713.1(15.5)	4.1(1.1)
σ_{true}	0.187	0.177	0.344	
Input	0.0	0.97	0.05	
HMC	-0.228	0.961	0.061	[94.6%]
σ	0.153	0.010	0.014	
2τ	1.5(1.1)	386.4(15.5)	707.3(16.1)	4.1(1.1)
σ_{true}	0.190	0.195	0.373	
Input	-1	0.5	0.05	
HMC	-1.036	0.052	0.117	[98.4%]
σ	0.038	0.289	0.050	
2τ	662.6(29.3)	6527.6(82.8)	4123.3(55.0)	1.0(1.0)
σ_{true}	0.982	23.324	3.229	
Input	-1	0.997	0.05	
HMC	-2.559	0.996	0.055	[94.2%]
σ	2.322	0.002	0.010	
2τ	1.2(1.1)	58.5(6.5)	402.5(12.5)	4.7(1.2)
σ_{true}	2.539	0.017	0.205	
Input	-1	0.97	0.01	
HMC	-1.086	0.957	0.013	[72.3%]
σ	0.081	0.027	0.010	
2τ	39.1(8.8)	3681.5(63.8)	3957.2(61.6)	6.7(1.5)
σ_{true}	0.504	1.640	0.652	
Input	-1	0.97	0.2	
HMC	-1.490	0.963	0.229	[98.6%]
σ	0.304	0.008	0.032	
2τ	1.0(1.0)	96.7(6.7)	272.5(9.6)	2.5(1.0)
σ_{true}	0.311	0.075	0.528	

Table 2: Parameters extracted by the HMC for length $N = 2000$ SV models, generated using various input parameters. σ is one standard deviation on the parameters, τ is the autocorrelation time calculated using equation (41), and σ_{true} is the true parameter error using equation (42). For 2τ the values in parentheses represent the error, calculated as explained in section 7.4. The ΔH acceptances for each HMC are shown in square brackets to the right of the extracted parameters.

original parameters (the second row of Table 1). From the third row of Table 2 we see that when using a smaller ϕ the HMC just extracts the correct parameters within two standard deviations, albeit with huge autocorrelation times. However this is only the case because the errors on ϕ and σ_η^2 are so large, as the extracted values are quite different. Whereas in the fourth row with $\phi \approx 1$ the parameters are all within one standard deviation (although again the error on μ is very large), and the autocorrelation time is small when compared to the original parameters. Then from the fifth row of Table 2 we see that when using a smaller σ_η^2 the HMC extracts the correct parameters within two standard deviations, but similarly to ϕ with much larger autocorrelation times. And finally from the last row using a larger σ_η^2 the HMC again extracts the correct parameters within two standard deviations, and also with smaller autocorrelation times for all the parameters. In general then we observe that the HMC works correctly when applied to different parameters as we hoped (within the limits of our implementation as discussed before), although sometimes this results in much larger autocorrelation times. As discussed in section 7.3 this is undesirable, and so we choose to continue using $(\mu, \phi, \sigma_\eta^2) = (-1, 0.97, 0.05)$ when improving our HMC in section 8.

8 Improving the Algorithm

From our results in section 7 we determine that we now have a working HMC implementation and have therefore fulfilled our primary goal, and so move onto improving our simulation algorithm. There were several existing ideas that were of interest in this matter, for example [41, 42]. However due to time constraints on this project, only two modifications based on the work in this paper [43] were implemented for our HMC.

8.1 Momentum Randomisation

The first smaller modification made was to be able to reduce the randomness of each subsequent set of momentum coordinates, during each iteration of the HMC. This was done by modifying step 4 of the algorithm as laid out in section 6.4. Instead of generating the momentum vector completely randomly (using a normal distribution) prior to the leapfrog, we now use [43]:

$$p_t^{old} = p_t^{n-1}\sqrt{1-\beta} + N(0,1)\sqrt{\beta} \tag{46}$$

where p_t^{n-1} is the final momentum vector from the previous iteration of the HMC (note that $p_t^{new} = p_t^{old}$ must now also be set in step 9(b) of the algorithm from section 6.4). $N(0,1)$ is a standard normal distribution of the same length as p_t, and $\beta \in [0,1]$ is a new HMC parameter used to control the level of randomness.

Usefully we can still reproduce our previous results by setting $\beta = 1$, for which equation (46) functions identically to our standard HMC and the momentum is completely random. By decreasing β we partly retain the momentum vector after each successful iteration of the HMC, which should be beneficial in achieving further successful transitions and hopefully causing faster decorrelation of the parameters. To determine whether this modification has improved our algorithm we take the same SV model from section 7.4 ($N = 2000$) with $(\mu, \phi, \sigma_\eta^2) = (-1, 0.97, 0.05)$, and apply the HMC several times with varying values of β. Each time we use the same initial seed (discussed in section 7.1), and run the HMC for $60,000$ leapfrog iterations (discarding the first $10,000$). By keeping the number of leapfrogs consistent between runs each HMC takes roughly the same amount of computational time to complete, and therefore the performance increase we are looking for are reductions in the parameter autocorrelation times. The results from these runs are shown in Table 3, and we can immediately see that when $\beta = 1$ the HMC has functioned identically to section 7.4 (Table 1, row $N = 2000$). From the other

		μ	ϕ	σ_η^2	$h_{100}[\Delta H\%]$
	Input	-1	0.97	0.05	
$\beta = 1.0$	HMC	-1.229	0.961	0.061	[94.7%]
	σ	0.153	0.010	0.014	
	2τ	1.6(1.1)	433.9(15.9)	784.3(16.2)	4.1(1.1)
	σ_{true}	0.191	0.209	0.400	
$\beta = 0.9$	HMC	-1.228	0.962	0.060	[94.5%]
	σ	0.154	0.010	0.014	
	2τ	1.3(1.1)	252.9(9.4)	442.7(10.3)	2.3(1.0)
	σ_{true}	0.178	0.161	0.303	
$\beta = 0.8$	HMC	-1.226	0.963	0.057	[94.2%]
	σ	0.157	0.010	0.015	
	2τ	1.3(1.0)	387.6(15.4)	654.5(17.7)	1.9(1.0)
	σ_{true}	0.176	0.202	0.383	
$\beta = 0.7$	HMC	-1.225	0.964	0.056	[94.2%]
	σ	0.158	0.010	0.014	
	2τ	1.2(1.0)	244.2(8.4)	427.1(9.6)	1.8(1.0)
	σ_{true}	0.176	0.154	0.290	
$\beta = 0.6$	HMC	-1.226	0.963	0.058	[94.4%]
	σ	0.157	0.010	0.015	
	2τ	1.2(1.0)	243.5(9.4)	422.2(10.4)	1.7(1.0)
	σ_{true}	0.173	0.158	0.300	

Table 3: Continued overleaf.

Continued.		μ	ϕ	σ_η^2	$h_{100}[\Delta H\%]$
	Input	-1	0.97	0.05	
$\beta = 0.5$	HMC	-1.225	0.963	0.057	[94.2%]
	σ	0.158	0.010	0.015	
	2τ	1.2(1.0)	300.7(10.3)	666.1(18.6)	1.6 (1.0)
	σ_{true}	0.174	0.178	0.389	
$\beta = 0.4$	HMC	-1.224	0.964	0.055	[94.1%]
	σ	0.159	0.010	0.014	
	2τ	1.2(1.0)	278.1(8.8)	471.7(9.3)	1.6(1.0)
	σ_{true}	0.175	0.164	0.303	
$\beta = 0.3$	HMC	-1.227	0.962	0.059	[94.4%]
	σ	0.155	0.010	0.014	
	2τ	1.2(1.0)	252.1(6.8)	433.4(7.1)	1.6(1.0)
	σ_{true}	0.171	0.160	0.301	
$\beta = 0.2$	HMC	-1.224	0.964	0.056	[94.2%]
	σ	0.159	0.010	0.015	
	2τ	1.2(1.0)	498.1(17.0)	829.3(17.7)	1.6(1.0)
	σ_{true}	0.175	0.229	0.437	
$\beta = 0.1$	HMC	-1.226	0.963	0.057	[94.1%]
	σ	0.157	0.010	0.016	
	2τ	1.2(1.0)	1144.2(39.7)	1853.7(39.9)	1.6(1.0)
	σ_{true}	0.174	0.352	0.671	

Table 3: Parameters extracted by the HMC using different values of β, on the same SV model of length $N = 2000$. σ is one standard deviation on the parameters, τ is the autocorrelation time calculated using equation (41), and σ_{true} is the true parameter error using equation (42). For 2τ the values in parentheses represent the error, calculated as explained in section 7.4. The ΔH acceptances for each HMC are shown in square brackets to the right of the extracted parameters.

nine rows we can also see that our improved algorithm has correctly extracted the input parameters when using different β, with very similar values and standard deviations. Additionally we observe that setting $\beta < 1$ causes a small decrease in the autocorrelation times of μ and h_{100}, although these values were already very small for $\beta = 1$ and therefore decreasing β actually has little effect on them. This is also supported by Figure 13, and so we consider the correlation behaviour of μ and h_t consistent when varying β. The autocorrelation times of ϕ and σ_η^2 however decrease much more significantly when setting $0.3 \leq \beta \leq 0.9$. This is further illustrated by Figures 14 and 15, where we observe faster decorrelation (a steeper autocorrelation function gradient) for both parameters when β is set in this range. From the figures it also appears that both ϕ and σ_η^2 decorrelate most

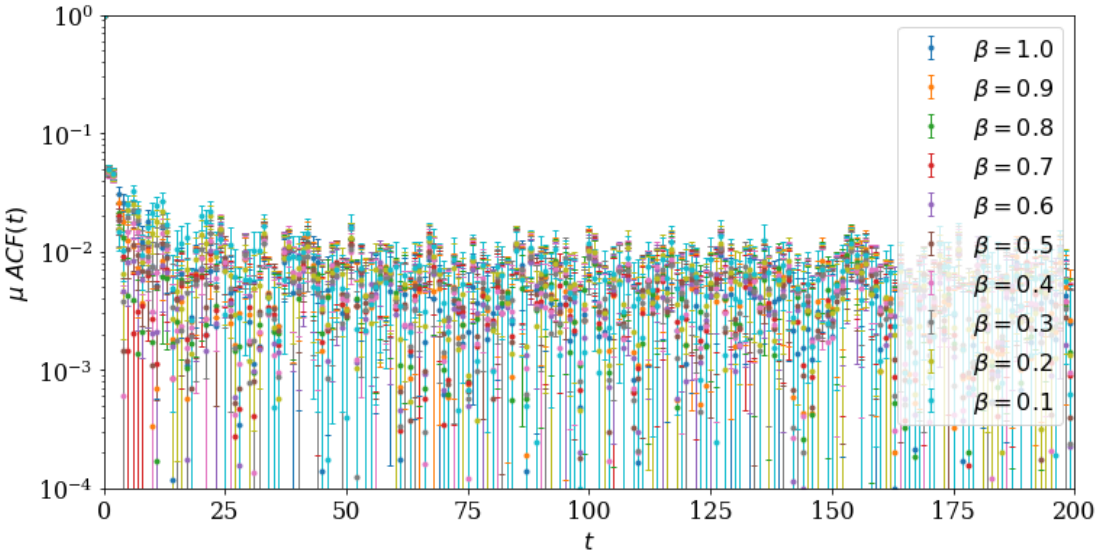

Figure 13: Autocorrelation functions for μ when using different values of β.

rapidly when β is set to 0.9, 0.7, 0.6 or 0.3, which matches the values of β with the lowest autocorrelation times from Table 3. We therefore determine that this improvement has been successful, as we are able to achieve lower parameter autocorrelation times by setting $0.3 \leq \beta \leq 0.9$ (while using our improved HMC which incorporates equation (46) into the algorithm).

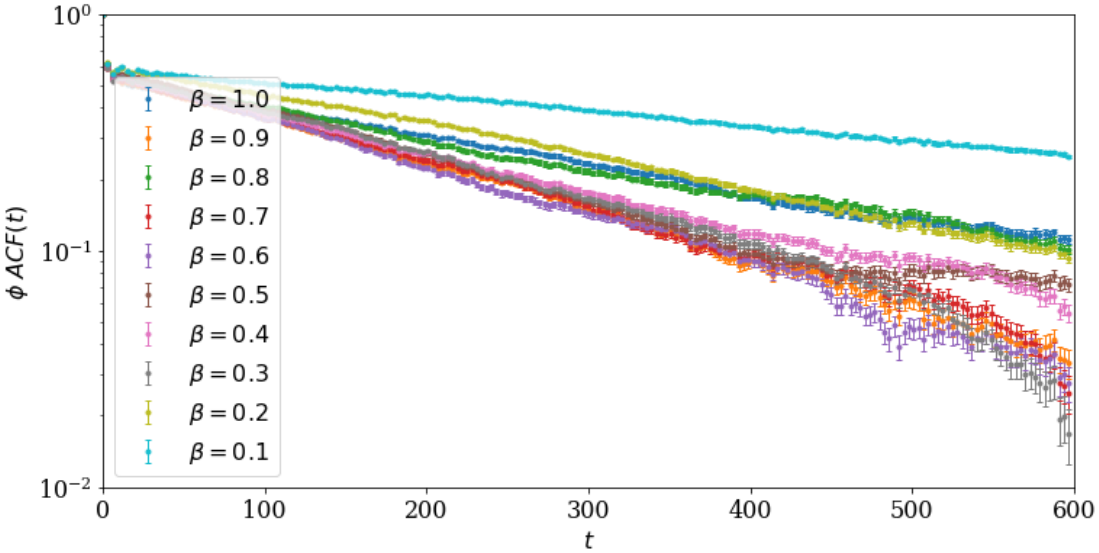

Figure 14: Autocorrelation functions for ϕ when using different values of β.

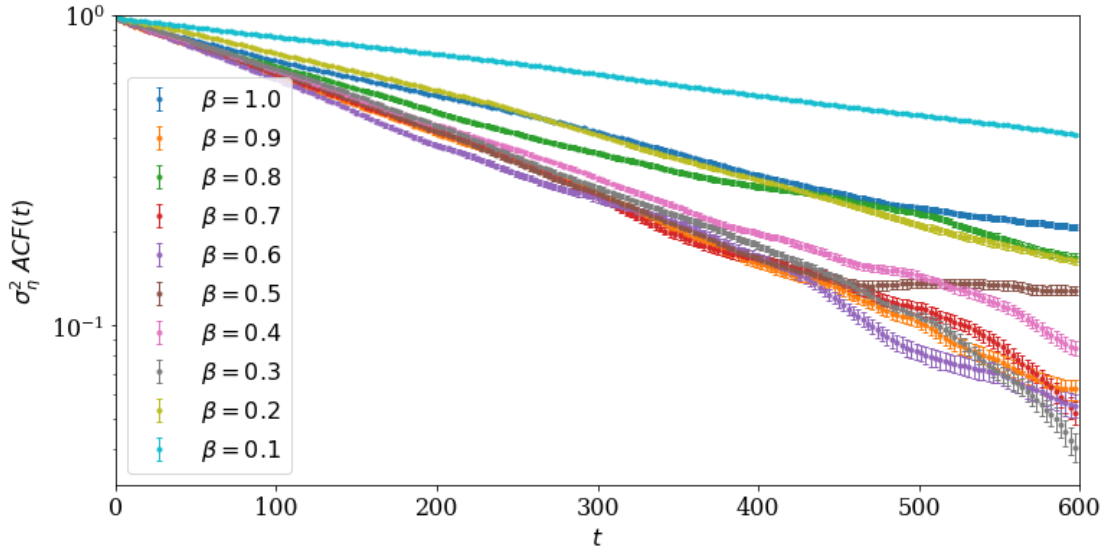

Figure 15: Autocorrelation functions for σ_η^2 when using different values of β.

8.2 Look Ahead HMC

The second larger modification made was to attempt further leapfrog integrations in the case of a transition rejection, instead of immediately reverting back to the previous state (as occurs during step 9(b) of the algorithm in section 6.4). This is potentially beneficial as we minimise the number of total rejections (which are essentially wasted computations), and therefore more thoroughly explore the Markov chain's phase space. Because of this behaviour we refer to the resulting algorithm as a Look Ahead Hybrid Monte Carlo (LAHMC), which importantly no longer satisfies the detailed balance relation in equation (17) like the standard HMC. In order to not violate the essential Markov transition relation in equation (16) however, we must also calculate the reverse transition probabilities and use these to modify the acceptance probabilities for each new leapfrog accordingly. We define the forward transitional probabilities between states b and a as [43, 44]:

$$\pi^a_{forward} = \min\left[(1 - \sum_{b<a} \pi^b_{forward}),\quad \exp(-H_a + H_b)(1 - \sum_{b<a} \pi^b_{backward})\right] \tag{47}$$

where H is the energy of the respective state determined using equation (33), and the backwards transitional probabilities are also calculated using equation (47) but with the probabilities reversed[12]. Equation (47) makes the probability

[12]Note that for any specific look ahead state state $\pi_{backward}$ must be calculated first, because $\pi_{forward}$ requires $\pi_{backward}$ for that state, whereas $\pi_{backward}$ is calculated using the probabilities related to the previous look ahead states).

of a forward transition as large as possible, in order to minimise the number of complete rejections as mentioned before. However it still necessarily restricts the total transition probability out of any state to 1, and ensures that the forward transition rate does not exceed that for reverse transitions. We limit the maximum number of look ahead steps per algorithm iteration using a new LAHMC parameter $k \in \mathbb{Z}^+$, which we will vary shortly in order to minimise the autocorrelation times. Similarly to section 8.1, we should also find that setting $k = 1$ causes the LAHMC to function equivalently to our HMC with its Metropolis accept/reject step. We determine how many look ahead steps to make using equation (47) and [43]:

$$x_n \to \begin{cases} x_{n+1} & \text{(1 look ahead step) with probability } \pi^1_{forward} \\ x_{n+2} & \text{(2 look ahead steps) with probability } \pi^2_{forward} \\ & \vdots \\ x_{n+k} & \text{(k look ahead steps) with probability } \pi^k_{forward} \end{cases} \tag{48}$$

where x_n is the current state, and x_{n+1}, x_{n+2} and x_{n+k} are the states after 1, 2 and k look ahead steps respectively. We calculate the transition probability after each leapfrog computation to maximise efficiency, and then only continue with the next leapfrog if the current transition is rejected. If no look ahead state has been accepted after k steps, we revert back to the original state (setting the next LAHMC state to the original state). One issue with performing multiple leapfrogs per LAHMC iteration however, is that this will increase the computational time compared to the HMC which only completes one leapfrog each time. To account for this we now consider each look ahead step as one iteration of the LAHMC, meaning that the total number of algorithm iterations will be lower than the HMC but the number of leapfrogs the same. We also have to take this into account when calculating the autocorrelation time, as there are essentially now gaps in the Markov chain where our LAHMC has jumped multiple states. When calculating $ACF(t)$ using equation (41), we therefore ignore bins if either the i-th or $(i+t)$-th element is one of these gaps.

In order to test our LAHMC we apply it with varying values of k, again to the same SV model from section 7.4 ($N = 2000$) with $(\mu, \phi, \sigma_\eta^2) = (-1, 0.97, 0.05)$. Each time we also use the same initial seed, and run the LAHMC for $60,000$ total leapfrog iterations (discarding the first $10,000$ samples). Note that while testing our LAHMC in this section we keep $\beta = 1$, in order to measure the improvement from varying k only. The results from these runs are shown in Table 4, and like in section 8.1 we see that when $k = 1$ the LAHMC has functioned identically to our standard HMC as designed. From the other nine rows we can also see that our LAHMC has correctly extracted the input parameters when using different k, with very similar values and standard deviations. Additionally we observe very similar

autocorrelation times for μ and h_{100} while varying k, and so we consider their correlation behaviour consistent when varying k (supported by Figure 16). The autocorrelation times of ϕ and σ_η^2 however increase slightly when setting $k = 2$,

		μ	ϕ	σ_η^2	h_{100}[LA%]
	Input	-1	0.97	0.05	
$k = 1$	LAHMC	-1.229	0.961	0.061	[94.7%]
	σ	0.153	0.010	0.014	
	2τ	1.6(1.1)	433.9(15.9)	784.3(16.2)	4.1(1.1)
	σ_{true}	0.191	0.209	0.400	
$k = 2$	LAHMC	-1.222	0.965	0.053	[99.5%]
	σ	0.163	0.010	0.015	
	2τ	1.4(1.1)	508.4(14.3)	816.5(15.3)	3.3(1.1)
	σ_{true}	0.192	0.230	0.429	
$k = 3$	LAHMC	-1.221	0.966	0.053	[100.0%]
	σ	0.163	0.009	0.013	
	2τ	1.4(1.1)	375.9(13.4)	689.7(14.8)	3.2(1.1)
	σ_{true}	0.195	0.183	0.346	
$k = 4$	LAHMC	-1.219	0.967	0.050	[100.0%]
	σ	0.164	0.009	0.012	
	2τ	1.5(1.1)	274.5(10.6)	513.0(11.3)	3.2(1.1)
	σ_{true}	0.198	0.152	0.282	
$k = 5$	LAHMC	-1.220	0.966	0.051	[100.0%]
	σ	0.165	0.010	0.014	
	2τ	1.5(1.1)	379.1(11.2)	649.6(11.4)	3.2(1.1)
	σ_{true}	0.202	0.190	0.362	
$k = 6$	LAHMC	-1.220	0.966	0.051	[100.0%]
	σ	0.165	0.010	0.014	
	2τ	1.5(1.1)	376.6(11.2)	646.2(11.4)	3.2(1.1)
	σ_{true}	0.203	0.189	0.361	
$k = 7$	LAHMC	-1.220	0.966	0.051	[100.0%]
	σ	0.165	0.010	0.014	
	2τ	1.5(1.1)	379.1(11.2)	649.6(11.4)	3.2(1.1)
	σ_{true}	0.202	0.190	0.362	
$k = 8$	LAHMC	-1.220	0.967	0.051	[100.0%]
	σ	0.165	0.010	0.014	
	2τ	1.5(1.1)	384.2(11.1)	656.2(11.4)	3.2(1.1)
	σ_{true}	0.201	0.192	0.365	

Table 4: Continued overleaf.

Continued.		μ	ϕ	σ_η^2	h_{100}[LA%]
	Input	-1	0.97	0.05	
$k = 9$	LAHMC	-1.220	0.967	0.051	[100.0%]
	σ	0.165	0.010	0.014	
	2τ	1.5(1.1)	384.2(11.1)	656.2(11.4)	3.2(1.1)
	σ_{true}	0.201	0.192	0.365	
$k = 10$	LAHMC	-1.220	0.967	0.051	[100.0%]
	σ	0.165	0.010	0.014	
	2τ	1.5(1.1)	384.2(11.1)	656.2(11.4)	3.2(1.1)
	σ_{true}	0.201	0.192	0.365	

Table 4: Parameters extracted by the LAHMC using different values of k, on the same $N = 2000$ SV model. σ is one standard deviation on the parameters, τ is the autocorrelation time, and σ_{true} is the true parameter error. For 2τ the values in parentheses represent the error (section 7.4). The look ahead acceptances (LA%) for each LAHMC are in square brackets to the right of the extracted parameters.

but then decrease (compared to $k = 1$) for $k \geq 3$. This behaviour is also shown in Figures 17 and 18, where we observe slower decorrelation for both parameters when $k = 2$ and faster decorrelation when $k \geq 3$. Specifically the parameters decorrelate slightly faster by setting $k = 3$, and then fastest when using $k = 4$. For $k \geq 5$ we observe very similar decorrelation behaviour from both parameters (although slightly slower than when $k = 4$), supported by similarly comparable

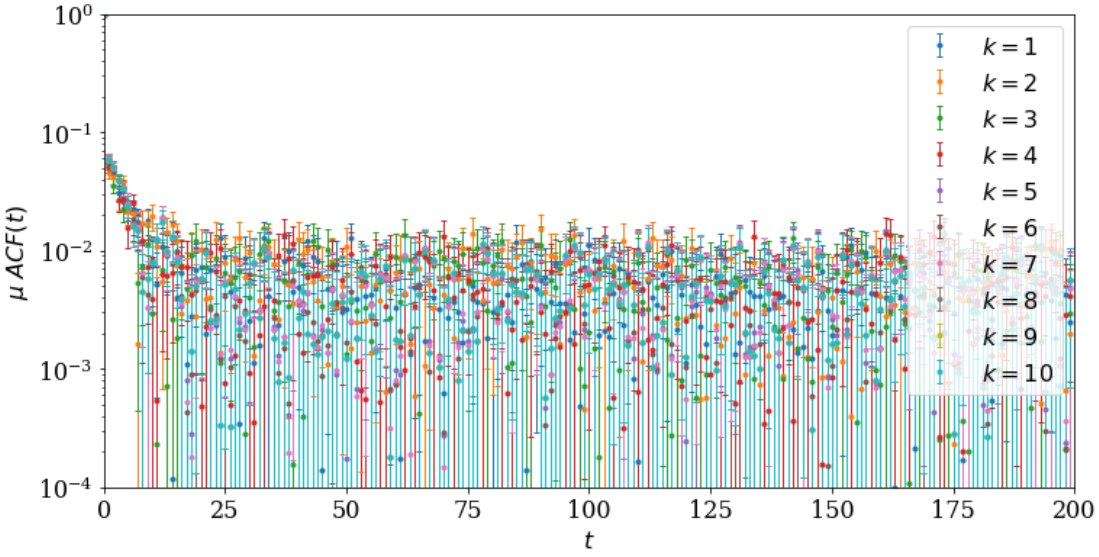

Figure 16: Autocorrelation functions for μ when using different values of k.

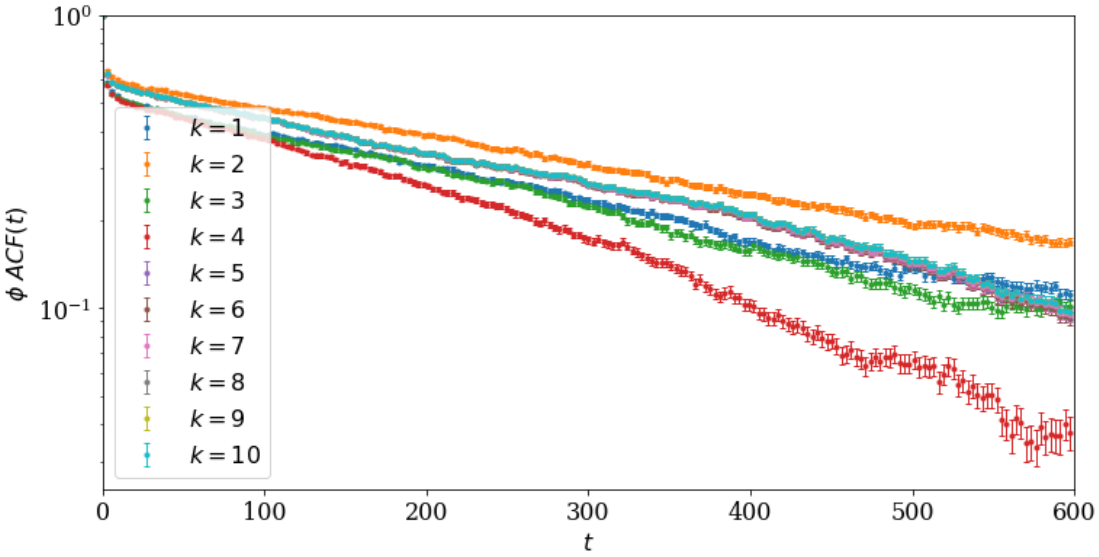

Figure 17: Autocorrelation functions for ϕ when using different values of k.

autocorrelation times in Table 4. This is presumably due to there always being an accepted state before this point (the LAHMC never actually reaching the maximum number of look ahead steps), and therefore the algorithm would perform equivalently even if there was no limit k. We would infer that the reason we gain faster autocorrelation for $k = 4$ is that the few (presumably extremely unfavourable) configurations which reach this maximum are then rejected, whereas

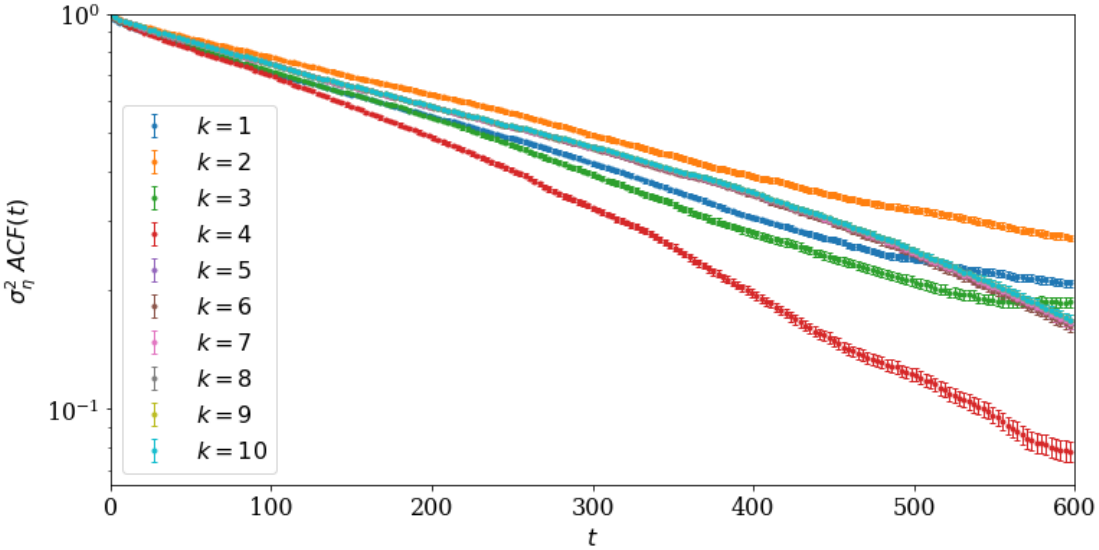

Figure 18: Autocorrelation functions for σ_η^2 when using different values of k.

with larger k eventually the algorithm effectively forces 100% acceptance (which in this case is likely detrimental). We therefore determine this improvement also a success, as we are able to achieve lower parameter autocorrelation times by setting $k \geq 3$ in our LAHMC (with the lowest times obtained by using $k = 4$).

8.3 Minimising the Autocorrelation Time

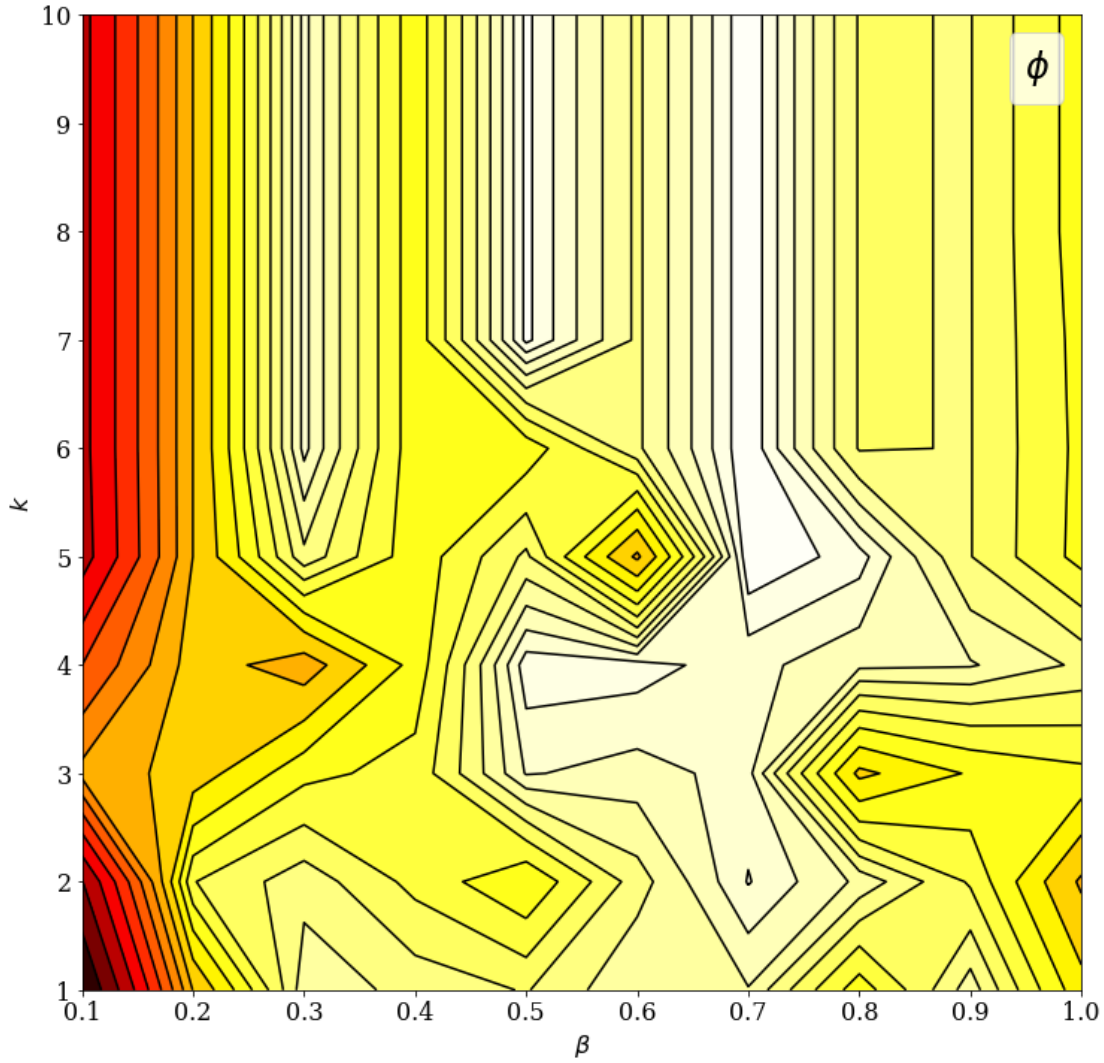

Figure 19: Contour plot (with 20 levels) showing the autocorrelation times for ϕ from 100 runs of the LAHMC, each with varying β (section 8.1) and k (section 8.2). The lighter the colour the smaller the autocorrelation time, and vice versa.

From sections 8.1 and 8.2 we determine that we now have a working LAHMC implementation, which is capable of minimising the parameter autocorrelation times by independently setting either $0.3 \leq \beta \leq 0.9$ or $k \geq 3$. We now finally attempt to combine our improvements by varying β and k simultaneously, in order to determine the ultimate LAHMC settings which maximise the reduction in the parameter autocorrelation times. We again apply out LAHMC to the same ($N = 2000$) SV model from the previous sections, varying β from 1 to 0.1 and for each of these varying k from 1 to 10. For each of the 100 LAHMC runs we use the

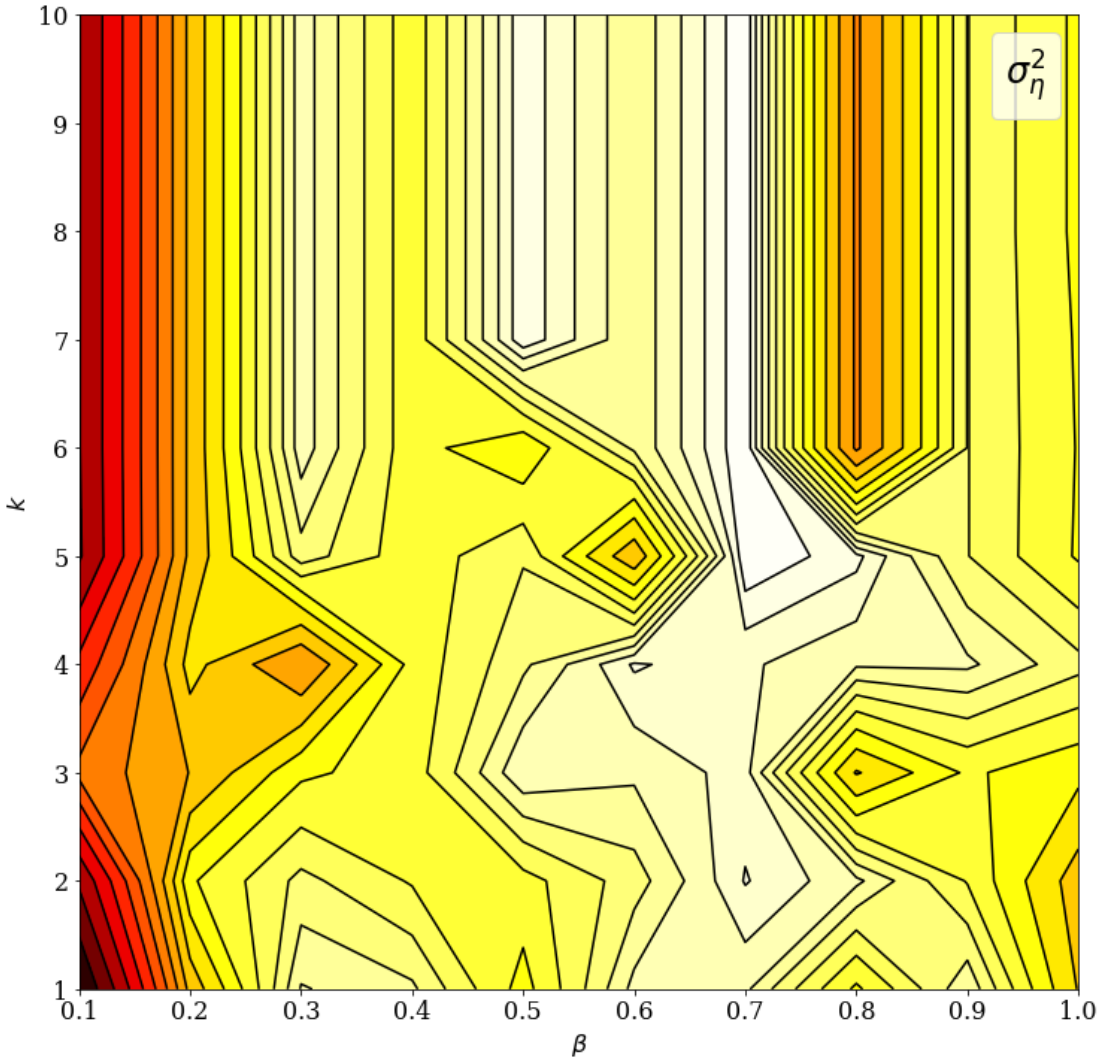

Figure 20: Contour plot (with 20 levels) showing the autocorrelation times for σ_η^2 from 100 runs of the LAHMC, each with varying β (section 8.1) and k (section 8.2). The lighter the colour the smaller the autocorrelation time, and vice versa.

same initial seed, and iterate $60,000$ total leapfrogs (discarding the first $10,000$ samples). This entire process took roughly 7 hours to complete, with the varying autocorrelation times for ϕ and σ_η^2 displayed as contour plots in Figures 19 and 20 respectively. We do not display contour plots for μ or any h_t, as we have already shown in previous sections that these remain very low when varying either β or k (as desired). From Figures 19 and 20 we observe that setting k above a certain value causes the correlation of both parameters to no longer vary (although this value changes for different β), which is consistent with our results in section 8.2. For both ϕ and σ_η^2 we find that the maximum autocorrelation time reduction is obtained by setting $\beta = 0.7$ and $k \geq 5$, with several other combinations giving almost as good reductions (for example $\beta = 0.6$ and $k = 4$, or $\beta = 0.7$ and $k = 2$). The numerical results of this optimal reduction are shown in Table 5, along with the results using $\beta = 1$ and $k = 1$ (effectively from the standard HMC for comparison). We can see that our optimised LAHMC has still extracted the parameters as effectively as the standard HMC, but has significantly reduced the parameter autocorrelation times. For μ we have gained roughly a 20% reduction, but for ϕ, σ_η^2 and h_{100} we have gained roughly a 60% reduction. This is most important for ϕ and σ_η^2 which previously had much larger autocorrelation times, (compared to μ and h_{100}).

		μ	ϕ	σ_η^2	h_{100}[LA%]
	Input	-1	0.97	0.05	
$\beta = 1.0$	LAHMC	-1.229	0.961	0.061	[94.7%]
$k = 1$	σ	0.153	0.010	0.014	
	2τ	1.6(1.1)	433.9(15.9)	784.3(16.2)	4.1(1.1)
	σ_{true}	0.191	0.209	0.400	
$\beta = 0.7$	LAHMC	-1.224	0.965	0.055	[100.0%]
$k = 5$	σ	0.160	0.009	0.013	
	2τ	1.3(1.0)	160.5(5.4)	297.5(5.6)	1.6(1.0)
	σ_{true}	0.180	0.118	0.217	

Table 5: Parameters extracted by the LAHMC using the standard and optimum settings for β and k, on the same $N = 2000$ SV model. σ is one standard deviation, τ is the autocorrelation time, and σ_{true} is the true error. For 2τ the values in parentheses represent the error (section 7.4). The look ahead acceptances (LA%) for each LAHMC are in square brackets to the right of the extracted parameters.

9 Conclusion

In this project we have verified that the implementation of our Stochastic Volatility (SV) model matches theory, and devised methods of drawing the parameters $\theta = (\mu, \phi, \sigma_\eta^2)$ from the likelihood function in equation (7) through the use of Bayesian inference. We have created a Hybrid Monte Carlo (HMC) algorithm with an appropriate leapfrog integrator capable of updating the volatility variables h_t along with θ, applied our HMC to similar artificial time-series as used by Takaishi [7], and corroborated his results. We have also found the limits of our implementation, and confirmed that within these limits our HMC is capable of extracting the SV input parameters to within acceptable error margins. While not directly attempted in this project our successful results suggest that we could apply our HMC to real stock market (returns) data, and extract the corresponding SV model parameters which could then be used to predict future behaviour.

We have made two improvements to our algorithm, the first of which introduced a parameter β which controls the randomness of each new set of momentum coordinates (Momentum Randomisation), and the second which attempts further leapfrog integrations in the event of a rejected configuration (up to a maximum number k), which we refer to as a Look Ahead Hybrid Monte Carlo (LAHMC). Using autocorrelation as a quantitative analysis we compared our improvements to our original implementation, and confirmed that by optimising our LAHMC with $\beta = 0.7$ and $k = 5$ we can improve the decorrelation of ϕ and σ_η^2 by roughly 60% (we also improved this for μ and h_t, although their autocorrelation times (τ) are small regardless). From our optimised LAHMC the longest autocorrelation time (τ) has reduced from roughly 400 to 150 (for σ_η^2), which following our discussion in section 7.3 means that our LAHMC needs to be run for approximately $15,000$ leapfrog iterations (whereas four times that amount are used throughout this paper). While we have not directly sped up our algorithm as originally proposed (indeed while testing our improvements we intentionally keep the computational time consistent), by decreasing the parameter autocorrelation times (τ) our LAHMC should now accurately extract them with fewer iterations in a shorter amount of time.

For further work beyond this paper we would recommend showing that the optimised LAHMC completes faster than the standard HMC, both with our standard parameters $(\mu, \phi, \sigma_\eta^2) = (-1, 0.97, 0.05)$ and other variations (such as those used in section 7.5). We also suggest applying the optimised LAHMC to historical stock market data, to determine if our model can accurately reproduce real world behaviour.

References

[1] Z. Zhao, *Stochastic volatility models with applications in finance*. Phd dissertation, University of Iowa, 2016. DOI:10.17077/etd.tkj2s3ik.

[2] J. Chen, "Derivative." Investopedia, 2018. URL: www.investopedia.com/terms/d/derivative.asp, accessed 4-January-2019.

[3] J. Chen, "Volatility." Investopedia, 2018. URL: www.investopedia.com/terms/v/volatility.asp, accessed 4-January-2019.

[4] W. Kenton, "Black scholes model." Investopedia, 2018. URL: www.investopedia.com/terms/b/blackscholes.asp, accessed 4-January-2019.

[5] W. Kenton, "Stochastic volatility." Investopedia, 2018. URL: www.investopedia.com/terms/s/stochastic-volatility.asp, accessed 4-January-2019.

[6] J. Gatheral and N. N. Taleb, *The Volatility Surface: A Practitioner's Guide*, vol. 357. Wiley Finance, 2006. ISBN:9780470068250.

[7] T. Takaishi, "Bayesian inference of stochastic volatility model by hybrid monte carlo," *Journal of Circuits, Systems, and Computers*, vol. 18, no. 8, pp. 1381–1396, 2009. arXiv:1001.0024.

[8] S. Kim, N. Shephard, and S. Chib, "Stochastic volatility: Likelihood inference and comparison with arch models," *The Review of Economic Studies*, vol. 65, no. 3, pp. 361–393, 1998. DOI:10.1111/1467-937X.00050.

[9] S. Huang Junying and J. Yu, "An efficient method for maximum likelihood estimation of a stochastic volatility model," *Statistics and Its Interface*, vol. 1, no. 2, pp. 289–296, 2008. DOI:10.4310/SII.2008.v1.n2.a6.

[10] A. Gelman, J. B. Carlin, H. S. Stern, D. B. Dunson, A. Vehtari, and D. B. Rubin, *Bayesian Data Analysis (Third Edition)*. Boca Raton: Chapman & Hall/CRC, 2013. ISBN:9781439898208.

[11] J. Joyce, "Bayes' theorem," in *The Stanford Encyclopedia of Philosophy (Winter 2016 Edition)* (E. N. Zalta, ed.), Metaphysics Research Lab, Stanford University, 2016. ISSN:1095-5054.

[12] C. J. Geyer, "Introduction to markov chain monte carlo," in *Handbook of Markov Chain Monte Carlo (First Edition)* (S. Brooks, A. Gelman, G. L. Jones, and X.-L. Meng, eds.), Chapman & Hall/CRC, 2011. ISBN:9781420079418.

[13] A. Doucet, N. de Freitas, and N. Gordon, "An introduction to sequential monte carlo methods," in *Sequential Monte Carlo Methods in Practice*, pp. 3–14, New York: Springer, 2001. DOI:10.1007/978-1-4757-3437-9.

[14] D. P. Kroese and R. Y. Rubinstein, "Monte carlo methods," *WIREs Computational Statistics*, vol. 4, no. 1, pp. 48–58, 2012. DOI:10.1002/wics.194.

[15] P. A. Gagniuc, *Markov Chains: From Theory to Implementation and Experimentation*. Wiley, 2017. ISBN:978-1-119-38755-8.

[16] D. D. Nolte, "The tangled tale of phase space," *Physics Today*, vol. 63, no. 4, pp. 33–38, 2010. DOI:10.1063/1.3397041.

[17] J. R. Norris, *Markov Chains*. Cambridge University Press, 1998. ISBN:978-0521633963.

[18] A. O'Hagan and J. J. Forster, "Markov chain monte carlo," in *Kendall's Advanced Theory of Statistics: Bayesian Inference*, vol. 2B, London: Arnold, 2004. ISBN:978-0470685693.

[19] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, "Equation of state calculations by fast computing machines," *The Journal of Chemical Physics*, vol. 21, pp. 1087–1092, 1953. DOI:10.1063/1.1699114.

[20] W. K. Hastings, "Monte carlo sampling methods using markov chains and their applications," *Biometrika*, vol. 57, no. 1, pp. 97–109, 1970. DOI:10.2307/2334940.

[21] Greenparker, "Sampling from an inverse gamma distribution." Cross Validated, 2016. URL: stats.stackexchange.com/q/224718, accessed 30-October-2018.

[22] E. W. Weisstein, "Normal distribution." MathWorld - A Wolfram Web Resource. URL: mathworld.wolfram.com/NormalDistribution.html, accessed 6-November-2018.

[23] N. Shephard and M. K. Pitt, "Likelihood analysis of non-gaussian measurement time series," *Biometrika*, vol. 84, no. 3, pp. 653–667, 1997.

[24] T. Watanabe and Y. Omori, "A multi-move sampler for estimating non-gaussian time series models: Comments on shephard & pitt (1997)," *Biometrika*, vol. 91, no. 1, pp. 246–248, 2004. ISSN:00063444.

[25] S. Duane, A. D. Kennedy, B. J. Pendleton, and D. Roweth, "Hybrid monte carlo," *Physics Letters B*, vol. 195, no. 2, pp. 216–222, 1987. DOI:10.1016/0370-2693(87)91197-X.

[26] S. K. Park and K. W. Miller, "Random number generators: Good ones are hard to find," *Communications of the ACM*, vol. 31, no. 10, pp. 1192–1201, 1988. DOI:10.1145/63039.63042.

[27] T. Takaishi, "Choice of integrator in the hybrid monte carlo algorithm," *Computer Physics Communications*, vol. 133, no. 1, pp. 6–17, 2000. DOI:10.1016/S0010-4655(00)00161-2.

[28] W. Kenton, "Autocorrelation." Investopedia, 2018. URL: www.investopedia.com/terms/a/autocorrelation.asp, accessed 9-January-2019.

[29] A. D. Kennedy and B. Pendleton, "Acceptances and autocorrelations in hybrid monte carlo," *Nuclear Physics B - Proceedings Supplements*, vol. 20, pp. 118–121, 1991. DOI:10.1016/0920-5632(91)90893-J.

[30] A. Beskos, N. S. Pillai, G. O. Roberts, J. M. Sanz-Serna, and A. M. Stuart, "Optimal tuning of the hybrid monte-carlo algorithm," *Bernoulli*, vol. 19, no. 5A, pp. 1501–1534, 2013. DOI:10.3150/12-BEJ414.

[31] J. A. Gubner, *Probability and Random Processes for Electrical and Computer Engineers*, pp. 388–395. Cambridge University Press, 2006. ISBN:978-0-521-86470-1.

[32] K. I. Park, *Fundamentals of Probability and Stochastic Processes with Applications to Communications*, pp. 165–169. Springer, 2018. DOI:10.1007/978-3-319-68075-0.

[33] A. Sokal, "Monte carlo methods in statistical mechanics: Foundations and new algorithms," in *Functional Integration: Basics and Applications* (C. DeWitt-Morette, P. Cartier, and A. Folacci, eds.), vol. 361 of *NATO ASI Series (Series B: Physics)*, pp. 131–192, Boston: Springer, 1997. DOI:10.1007/978-1-4899-0319-8.

[34] M. Wallerberger, "Efficient estimation of autocorrelation spectra." arXiv, 2018. arXiv:1810.05079.

[35] U. Wolff, "Monte carlo errors with less errors," *Computer Physics Communications*, vol. 156, no. 2, pp. 143–153, 2004. DOI:10.1016/S0010-4655(03)00467-3.

[36] B. A. Berg, "Introduction to markov chain monte carlo simulations and their statistical analysis." arXiv, 2004. arXiv:cond-mat/0410490.

[37] A. Kramer, B. Calderhead, and N. Radde, "Hamiltonian monte carlo methods for efficient parameter estimation in steady state dynamical systems," *BMC Bioinformatics*, vol. 15, no. 1, p. 253, 2014. DOI:10.1186/1471-2105-15-253.

[38] B. Efron and G. Gong, "A leisurely look at the bootstrap, the jackknife, and cross-validation," *The American Statistician*, vol. 37, no. 1, pp. 36–48, 1983. DOI:10.2307/2685844.

[39] A. McIntosh, "The jackknife estimation method." arXiv, 2016. arXiv:1606.00497.

[40] C. F. Chung, "Use of the jackknife method to estimate autocorrelation functions (or variograms)," in *Geostatistics for Natural Resources Characterization* (G. Verly, M. David, A. G. Journel, and A. Marechal, eds.), pp. 55–69, Dordrecht: Springer, 1984. DOI:10.1007/978-94-009-3699-7.

[41] M. Girolami, B. Calderhead, and S. A. Chin, "Riemannian manifold hamiltonian monte carlo." arXiv, 2009. arXiv:0907.1100.

[42] M. Girolami and B. Calderhead, "Riemann manifold langevin and hamiltonian monte carlo methods," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 73, no. 2, pp. 123–214, 2011. DOI:10.1111/j.1467-9868.2010.00765.x.

[43] J. Sohl-Dickstein, M. Mudigonda, and M. R. DeWeese, "Hamiltonian monte carlo without detailed balance." arXiv, 2014. arXiv:1409.5191.

[44] C. M. Campos and J. M. Sanz-Serna, "Extra chance generalized hybrid monte carlo." arXiv, 2014. arXiv:1407.8107.

A Appendix of Derivations

A.1 Derivation of equation (3), $\langle \ln \sigma_t^2 \rangle = \mu$

$$\langle \ln \sigma_t^2 \rangle = \langle h_t \rangle = \langle \mu + \phi(h_{t-1} - \mu) + \eta_t \rangle = \langle \mu \rangle + \langle \phi h_{t-1} \rangle - \langle \phi \mu \rangle + \langle \eta_t \rangle$$

μ and ϕ are parameters so $\langle \mu \rangle = \mu$ and $\langle \phi \rangle = \phi$. η_t is a normal distribution centred on zero $N(0, \sigma_\eta^2)$, so $\langle \eta_t \rangle = 0$:

$$\langle \ln \sigma_t^2 \rangle = \mu + \phi \langle h_{t-1} \rangle - \phi\mu = \mu(1 - \phi) + \phi \langle \ln \sigma_{t-1}^2 \rangle$$

$$\langle \ln \sigma_t^2 \rangle - \phi \langle \ln \sigma_{t-1}^2 \rangle = \mu(1 - \phi)$$

While $\ln \sigma_t^2$ is different for different t, the expectation value $\langle \ln \sigma_t^2 \rangle$ is the same for any t, so:

$$\langle \ln \sigma_t^2 \rangle (1 - \phi) = \mu(1 - \phi)$$

$$\boxed{\langle \ln \sigma_t^2 \rangle = \langle h_t \rangle = \mu} \tag{3}$$

A.2 Derivation of equation (4), $\mathrm{Var}(\ln \sigma_t^2) = \frac{\sigma_\eta^2}{1-\phi^2}$

$$\mathrm{Var}(\ln \sigma_t^2) = \mathrm{Var}(h_t) = \langle h_t^2 \rangle - \langle h_t \rangle^2 = \langle h_t^2 \rangle - \mu^2$$

$$\langle h_t^2 \rangle = \langle (\mu + \phi h_{t-1} - \phi\mu + \eta_t)^2 \rangle$$

Expanding and removing terms containing $\langle \eta_t \rangle = 0$:

$$\langle h_t^2 \rangle = \mu^2 + 2\mu\phi \langle h_{t-1} \rangle - 2\phi\mu^2 + \phi^2 \langle h_{t-1}^2 \rangle - 2\phi^2 \mu \langle h_{t-1} \rangle + \phi^2\mu^2 + \langle \eta_t^2 \rangle$$

Substituting $\langle h_{t-1} \rangle = \langle h_t \rangle = \mu$ and then eliminating like terms:

$$\langle h_t^2 \rangle = \mu^2 + 2\mu^2\phi - 2\phi\mu^2 + \phi^2 \langle h_{t-1}^2 \rangle - 2\phi^2\mu^2 + \phi^2\mu^2 + \langle \eta_t^2 \rangle$$

$$\langle h_t^2 \rangle = \mu^2 + \phi^2 \langle h_{t-1}^2 \rangle - \phi^2\mu^2 + \langle \eta_t^2 \rangle$$

Substituting $\langle h_t^2 \rangle$ back into initial equation for $\mathrm{Var}(\ln \sigma_t^2)$:

$$\mathrm{Var}(\ln \sigma_t^2) = \langle h_t^2 \rangle - \mu^2 = \phi^2 \langle h_{t-1}^2 \rangle - \phi^2\mu^2 + \langle \eta_t^2 \rangle$$

Re-substituting $\mu = \langle h_t \rangle = \langle h_{t-1} \rangle$, and subtracting $\langle \eta_t \rangle^2$ (as $\langle \eta_t \rangle^2 = 0^2 = 0$):

$$\mathrm{Var}(\ln \sigma_t^2) = \phi^2 \langle h_{t-1}^2 \rangle - \phi^2 \langle h_{t-1} \rangle^2 + \langle \eta_t^2 \rangle = \phi^2 (\langle h_{t-1}^2 \rangle - \langle h_{t-1} \rangle^2) + \langle \eta_t^2 \rangle - \langle \eta_t \rangle^2$$

$$\mathrm{Var}(\ln \sigma_t^2) = \phi \mathrm{Var}(h_{t-1}) + \mathrm{Var}(\eta_t)$$

Similar to the final step in A.1, variance of $\ln \sigma_t^2$ is also independent of t:

$$(1 - \phi)\mathrm{Var}(\ln \sigma_t^2) = \mathrm{Var}(\eta_t)$$

Variance is standard deviation squared, so $\mathrm{Var}(\eta_t) = \sigma_\eta^2$ and therefore:

$$\boxed{\mathrm{Var}(\ln \sigma_t^2) = \mathrm{Var}(h_t) = \frac{\sigma_\eta^2}{1 - \phi^2}} \tag{4}$$

A.3 Derivation of equation (20), σ_η^2 distribution

Explicit likelihood function is given by substituting equations (8), (9) and (10) into equation (7), and multiplying by $p(\theta)$ to obtain the posterior probability:

$$p(\theta|y) = \left[\int \prod_{t=1}^{N} \frac{1}{\sqrt{2\pi\sigma_t^2}} \exp\left(-\frac{y_t^2}{2\sigma_t^2}\right) \sqrt{\frac{1-\phi^2}{2\pi\sigma_\eta^2}} \exp\left(-\frac{(h_1-\mu)^2}{2\sigma_\eta^2/(1-\phi^2)}\right) \cdot \prod_{t=2}^{N} \frac{1}{\sqrt{2\pi\sigma_\eta^2}} \exp\left(-\frac{(h_t-\mu-\phi(h_{t-1}-\mu))^2}{2\sigma_\eta^2}\right) dh_1 dh_2 ... dh_N \right] p(\theta) \tag{49}$$

Extracting terms which depend on σ_η^2 and setting prior probability $p(\sigma_\eta^2) = (\sigma_\eta^2)^{-1}$:

$$P(\sigma_\eta^2) = \frac{1}{\sigma_\eta^2} \int \frac{1}{\sqrt{\sigma_\eta^2}} e^{-\frac{(h_1-\mu)^2}{2\sigma_\eta^2/(1-\phi^2)}} \prod_{t=2}^{N} \frac{1}{\sqrt{\sigma_\eta^2}} e^{-\frac{(h_t-\mu-\phi(h_{t-1}-\mu))^2}{2\sigma_\eta^2}} dh_1 dh_2 ... dh_N$$

$$P(\sigma_\eta^2) = \frac{1}{\sigma_\eta^2} \frac{1}{\sqrt{\sigma_\eta^2}} e^{-\frac{(h_1-\mu)^2(1-\phi^2)}{2\sigma_\eta^2}} (\frac{1}{\sqrt{\sigma_\eta^2}})^{N-1} e^{-\sum_{t=2}^{N} \frac{(h_t-\mu-\phi(h_{t-1}-\mu))^2}{2\sigma_\eta^2}}$$

$$P(\sigma_\eta^2) = (\sigma_\eta^2)^{-1} (\sigma_\eta^2)^{-\frac{1}{2}} (\sigma_\eta^2)^{-\frac{N-1}{2}} e^{-\frac{(h_1-\mu)^2(1-\phi^2)}{2\sigma_\eta^2} - \sum_{t=2}^{N} \frac{(h_t-\mu-\phi(h_{t-1}-\mu))^2}{2\sigma_\eta^2}}$$

$$P(\sigma_\eta^2) = (\sigma_\eta^2)^{-\frac{N}{2}-1} \exp\left(-\frac{(h_1-\mu)^2(1-\phi^2)}{2\sigma_\eta^2} - \sum_{t=2}^{N} \frac{(h_t-\mu-\phi(h_{t-1}-\mu))^2}{2\sigma_\eta^2}\right)$$

Then by substituting $A = \frac{1}{2}[(1-\phi^2)(h_1-\mu)^2 + \sum_{t=2}^{N}(h_t-\mu-\phi(h_{t-1}-\mu))^2]$:

$$\boxed{P(\sigma_\eta^2) = (\sigma_\eta^2)^{-\frac{N}{2}-1} \exp\left(-\frac{A}{\sigma_\eta^2}\right)} \tag{20}$$

A.4 Derivation of equation (22), μ distribution

Extracting terms which depend on μ from equation (49) and setting $p(\mu) = const$:

$$P(\mu) = \int \exp\left(-\frac{(h_1-\mu)^2}{2\sigma_\eta^2/(1-\phi^2)}\right) \prod_{t=2}^{N} \exp\left(-\frac{(h_t-\mu-\phi(h_{t-1}-\mu))^2}{2\sigma_\eta^2}\right) dh_1 ... dh_N$$

$$P(\mu) = \exp\left(-\frac{1}{2\sigma_\eta^2} \underbrace{\left[(h_1-\mu)^2(1-\phi^2) + \sum_{t=2}^{N}(h_t-\mu-\phi(h_{t-1}-\mu))^2\right]}\right)$$

$$\Rightarrow (h_1^2 - 2h_1\mu + \mu^2)(1-\phi^2) + \sum_{t=2}^{N}((h_t - \phi h_{t-1}) - \mu(1-\phi))^2$$

Expanding and removing terms with no μ dependence:

$$\Rightarrow \mu^2(1-\phi^2) - 2\mu h_1(1-\phi^2) - 2\sum_{t=2}^{N}\mu(1-\phi)(h_t - \phi h_{t-1}) + \sum_{t=2}^{N}\mu^2(1-\phi)^2$$

$$\Rightarrow \mu^2[(1-\phi^2) + (N-1)(1-\phi)^2] - 2\mu[h_1(1-\phi^2) + (1-\phi)\sum_{t=2}^{N}(h_t - \phi h_{t-1})]$$

By substituting $B = (1-\phi^2) + (N-1)(1-\phi)^2$ and
$C = (1-\phi^2)h_1 + (1-\phi)\sum_{t=2}^{N}(h_t - \phi h_{t-1})$ we obtain:

$$\Rightarrow \mu^2 B - 2\mu C$$

Completing the square $ax^2 + bx + c = a\left(x + \frac{b}{2a}\right)^2 + c - \frac{b^2}{4a}$ but ignoring the $c - \frac{b^2}{4a}$ term as it has no x dependence, and using constants $a = B$, $b = -2C$, $c = 0$:

$$\Rightarrow B(\mu - \frac{C}{B})^2$$

$$\boxed{P(\mu) = \exp\left(-\frac{B}{2\sigma_\eta^2}(\mu - \frac{C}{B})^2\right)} \tag{22}$$

A.5 Derivation of equation (25), ϕ distribution

Extracting terms which depend on ϕ from equation (49) and setting $p(\phi) = const$:

$$P(\mu) = \int \sqrt{1-\phi^2}\exp\left(-\frac{(h_1-\mu)^2}{2\sigma_\eta^2/(1-\phi^2)}\right)\prod_{t=2}^{N} e^{-\frac{(h_t-\mu-\phi(h_{t-1}-\mu))^2}{2\sigma_\eta^2}} dh_1 dh_2 ... dh_N$$

$$P(\mu) = \sqrt{1-\phi^2}\exp\left(-\frac{1}{2\sigma_\eta^2}\underbrace{\left[(h_1-\mu)^2(1-\phi^2) + \sum_{t=2}^{N}(h_t - \mu - \phi(h_{t-1}-\mu))^2\right]}\right)$$

Expanding terms inside square bracket and removing those with no ϕ dependence:

$$\Rightarrow -\phi^2(h_1-\mu)^2 - 2\sum_{t=2}^{N}\phi(h_t-\mu)(h_{t-1}-\mu) + \sum_{t=2}^{N}\phi^2(h_{t-1}-\mu)^2$$

$$\Rightarrow \phi^2[-(h_1-\mu)^2+\sum_{t=2}^{N}(h_{t-1}-\mu)^2]-2\phi\sum_{t=2}^{N}(h_t-\mu)(h_{t-1}-\mu)$$

By substituting $D=-(h_1-\mu)^2+\sum_{t=2}^{N}(h_{t-1}-\mu)^2$ and
$E=\sum_{t=2}^{N}(h_t-\mu)(h_{t-1}-\mu)$ we obtain:

$$\Rightarrow \phi^2 D-2\phi E$$

Completing the square like in A.4, and using constants $a=D$, $b=-2E$, $c=0$:

$$\Rightarrow D(\phi-\frac{E}{D})^2$$

$$\boxed{P(\phi)=\sqrt{1-\phi^2}\exp\left(-\frac{D}{2\sigma_\eta^2}(\phi-\frac{E}{D})^2\right)} \tag{25}$$

A.6 Derivation of equation (30), $P^{\phi}_{metro}(\phi_{new},\phi)$

Using equations (25), (28) and (29) and setting $T_0(x_n|x_{n+1})=P_2(\phi)$, $P(x_{n+1})=P(\phi_{new})$, $T_0(x_{n+1}|x_n)=P_2(\phi_{new})$ and $P(x_n)=P(\phi)$ in equation (18):

$$P^{\phi}_{metro}(\phi_{new},\phi)=\min\left[\frac{T_0(x_n|x_{n+1})P(x_{n+1})}{T_0(x_{n+1}|x_n)P(x_n)},1\right]=\min\left[\frac{P_2(\phi)P(\phi_{new})}{P_2(\phi_{new})P(\phi)},1\right]$$

$$P^{\phi}_{metro}(\phi_{new},\phi)=\min\left[\frac{P_2(\phi)P_1(\phi_{new})P_2(\phi_{new})}{P_2(\phi_{new})P_1(\phi)P_2(\phi)},1\right]=\min\left[\frac{P_1(\phi_{new})}{P_1(\phi)},1\right]$$

$$\boxed{P^{\phi}_{metro}(\phi_{new},\phi)=\min\left[\sqrt{\frac{1-\phi^2_{new}}{1-\phi^2}},1\right]} \tag{30}$$

A.7 Derivation of equation (31), h_t distribution

Extracting terms which depend on h_t from equation (49) and setting $p(h_t)=const$:

$$P(h_t)=\int\prod_{t=1}^{N}\frac{1}{\sqrt{\sigma_t^2}}e^{-\frac{y_t^2}{2\sigma_t^2}}e^{-\frac{(h_1-\mu)^2}{2\sigma_\eta^2/(1-\phi^2)}}\prod_{t=2}^{N}e^{-\frac{(h_t-\mu-\phi(h_{t-1}-\mu))^2}{2\sigma_\eta^2}}dh_1dh_2...dh_N$$

Then by making the substitution $\sigma_t^2=e^{h_t}$:

$$P(h_t)=\int\prod_{t=1}^{N}e^{-\frac{h_t}{2}}e^{-\frac{y_t^2}{2}e^{-h_t}}e^{-\frac{(h_1-\mu)^2}{2\sigma_\eta^2/(1-\phi^2)}}\prod_{t=2}^{N}e^{-\frac{(h_t-\mu-\phi(h_{t-1}-\mu))^2}{2\sigma_\eta^2}}dh_1dh_2...dh_N$$

$$P(h_t) = e^{\sum_{t=1}^{N}(-\frac{h_t}{2}-\frac{y_t^2}{2}e^{-h_t})} e^{-\frac{(h_1-\mu)^2}{2\sigma_\eta^2/(1-\phi^2)}} e^{\sum_{t=2}^{N}(-\frac{[h_t-\mu-\phi(h_{t-1}-\mu)]^2}{2\sigma_\eta^2})}$$

$$\boxed{\begin{aligned} &P(h_t) \equiv P(h_1, h_2, ...h_N) = \\ &\exp\left[-\sum_{t=1}^{N}\left(\frac{h_t}{2}+\frac{y_t^2}{2}e^{-h_t}\right) - \frac{(h_1-\mu)^2}{2\sigma_\eta^2/(1-\phi^2)} - \sum_{t=2}^{N}\frac{(h_t-\mu-\phi(h_{t-1}-\mu))^2}{2\sigma_\eta^2}\right] \end{aligned}} \tag{31}$$

A.8 Derivation of equation (35), $\frac{dp_t}{dt} = -\frac{\partial H}{dh_t}$

From equation (33) the Hamiltonian $H(p_t, h_t)$ is given by:

$$H = \sum_{t=1}^{N}\left(\frac{1}{2}p_t^2 + \frac{h_t}{2} + \frac{y_t^2}{2}e^{-h_t}\right) + \frac{(h_1-\mu)^2}{2\sigma_\eta^2/(1-\phi^2)} + \underbrace{\sum_{t=2}^{N}\frac{(h_t-\mu-\phi(h_{t-1}-\mu))^2}{2\sigma_\eta^2}}$$

We partly write the sum in the final term out explicitly for clarity:

$$\sum_{t=2}^{N}\frac{(h_t-\mu-\phi(h_{t-1}-\mu))^2}{2\sigma_\eta^2} = \frac{(h_2-\mu-\phi(h_1-\mu))^2}{2\sigma_\eta^2} + \frac{(h_3-\mu-\phi(h_2-\mu))^2}{2\sigma_\eta^2}$$
$$+\sum_{t=4}^{N-1}\frac{(h_t-\mu-\phi(h_{t-1}-\mu))^2}{2\sigma_\eta^2} + \frac{(h_N-\mu-\phi(h_{N-1}-\mu))^2}{2\sigma_\eta^2}$$

The equation of motion is given by differentiating the Hamiltonian:

$$\frac{dp_t}{dt} = -\frac{\partial H}{dh_t}$$

As can be seen from the equations above, this differential will have a general form for $1 < t < N$, and then there will be two unique cases for $t = 1$ and $t = N$.

For $t = 1$:

$$\frac{\partial H}{dh_1} = \frac{1}{2} - \frac{y_1^2}{2}e^{-h_1} + \frac{(h_1-\mu)(1-\phi^2)}{\sigma_\eta^2} - \phi\frac{((h_2-\mu)-\phi(h_1-\mu))}{\sigma_\eta^2}$$
$$\frac{\partial H}{dh_1} = \frac{1}{2} - \frac{y_1^2}{2}e^{-h_1} + \frac{(h_1-\mu)-\phi^2(h_1-\mu)-\phi(h_2-\mu)+\phi^2(h_1-\mu)}{\sigma_\eta^2}$$

Cancelling like terms and simplifying:

$$\frac{\partial H}{dh_1} = \frac{1}{2} - \frac{y_1^2}{2}e^{-h_1} + \frac{1}{\sigma_\eta^2}(h_1-\mu-\phi(h_{t-2}-\mu))$$

For $1 < t < N$ (or $t = t$):

$$\frac{\partial H}{dh_t} = \frac{1}{2} - \frac{y_t^2}{2}e^{-h_t} + \frac{(h_t - \mu - \phi(h_{t-1} - \mu))}{\sigma_\eta^2} + \frac{(h_{t+1} - \mu - \phi(h_t - \mu))}{\sigma_\eta^2}$$

$$\frac{\partial H}{dh_t} = \frac{1}{2} - \frac{y_t^2}{2}e^{-h_t} + \frac{1}{\sigma_\eta^2}\underbrace{[h_t - \mu - \phi(h_{t-1} - \mu) - \phi(h_{t+1} - \mu - \phi(h_t - \mu))]}$$

$$\Rightarrow h_t - \mu - \phi h_{t-1} + \phi\mu - \phi h_{t+1} + \phi\mu + \phi^2 h_t - \phi^2\mu$$

$$\Rightarrow h_t(1 + \phi^2) - \phi(h_{t-1} + h_{t+1}) + \mu(2\phi - 1 - \phi^2)$$

$$\Rightarrow h_t(1 + \phi^2) - \phi(h_{t-1} + h_{t+1}) - \mu(\phi - 1)^2$$

$$\frac{\partial H}{dh_t} = \frac{1}{2} - \frac{y_t^2}{2}e^{-h_t} + \frac{1}{\sigma_\eta^2}(h_t(1 + \phi^2) - \phi(h_{t-1} + h_{t+1}) - \mu(\phi - 1)^2)$$

For $t = N$:

$$\frac{\partial H}{dh_N} = \frac{1}{2} - \frac{y_N^2}{2}e^{-h_N} + \frac{1}{\sigma_\eta^2}(h_N - \mu - \phi(h_{N-1} - \mu))$$

$$\frac{dp_t}{dt} = -\frac{\partial H}{dh_t} = \begin{cases} -\frac{\partial H}{dh_1} & \text{for } t = 1 \\ -\frac{\partial H}{dh_t} & \text{for } 1 < t < N \\ -\frac{\partial H}{dh_N} & \text{for } t = N \end{cases}$$

$$\frac{dp_t}{dt} = -\frac{\partial H}{dh_t} =$$

$$\begin{cases} \frac{y_1^2}{2}e^{-h_1} - \frac{1}{2} - \frac{1}{\sigma_\eta^2}(h_1 - \mu - \phi(h_{t-2} - \mu)) & \text{for } t = 1 \\ \frac{y_t^2}{2}e^{-h_t} - \frac{1}{2} - \frac{1}{\sigma_\eta^2}(h_t(1 + \phi^2) - \phi(h_{t-1} + h_{t+1}) - \mu(\phi - 1)^2) & \text{for } 1 < t < N \\ \frac{y_N^2}{2}e^{-h_N} - \frac{1}{2} - \frac{1}{\sigma_\eta^2}(h_N - \mu - \phi(h_{N-1} - \mu)) & \text{for } t = N \end{cases} \tag{35}$$

A.9 Derivation of equations (41) and (42), τ and σ_{true}

Using the same notation as equation (43), we define the unnormalised autocorrelation function for an observable X as:

$$\Gamma(t) = \frac{1}{n}\sum_{i=1}^{n}[X_i - \langle X\rangle][X_{i+t} - \langle X\rangle] = \langle[X_i - \langle X\rangle][X_{i+t} - \langle X\rangle]\rangle$$

$$\Gamma(t) = \langle[X_i X_{i+t} - X_i\langle X\rangle - X_{i+t}\langle X\rangle + \langle X\rangle^2]\rangle$$

Now we use the fact that an expectation value is just a single number (so its expected value is just itself), to write:

$$\Gamma(t) = \langle X_i X_{i+t} \rangle - \langle X_i \rangle \langle X \rangle - \langle X_{i+t} \rangle \langle X \rangle + \langle X \rangle^2$$

The properties of the data from our HMC (θ and h_t) are invariant in time, so their expectation values do not depend on i and therefore $\langle X \rangle = \langle X_i \rangle = \langle X_{i+t} \rangle$:

$$\Gamma(t) = \langle X_i X_{i+t} \rangle - 2\langle X \rangle^2 + \langle X \rangle^2 = \langle X_i X_{i+t} \rangle - \langle X \rangle^2$$

The variance of X is a special case of the autocorrelation function when $t = 0$:

$$\Gamma(0) = \langle X_i^2 \rangle - \langle X \rangle^2 = \langle X^2 \rangle - \langle X \rangle^2 = \sigma_X^2 \tag{50}$$

The standard way of estimating the mean for a set of data X is given by $\langle X \rangle = \frac{1}{n} \sum_{i=1}^{n} X_i$, where n is the number data samples in X. The variance of this estimator $\langle X \rangle$ is then given by [33–36]:

$$\sigma_{\langle X \rangle}^2 = \frac{1}{n} \sum_{t=-n}^{n} \left(1 - \frac{|t|}{n}\right) \Gamma(t) = \frac{1}{n} \left[\Gamma(0) + 2 \sum_{t=1}^{n} \left(1 - \frac{|t|}{n}\right) \Gamma(t)\right]$$

Substituting in $\Gamma(0) = \sigma_X^2$ from equation (50):

$$\sigma_{\langle X \rangle}^2 = \frac{1}{n} \left[\sigma_X^2 + 2 \sum_{t=1}^{n} \left(1 - \frac{|t|}{n}\right) \Gamma(t)\right] = \frac{2\sigma_X^2}{n} \left[\frac{1}{2} + \sum_{t=1}^{n} \left(1 - \frac{|t|}{n}\right) \frac{\Gamma(t)}{\sigma_X^2}\right] \tag{51}$$

Comparing this with the equivalent equation for uncorrelated random variables $\sigma_{\langle X \rangle}^2 = \sigma_X^2 / n$, we define the autocorrelation time τ as:

$$\tau = \left[\frac{1}{2} + \sum_{t=1}^{n} \left(1 - \frac{|t|}{n}\right) \frac{\Gamma(t)}{\sigma_X^2}\right]$$

And as we set $n \gg 1$ for our HMC, we can recast τ in the limit $n \to \infty$ as:

$$\boxed{\tau = \left[\frac{1}{2} + \sum_{t=1}^{\infty} \frac{\Gamma(t)}{\sigma_X^2}\right] = \frac{1}{2} + \sum_{t=1}^{\infty} ACF(t)} \tag{41}$$

Substituting our definition of τ into equation (51) we can also write:

$$\sigma_{\langle X \rangle}^2 = \frac{2\tau\sigma_X^2}{n} \tag{52}$$

And then by defining $\sigma_{\langle X \rangle}^2$ and σ_X^2 / n as the true and naive variances respectively:

$$\boxed{\sigma_{true} = \sigma_{naive} \sqrt{2\tau}} \tag{42}$$