M5MS16 Advanced Simulation - Coursework 2

Question 1

Question 1 (i)

State a theoretical result of your choice that can be used to justify particle filters. Mention its significance and relevance to practical applications. [1 mark]

Particle filters are motivated by the same reasoning as importance sampling, which can be applied successfully in a sequential setting as follows. Given a sequence of target densities $\{\pi_n(x_{0:n})\}_{n\leq T}$, assume that $\pi_n(x_{0:n}) = \mathcal{Z}_n^{-1}\gamma_n(x_{0:n})$ where γ_n is known but the constant $\mathcal{Z}_n$ is not. In this scenario we have enough information to approximate each π_n: we sample N values $x^i_{0:n}$ according to proposal density $q_n(x_{0:n})$, compute their associated weights

$$w(x^i_{0:n}) = \frac{\gamma_n(x^i_{0:n})}{q_n(x^i_{0:n})}, \tag{1}$$

normalize these to form W^i_n, then approximate the density according to:

$$\pi_n(dx_{0:n}) = \sum_{i=1}^{N} W^i_n \delta_{x^i_{0:n}}(dx_{0:n}) \tag{2}$$

where for $A \subseteq \mathbb{R}^n$, $\delta_{x_{0:n}}(A)$ is the Dirac measure that equals 1 if $x_{0:n} \in A$ and 0 otherwise.

Particle filters are functionally identical to the batch importance sampling procedure described above, but are designed to exploit the structure of a Hidden Markov Model. The proposal densities q_n are chosen such that they can be factorized:

$$q_n(x_{0:n}) = q_n(x_n|x_{0:n-1})q_n(x_{0:n-1}) \tag{3}$$

and the weights in Equation 1 are computed recursively according to:

$$\begin{aligned} w(x_{0:n}) = \frac{\gamma_n(x_{0:n})}{q_n(x_{0:n})} &= \frac{\gamma_{n-1}(x_{0:n-1})}{q_{n-1}(x_{0:n-1})}\frac{f(x_n|x_{n-1})g(y_n|x_n)}{q_n(x_n|x_{0:n-1})} \\ &= w(x_{0:n-1})\frac{f(x_n|x_{n-1})g(y_n|x_n)}{q_n(x_n|x_{0:n-1})} \end{aligned} \tag{4}$$

Factorizing the proposal density allows a path to be sampled from $q_n(x_{0:n})$ by appending a value to the path sampled from $q_{n-1}(x_{0:n-1})$, where the value is sampled according to the density $q(x_n|x_{0:n-1})$. This is less computationally expensive than sampling a full path $x_{0:n}$, which has a cost that grows in n. Furthermore, because we update the weights rather than re-compute them, it is not necessary to repeatedly evaluate the (possibly expensive) functions $\gamma_n(x_{0:n})$ and $q_n(x_{0:n})$.

Because particle filters are a variant of importance sampling, they benefit from the same theoretical results - asymptotic consistency in N with respect to target densities π_n, a central limit result concerning functional expectations, and a Monte Carlo variance that decreases at a rate proportional to $\frac{1}{N}$. In addition to this, if the state-space model has the exponential forgetting property, i.e. for any x_0, $x_0' \in \mathcal{X}$ and observation $y_{0:n} \in \mathcal{Y}$,

$$\int \left|p_\theta\left(x_n|y_{0:n}, x_0\right) - p_g\left(x_n|y_{0:n}, x_0'\right)\right| dx_n \leq C\lambda^n \tag{5}$$

where $\lambda \in [0,1)$ and C is a constant, then for an integer L and bounded test function $\psi_L : \mathcal{X}^L \to \mathbb{R}$ there exist finite constants $D_{\theta,L,p}$ such that for each $p > 0$, there exists a bound on the mean L^p error:

$$\mathbb{E}^N\left[\left|\int \varphi_L\left(x_{n-L+1:n}\right)\epsilon_{\theta,L}\left(dx_{n-L+1:n}\right)\right|^p\right]^{\frac{1}{p}} \leq \frac{D_{\theta,L,p}\overline{\varphi_L}}{N^{1/2}} \tag{6}$$

where $\epsilon_{\theta,L}\left(dx_{n-L+1:n}\right) = \int_{\mathcal{X}^{n-L+1}} \epsilon_{\theta,n}\left(dx_{0:n}\right)$ and $\epsilon_{\theta,n}\left(dx_{0:n}\right) = \widehat{p}_\theta\left(dx_{0:n}|y_{0:n}\right) - p_\theta\left(dx_{0:n}|y_{0:n}\right)$, with the hat indicating the SIR filter approximation. The practical relevance of this result is that the bound on the mean-squared error of an SIR approximation to an expectation is proportional to $\frac{1}{N}$ for all n (i.e. the error does not grow in n).

Question 1 (ii)

Explain why approximating $p(x_m|y_{0:n})$ with $m < n$ using particle filters is important. State two different algorithms for approximating $p(x_m|y_{0:n})$, with $m < n$, using particle filters. Mention briefly one advantage and disadvantage for each method. There is no need to provide expressions or equations. [2 marks]

Approximating $p(x_m|y_{0:n})$ with $m < n$ using particle filters is important because it constitutes a marginal smoothing problem, which has applications in GPS tracking, inertial navigation (IMU dead-reckoning), streaming classification and regression problems, stochastic optimal control, and epidemiological modeling. Furthermore, the smoothing distribution may be useful for parameter estimation in a general setting, as will be mentioned in Question 3.2 (ii).

Two different algorithms for approximating $p(x_m|y_{0:n})$ with $m < n$ using particle filters are the fixed-lag smoother and forward-backward smoothers.

An advantage of the fixed-lag smoother is that it does not require any additional computation expense beyond that of a standard SIR filter, which has time complexity $\mathcal{O}(NT)$. A disadvantage of the fixed-lag smoother is that if the state-space model does not possess the exponential forgetting property for moderately small L then its approximation to $p(x_m|y_{0:n})$ will be asymptotically biased. A further disadvantage is that it is unlikely to be immediately obvious what choice of L is appropriate.

The forward-backward smoothers, FFBSa and FFBSm, sample from N^T possible paths (as opposed to the NT accessible to the standard SIR filter) and, when used to estimate an additive functional $\mathcal{S}_n^\theta$, have a variance that increases more slowly in N than an SIR filter approximation (where it grows at at least n^2/N), at a rate proportional to n/N. The disadvantage of these methods is that they have time complexity $\mathcal{O}(N^2T)$.

Question 1 (iii) (a)

Assume you have a data-set $y_{0:T}$ that you want to model with a hidden Markov model. [4 marks]

For the purpose of parameter estimation, describe a likelihood-based method of your choice that uses a particle filtering method. The description should include pseudo-code for the particle filter and computing the MLE. State also the computational cost of your method as a function of T, N, and any other important variables e.g. in an iterative scheme the number of iterations K.

Gradient ascent on the log-likelihood is a likelihood-based method for parameter estimation. Given a hidden Markov model that requires inference for parameter vector θ, an initialization value θ_0, and a sequence of step sizes $\{\gamma_k\}$, exact gradient ascent consists of computing updates:

$$\theta_k = \theta_{k-1} + \gamma_k \nabla_\theta \ell(\theta)\big|_{\theta=\theta_{k-1}} \tag{7}$$

where $\ell(\theta) = \log p(y_{0:T}|\theta)$. Recall Fisher's identity:

$$\nabla_\theta \ell(\theta) = \int p_\theta(x_{0:T}|y_{0:T}) \nabla_\theta \log p_\theta(x_{0:T}, y_{0:T}) dx_{0:T} \tag{8}$$

This identity is useful because we can use an SIR filter to approximate $p_\theta(x_{0:T}|y_{0:T})$ and we can obtain a convenient expression for $\nabla_\theta \log p_\theta(x_{0:T}, y_{0:T})$:

$$\nabla_\theta \log p_\theta(x_{0:T}, y_{0:T}) = \nabla_\theta \log\left(\prod_{k=0}^{T} g(y_k|x_k) f(x_k|x_{k-1})\right) \tag{9}$$

$$= \sum_{k=0}^{T} \big(\nabla_\theta \log g(y_k|x_k) + \nabla_\theta \log f(x_k|x_{k-1})\big) \tag{10}$$

where we adopt the convention that $f(x_0|x_{-1}) = f(x_0)$. Assuming that $\log g(y_k|x_k)$ and $\log f(x_k|x_{k-1})$ are differentiable, gradient ascent can be implemented via the filter approximation:

$$\hat{p}_\theta(dx_{0:T}|y_{0:T}) = \frac{1}{N}\sum_{i=1}^{N} \delta_{\bar{X}^i_{0:T}}(dx_{0:T}) \tag{11}$$

such that SIR approximation to the gradient of the log-likelihood is:

$$\nabla_\theta \ell(\theta) = \frac{1}{N}\sum_{i=1}^{N}\sum_{k=0}^{T} \big(\nabla_\theta \log g(y_k|\bar{X}^i_k) + \nabla_\theta \log f(\bar{X}^i_k|\bar{X}^i_{k-1})\big) \tag{12}$$

where the components $\bar{X}^i_k$ are of the resampled particles $\bar{X}^i_{0:T}$. Computing this gradient approximation requires an SIR filter of N particles to be run across T timesteps, which has cost $\mathcal{O}(NT)$. If the gradient ascent converges in K iterations, then the time complexity of this algorithm is $\mathcal{O}\big(KNT\big)$.

Pseudo-code for this procedure is presented below.

Algorithm 1: Question 1 (iii) (a) - Pseudo-code for the gradient ascent procedure and particle filter.

```
Initialize $\theta_0$, $K$, $\{\gamma_k\}$, $N$, $y_{0:T}$, $T$, $f(x_0)$, $f(x_k|x_{k-1})$, $g(y_k|x_k)$.
def estimate_ll_grad($\theta$: vec):
    # Approximate log-likelihood's gradient via SIR filter
    sir.x := zeros(T, N) # particles
    sir.w := zeros(N) # weight increments
    grad_log_fxk := zeros(T, dim($\theta$)) # $\nabla_\theta f(x_k|x_{k-1})$ approximations
    for $j := 1 : N$ do
        # NB this loop can be vectorized
        sir.x[1, j] $\sim f(x_0)$  # Sample initial state
        sir.w[j] := $g$(y[1] | sir.x[1, j]) # Compute weight increment
    sir.x := sir.x[ , sample(1:N, size=N, replace=TRUE, prob=sir.w)] # Resample particles
    grad_log_fxk[1, ] := $\frac{1}{N}\sum_{i=1}^{N} \nabla_\theta f$(sir.x[1, i]) # $\nabla_\theta f(x_0)$ approximation
    for $k := 2 : T$ do
        for $j := 1 : N$ do
            # NB this loop can be vectorized
            sir.x[k, j] $\sim f(x_k|$ sir.x[k-1, j] )  # Sample state via bootstrap density
            sir.w[j] := $g$(y[k] | sir.x[k, j]) # Compute weight increment
        sir.x := sir.x[ , sample(1:N, size=N, replace=TRUE, prob=sir.w)] # Resample particles
        for $j := 1 : N$ do
            # NB this loop can be vectorized
            grad_log_fxk[k, ] += $\nabla_\theta f$(sir.x[k, j] | sir.x[k-1, j])
        grad_log_fxk[k, ] := $\frac{1}{N}$grad_log_fxk[k, ] # $\nabla_\theta f(x_k|x_{k-1})$ approximation
    return(colSums(grad_log_fxk))
for $k := 1 : K$ do
    Compute approximation to log-likelihood gradient,
        $$\hat{\nabla}_\theta \log \ell(\theta)\Big|_{\theta=\theta_{k-1}} := \texttt{estimate\_ll\_grad}(\theta_{k-1})$$
    Update parameter vector:
        $$\theta_k := \theta_{k-1} + \gamma_k \hat{\nabla}_\theta \log \ell(\theta)\Big|_{\theta=\theta_{k-1}}$$
```

Question 1 (iii) (b)

Suppose T is a of moderate size and assume also you are given a small computational budget. Would you spend more of this budget in N or K? Give reasons or examples for your choice.

As will be discussed in Question 3.2 (i), N needs only be sufficiently large that the variance of the gradient estimate does not obscure the shape of the log-likelihood surface $\log \ell(\theta)$. This means that N should be chosen such that the variance of the gradient estimate is much smaller than the magnitude of the observed gradients when in the region that needs to be explored. Nearby to a maximum, a large N is necessary, whereas in a region where the surface of the log-likelihood has a large gradient, a small N is satisfactory. The shape of the log-likelihood surface is problem-dependent, but having N depend on the local gradient, rather than taking on a fixed value, might be sensible.

Provided N is large enough that the variance of the gradient estimate is adequately small, investing in K is desirable since more iterations towards a maximum can then be taken. The K budget might be distributed by initializing the algorithm at multiple locations in parameter space, or by investing it in a single initialization with a step sequence $\{\gamma_k\}$ that permits the maximum the be more finely resolved (i.e. the sequence is long and the steps become very small). Multiple initialization is especially relevant if good starting values for θ are not known.

Question 1 (iv) (a)

In the Resample Move algorithm, a practitioner proposes a modification of the algorithm: instead of using an MCMC algorithm simply add a Student t-distributed noise whose variance is finite and proportional to $\frac{1}{N}$ with N being the number of particles.

How can this approach be justified theoretically?

This approach can be justified theoretically by noting that it addresses the path degeneracy problem - adding t-distributed noise to all components of the particles at each time-step means that small-k (relative to n) particle components X_k^i will not be identical across all particles.

Question 1 (iv) (b)

Mention a weakness of this method.

The approach described corresponds to a single iteration of an MCMC algorithm. Assume that the particles jittered are $\{\bar{X}_{0:n}^i\}_{i=1,\dots,N}$ (i.e. these are the resampled particles) and the jittering procedure constitutes sampling:

$$\tilde{X}_{0:n}^i \sim K_n(\cdot|\bar{X}_{0:n}^i) \tag{13}$$

We can see that the kernel K_n suggested by the practitioner is a t-distribution located at $\bar{X}_{0:n}^i$ with variance proportional to $1/N$. A weakness of this method is that the the kernel's invariant distribution is not guaranteed to be $p(x_{0:n}|y_{0:n})$. In general, the invariant distribution will not be $p(x_{0:n}|y_{0:n})$, meaning that the proposed approach will bias Monte Carlo estimates based on the filter.

Question 1 (iv) (c)

How could this approach be useful in the context of parameter estimation?

As will be discussed in Question 2 (iv) (a), particle filter methods for parameter estimation can be enhanced by introducing artificial dynamics to the parameters. The practitioner's idea for preserving sample diversity would be useful in this context of on-line parameter estimation since it would prevent the posterior being dominated by a single particle.

Question 2

Consider the following scalar linear Gaussian model

$$X_n = \rho X_{n-1} + \tau V_n \tag{14}$$

$$Y_n = X_n + \sigma W_n \tag{15}$$

where $W_n, V_n \overset{i.i.d.}{\sim} \mathcal{N}(0,1)$, $X_0 \sim \mathcal{N}(0,1)$. *Run the model to synthesize a data-set* $y_{0:T}$ *for* $T = 100, \rho = 0.8, \tau = 1, \sigma = 0.5$. *Store the real state trajectory,* $x^*_{0:T}$. *We wish to perform Bayesian inference for* $p(\rho, \tau^2|y_{0:T})$ *assuming* σ *is known and set to its true value. Use as priors the uniform* $\mathcal{U}[1,1]$ *and inverse Gamma* $\mathcal{IG}(1,1)$ *distributions respectively.*

Question 2 (i)

Using $p_\theta(y_{0:T})$ *as computed from the Kalman filter recursions, design an ideal Marginal Metropolis Hastings chain to target* $p(\rho, \tau^2|y_{0:T})$. *Provide details on the algorithm implemented (what proposal is used, step size used, resulting average acceptance ratio) and plot the resulting histograms for* ρ, τ^2 *after* 25000 *iterations along with two MCMC diagnostics of your choice.*

Let $\eta = (\rho, \tau^2)$. We can compute the likelihood by recursively applying the definition of a conditional density:

$$p_\sigma(y_{0:n}|\eta) = p_\sigma(y_0|\eta) \prod_{k=1}^{n} p_\sigma(y_k|y_{0:k-1}, \eta) \tag{16}$$

Since the state-space model is linear and Gaussian, the Kalman filter allows us to evaluate the densities $p_\sigma(y_k|y_{0:k-1}, \eta)$ as those of $\mathcal{N}(m_n, \Sigma_n)$, while $p_\sigma(y_0|\eta)$ is the density of $\mathcal{N}(0, 1+\sigma^2)$ (from Equation 15).

An ideal marginal Metropolis Hastings chain that targets $p(\rho, \tau^2|y_{0:T})$ can be derived by applying Bayes' theorem:

$$p(\eta|y_{0:T}) = \frac{p_\sigma(y_{0:T}|\eta)p(\eta)}{p_\sigma(y_{0:T})} \tag{17}$$

The joint density $p(\eta) = p(\rho, \tau^2)$ is a product of $\mathcal{U}[-1,1]$ and $\mathcal{IG}(1,1)$. No method for computing $p(y_{0:T})$ suggests itself, but it is possible to evaluate the likelihood and prior densities for a given η, meaning that Markov chain Monte Carlo methods are applicable.

Define $\gamma(\eta) = p_\sigma(y_{0:T}|\eta)p(\eta)$. Let $\tilde{\eta}_k$ and η_k denote the proposed and accepted parameter values at the kth iteration of the chain respectively, and let $q(x|y)$ be the proposal density. Then, under the assumption that the proposal density is symmetric : $q(\eta|\tilde{\eta}) = q(\tilde{\eta}|\eta)$, the acceptance ratio is $\alpha(\eta_{k-1}, \tilde{\eta}_k) = 1 \wedge \frac{\gamma(\tilde{\eta}_k)}{\gamma(\eta_{k-1})}$. This is the case here, since we will use the Gaussian proposal density:

$$\tilde{\eta}_k|\eta_{k-1} \sim \mathcal{N}(\eta_{k-1}, \varrho^2) \tag{18}$$

Algorithm 2 specifies the proposed ideal marginal Metropolis Hastings algorithm. The initial value η_0 is sampled from the prior $p(\eta)$, although a fixed initial value could be used.

The implemented chain used a step size of $\varrho = 0.1$. This value was chosen so that the sum over the chain's autocorrelation sequence was small. The chain was run for 25000 iterations to produce the trace displayed in the bottom two panels of Figure 1. This figure also presents smoothed kernel density estimates[1] of the posterior marginal densities based on the chain's samples. The trace plot indicates that the chain transitioned regularly and made moderately-sized steps relative to the size of the parameter's support. The estimated autocorrelation values for the chain, presented in the middle

[1] A Gaussian kernel was used with a bandwidth of 0.02.

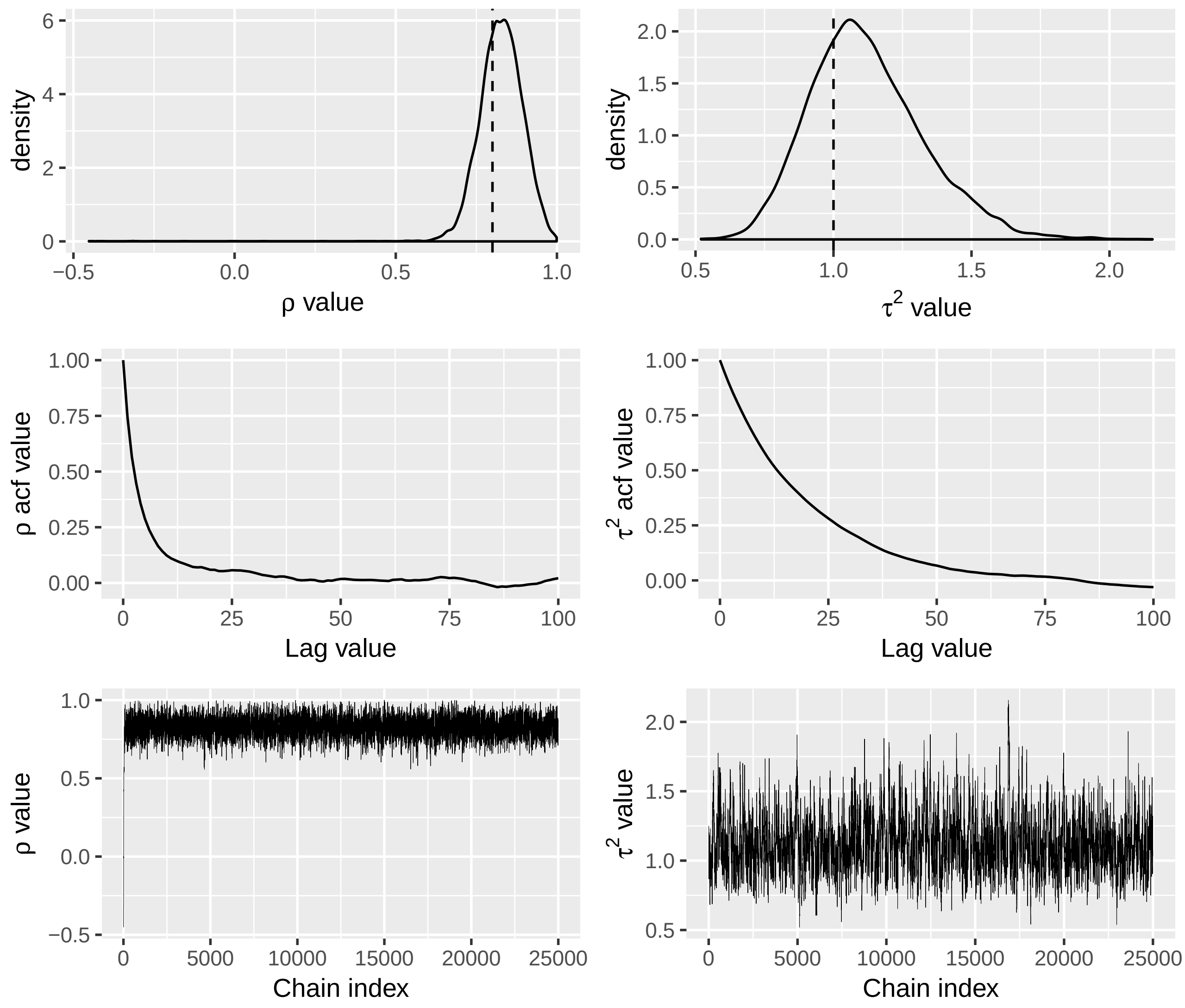

Figure 1: Question 2 (i) - ideal marginal Markov chain Monte Carlo via the Metropolis-Hastings algorithm using a Gaussian random walk proposal with step size $\varrho = 0.1$ and $M = 25000$ iterations. The chain was initialized by sampling from the prior distributions $\mathcal{U}[-1, 1]$ and $\mathcal{IG}(1, 1)$.

panels of Figure 1, indicate that the sum of autocorrelations was moderate: the effective sample size was 1809 in ρ and 646 in τ^2. The acceptance ratio was 0.52.

Algorithm 2: Question 2 (i) - ideal Metropolis-Hastings MCMC sampler for $p(\rho, \tau^2|y_{0:T})$.

```
Note that $\gamma(\eta) = p_\sigma(y_{0:T}|\eta)p(\eta)$, where $p_\sigma(y_{0:T}|\eta)$ is computed via the Kalman filter.
Sample initial values $\rho_0 \sim \mathcal{U}[-1, 1]$, $\tau_0^2 \sim \mathcal{IG}(1, 1)$.
Set $\eta_0 := (\rho_0, \tau_0^2)$.
Compute and store $\gamma(\eta_0) := p_\sigma(y_{0:T}|\eta_0)p(\eta_0)$.
for $k$ := 1:$M$ do
    Sample proposal $\tilde{\eta}_k \sim \mathcal{N}(\eta_{k-1}, \varrho^2 I)$.
    Compute proposal unnormalized density $\gamma(\tilde{\eta}_k) := p_\sigma(y_{0:T}|\tilde{\eta}_k)p(\tilde{\eta}_k)$.
    Compute proposal acceptance ratio $\alpha(\eta_{k-1}, \tilde{\eta}_k) := 1 \wedge \frac{\gamma(\tilde{\eta}_k)}{\gamma(\eta_{k-1})}$.
    With probability $\alpha(\eta_{k-1}, \tilde{\eta}_k)$, set $\eta_k := \tilde{\eta}_k$, $\gamma(\eta_k) := \gamma(\tilde{\eta}_k)$. Otherwise set $\eta_k := \eta_{k-1}$,
     $\gamma(\eta_k) := \gamma(\eta_{k-1})$.
```

Question 2 (ii) (a)

Using the same MCMC algorithm design (initialisation, iteration number, step sizes in proposals etc.), implement a particle Marginal Metropolis Hastings method.

Plot the resulting histograms for ρ, τ^2 together with the MCMC diagnostics used earlier.

The particle Marginal Metropolis Hastings method requires only a slight modification to the approach taken in Question 2 (i): instead of computing $p_\theta(y_{0:T}) = p_\sigma(y_{0:T}|\tilde{\eta})$ using the Kalman filter, we use the SIR particle approximation:

$$\hat{p}_\theta(y_{0:T}) = \hat{p}_\theta(y_0) \prod_{k=1}^{n} \hat{p}_\theta(y_k|y_{0:k-1}) \tag{19}$$

where:

$$\hat{p}_\theta(y_k|y_{0:k-1}) = \frac{1}{N}\sum_{i=1}^{N} w_n(X^i_{n-1:n}) \tag{20}$$

and:

$$w_n(X^i_{n-1:n}) = \frac{f(X^i_n|X^i_{n-1})g(X^i_n|y_n)}{q(X^i_n|X^i_{n-1})} \tag{21}$$

In the interest of numerical stability, the log-likelihood $\log p_\theta(y_{0:T})$ was estimated rather than the likelihood. The estimator $\log \hat{p}_\theta(y_{0:T})$ was used.

The requested plots are shown in Figure 2. The average acceptance ratio for the implementation was 0.334, while the effective sample sizes were 1572 and 462 for ρ and τ^2 respectively.

Question 2 (ii) (b)

Comment on how you chose N.

N was chosen such that the variance of the acceptance probability $\alpha(\eta_{k-1}, \tilde{\eta}_k)$ was small (i.e. such that the variance of $\log \hat{p}_\theta(y_{0:T})$ was ≈ 1, as suggested in the course notes). $\log \hat{p}_\theta(y_{0:T})$ was repeatedly evaluated for a selection of parameter values sampled from the prior to identify that $N = 25$ yielded the stated variance in $\log \hat{p}_\theta(y_{0:T})$ when the optimal proposal was used. Using the optimal proposal yielded a significant computational speed-up relative to the bootstrap proposal: when using a bootstrap proposal, a value of $N = 350$ when the bootstrap proposal was necessary to obtain $\log \hat{p}_\theta(y_{0:T}) \approx 1$. Only the results for the optimal proposal are presented here.

Question 2 (ii) (c)

State how this algorithm can be used to estimate $x^*_{0:T}$.

The particle MCMC algorithm can be used to estimate $x^*_{0:T}$ by sampling $\tilde{X}_{0:T}$ at each MCMC iteration from the particle filter, i.e. by sampling according to the density

$$\bar{p}_\sigma(dx_{0:T}|y_{0:T}, \tilde{\eta}) = \frac{1}{N}\sum_{i=1}^{N} \delta_{\bar{X}^i_{0:T}}(dx_{0:T}) \tag{22}$$

with sampled paths stored or rejected according to the Metropolis-Hastings acceptance outcome. These paths correspond to samples from $p(x_{0:T}, \eta|y_{0:T})$. The samples can be used to fabricate an estimator for $x^*_{0:T}$, although the optimal choice of estimator depends on the choice of loss function. If mean squared error is to be minimized, then the average of the MCMC samples would be optimal.

Question 2 (ii) (d)

Using the output of this algorithm, produce a plot with an estimate of $X_{0:T}$ against time, together with 95% credible intervals. In the same plot include $x^*_{0:T}$.

The requested plot is shown in Figure 3.

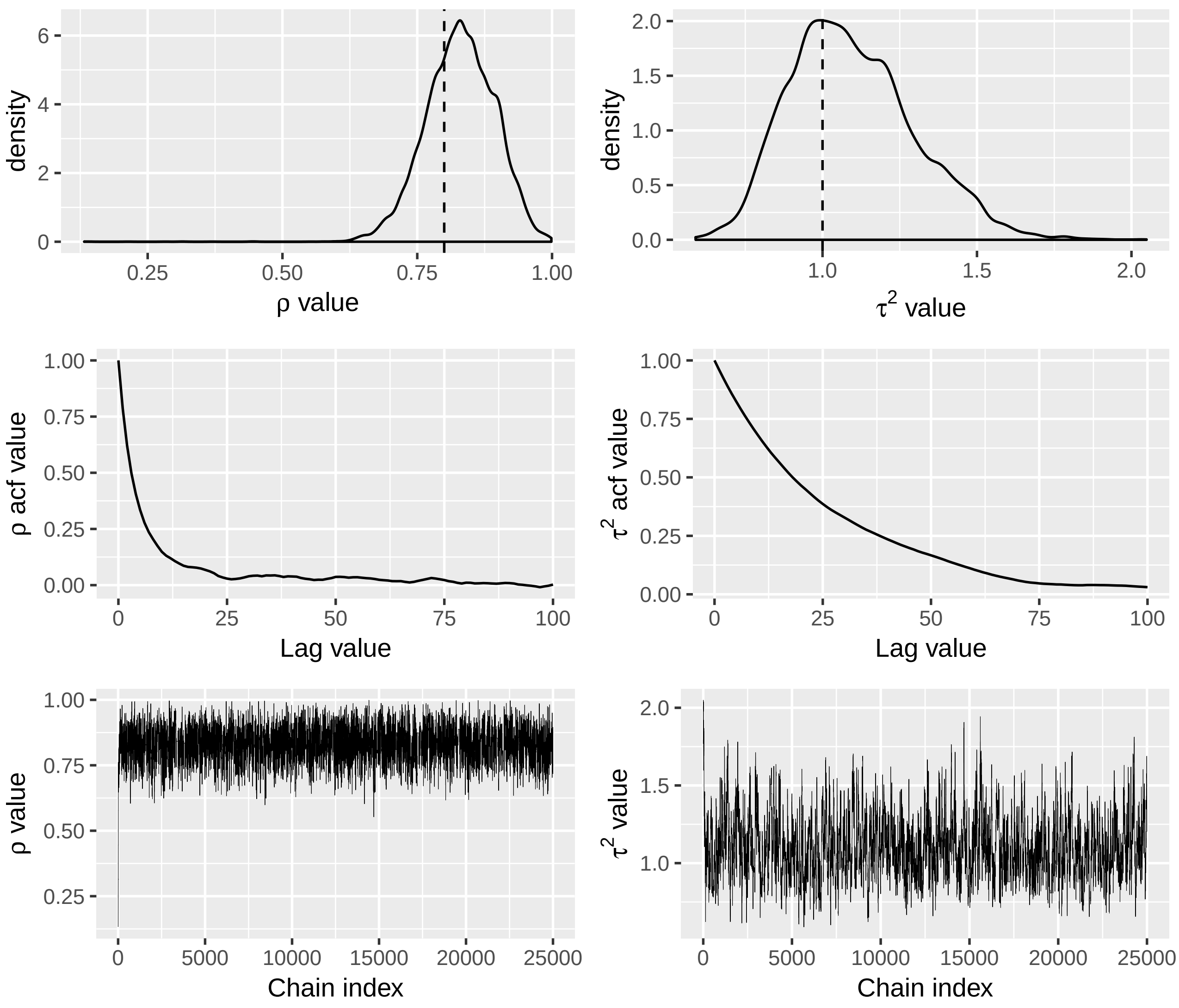

Figure 2: Question 2 (ii) (a) - particle marginal Markov chain Monte Carlo via the Metropolis-Hastings algorithm using a Gaussian random walk proposal with step size $\varrho = 0.1$ and $M = 25000$ iterations. The chain was initialized by sampling from the prior distributions $\mathcal{U}[-1, 1]$ and $\mathcal{IG}(1, 1)$. The particle filter used was an SIR filter with an optimal proposal density and $N = 25$.

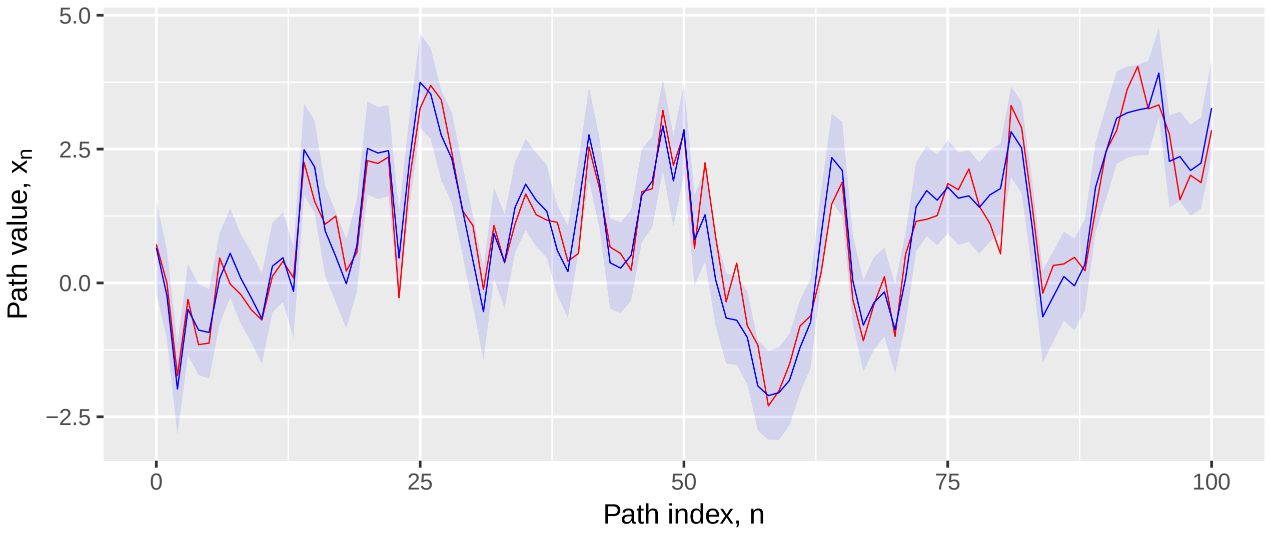

Figure 3: Question 2 (ii) (d) - pMMH estimate of $\mathrm{E}[X_{0:T}|Y_{0:T}]$ versus $x^*_{0:T}$. The red line corresponds to $x^*_{0:T}$. The blue line is the pMMH estimate of $\mathrm{E}[X_{0:T}|Y_{0:T}]$, formed by averaging 25 000 MCMC paths $\bar{X}^i_{0:T}$ sampled from SIR filter approximations $\bar{p}(dx_{0:T}|y_{0:T})$. The filled region corresponds to the empirical 95% credible interval for $X_{0:T}$ implied by the pMMH samples and was formed by taking quantiles 0.025 and 0.975 at each path index value.

Question 2 (iii)

Compare the results of the ideal and particle Marginal Metropolis Hastings. Relate your answer to any theoretical results for particle filters.

Comparing Figure 1 and Figure 3, we can see that the marginal posterior approximations obtained by both ideal marginal MCMC and particle marginal MCMC are consistent with one another to the degree that they are almost indistinguishable.

Ignoring the small bias of the log-likelihood estimate, even were we to decrease N, the number of particles used in the particle approximation, we would expect the particle MCMC posterior approximation to remain similar to the ideal MCMC posterior. This is because the invariant distribution of the Markov chain is not affected by our choice of N. What is affected by using a particle approximation is the MCMC algorithm's mixing properties: high-variance log-likelihood estimates prevent chain transitions and result in a smaller effective sample size, which means more MCMC iterations are necessary to achieve a given approximation accuracy. Furthermore, an important difference between the two approaches is that ideal MMH is less computationally expensive still than pMMH since it does not rely on resampling.

Question 2 (iv) (a)

Implement a particle filter that targets $p(\rho, \tau^2, x_{0:n}|y_{0:n})$. For each of the marginals $p(\rho|y_{0:n})$ and $p(\tau^2|y_{0:n})$ plot the 95% credible intervals and the median for $n = 0, 1, \dots, T$. In your answer explain briefly how the median and credible intervals were computed.

Recall that our importance sampling procedure relies on computing the weights

$$w(x_{0:n}) = \frac{\gamma(x_{0:n})}{q(x_{0:n})} = \prod_{k=0}^{n} \frac{f(x_k|x_{k-1})g(y_k|x_k)}{q(x_k|x_{k-1}, y_k)} \tag{23}$$

where we define $f(x_0|x_{-1}) = f(x_0)$ and $q(x_0|x_{-1}, y_0) = q(x_0|y_0)$. These weights are applied to our particles $x^i_{0:n}$ to yield our density estimate.

To use a particle filter for parameter estimation, we can augment the hidden state with parameter η.

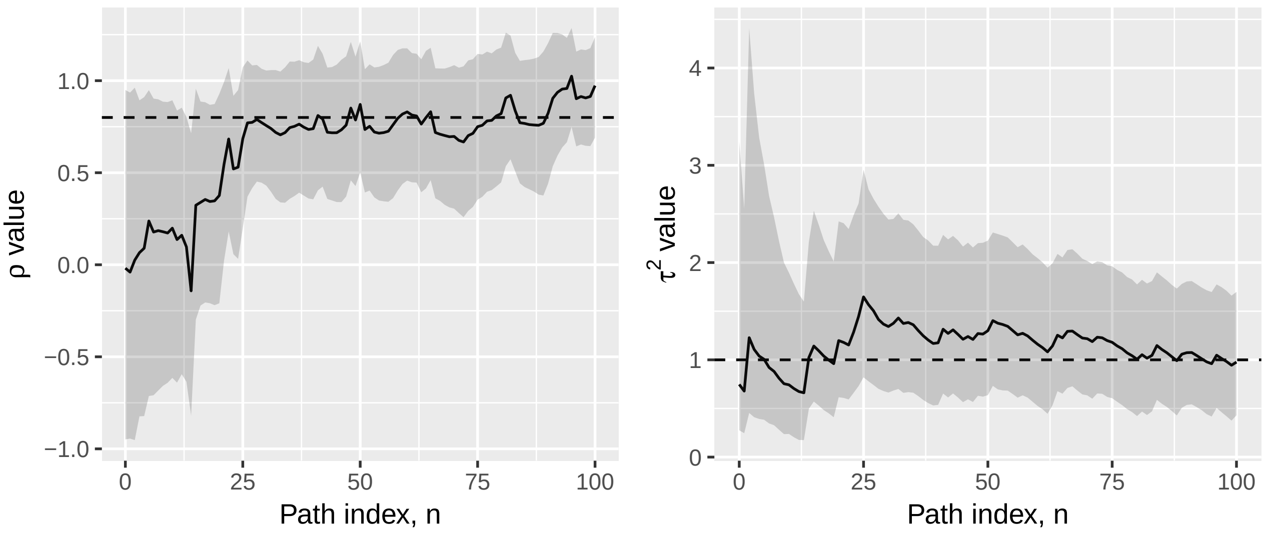

Figure 4: Question 2 (iv) (a) - Plot of 95% credible intervals (shaded interval) and the median (black lines) for particle filter ρ and τ^2 particle distributions at $n = 0, 1, \dots, T$. The dashed lines correspond to the state-space model values of ρ and τ.

The particle filter weight recursion then has the form:

$$w((x,\eta)_{0:n}) = \prod_{k=0}^{n} \frac{f(x_k, \eta_k | x_{k-1}, \eta_{k-1}) g(y_k | x_k, \eta_k)}{q(x_k, \eta_k | x_{k-1}, \eta_{k-1}, y_k)} \tag{24}$$

We specify that η_0 is sampled from the density $p(\eta_0) = p(\rho_0, \tau_0^2)$, a product of $\mathcal{U}[-1,1]$ and $\mathcal{IG}(1,1)$, and that $f(x_k, \eta_k | x_{k-1}, \eta_{k-1}) = f(x_k | x_{k-1}, \eta_{k-1}) \delta_{\eta_{k-1}}(\eta_k)$. We also stipulate $q(x_k, \eta_k | x_{k-1}, \eta_{k-1}, y_k) = q(x_k | x_{k-1}, \eta_{k-1}, y_k) \delta_{\eta_{k-1}}(\eta_k)$. We use an SIR filter with the bootstrap proposal $q(x_k | x_{k-1}, \eta_0, y_k) = f(x_k | x_{k-1}, \eta_0)$.

Implementing the procedure described above resulted in coarse marginal distributions dominated by only a few η values, even for N as large as 100 000. The artificial dynamics:

$$\eta_k = \eta_{k-1} + \epsilon_k \quad : \epsilon \sim \mathcal{N}(0, \sigma_\epsilon^2) \tag{25}$$

were therefore used to improve exploration of the support.

The requested plot is shown in Figure 4. The σ_ϵ value used was 0.05. The median and credible intervals were computed using sample quantiles 0.5, 0.025, and 0.975 from the particle filter collection of ρ, τ^2 values at time T. N was set to 100 000, although a value as high as 625 000 could be justified on the basis that this number would yield an equivalent number of SIR iterations as the pMMH method of Question 2 (ii) (a).

Question 2 (iv) (b)

Produce the same plot as in 2 (ii) (d) above.

The requested plot is shown in Figure 5.

Question 2 (v)

Of the estimate for ρ and τ^2 presented in (ii) and (iv) above, which would you have more faith in and why?

I would have more faith in the MCMC methods, since the artificial dynamics used in the particle filter method are exactly that - artificial! The true state-space model has fixed parameters. Using artificial dynamics means that the marginal distribution of ρ and τ^2 that is constructed is excessively diffuse

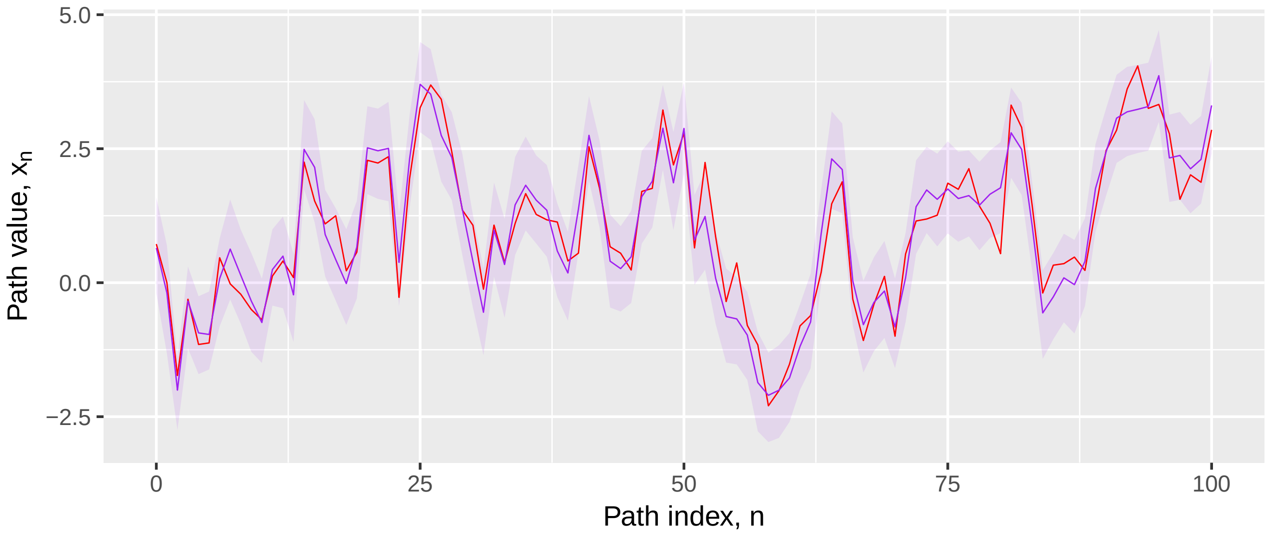

Figure 5: Question 2 (iv) (b) - sample average and variance of particle filter targeting $p(\rho, \tau^2, x_{0:n}|y_{0:n})$. The purple line is an estimate of $x^*_{0:T}$ against time (the sample average of the particles available at time T), together with 95% credible intervals (quantile-based). The red line corresponds to the true hidden state path, $x^*_{0:T}$. This plot is almost indistinguishable from Figure 3 - it was verified that the plots were constructed correctly by checking that the difference in sample average and variance between the algorithms was non-zero.

(see reference [50] of the course notes, Liu and West (2001), *Combined parameter and state estimation in simulation-based filtering*). The MCMC methods, on the other hand, generate samples that are precisely from $p(\rho, \tau^2|y_{0:T})$. Of the two MCMC methods, the ideal one is preferable since it uses the exact value of the log-likelihood, as opposed to the SIR approximation. As has been discussed, the particle method still samples from $p(\rho, \tau^2|y_{0:T})$, but it will tend to have a lower effective sample size for given N than the ideal method (i.e. it its chain will have worse mixing properties).

Question 3

Consider the following scalar model

$$X_n = \rho X_{n-1} + \sigma V_n \tag{26}$$
$$Y_n = \beta \exp(X_n/2) W_n \tag{27}$$

where $W_n, V_n \overset{i.i.d.}{\sim} \mathcal{N}(0,1), X_0 \sim \mathcal{N}\big(0, \frac{\sigma^2}{1-\rho^2}\big)$. *Run the model to synthesise a data-set with* $y_{0:T}$ *for* $T = 100$, $\rho = 0.91$, $\sigma = 1$, $\beta = 0.5$.

Question 3.1 (i)

Assume the static parameters are known. We are interested in performing a fair comparison (i.e. using the same computational budget) for three methods approximating $p(x_{0:T}|y_{0:T})$*:*

(a) *a single forward run of an SIR particle filter,*

(b) *a dedicated particle smoothing algorithm (either fixed-lag smoothing, FFBSm, or FFBSa),*

(c) *particle MCMC.*

Mention what method you implement for (b) and its tuning parameters (size of lag for fixed-lag smoothing or number of sampled paths for FFBSa). State your values of the number of particles used for each method and the number of MCMC iterations for (c). Explain how you came up with all these values, and in particular explain how the comparison is fair. Some hints: if θ *is initialized to its true value, you can use a very moderate total number of particle MCMC iterations.*

The single forward run of the SIR particle filter was implemented using a bootstrap proposal. A fixed-lag smoothing algorithm with $L = 5$ and the bootstrap proposal were chosen for (b). This algorithm was almost identical to the SIR filter, with the caveat that resampling at iteration k takes place over $x^i_{k-L+1:k}$ rather than $x^i_{0:k}$ (as in the filter), a technique that mitigates path degeneracy in the SIR filter approximation to $p(x_{0:T}|y_{0:T})$ when T is large.

Two approaches to particle MCMC were considered. Both methods held the parameter vector θ fixed to its true value and used a Metropolis-Hastings algorithm. The first approach involved sampling an initial path $X^{(0)}_{0:T}$ via SMC then using it to generate proposals, for example by adding Gaussian noise. This procedure seemed likely to suffer from a very low acceptance rate since $X_{0:T}$ is high-dimensional and an arbitrary perturbation to an SMC particle would, in general, therefore be unlikely to constitute a probable sample from $p(x_{0:T}|y_{0:T})$.

Given the Question's hint concerning the 'very moderate' number of MCMC iterations, walking on $X_{0:T}$ was forgone in favour of another strategy: separately run several particle filters, sample a particle from each, then accept these particles with probability 1. This approach can be expressed as a Metropolis-Hastings algorithm, as shown in Algorithm 3. This sampling approach obtains samples from $p_\theta(x_{0:T}|y_{0:T})$ while addressing the path degeneracy problem and is similar to particle independent Metropolis-Hastings[2] with the exception that the acceptance probability is 1 rather than $1 \wedge \frac{\hat{p}_\theta(y_{0:T})\tilde{()}}{\hat{p}_\theta(y_{0:T})(k-1)}$, where the tilde indicates the proposal.

In all three algorithms, the computational budget is expended by executing SIR filter iterations. Let N_f, N_s, and N_m denote the number of SIR iterations executed by the single run of the particle filter, the smoother, and one of the particle filters in the MCMC method respectively. A run of the fixed-lag smoother has the same computational cost as a run of the SIR particle filter, meaning that their costs can be matched by requiring $N_f = N_s$. The MCMC algorithm devised consists of running M SIR particle filters, so the computational budget can be matched by setting N_m such that $MN_m = N_s = N_f$.

[2]See page 276 of *Particle Markov Chain Monte Carlo Methods* by Doucet, Andrieu, and Holenstein.

In the experiments, N_s and N_f were initially set to 10000 such that N_m and M were both set to 100. The value of N_m was chosen on the basis that it is the minimum number of particles that can generate moderately good filter approximations for $T = 100$. A further experiment to illustrate the effect of path degeneracy was run with $N_s = N_f = 100$ and $N_m = M = 10$.

Algorithm 3: Question 3.1 (i) - particle Metropolis-Hastings sampler for the density $p(x_{0:T}|y_{0:T})$.

```
Initialize $\theta_0 = \theta^*$, where $\theta^*$ is the value of $\theta$ in the state-space model.
Run an SIR particle filter with $\theta = \theta_0$ to generate $N$ particles $\{\bar{X}^i_{0:T}\}^{(0)}_{i=1,...,N}$.
Uniformly sample a single value from $\{\bar{X}^i_{0:T}\}^{(0)}_{i=1,...,N}$ and set it to $X^{(0)}_{0:T}$.
for $k$ := 1:$M$ do
    Sample proposal $\tilde{\theta}_k$ according to density $\delta_{\theta_{k-1}}(d\theta)$.
    Run an SIR particle filter with $\theta = \theta_k$ to generate $N$ particles $\{\bar{X}^i_{0:T}\}^{(k)}_{i=1,...,N}$.
    Uniformly sample a single value from $\{\bar{X}^i_{0:T}\}^{(k)}_{i=1,...,N}$ and set it to $\tilde{X}^{(k)}_{0:T}$.
    Accept with probability 1: set $X^{(k)}_{0:T} := \tilde{X}^{(k)}_{0:T}$, $\theta_k := \tilde{\theta}_k$.
```

Question 3.1 (ii), (iii), (iv)

Produce a plot that displays x^*_n and the particle approximations for $\mathrm{E}[X_n|Y_{0:T}]$ against n.

Produce an additional plot that displays the variance of $p(x_n|y_{0:T})$ against n for each method.

Discuss which method between (a), (b), and (c) you prefer in this case and why.

The requested plots are shown for $N = N_f = N_s = 10\ 000$ and $N = N_f = N_s = 100$ in Figures 6 and 7.

We can see that the methods are very similar in terms of their estimates for $\mathrm{E}[X_n|Y_{0:T}]$ and $\mathrm{Var}[X_n|Y_{0:T}]$ for $N = 10\ 000$, but that the SIR filter tends to perform worse (relative to the true path and plausible variance) for $n << T$ when $N = 100$. This is because of path degeneracy: a single particle dominates the SIR filter's approximation for $n \leq 60$, as suggested by the zero-variance estimate in Figure 7, meaning that the path approximation for small n is worse than those of the smoother and MCMC algorithm. For $N = 10\ 000$ i.e. $N >> T$, path degeneracy ceases to be a problem for the filter, which is why its variance and path estimate are very similar to those of the fixed-lag smoother and MCMC method, as seen in the bottom panel of Figure 6. The close agreement between the MCMC method and fixed-lag smoother suggests that the value of L results in negligible bias.

For $N >> T$, it is difficult to argue that any of the three methods is outright preferable. The MCMC method and SIR filter require no tuning, whereas a suitable lag L has to be obtained for the fixed-lag smoother. The smoother can generate good approximations in an on-line setting, however. If T were large such that M had to be small in order to satisfy $N_m \geq T$, then the MCMC approximation to $p(x_{0:T}|y_{0:T})$ would consist of only a few particles and would probably yield inadequate variance estimates. For large T, N_f would be near T too and path degeneracy might be a problem for the SIR filter and the MCMC algorithm. This reasoning suggests that for $T \approx N$, the number of SIR iterations available, the fixed-lag smoother is most appropriate.

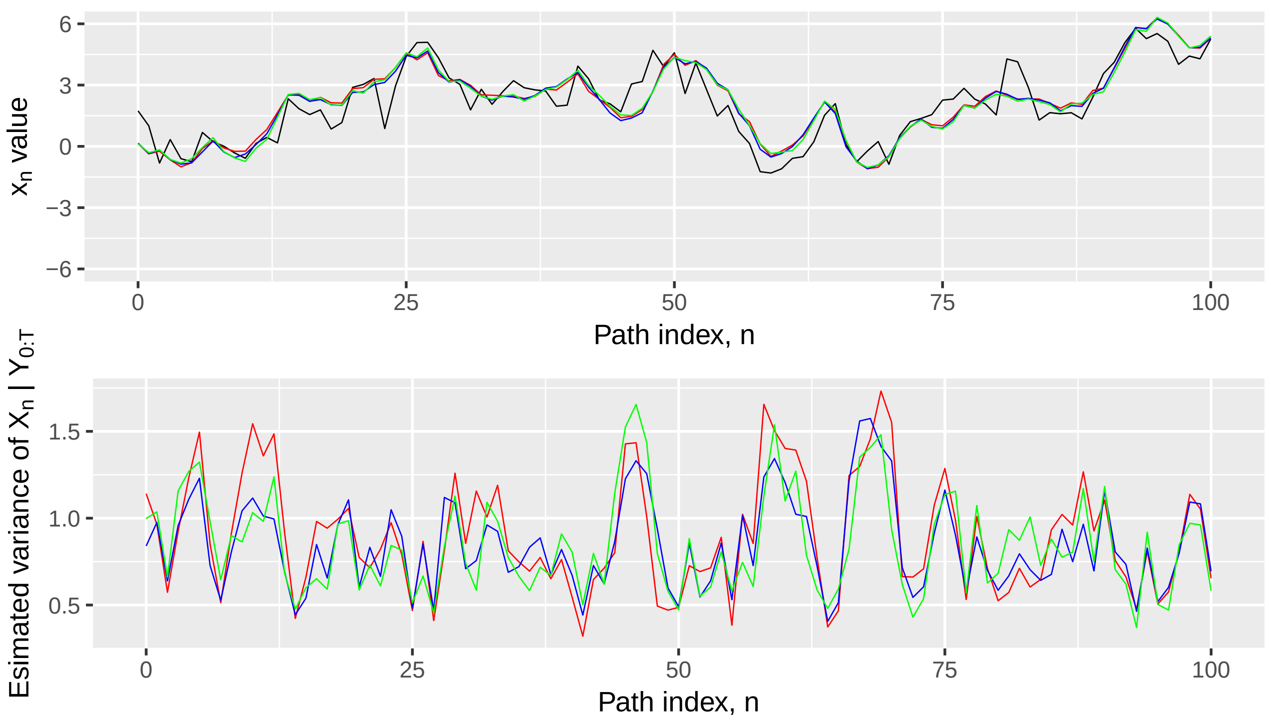

Figure 6: Question 3 Part 1 - results for $N_f = N_s = 10\ 000$, $N_m = 100$, $M = 100$. (Top) Estimate of $\mathrm{E}[X_n|Y_{0:T}]$ as a function of n for the three methods, obtained by taking a sample average of the particles available at time T for the smoother and filter and by averaging the particles collected by the MCMC algorithm. (Bottom) Estimate of $\mathrm{Var}[X_n|Y_{0:T}]$, obtained by computing the variance of the time-T particle approximation at each time-step n. . Red corresponds to the SIR filter. Blue corresponds to the smoother with $L = 5$. Green corresponds to the MCMC method. The black line is $x^*_{0:T}$.

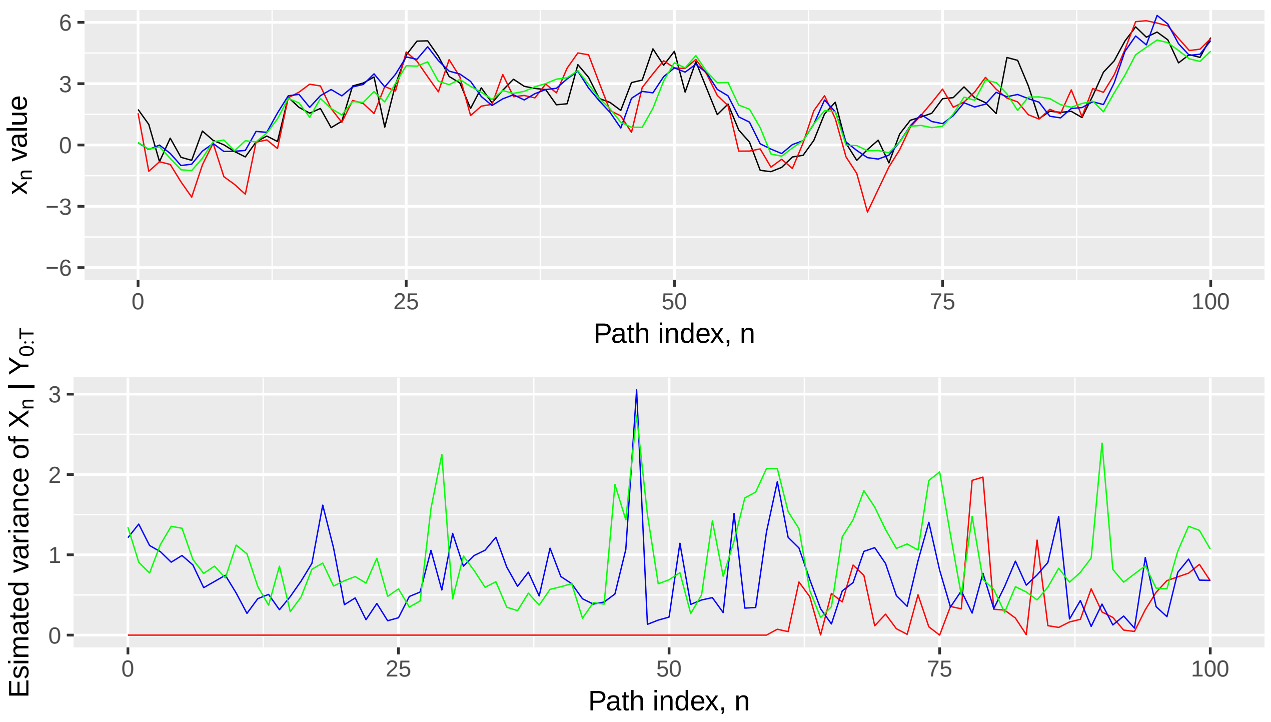

Figure 7: Question 3 Part 1 - results for $N_f = N_s = 100$, $N_m = 10$, $M = 10$. The plots are otherwise identical to those of Figure 7.

Question 3.2 (i)

Assume that ρ, σ^2 are unknown and we would like to perform likelihood inference. Regard β as known and set to its true value.

Implement the algorithm you chose to present in Question 1.3.a and present a figure of your estimates of ρ, σ^2 against each iteration.

The algorithm presented in Question 1.3.a. was gradient ascent. In this question $g(y_k|x_k)$ is the density of $\mathcal{N}\big(0, \beta^2\exp(x_k)\big)$ such that for $\theta = (\rho, \sigma^2)$ we have $\nabla_\theta \log g(y_k|x_k) = \mathbf{0}$. $f(x_k|x_{k-1})$ is the density of $\mathcal{N}(\rho x_{k-1}, \sigma^2)$, implying that:

$$\nabla_\theta \log f(x_k|x_{k-1}) = \begin{pmatrix} \frac{\partial\cdot}{\partial\rho} \\ \frac{\partial\cdot}{\partial\sigma^2} \end{pmatrix} \log f(x_k|x_{k-1}) = \begin{pmatrix} \frac{1}{\sigma^2}x_{k-1}(x_k - \rho x_{k-1}) \\ \frac{1}{2\sigma^2}(-1 + \frac{1}{\sigma^2}(x_k - \rho x_{k-1})^2) \end{pmatrix} \tag{28}$$

Similarly, $f(x_0)$ is the density of $\mathcal{N}\big(0, \frac{\sigma^2}{1-\rho^2}\big)$, from which it follows that:

$$\frac{\partial}{\partial\rho}\log f(x_0) = \frac{\rho s^2}{\sigma^2}\bigg(-1 + \frac{x_0^2}{s^2}\bigg) \tag{29}$$

and

$$\frac{\partial}{\partial\sigma^2}\log f(x_0) = \frac{1}{2\sigma^2}\bigg(-1 + \frac{x_0^2}{s^2}\bigg) \tag{30}$$

where $s^2 = \frac{\sigma^2}{1-\rho^2}$.

The constraints on ρ ($\in (-1, 1)$) and σ^2 (> 0) were respected by performing gradient descent on (θ_1, θ_2) where:

$$\theta_1 = \tanh^{-1}(\rho) \tag{31}$$

$$\theta_2 = \sigma \tag{32}$$

with the gradient of the log-likelihood computed with respect to θ_1 and θ_2 as opposed to ρ and σ^2 by applying the chain rule with:

$$\frac{\partial\rho}{\partial\theta_1} = 1 + \tanh^2(\theta_1) \tag{33}$$

$$\frac{\partial\sigma^2}{\partial\theta_2} = 2\theta_2 \tag{34}$$

Step-sizes of $\gamma_1 = 10^{-2}$ and $\gamma_2 = 10^{-3}$ were obtained by testing values in $\{10^0, 10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}\}$ and selecting those that yielded stable convergence within 200 iterations. To determine appropriate initialization values, the algorithm was run for $K = 200$ iterations across the grid of pairs $\{-0.99, 0, 0.99\} \times \{0.1, 1, 10\}$ of ρ and σ^2 values respectively. This experiment yielded the results displayed in Table 1. For almost all initialization pairs (ρ_0, σ_0^2), the algorithm converges to $(\rho_K, \sigma_K^2) \approx (0.8, 1.5)$, with the results less unanimous w.r.t. σ_K^2, which takes on values between 0.8 and 2.5. The estimated log-likelihood is larger for σ_K^2 in 1-1.5 and takes on its largest value for $(\rho_K, \sigma_K^2) = (0.83, 0.99)$, which was reached from initialization $(\rho_0, \sigma_0^2) = (0.99, 1)$. The standard deviation of the log-likelihood estimate at convergence was approximately 0.3 and was estimated by computing the sample variance across the last 20 iterations of the gradient ascent. Note that this value is sufficiently small for the differences in the estimates of $\log \ell(\theta_K)$ to be deemed meaningful.

Based on the results of the preliminary experiment, the algorithm was re-run using initialization pair $(\rho_0, \sigma_0^2) = (0.99, 0.1)$, $K = 500$ and $N = 1000$. σ_0^2 was not used since this is the true value of σ^2 and arguably provides less insight into the algorithm's behaviour.

Estimates of ρ, σ^2 at each iteration of the algorithm are presented in Figure 8. The value of ρ_k converges towards 0.829, while the value of σ_k^2 converges towards 1.02. The discrepancy between ρ_K and $\rho = 0.91$, the state-space model value, is attributable to sample variance in $y_{0:T}$ - running the gradient ascent against other samples of the observation sequence $y_{0:T}$ consistently yielded $\rho_K \approx 0.91$.

Table 1: Question 3.2 (a) - Experimental results for gradient ascent with different initialization values ρ_0 and σ_0^2. ρ_K and σ_K^2 are the parameter values at termination of gradient ascent after $K = 200$ iterations.

ρ_0	σ_0^2	ρ_K	σ_K^2	$\log \ell(\theta_K)$	$\nabla_\rho \log \ell(\theta_K)$	$\nabla_{\sigma^2} \log \ell(\theta_K)$
0.99	0.1	0.85	0.87	-99.25	-1.94	0.45
0.99	1	0.83	0.99	-98.20	1.17	0.14
0.99	10	0.70	1.94	-98.48	1.86	-1.46
0	0.1	0.85	0.95	-98.42	0.15	0.02
0	1	0.83	1.04	-98.53	-1.08	-0.24
0	10	0.71	1.92	-99.98	0.17	-1.29
-0.99	0.1	0.78	1.34	-98.69	-0.29	-0.45
-0.99	1	0.77	1.38	-99.27	-0.59	-1.22
-0.99	10	0.66	2.38	-99.96	1.55	-1.65

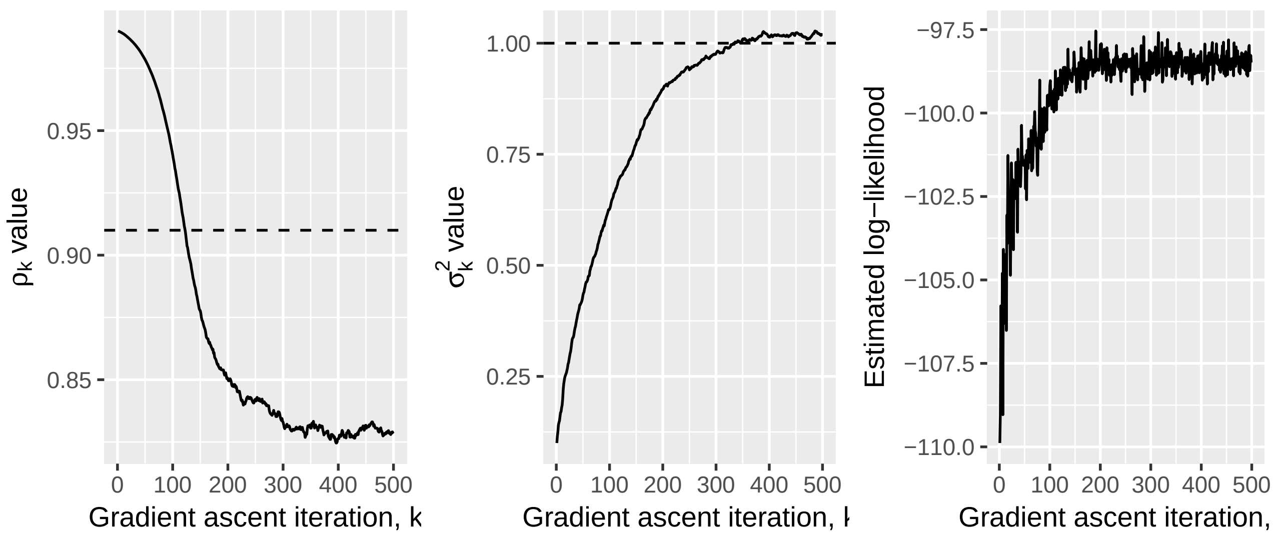

Figure 8: Question 3.2 (a) - Gradient ascent estimate trajectory for $N = 1000$, $K = 500$, $\gamma_1 = 10^{-2}$, $\gamma_2 = 10^{-3}$, $\rho_0 = 0.99$, $\sigma_0^2 = 0.01$. The dashed lines indicate the values of ρ and σ^2 that are used in the state-space model.

Question 3.2 (ii)

Discuss briefly whether and how path degeneracy will affect the performance of this scheme. Mention also a separate/additional weakness of your method and suggest a possible improvement.

Path degeneracy will affect this scheme by way of the filter approximation: for $k << T$, the value of a given $\bar{X}_k^i$ is likely to be shared across many $i \in 1, \dots, N$. This makes our approximation to the gradient of the log-likelihood have a higher mean-squared error, meaning that the gradient is 'noisier', causing the algorithm to take longer to converge or, if the variance of the gradient estimate is excessively high, causing it to fail to converge (see asymptotic variance result on page 42). Furthermore, the variance of the gradient estimate will increase as T grows. This problem could be mitigated by using a smoother (as opposed to a filter) to obtain particles from $p(x_{0:T}|y_{0:T})$, resulting in lower-variance gradient estimates.

An additional weakness of this algorithm is that an appropriate step size sequence $\{\gamma_k\}$ is not obvious and is not necessarily the same for each component of θ. In this study, values for γ were obtained manually. This was laborious and would be undesirable if we wanted to automate the inference process. A possible improvement would be to use Newton's method (which would require the inverse of the Hessian of $\log \ell(\theta)$ to be approximated) or to use the Barzilai-Borwein step-size, which is the solution to the equation:

$$\gamma_k = \arg\min_{\gamma} \left|(\theta_k - \theta_{k-1}) - \gamma\big(\nabla_\theta f(\theta_k) - \nabla_\theta f(\theta_{k-1})\big)\right|_2^2 \tag{35}$$

where in our case $f(\theta) = \log \ell(\theta)$. The solution to this equation is:

$$\gamma_k = \frac{\big(\nabla_\theta f(\theta_k) - \nabla_\theta f(\theta_{k-1})\big)^T (\theta_k - \theta_{k-1})}{|\nabla_\theta f(\theta_k) - \nabla_\theta f(\theta_{k-1})|_2^2} \tag{36}$$

This step-size is motivated by quasi-Newton methods and yields convergence that is often substantially faster than what is obtained by using the Cauchy step size (which involves a line search and is therefore not practical here).

END OF REPORT

Appendix

```
# Question 2 Part 1 - ideal MMH

rm(list=ls())
set.seed(60)
T_    <- 100 # n = 10, 50, 100
rho   <- 0.8
tau   <- 1
sigma <- 0.5

y <- rep(NA, T_+1)
x.star <- rep(NA, T_+1)

# Sample initial state
x.star[1] <- rnorm(1, mean=0, sd=1)
y[1] <- x.star[1] + sigma*rnorm(1, mean=0, sd=1)

# Synthesize a trajectory
for (t in 2:(T_+1)) {
  x.star[t] <- rho*x.star[t-1] + tau*rnorm(1, mean=0, sd=1)
  y[t] <- x.star[t] + sigma*rnorm(1, mean=0, sd=1)
}

# Function for computing exact log-likelihood via Kalman filter
log_kf_likelihood <- function(p_rho, p_tau2) {
  # Computes log(p_sigma(y0:T|rho, tau2)) using the Kalman filter

  # ! Kalman filter ! #
  kf.mu_nln_1 <- rep(NA, T_+1)
  kf.Sigma_nln_1 <- rep(NA, T_+1)

  kf.mu_nln    <- rep(NA, T_+1)
  kf.Sigma_nln <- rep(NA, T_+1)

  kf.m_n <- rep(NA, T_+1)
  kf.S_n <- rep(NA, T_+1)

  kf.K_n <- rep(NA, T_+1)

  A <- p_rho
  B <- sqrt(p_tau2)
  C <- 1
  D <- sigma

  # Recursively compute updates
  kf.mu_nln[1] <- y[1] # EV of p(x0|y0)
  kf.Sigma_nln[1] <- D^2 # Variance of p(x0|y0)

  for (t in 2:(T_+1)){
    # Hidden state predictive density p(x_n|y_0:n-1)
    kf.mu_nln_1[t]    <- A*kf.mu_nln[t-1]
    kf.Sigma_nln_1[t] <- A*kf.Sigma_nln[t-1]*A + B*B
    # Observed state predictive density p(y_n|y_0:n-1)
    kf.m_n[t] <- C*kf.mu_nln_1[t]
    kf.S_n[t] <- C*kf.Sigma_nln_1[t]*C + D*D

    # Filter density p(x_n|y_0:n)
    kf.K_n[t] <- kf.Sigma_nln_1[t]*C/kf.S_n[t]
    kf.mu_nln[t] <- kf.mu_nln_1[t] + kf.K_n[t]*(y[t] - kf.m_n[t])
    kf.Sigma_nln[t] <- kf.Sigma_nln_1[t] - kf.K_n[t]*C*kf.Sigma_nln_1[t]
  }

  # Compute likelihood (may need to log)
  log_py0Tlrhotau2 <- log(dnorm(y[1], mean=0, sd=sqrt(1 + sigma^2))) +
                      sum(log(dnorm(y[2:(T_+1)], mean=kf.m_n[2:(T_+1)],
                                    sd=sqrt(kf.S_n[2:(T_+1)]))))

  return(log_py0Tlrhotau2)
}

# ! MCMC sampler !
# install.packages('invgamma')
require(invgamma)
require(MASS)

# Configure chain
M <- 25000 # 25000 # chain length
varrho <- 0.1 # Random walk step size
eta <- matrix(rep(NA, M*2), ncol=2) # rho left, tau^2 right
log_gamma_eta <- rep(NA, M)

# Initialize chain
eta[1, 1] <- runif(1, min=-1, max=1)
eta[1, 2] <- rinvgamma(1, shape=1, rate=1)
log_gamma_eta[1] <- log_kf_likelihood(eta[1, 1], eta[1, 2]) +
                    log(dunif(eta[1, 1], min=-1, max=1)) +
                    log(dinvgamma(eta[1, 2], shape=1, rate=1)) # likelihood*prior

# Run chain
for (k in 2:M) {
  # Generate proposal via random walk transition
  tilde_eta <- mvrnorm(1, mu=eta[k-1, ], Sigma=(varrho^2)*diag(c(1, 1)))

  # Compute acceptance probability
  if (tilde_eta[2] <= 0){
    alpha <- 0
  }
  else {
    log_gamma_tilde_eta <- log_kf_likelihood(tilde_eta[1], tilde_eta[2]) +
                           log(dunif(tilde_eta[1], min=-1, max=1)) +
                           log(dinvgamma(tilde_eta[2], shape=1, rate=1))
    alpha <- min(1, exp(log_gamma_tilde_eta - log_gamma_eta[k-1]))
  }

  # Accept/reject proposal
```

```r
  if (runif(1) <= alpha) {
    eta[k, ] <- tilde_eta # Store proposal
    log_gamma_eta[k] <-log_gamma_tilde_eta # Store gamma(proposal)
  } else {
    eta[k, ] <- eta[k-1, ]
    log_gamma_eta[k] <- log_gamma_eta[k-1]
  }
}

# Compute acceptance ratio
print(mean(eta[2:M, 1] != eta[1:(M-1), 1]))

# Evaluate effective sample size for marginal distributions
rho_ESS <- M/(1 + 2*sum(rho_acf))
tau2_ESS <- M/(1 + 2*sum(tau2_acf))
```

```r
# Question 2 Part 2 - Particle MMH

rm(list=ls())
set.seed(60)
T_    <- 100 # n = 10, 50, 100
rho   <- 0.8
tau   <- 1
sigma <- 0.5

y <- rep(NA, T_+1)
x.star <- rep(NA, T_+1)

# Sample initial state
x.star[1] <- rnorm(1, mean=0, sd=1)
y[1] <- x.star[1] + sigma*rnorm(1, mean=0, sd=1)

# Synthesize a trajectory
for (t in 2:(T_+1)) {
  x.star[t] <- rho*x.star[t-1] + tau*rnorm(1, mean=0, sd=1)
  y[t] <- x.star[t] + sigma*rnorm(1, mean=0, sd=1)
}

log_sir_likelihood <- function(p_rho, p_tau2) {
  # Computes SIR-based approximation to log-likelihood
  N     <- 350
  sir.x <- matrix(rep(NA, N*(T_+1)), ncol=N)
  sir.w <- matrix(rep(NA, N*(T_+1)), ncol=N)

  # Sample initial state
  sir.x[1, ] <- rnorm(N, mean=0, sd=1)
  sir.w[1, ] <- dnorm(y[1], mean=sir.x[1, ], sd=sigma)
  sir.x      <- sir.x[ , sample(1:N, size=N, replace=T, prob=sir.w[1,])]

  for (k in 2:(T_+1)){
    sir.x[k, ] <- rnorm(N, mean=p_rho*sir.x[k-1, ], sd=sqrt(p_tau2))
    sir.w[k, ] <- dnorm(y[k], mean=sir.x[k, ], sd=sigma)
    sir.x      <- sir.x[ , sample(1:N, size=N, replace=T, prob=sir.w[k, ])]
  }

  # Compute log-likelihood estimate
  hat_log_py0Tlrhotau2 <- sum(log(rowMeans(sir.w)))
  return(hat_log_py0Tlrhotau2)
}

# Tune number of particles, N
var(replicate(100, log_sir_likelihood(rho, tau^2))) # approx. 1 for N = 25

# ! MCMC sampler !
# install.packages('invgamma')
require(invgamma)
require(MASS)

# Configure chain
M <- 25000 # chain length
varrho <- 0.1 # Random walk step size
eta <- matrix(rep(NA, M*2), ncol=2) # rho left, tau^2 right
log_gamma_eta <- rep(NA, M)

# Initialize chain
eta[1, 1] <- runif(1, min=-1, max=1)
eta[1, 2] <- rinvgamma(1, shape=1, rate=1)
log_gamma_eta[1] <- log_sir_likelihood(eta[1, 1], eta[1, 2]) +
  log(dunif(eta[1, 1], min=-1, max=1)) +
  log(dinvgamma(eta[1, 2], shape=1, rate=1)) # likelihood*prior

# Run chain
for (k in 2:M) {
  # Generate proposal via random walk transition
  tilde_eta <- mvrnorm(1, mu=eta[k-1, ], Sigma=(varrho^2)*diag(c(1, 1)))

  # Compute acceptance probability
  if (tilde_eta[2] <= 0){
    alpha <- 0
  }
  else {
    log_gamma_tilde_eta <- log_sir_likelihood(tilde_eta[1], tilde_eta[2]) +
      log(dunif(tilde_eta[1], min=-1, max=1)) +
      log(dinvgamma(tilde_eta[2], shape=1, rate=1))
    alpha <- min(1, exp(log_gamma_tilde_eta - log_gamma_eta[k-1]))
  }

  # Accept/reject proposal
  if (runif(1) <= alpha) {
    eta[k, ] <- tilde_eta # Store proposal
    log_gamma_eta[k] <-log_gamma_tilde_eta # Store gamma(proposal)
  } else {
    eta[k, ] <- eta[k-1, ]
    log_gamma_eta[k] <- log_gamma_eta[k-1]
  }
}
```

```r
# Compute acceptance ratio
print(mean(eta[2:M, 1] != eta[1:(M-1), 1]))

# Evaluate effective sample size for marginal distributions
rho_ESS <- M/(1 + 2*sum(rho_acf))
tau2_ESS <- M/(1 + 2*sum(tau2_acf))
```

```r
# Question 2 Part 3 - same as Part 2 but w/ path sampling & optimal proposal

rm(list=ls())
set.seed(60)
T_    <- 100 # n = 10, 50, 100
rho   <- 0.8
tau   <- 1
sigma <- 0.5

y <- rep(NA, T_+1)
x.star <- rep(NA, T_+1)

# Sample initial state
x.star[1] <- rnorm(1, mean=0, sd=1)
y[1] <- x.star[1] + sigma*rnorm(1, mean=0, sd=1)

# Synthesize a trajectory
for (t in 2:(T_+1)) {
  x.star[t] <- rho*x.star[t-1] + tau*rnorm(1, mean=0, sd=1)
  y[t] <- x.star[t] + sigma*rnorm(1, mean=0, sd=1)
}

sir <- function(p_rho, p_tau2) {
  # Computes SIR-based path sample and approximation to log-likelihood

  # Observed state likelihood density, g(Y_n | X_n)
  g.d <- function(y_n, x_n) dnorm(y_n, mean=x_n, sd=sigma)

  # Initial state density, eta(X_0) = f(X_0)
  nu.d <- function(x_0) dnorm(x_0, mean=0, sd=1)
  nu.r <- function() rnorm(1, mean=0, sd=1)

  # Hidden state transition density, f(X_n | X_{n-1})
  f.d <- function(x_n, x_n_1) dnorm(x_n, mean=p_rho*x_n_1, sd=sqrt(p_tau2))
  f.r <- function(x_n_1) rnorm(1, mean=p_rho*x_n_1, sd=sqrt(p_tau2))

  # Hidden state initial sampling density, q0(X_0|Y_0)
  q0.d <- function(x0, y0) dnorm(x0, mean=y0, sd=sigma)
  q0.r <- function(y0) rnorm(1, mean=y0, sd=sigma)

  # Hidden state transition sampling density, q(X_n | X_{n-1}, Y_n)
  q.d <- function(x_n, x_n_1, y_n) dnorm(x_n, mean=(p_rho*x_n_1*sigma^2 +
                                                 y_n*p_tau2)/(p_tau2 + sigma^2),
                                          sd=sqrt(p_tau2*sigma^2/(p_tau2 + sigma^2)))
  q.r <- function(x_n_1, y_n) rnorm(1, mean=(p_rho*x_n_1*sigma^2 +
                                              y_n*p_tau2)/(p_tau2 + sigma^2),
                                         sd=sqrt(p_tau2*sigma^2/(p_tau2 + sigma^2)))

  # Initialize particle and weight matrix
  N <- 100 # Number of particles
  siro.x <- matrix(rep(NA, N*(T_+1)), nrow=(T_+1)) # Each column is a particle
  siro.w <- matrix(rep(NA, N*(T_+1)), nrow=(T_+1))
  siro.W <- matrix(rep(NA, N*(T_+1)), nrow=(T_+1)) # Normalized w

  # Perform the initial step
  siro.x[1, ] <- sapply(1:N, function(dmmy) q0.r(y[1]))
  siro.w[1, ] <- sapply(siro.x[1, ], nu.d)* # f(x0)
    sapply(siro.x[1, ], function(xi0) g.d(y[1], xi0)) / # g(y0|x0)
    sapply(siro.x[1, ], function(xi0) q0.d(xi0, y[1]) ) # q0(xi0|y0)
  siro.x[1, ] <- siro.x[1, sample(1:N, size=N, replace=T, prob=siro.w[1, ])]

  for (t in 2:(T_+1)){
    # Sample X^i_n from q(X_n | X_{n-1}, Y_n)
    siro.x[t, ] <- sapply(siro.x[t-1, ], function(x_t_1) q.r(x_t_1, y[t]) )

    # Compute importance weight increments
    siro.w[t, ] <- apply(siro.x[(t-1):t, ], 2, function(xi) f.d(xi[2], xi[1]))*
      sapply(siro.x[t, ], function(xi_t) g.d(x_n=xi_t, y_n=y[t]))/
      apply(siro.x[(t-1):t, ], 2, function(xi) q.d(xi[2], xi[1], y[t]))

    # Resample particles
    siro.x <- siro.x[ , sample(1:N, size=N, replace=T, prob=siro.w[t, ])]
  }

  # Compute log-likelihood estimate
  hat_log_py0Tlrhotau2 <- sum(log(rowMeans(siro.w)))

  # Sample path
  tilde_X <- siro.x[ , sample(1:N, size=1)]
  return(list(tilde_X, hat_log_py0Tlrhotau2))
}

# ! MCMC sampler !
# install.packages('invgamma')
require(invgamma)
require(MASS)

# Configure chain
M <- 500 # chain length
varrho <- 0.1 # Random walk step size
eta <- matrix(rep(NA, M*2), ncol=2) # rho left, tau^2 right
X   <- matrix(rep(NA, M*(T_+1)), ncol=T_+1) # each row is a path
log_gamma_eta <- rep(NA, M)

# Initialize chain
eta[1, 1] <- runif(1, min=-1, max=1)
eta[1, 2] <- rinvgamma(1, shape=1, rate=1)
sir_output <- sir(eta[1, 1], eta[1, 2]) # sampled path, log-likelih. est.
X[1, ] <- sir_output[[1]] # sampled path
```

```
log_sir_likelihood <- sir_output[[2]] # log-likelih. est.
log_gamma_eta[1] <- log_sir_likelihood +
  log(dunif(eta[1, 1], min=-1, max=1)) +
  log(dinvgamma(eta[1, 2], shape=1, rate=1)) # likelihood*prior

# Run chain
for (k in 2:M) {
  # Generate proposal via random walk transition
  tilde_eta <- mvrnorm(1, mu=eta[k-1, ], Sigma=(varrho^2)*diag(c(1, 1)))

  # Compute acceptance probability
  if (tilde_eta[2] <= 0){ # if proposed variance is negative
    alpha <- 0
  }
  else {
    sir_output <- sir(tilde_eta[1], tilde_eta[2]) # outputs sampled path, log-likelih. est.
    tilde_X  <- sir_output[[1]]
    log_sir_likelihood <- sir_output[[2]]
    log_gamma_tilde_eta <- log_sir_likelihood +
      log(dunif(tilde_eta[1], min=-1, max=1)) +
      log(dinvgamma(tilde_eta[2], shape=1, rate=1))
    alpha <- min(1, exp(log_gamma_tilde_eta - log_gamma_eta[k-1]))
  }

  # Accept/reject proposal
  if (runif(1) <= alpha) {
    eta[k, ] <- tilde_eta # Store proposal
    X[k, ]   <- tilde_X # Store sampled path
    log_gamma_eta[k] <-log_gamma_tilde_eta # Store gamma(proposal)
  } else {
    eta[k, ] <- eta[k-1, ]
    X[k, ]   <- X[k-1, ]
    log_gamma_eta[k] <- log_gamma_eta[k-1]
  }
}
```

```
# Coursework 2, Question 2, Part 4 (a)
# Artificial dynamics SIR particle filter for parameter estimation

rm(list=ls())
set.seed(60)
T_    <- 100
rho   <- 0.8
tau   <- 1
sigma <- 0.5

y <- rep(NA, T_+1)
x.star <- rep(NA, T_+1)

# Sample initial state
x.star[1] <- rnorm(1, mean=0, sd=1)
y[1] <- x.star[1] + sigma*rnorm(1, mean=0, sd=1)
```

```r
# Synthesise a trajectory
for (t in 2:(T_+1)) {
  x.star[t] <- rho*x.star[t-1] + tau*rnorm(1, mean=0, sd=1)
  y[t] <- x.star[t] + sigma*rnorm(1, mean=0, sd=1)
}

# Initialize particle matrix
require(invgamma)
N        <- 100000 # Number of particles
sir.x    <- matrix(rep(NA, N*(T_+1)), nrow=T_+1)
sir.rho  <- matrix(rep(NA, N*(T_+1)), nrow=T_+1)
sir.tau2 <- matrix(rep(NA, N*(T_+1)), nrow=T_+1)
sir.w    <- rep(NA, N) # weights
sir.EX   <- rep(NA, T_+1)

# Sample initial values
sir.rho[1, ]  <- runif(N, min=-1, max=1)
sir.tau2[1, ] <- rinvgamma(N, shape=1, rate=1)
sir.x[1, ]    <- rnorm(N, mean=0, sd=1)

# Compute weights
p_rho_tau2 <- dunif(sir.rho[1, ], min=-1, max=1)*
               dinvgamma(sir.tau2[1, ], shape=1, rate=1) # p(eta0)
sir.w      <- p_rho_tau2*dnorm(y[1], mean=sir.x[1, ], sd=sigma) # p(eta0)*g(y0 | x0)

# Resample particles and parameters
resample_ix  <- sample(1:N, size=N, replace=T, prob=sir.w)
sir.x        <- sir.x[ , resample_ix]
sir.rho[1, ] <- sir.rho[1, resample_ix]
sir.tau2[1, ] <- sir.tau2[1, resample_ix]
sir.EX[1]    <- mean(sir.x[1, ])

# Run SMC
for (k in 2:(T_+1)) {
  # Sample parameters
  sir.rho[k, ]  <- sir.rho[k-1, ] + rnorm(N, mean=0, sd=0.05)
  sir.tau2[k, ] <- abs(sir.tau2[k-1, ] + rnorm(N, mean=0, sd=0.05))

  # Sample hidden state via transition f(x_k | x_{k-1}, eta_{k-1})
  sir.x[k, ] <- rnorm(N, mean=sir.rho[k, ]*sir.x[k-1, ],
                      sd=sqrt(sir.tau2[k, ]))

  # Compute weights
  sir.w      <- dnorm(y[k], mean=sir.x[k, ], sd=sigma) # g(y_k | x_k)

  # Resample particles and parameters
  resample_ix <- sample(1:N, size=N, replace=T, prob=sir.w)
  sir.x       <- sir.x[ , resample_ix]
  sir.rho[k, ]  <- sir.rho[k, resample_ix]
  sir.tau2[k, ] <- sir.tau2[k, resample_ix]
  sir.EX[k]     <- mean(sir.x[k, ])
}
```

```r
# Coursework 2, Question 3.2 (a) - Experiment script
# Experiment seeking appropriate initialization values
rm(list=ls())
set.seed(69)

rho   <- 0.91
sigma <- 1
beta  <- 0.5
T_    <- 100

# Simulate path
x.star <- rep(NA, T_+1)
y      <- rep(NA, T_+1)

x.star[1] <- rnorm(1, mean=0, sd=sqrt(sigma^2/(1 - rho^2)))
y[1] <- beta*exp(x.star[1]/2)*rnorm(1, mean=0, sd=1)

for (n in 2:(T_+1)) {
  x.star[n] <- rho*x.star[n-1] + sigma*rnorm(1, mean=0, sd=1)
  y[n] <- beta*exp(x.star[n]/2)*rnorm(1, mean=0, sd=1)
}

# ! Gradient ascent for rho and sigma2 !
# SIR-based gradient approximator
get_loglikelihood_gradient <- function(rho_, sigma2_) {
  # Gets approximation of log-likelihood's gradient via SIR filter
  N <- 500 # Number of particles
  sir.x <- matrix(rep(NA, N*(T_+1)), ncol=N) # particles
  sir.w <- matrix(rep(NA, N*(T_+1)), ncol=N) # weight increments
  grad_log_fxk <- matrix(rep(NA, 2*(T_+1)), ncol=2)

  # Use particles to compute gradient approximation
  # Sample initial state
  sir.x[1, ] <- rnorm(N, mean=0, sd=sqrt(sigma2_/(1 - rho_^2)))

  # Compute weight increment (observation likelihood)
  sir.w[1, ] <- dnorm(y[1], mean=0, sd=beta*exp(sir.x[1, ]/2))

  # Resample particles
  sir.x <- sir.x[ , sample(1:N, size=N, replace=T, prob=sir.w[1, ])]

  # Compute grad 1/N sum_i(log f(x^i_0))
  s2 <- sigma2_/(1 - rho_^2)
  grad_log_fxk[1, 1] <- sum(rho_*s2*(-1 + sir.x[1, ]^2/s2)/sigma2_)/N
  grad_log_fxk[1, 2] <- sum(0.5*(-1 + sir.x[1, ]^2/s2)/sigma2_)/N

  for (k in 2:(T_+1)) {
    # Sample particles via transition density
    sir.x[k, ] <- rnorm(N, mean=rho_*sir.x[k-1, ], sd=sqrt(sigma2_))

    # Compute weight increment
    sir.w[k, ] <- dnorm(y[k], mean=0, sd=beta*exp(sir.x[k, ]/2))
    # Resample particles
    sir.x <- sir.x[ , sample(1:N, size=N, replace=T, prob=sir.w[k, ])]

    # Compute grad 1/N sum_i(log f(x^i_k | x^i_{k-1}))
    grad_log_fxk[k, 1] <- sum(sir.x[k-1, ]*(sir.x[k, ] - rho_*
                              sir.x[k-1, ])/sigma2_)/N
    grad_log_fxk[k, 2] <- sum((-1 + (sir.x[k, ] - rho_*sir.x[k-1, ])^2
                              /sigma2_)/(2*sigma2_))/N
  }

  hat_log_like <- sum(log(rowMeans(sir.w)))

  # Return gradient approximation and log-likelihood estimate
  return(c(colSums(grad_log_fxk), hat_log_like))
}

# Configure optimization
K <- 200 # Number of iterations of g.a - 1
results <- c() # rho0, sigma20, rho_con, sigma2_con,
               # log_like, rho_grad_con, sigma2_grad_con
list_rho_0 <- c(0.99, 0, -0.99)
list_sigma2_0 <- c(0.1, 1, 10)

list_rho_0_sigma2_0 <- c()
for (r in list_rho_0) {
  for (s in list_sigma2_0) {
    list_rho_0_sigma2_0 <- rbind(list_rho_0_sigma2_0, c(r, s))
  }
}

for (e in 1:nrow(list_rho_0_sigma2_0)) { # e is experiment index
  theta <- matrix(rep(NA, K*2), ncol=2) # first col is tanh-1(rho), second is sigma
  grad_ll  <- matrix(rep(NA, K*3), ncol=3) # grad_rho, grad_sigma2, log_like
  h_grad   <- matrix(rep(NA, K*2), ncol=2)
  log_like <- rep(NA, K) # log-likelihood estimates
  gamma1 <- 1e-2 # Step size in theta1
  gamma2 <- 1e-3 # Step size in theta2
  rho_0 <- list_rho_0_sigma2_0[e, 1] # -0.99, 0, 0.99
  sigma2_0 <- list_rho_0_sigma2_0[e, 2] # 0.1, 1, 10
  theta[1, 1] <- atanh(rho_0) # Initial atanh(rho) value; rho = tanh(theta1)
  theta[1, 2] <- sqrt(sigma2_0) # Initial sigma value; theta2^2 = sigma^2

  for (j in 2:K) {
    # Compute gradient approximation
    grad_ll[j-1, ]  <- get_loglikelihood_gradient(
                          tanh(theta[j-1, 1]), theta[j-1, 2]^2)
    h_grad[j-1, 1] <- (1 - tanh(theta[j-1, 1])^2)*grad_ll[j-1, 1]
    h_grad[j-1, 2] <- 2*theta[j-1, 2]*grad_ll[j-1, 2]
    log_like[j-1]  <- grad_ll[j-1, 3]

    # Update theta
    theta[j, 1] <- theta[j-1, 1] + gamma1*h_grad[j-1, 1]
    theta[j, 2] <- theta[j-1, 2] + gamma2*h_grad[j-1, 2]
  }

  # Gradient and log-likelihood at termination
  grad_ll[K, ]  <- get_loglikelihood_gradient(tanh(theta[K, 1]), theta[K, 2]^2)
  log_like[K]   <- grad_ll[K, 3]

  # rho0, sigma20, rho_con, sigma2_con,
  # log_like, rho_grad_con, sigma2_grad_con
  results <- cbind(results, c(rho_0, sigma2_0, tanh(theta[K, 1]),
                              theta[K, 2]^2, log_like[K], grad_ll[K, 1],
                              grad_ll[K, 2]))
}
```

```
# Coursework 2, Question 3.2 (a) - gradient ascent on log-likelihood
rm(list=ls())
set.seed(69)

rho   <- 0.91
sigma <- 1
beta  <- 0.5
T_    <- 100

# Simulate path
x.star <- rep(NA, T_+1)
y      <- rep(NA, T_+1)

x.star[1] <- rnorm(1, mean=0, sd=sqrt(sigma^2/(1 - rho^2)))
y[1] <- beta*exp(x.star[1]/2)*rnorm(1, mean=0, sd=1)

for (n in 2:(T_+1)) {
  x.star[n] <- rho*x.star[n-1] + sigma*rnorm(1, mean=0, sd=1)
  y[n] <- beta*exp(x.star[n]/2)*rnorm(1, mean=0, sd=1)
}

# ! Gradient ascent for rho and sigma2 !
# SIR-based gradient approximator
get_loglikelihood_gradient <- function(rho_, sigma2_) {
  # Gets approximation of log-likelihood's gradient via SIR filter
  N <- 1000 # Number of particles
  sir.x <- matrix(rep(NA, N*(T_+1)), ncol=N) # particles
  sir.w <- matrix(rep(NA, N*(T_+1)), ncol=N) # weight increments
  grad_log_fxk <- matrix(rep(NA, 2*(T_+1)), ncol=2)

  # Use particles to compute gradient approximation
  # Sample initial state
  sir.x[1, ] <- rnorm(N, mean=0, sd=sqrt(sigma2_/(1 - rho_^2)))

  # Compute weight increment (observation likelihood)
  sir.w[1, ] <- dnorm(y[1], mean=0, sd=beta*exp(sir.x[1, ]/2))

  # Resample particles
  sir.x <- sir.x[ , sample(1:N, size=N, replace=T, prob=sir.w[1, ])]
  # Compute grad 1/N sum_i(log f(x^i_0))
  s2 <- sigma2_/(1 - rho_^2)
  grad_log_fxk[1, 1] <- sum(rho_*s2*(-1 + sir.x[1, ]^2/s2)/sigma2_)/N
  grad_log_fxk[1, 2] <- sum(0.5*(-1 + sir.x[1, ]^2/s2)/sigma2_)/N

  for (k in 2:(T_+1)) {
    # Sample particles via transition density
    sir.x[k, ] <- rnorm(N, mean=rho_*sir.x[k-1, ], sd=sqrt(sigma2_))

    # Compute weight increment
    sir.w[k, ] <- dnorm(y[k], mean=0, sd=beta*exp(sir.x[k, ]/2))

    # Resample particles
    sir.x <- sir.x[ , sample(1:N, size=N, replace=T, prob=sir.w[k, ])]

    # Compute grad 1/N sum_i(log f(x^i_k | x^i_{k-1}))
    grad_log_fxk[k, 1] <- sum(sir.x[k-1, ]*(sir.x[k, ] - rho_*
                              sir.x[k-1, ])/sigma2_)/N
    grad_log_fxk[k, 2] <- sum((-1 + (sir.x[k, ] - rho_*sir.x[k-1, ])^2/
                                 sigma2_)/(2*sigma2_))/N
  }s

  hat_log_like <- sum(log(rowMeans(sir.w)))

  # Return gradient approximation and log-likelihood estimate
  return(c(colSums(grad_log_fxk), hat_log_like))
}

# Configure optimization
K <- 500 # Number of iterations of g.a - 1
theta <- matrix(rep(NA, K*2), ncol=2) # first col is tanh-1(rho), second is sigma
grad_ll  <- matrix(rep(NA, K*3), ncol=3) # grad_rho, grad_sigma2, log_like
h_grad   <- matrix(rep(NA, K*2), ncol=2)
log_like <- rep(NA, K) # log-likelihood estimates
gamma1 <- 1e-2 # Step size in theta1
gamma2 <- 1e-3 # Step size in theta2
rho_0 <- 0.99 # -0.99, 0, 0.99
sigma2_0 <- 0.1 # 0.1, 1, 10
theta[1, 1] <- atanh(rho_0) # Initial atanh(rho) value; rho = tanh(theta1)
theta[1, 2] <- sqrt(sigma2_0) # Initial sigma value; theta2^2 = sigma^2

for (j in 2:K) {
  # Compute gradient approximation
  grad_ll[j-1, ]  <- get_loglikelihood_gradient(
                       tanh(theta[j-1, 1]), theta[j-1, 2]^2)
  h_grad[j-1, 1] <- (1 - tanh(theta[j-1, 1])^2)*grad_ll[j-1, 1]
  h_grad[j-1, 2] <- 2*theta[j-1, 2]*grad_ll[j-1, 2]
  log_like[j-1]  <- grad_ll[j-1, 3]

  # Update theta
  theta[j, 1] <- theta[j-1, 1] + gamma1*h_grad[j-1, 1]
  theta[j, 2] <- theta[j-1, 2] + gamma2*h_grad[j-1, 2]
}
```