M5MS16 Advanced Simulation - Coursework 1

Question 1

Question 1 (i)

The following equations

$$\mathbf{K}_n = \boldsymbol{\Sigma}_{n|n-1}\mathbf{C}^T\mathbf{S}_n^{-1} \tag{1}$$
$$\boldsymbol{\mu}_{n|n} = \boldsymbol{\mu}_{n|n-1} + \mathbf{K}_n(\mathbf{y}_n - \mathbf{m}_n) \tag{2}$$
$$\boldsymbol{\Sigma}_{n|n} = \boldsymbol{\Sigma}_{n|n-1} - \mathbf{K}_n\mathbf{C}\boldsymbol{\Sigma}_{n|n-1} \tag{3}$$

appear in the Kalman filter.

(a) What quantities are they meant to approximate and state the state space model equations in the question?

(b) Derive the expressions for $\boldsymbol{\mu}_{n|n}$ and $\boldsymbol{\Sigma}_{n|n}$.

We answer both of these questions by deriving the Kalman filter.

Consider a Bayesian filtering problem involving hidden state path $\mathbf{x}_{0:n}$ and observation path $\mathbf{y}_{0:n}$, both of which are sequences of random vectors. Assume that the state-space model is linear:

$$\mathbf{x}_n = \mathbf{A}\mathbf{x}_{n-1} + \mathbf{B}\mathbf{w}_n \tag{4}$$
$$\mathbf{y}_n = \mathbf{C}\mathbf{x}_n + \mathbf{D}\mathbf{v}_n \tag{5}$$

where $\mathbf{w}_n \overset{i.i.d.}{\sim} \mathcal{N}(\mathbf{0}, I)$, $\mathbf{v}_n \overset{i.i.d.}{\sim} \mathcal{N}(\mathbf{0}, I)$, and $\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}$ are fixed matrices. Assume that $\mathbf{x}_0 \sim \mathcal{N}(\mathbf{0}, I)$. Our objective is to recursively compute the filter density $p(x_n|y_{0:n})$, where x_n is in the support of $\mathbf{x}_n$ and $y_{0:n}$ is in the support of $\mathbf{y}_{0:n}$.

Equation 4 implies that the distribution of $\mathbf{x}_n$ conditional on $\mathbf{x}_{n-1}$ is $\mathcal{N}(\mathbf{A}\mathbf{x}_{n-1}, \mathbf{B}\mathbf{B}^T)$. Denote the density associated with this distribution by $f(x_n|x_{n-1})$. Similarly, Equation 5 implies that the distribution of $\mathbf{y}_n$ conditional on $\mathbf{x}_n$ is $\mathcal{N}(\mathbf{C}\mathbf{x}_n, \mathbf{D}\mathbf{D}^T)$, which has associated density function $g(y_n|x_n)$.

As a result of Equation 5, $\mathbf{x}_0|\mathbf{y}_0$ has distribution $\mathcal{N}(\mathbf{C}^{-1}\mathbf{y}_0, \mathbf{C}^{-1}\mathbf{D}(\mathbf{C}^{-1}\mathbf{D})^T)$. Assume that the filter $p(x_{n-1}|y_{0:n-1})$ is known be the density of $\mathcal{N}(\boldsymbol{\mu}_{n-1|n-1}, \boldsymbol{\Sigma}_{n-1|n-1})$. Equation 4 implies that $\mathbf{x}_n$ conditional on $\mathbf{y}_{0:n-1}$ is the sum of Gaussian random vectors $\mathbf{A}\mathbf{x}_{n-1}$ and $\mathbf{B}\mathbf{w}_n$. It follows that $\mathbf{x}_n$ conditional on $\mathbf{y}_{0:n-1}$ has distribution $\mathcal{N}(\boldsymbol{\mu}_{n|n-1}, \boldsymbol{\Sigma}_{n|n-1})$, where:

$$\begin{aligned}\boldsymbol{\mu}_{n|n-1} &= \mathrm{E}[\mathbf{x}_n|\mathbf{y}_{0:n-1}] \\ &= \mathrm{E}[\mathbf{A}\mathbf{x}_{n-1} + \mathbf{B}\mathbf{w}_n|\mathbf{y}_{0:n-1}] \\ &= \mathbf{A}\boldsymbol{\mu}_{n-1|n-1} \end{aligned} \tag{6}$$
$$\begin{aligned}\boldsymbol{\Sigma}_{n|n-1} &= \mathrm{E}[(\mathbf{A}\mathbf{x}_{n-1} + \mathbf{B}\mathbf{w}_n)(\mathbf{A}\mathbf{x}_{n-1} + \mathbf{B}\mathbf{w}_n)^T|\mathbf{y}_{0:n-1}] \\ &= \mathbf{A}\boldsymbol{\Sigma}_{n-1|n-1}\mathbf{A}^T + \mathbf{B}\mathbf{B}^T \end{aligned} \tag{7}$$

Knowing the distribution of $\mathbf{x}_n|\mathbf{y}_{0:n-1}$ allows us to compute the distribution of $\mathbf{y}_n|\mathbf{y}_{0:n-1}$. Equation 5 implies that $\mathbf{y}_n|\mathbf{y}_{0:n-1}$ is a Gaussian random vector with distribution $\mathcal{N}(\mathbf{m}_n, \mathbf{S}_n)$, where:

$$\begin{aligned}\mathbf{m}_n &= \mathrm{E}[\mathbf{y}_n|\mathbf{y}_{0:n-1}] \\ &= \mathrm{E}[\mathbf{C}\mathbf{x}_n + \mathbf{D}\mathbf{v}_n|\mathbf{y}_{0:n-1}] \\ &= \mathbf{C}\boldsymbol{\mu}_{n|n-1} \end{aligned} \tag{8}$$
$$\begin{aligned}\mathbf{S}_n &= \mathrm{E}[(\mathbf{C}\mathbf{x}_n + \mathbf{D}\mathbf{v}_n)(\mathbf{C}\mathbf{x}_n + \mathbf{D}\mathbf{v}_n)^T|\mathbf{y}_{0:n-1}] \\ &= \mathbf{C}\boldsymbol{\Sigma}_{n|n-1}\mathbf{C}^T + \mathbf{D}\mathbf{D}^T \end{aligned} \tag{9}$$

We can then use the distribution of $\mathbf{y}_n|\mathbf{y}_{0:n-1}$ to derive the filter distribution $\mathbf{x}_n|\mathbf{y}_{0:n}$ by considering the joint distribution of $\mathbf{x}_n, \mathbf{y}_n$ conditional on $\mathbf{y}_{0:n-1}$:

$$\mathcal{N}\left(\begin{bmatrix} \boldsymbol{\mu}_{n|n-1} \\ \mathbf{m}_n \end{bmatrix}, \begin{bmatrix} \boldsymbol{\Sigma}_{n|n-1} & \tilde{\mathbf{K}}_n \\ \tilde{\mathbf{K}}_n^T & \mathbf{S}_n \end{bmatrix} \right) \tag{10}$$

where:

$$\begin{aligned} \tilde{\mathbf{K}}_n = \mathrm{Cov}(\mathbf{x}_n, \mathbf{y}_n|\mathbf{y}_{0:n-1}) &= \mathrm{E}[(\mathbf{x}_n - \boldsymbol{\mu}_{n|n-1})(\mathbf{y}_n - \mathbf{m}_n)^T|\mathbf{y}_{0:n-1}] \\ &= \mathrm{E}[\mathbf{x}_n\mathbf{y}_n^T|\mathbf{y}_{0:n-1}] - \boldsymbol{\mu}_{n|n-1}\mathbf{m}_n^T \\ &= \mathrm{E}[\mathbf{x}_n(\mathbf{C}\mathbf{x}_n + \mathbf{D}\mathbf{v}_n)^T|\mathbf{y}_{0:n-1}] - \boldsymbol{\mu}_{n|n-1}\mathbf{m}_n^T \\ &= \mathrm{E}[\mathbf{x}_n\mathbf{x}_n^T|\mathbf{y}_{0:n-1}]\mathbf{C}^T - \boldsymbol{\mu}_{n|n-1}\mathbf{m}_n^T \\ &= (\boldsymbol{\Sigma}_{n|n-1} + \boldsymbol{\mu}_{n|n-1}\boldsymbol{\mu}_{n|n-1}^T)\mathbf{C}^T - \boldsymbol{\mu}_{n|n-1}\mathbf{m}_n^T \\ &= \boldsymbol{\Sigma}_{n|n-1}\mathbf{C}^T \end{aligned} \tag{11}$$

The Gaussian conditioning lemma can then be applied to Equation 10 to reveal that the distribution of $\mathbf{x}_n$ conditional on $\mathbf{y}_{0:n}$ is Gaussian with expectation:

$$\boldsymbol{\mu}_{n|n} = \boldsymbol{\mu}_{n|n-1} + \tilde{\mathbf{K}}_n\mathbf{S}_n^{-1}(\mathbf{y}_n - \mathbf{m}_n) \tag{12}$$

and covariance matrix:

$$\boldsymbol{\Sigma}_{n|n} = \boldsymbol{\Sigma}_{n|n-1} - \tilde{\mathbf{K}}_n\mathbf{S}_n^{-1}\tilde{\mathbf{K}}_n^T \tag{13}$$

Define the Kalman gain as:

$$\mathbf{K}_n = \tilde{\mathbf{K}}_n\mathbf{S}_n^{-1} \tag{14}$$

to obtain the Kalman filter update equation as it is stated in the question:

$$\boldsymbol{\mu}_{n|n} = \boldsymbol{\mu}_{n|n-1} + \mathbf{K}(\mathbf{y}_n - \mathbf{m}_n) \tag{15}$$

$$\boldsymbol{\Sigma}_{n|n} = \boldsymbol{\Sigma}_{n|n-1} - \mathbf{K}_n\mathbf{C}\boldsymbol{\Sigma}_{n|n-1} \tag{16}$$

In summary, the Kalman gain $\mathbf{K}_n$ describes how conditioning on the observation $\mathbf{y}_n$ affects the covariance of the predictive distribution. $\boldsymbol{\mu}_{n|n}$ and $\Sigma_{n|n}$ are the expected value and covariance of the filter $p(x_n|y_{0:n})$ respectively.

Question 1 (ii)

Mention a deterministic approximation to the non-linear filtering problem and state three advantages for it.

The extended Kalman filter is a deterministic approximation to the non-linear filtering problem. Non-linear state-space equations

$$\mathbf{x}_n = \phi(\mathbf{x}_{n-1}, \mathbf{w}_n) \tag{17}$$

$$\mathbf{y}_n = \psi(\mathbf{x}_n, \mathbf{v}_n) \tag{18}$$

can be approximated using their linear expansions around the points

$$(\mathbf{x}_{n-1} = \mu_{n-1|n-1}, \mathbf{w}_n = 0) \tag{19}$$

$$(\mathbf{x}_n = \mu_{n|n-1}, \mathbf{v}_n = 0) \tag{20}$$

respectively. The Kalman filter is then the deterministic solution to linear filtering problem and represents an approximate solution to the corresponding non-linear filtering problem.

The advantages of the unscented Kalman filter are similar to those of the Kalman filter. It is easy to implement, it is deterministic, the computational cost of updates does not vary with trajectory length, and the computational cost of updates is relatively low.

Question 1 (iii)

State two advantages and two disadvantages of using MCMC for sampling from a target distribution of the form $\pi = \frac{\gamma}{Z}$, where Z is an unknown normalising constant.

Markov-chain Monte Carlo (MCMC) methods require a burn-in period to be specified and are not guaranteed to explore the entire support of the target distribution (especially if it is not unimodal and/or the chain relies on an inappropriate proposal mechanism). Insufficiently long chains that suffer from high autocovariance are also unlikely to have an empirical distribution that accurately approximates the target distribution. These are the disadvantages of MCMC.

MCMC methods have two key advantages. First, they permit a density π to be sampled from when only $Z\pi$ is known, where Z is some unknown constant. This is especially useful in the context of analytically intractable Bayesian inference. The second major advantage of MCMC methods is that, unlike quadrature methods, the variance of Monte Carlo estimates of expected values decreases at a rate ($\propto N^{-1/2}$) that is independent of problem dimension (i.e. the dimension of the distribution being sampled from).

Question 1 (iv)

What algorithm is this code implementing? Also state what are the variables `rho`, `muCond`, `varCond` *and* `x`?

The algorithms appears to be for a Gibbs sampler with stationary distribution:

$$\mathcal{N}\left(\begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix}, \begin{bmatrix} 1 & \rho \\ \rho & 1 \end{bmatrix} \right) \tag{21}$$

The initial value (x_1, x_2) is drawn the product of two Uniform$(-3, 3)$ distributions. Let $\mu = (0, 0)$, $\rho = \rho_1 = 0.8 = \rho_2 = 0.8$. The equations implemented in the code are:

$$\mu_c = \mu_1 + \rho_1(x_{t-1}^2 - \mu_2) \tag{22}$$

$$\sigma_c = \sqrt{1 - \rho_1^2} \tag{23}$$

$$x_t^1 \sim \mathcal{N}(\mu_c, \sigma_c^2) \tag{24}$$

and

$$\mu_c = \mu_2 + \rho_2(x_t^1 - \mu_1) \tag{25}$$

$$\sigma_c = \sqrt{1 - \rho_2^2} \tag{26}$$

$$x_t^2 \sim \mathcal{N}(\mu_c, \sigma_c^2) \tag{27}$$

The above equations imply that the sampler's proposal distributions are:

$$X_t^1 \sim \pi_1\big(\cdot \,|X_{t-1}^2) = \mathcal{N}\Big(\mu_1 + \rho_1(X_{t-1}^2 - \mu_2),\ 1 - \rho_1^2 \Big) \tag{28}$$

$$X_t^2 \sim \pi_2(\cdot|X_t^1) = \mathcal{N}\Big(\mu_2 + \rho_2(X_t^1 - \mu_1),\ 1 - \rho_2^2 \Big) \tag{29}$$

`rho` determines the covariance of the target distribution. `muCond` and `varCond` correspond to the mean and variance of the normal distribution from which a sample is drawn. `x` is a matrix of samples.

Question 2

Consider the following scalar linear Gaussian model

$$X_n = \rho X_{n-1} + \tau V_n \tag{30}$$

$$Y_n = X_n + \sigma W_n \tag{31}$$

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 with* $y_{0:T}$ *for* $T = 100$, $\rho = 0.8$, $\tau = 1$, $\sigma = 0.5$. *Denote the real state trajectory* $x^*_{0:T}$. *You are asked to implement Monte Carlo filters for this model.*

Question 2 (i)

Write down an expression for the optimal importance density w.r.t. the variance of the importance weights. Denote it as $q^{opt}(x_n|y_n, x_{n-1})$.

Referring to page 19 of the course notes, the optimal importance density w.r.t. the variance of the importance weights is

$$q^{opt}(x_n|y_n, x_{n-1}) = p(x_n|y_n, x_{n-1}) \tag{32}$$

The joint distribution of (X_n, Y_n) given X_{n-1} is

$$\mathcal{N}\left(\begin{bmatrix} \rho X_{n-1} \\ \rho X_{n-1} \end{bmatrix}, \begin{bmatrix} \tau^2 & \tau^2 \\ \tau^2 & \tau^2 + \sigma^2 \end{bmatrix} \right) \tag{33}$$

It follows that $X_n|Y_n, X_{n-1}$ has distribution:

$$\mathcal{N}\left(\rho X_{n-1} + \frac{\tau^2}{\tau^2 + \sigma^2}(Y_n - \rho X_{n-1}),\ \tau^2\big(1 - \frac{\tau^2}{\tau^2 + \sigma^2}\big) \right) \tag{34}$$

The former expression is:

$$\frac{\sigma^2 \rho X_{n-1} + \tau^2 Y_n}{\tau^2 + \sigma^2} \tag{35}$$

While the latter is:

$$\frac{\tau^2 \sigma^2}{\tau^2 + \sigma^2} \tag{36}$$

The optimal initial proposal density is $p(x_0|y_0)$, which for this problem is the density of $\mathcal{N}(y_0, \sigma^2)$.

Question 2 (ii)

Implement the Kalman filter, SIS, and SIR for $N = 10,\ 100,\ 500$ *using the following proposals:* $f(x_n|x_{n-1}),\ q^{opt}(x_n|y_n, x_{n-1})$.

Code implementing the requested filters is available in the Appendix.

Question 2 (ii) (a)

Briefly mention one advantage and disadvantage of each of the importance densities.

The advantage of using the bootstrap proposal $f(x_n|x_{n-1})$ is that it is readily available, is simple to sample according to, and is computationally inexpensive to evaluate. The disadvantage of using the bootstrap proposal is that it is not optimal in the sense of minimizing the variance of the importance weights, especially when the observation y_n is informative with respect to the value of x_n. This means that the rate of decay for the majority of the importance weights is higher than it might be otherwise within SIS, further meaning that the SIS filter is a poor approximation for $p(x_n|y_{0:n})$.

The advantage of using the optimal importance density q^{opt} is that it minimizes the variance of the importance weights, meaning that the SIS weights decay less quickly and the filter therefore performs better. The disadvantage of using the optimal importance density is that it cannot in general be obtained analytically for non-linear filtering problems.

Question 2 (ii) (b)

Plot the SIS and SIR errors against n when estimating $\mathrm{Var}[X_n|Y_{0:n}]$. *Write down how you compute this error. Using numerical results or otherwise, comment on whether the first or second moment of* $p(X_n|Y_{0:n})$ *dominates in the plotted errors.*

Since we are working a linear Gaussian state-space model, the filter $p(x_n|y_{0:n})$ is a Kalman filter. This implies that:

$$\mathrm{Var}[X_n|Y_{0:n}] = \mathbf{\Sigma}_{n|n} \tag{37}$$

To compute the SIS and SIR estimate of $\mathrm{Var}[X_n|Y_{0:n}]$ we note that:

$$\mathrm{Var}[X_n|Y_{0:n}] = \mathrm{E}[X_n^2|Y_{0:n}] - \mathrm{E}[X_n|Y_{0:n}]^2 \tag{38}$$

and that the first and second moments of the filter $p(x_n|y_{0:n})$ are:

$$\mathrm{E}[X_n|Y_{0:n}] = \mu_{n|n} \tag{39}$$

$$\mathrm{E}[X_n^2|Y_{0:n}] = \mathrm{Var}[X_n|Y_{0:n}] + \mu_{n|n}^2 \tag{40}$$

Then use the SIS and SIR moment estimators:

$$\mathrm{E}[X_n^2|Y_{0:n}] \approx \sum_{i=1}^{N} W_n^i (X_n^i)^2 \tag{41}$$

$$\mathrm{E}[X_n|Y_{0:n}] \approx \sum_{i=1}^{N} W_n^i X_n^i \tag{42}$$

The SIS and SIR[1] moment estimates differ on account of the way in which the samples X_n^i are generated. The SIR samples are generated based on particles that have been the result of successive resampling and branching. The SIS samples are generated using particles that have all existed separately of one another since the filter was initialized.

The error of the variance estimates is computed according to:

$$\Sigma_{n|n} - \Big(\sum_{i=1}^{N} W_n^i (X_n^i)^2 - \big(\sum_{i=1}^{N} W_n^i X_n^i \big)^2 \Big) \tag{43}$$

While the errors of the second and first moment estimates are computed as

$$\sum_{i=1}^{N} W_n^i (X_n^i)^2 - \big(\mathrm{Var}[X_n|Y_{0:n}] + \mu_{n|n}^2 \big) \tag{44}$$

$$\big(\sum_{i=1}^{N} W_n^i X_n^i \big)^2 - \mu_{n|n}^2 \tag{45}$$

respectively. The error in estimating the first moment squared is evaluated so that its contribution to error in estimating the variance is more clearly delineated.

Figures 1, 2, and 3 contain plots of the error of the SIS and SIR filter variance estimates for sampling densities q^{opt} and f under $N = 500$, 100, and 10 respectively. The second and third columns of these matrices contain plots of the error of the SIS and SIR estimates for the filters squared first moment and second moment with path index n. We can see that for both filter approximations, the error of the squared first moment tends to share the same sign as the error of the second moment. This is fortuitous since the variance estimate is the difference between the second moment and first moment squared, meaning that the errors in the moment estimates tend to mitigate each other. To say that one dominates the other in the estimate of the variance would be inaccurate - both the second moment estimate and squared first moment estimate might be grossly inaccurate but be such that when their difference is taken, an excellent estimate of the variance is obtained.

[1] NB the SIR estimate is computed prior to resampling.

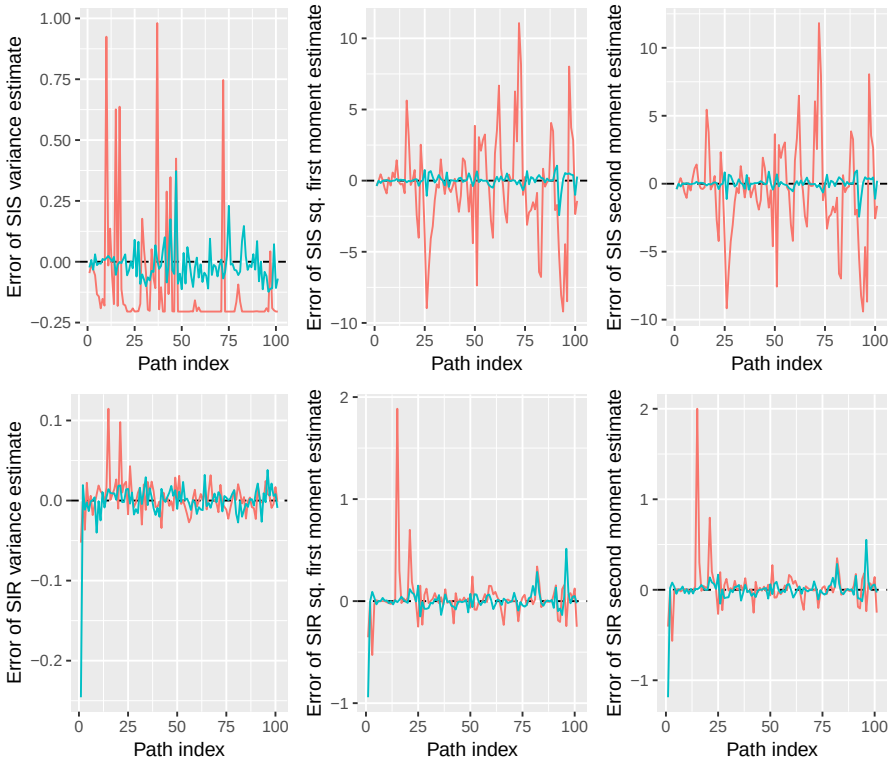

Figure 1: Error of SIR and SIS estimates of $\mathrm{Var}[X_n|Y_{0:n}]$, $\mathrm{E}[X_n|Y_{0:n}]^2$, and $\mathrm{E}[X_n^2|Y_{0:n}]$ for $N = 500$ particles. Blue line corresponds to optimal proposal density q^{opt}. Red line corresponds to bootstrap proposal density f. Path index is the value of n.

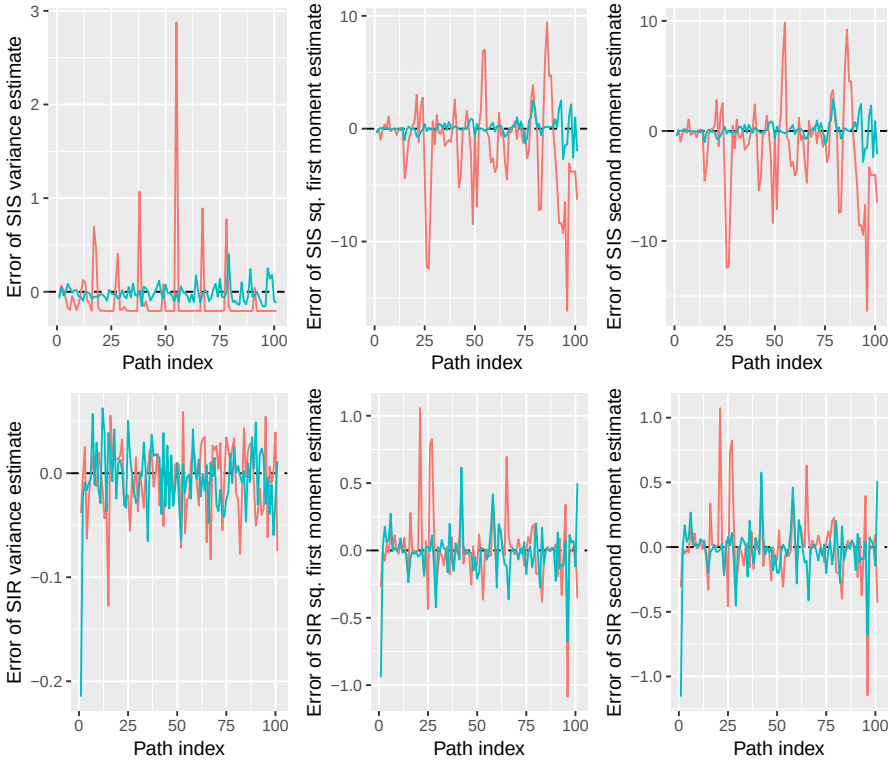

Figure 2: Error of SIR and SIS estimates of $\mathrm{Var}[X_n|Y_{0:n}]$, $\mathrm{E}[X_n|Y_{0:n}]^2$, and $\mathrm{E}[X_n^2|Y_{0:n}]$ for $N = 100$ particles. Blue line corresponds to optimal proposal density q^{opt}. Red line corresponds to bootstrap proposal density f. Path index is the value of n.

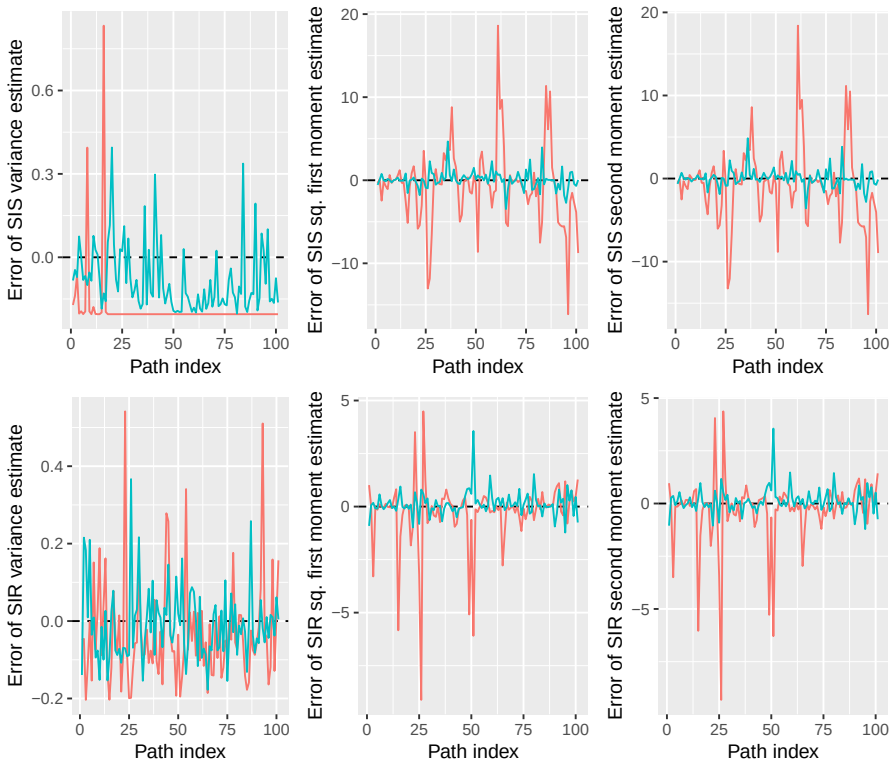

Figure 3: Error of SIR and SIS estimates of $\text{Var}[X_n|Y_{0:n}]$, $\text{E}[X_n|Y_{0:n}]^2$, and $\text{E}[X_n^2|Y_{0:n}]$ for $N = 10$ particles. Blue line corresponds to optimal proposal density q^{opt}. Red line corresponds to bootstrap proposal density f. Path index is the value of n.

Question 2 (ii) (c)

For $n = 10, 50, 100$ only plot the particle approximations for $p(X_n|Y_{0:n})$ using SIR and SIS separately. In each figure also overlay the true PDF, $p(X_n|Y_{0:n})$. You may use either histograms or smoothed kernel density estimates to estimate the filter's PDF.

The true PDF is that of the Kalman filter, $\mathcal{N}(\mu_{n|n}, \Sigma_{n|n})$. Figures 4, 5, and 6 present smoothed kernel density estimates for the SIS and SIR approximations to the filter $p(x_n|y_{0:n})$ at path indices $n = 10$, $n = 50$, $n = 100$ for sample sizes of $N = 10$, $N = 100$, $N = 500$. The kernel density estimator used a Gaussian kernel with bandwidth value of 0.05. A small bandwidth value was chosen so that it was clear how individual particles contributed to the density estimate.

Question 2 (iii) (a)

For which models is $x^*_{0:T}$ and $\text{E}[X_n|Y_{0:n}]$ appropriate to be used as a benchmark for estimating the hidden state.

$x^*_{0:T}$ is a somewhat appropriate benchmark for estimating the hidden state, since it *is* the hidden state - ideally, we would have a model that simply returned the value of $x^*_{0:T}$. That being said, $x^*_{0:T}$ represents only a single instantiation of the process and evaluations of a filter against it may not represent that filter's performance against all possible paths. Evaluating a filter with respect to the distribution over hidden state paths (or a sample of them) would be a better idea.

Whether or not $\text{E}[X_n|Y_{0:n}]$ is an appropriate benchmark for evaluating estimates of the hidden state depends on the nature of the loss function being considered for the problem. If an estimator is being evaluated according to the L^2 loss then the expected value is the optimal estimate - estimating it accurately is therefore very relevant indeed. If another loss function is being used, for example an L^1 loss $L(\hat{x}_n, x_n) = |\hat{x}_n - x_n|$, then other statistics would be more appropriate (for L_1, the median would be optimal prediction and most appropriate benchmark). In terms of representing whether a filter

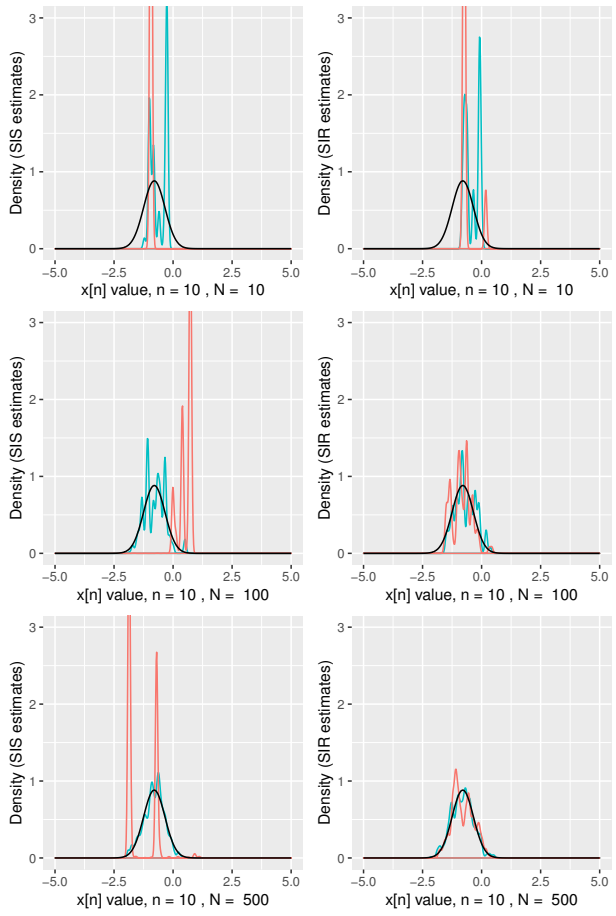

Figure 4: Black is the Kalman filter $p(x_n|y_{0:n})$. Red corresponds to proposal density f. Blue corresponds to proposal density q^{opt}. $n = 10$, (top) $N = 10$, (center) $N = 100$, (bottom) $N = 500$. The SIR and SIS densities were derived using a kernel density estimator with a Gaussian kernel and a bandwidth of 0.05.

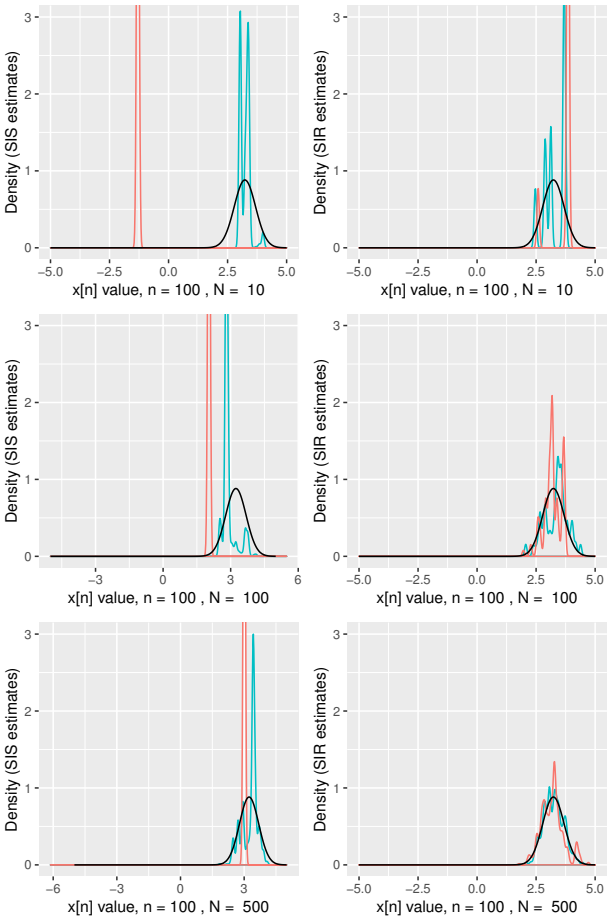

Figure 5: Black is the Kalman filter $p(x_n|y_{0:n})$. Red corresponds to proposal density f. Blue corresponds to proposal density q^{opt}. $n = 100$, (top) $N = 10$, (center) $N = 100$, (bottom) $N = 500$. The SIR and SIS densities were derived using a kernel density estimator with a Gaussian kernel and a bandwidth of 0.05.

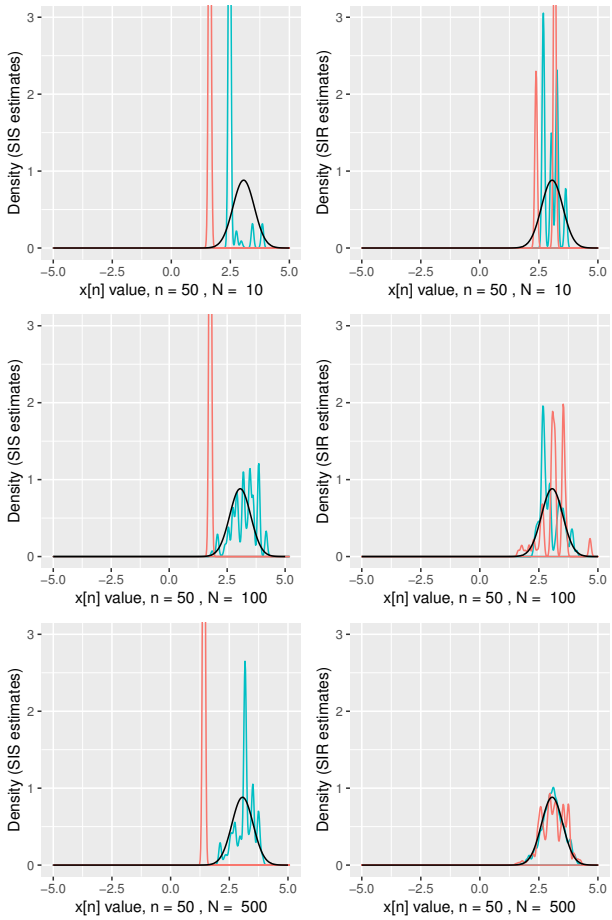

Figure 6: Black is the Kalman filter $p(x_n|y_{0:n})$. Red corresponds to proposal density f. Blue corresponds to proposal density q^{opt}. $n = 50$, (top) $N = 10$, (center) $N = 100$, (bottom) $N = 500$. The SIR and SIS densities were derived using a kernel density estimator with a Gaussian kernel and a bandwidth of 0.05.

accurately approximates the distribution over paths, the expected value may be misleading if e.g. the distribution is bimodal. In summary, the relevance of estimating a statistic of the filter distribution depends on how that statistic is related to the loss function of interest.

Question 2 (iii) (b)

Provide comments on the accuracy of SIS and SIR. In your discussion include a brief comment on your plots in 2 (b)-(c).

Referring to Figures 1, 2, and 3 the SIR variance estimate appears to unbiased and low-variance for each N, with variance decreasing as the number of particles N increases. The choice of proposal density $q(x_n|x_{n-1}, y_n)$ seems to make little difference to the quality of the estimate. Furthermore, the estimates do not seem to be strongly auto-correlated in n. The SIS variance estimate for $q = q^{opt}$, on the other hand, are auto-correlated, tend to be under-estimates, and are slightly higher-variance than the SIR estimates. The bias and variance of these estimates decreases as N increases, with the bias substantially negative for $N = 10$. The SIS variance estimates for $q = f$ seems to converge to 0 quickly (creating an error of ≈ -0.2), meaning that the filter approximation systematically under-estimates the true filter's variance. The variance of the SIS $q = f$ variance estimate is high for all N.

The variance estimate for SIS is better for the optimal proposal density since its weights decay less quickly than those for the bootstrap proposal density. This is because the weights of the former have a lower variance. Weight decay does not represent an issue for SIR since resampling means that low-weight particles are unlikely to persist and there are therefore no weights that decay to ≈ 0. This explains why using the optimal proposal density in SIR does not substantially improve the variance estimate.

Figures 4, 5, and 6 all indicate that SIR provides more accurate estimates of $p(x_n|y_{0:n})$ than SIS, irrespective of proposal density considered. Performance of both algorithms improves with increasing numbers of particles N, although this claim is marginal for SIS with $q = f$, since its approximation is degenerate (i.e. consists of only one or two particles) for all N and n. Even when using the optimal proposal density for SIS, there still seem to exist individual particles with especially large weights, as shown in the plots by tall, thin peaks. In general $\mathrm{E}[X_n|Y_{0:n}]$ tends to be well-approximated, even in the pathological cases.

Question 2 (iii) (c)

For each of the static model parameters comment on their role in order to recover good estimates in plots 2 (b)-(c).

The static model parameters are ρ, τ, and σ. If τ becomes large relative to ρ, then the sequence $X_{0:n}$ essentially becomes a sequence of i.i.d. draws from a zero-mean Gaussian with variance τ^2 (i.e. it becomes a white noise process). As τ approaches 0, the hidden state's path becomes deterministic for given X_0 ($X_n = \rho X_{n-1}$). We can also see that as σ approaches 0, the observed state Y_n becomes a better approximation of the hidden state X_n. By contrast, if σ becomes large then the observation Y_n becomes less informative with respect to the hidden state X_n.

One approach that can be taken to this question is to ask when the optimal proposal density q^{opt} is well-approximated by the bootstrap proposal density f. Recall that:

$$f(x_n|x_{n-1}) \text{ is the density of } \mathcal{N}(\rho X_{n-1}, \tau^2) \tag{46}$$

$$q^{opt}(x_n|y_n, x_{n-1}) \text{ is the density of } \mathcal{N}\left(\frac{\sigma^2 \rho X_{n-1} + \tau^2 Y_n}{\tau^2 + \sigma^2}, \frac{\tau^2 \sigma^2}{\tau^2 + \sigma^2}\right) \tag{47}$$

We can see that if σ^2 is large relative to τ, then f and q^{opt} will be similar, implying that the bootstrap proposal will be approximately optimal and result in low-variance weights with associated improved SIS variance estimates. If τ^2 is large relative to σ however, then the variance of q^{opt} approaches zero and its mean approaches Y_n, creating a point mass at Y_n. This would make f a poor approximation

of the optimal sampling density, meaning that SIS filters using it would struggle to approximate $p(x_n|y_{0:n})$ as accurately as the optimal proposal density.

Question 2 (iii) (d)

Suppose one changes the observation function into $Y_n = X_n^2 + \sigma W_n$. Without providing any new results or plots, how do you think the PDFs in Question 2 (ii) (c) will change?

Assume that the proposal densities are updated accordingly.

An immediate consequence of this change would be that it would not be possible to determine whether $-x_n$ or x_n was more likely for a given y_n. This suggests that $p(x_n|y_{0:n})$ would be symmetric about $x_n = 0$, with a mode either side of this value. This would affect the behaviour of the particle filters in that two groups of chains, one group with $x_{0:n}^i$ taking on negative values, another group with $x_{0:n}^i$ taking on positive values, could coexist for a sequence $y_{0:n}$ far from zero (w.r.t. scale σ). In spite of this, it seems plausible that they could still produce accurate filter approximations, although it would not be clear whether an individual particle was a good approximation to $x_{0:n}^*$.

Question 3

For the following scalar model

$$X_n = \rho X_{n-1} + \sigma V_n \tag{48}$$

$$Y_n = \beta \exp\left(\frac{X_n}{2}\right) W_n \tag{49}$$

where $W_n, V_n \overset{i.i.d.}{\sim} \mathcal{N}(0,1)$, $X_0 \sim \mathcal{N}(0, \frac{\sigma^2}{1-\rho^2})$, run the model to synthesize a data-set with $\rho = 0.91$, $\sigma = 1$, $\beta = 0.5$. Let $\theta = (\rho, \sigma, \beta)$. Store the real state trajectory $x_{0:T}^*$ for future comparisons. For known values ρ, σ, β you will implement a particle filter of your choice.

Question 3 (i) (a)

Mention clearly which importance proposal you will be using below.

We will be using the bootstrap proposal density,

$$q(x_n|x_{n-1}, y_n) = f(x_n|x_{n-1}), \text{ the density of } \mathcal{N}(\rho x_{n-1}, \sigma^2) \tag{50}$$

Question 3 (i) (b)

Is $\log p_\theta^N(y_{0:n})$ an unbiased estimate of $\log p_\theta(y_{0:n})$ and why?

Referring to page 26 in the lecture notes, the SIR weight increments can be used to construct an unbiased estimate of the likelihood provided that the resampling method is unbiased, i.e. we have:

$$\mathrm{E}[p_\theta^N(y_{0:n})] = p_\theta(y_{0:n}) \tag{51}$$

log is a strictly concave function[2]. Jensen's inequality therefore implies that:

$$\mathrm{E}[\log p_\theta^N(y_{0:n})] < \log \mathrm{E}[p_\theta^N(y_{0:n})] = \log p_\theta(y_{0:n}) \tag{52}$$

It follows that $\log p_\theta^N(y_{0:n})$ has a negative bias.

[2] A function h is strictly concave on interval $\mathcal{I}$ if for each pair $x, x' \in \mathcal{I}$ and constant $\lambda \in [0,1]$, $h(\lambda x + (1-\lambda)x') > \lambda h(x) + (1-\lambda)h(x')$.

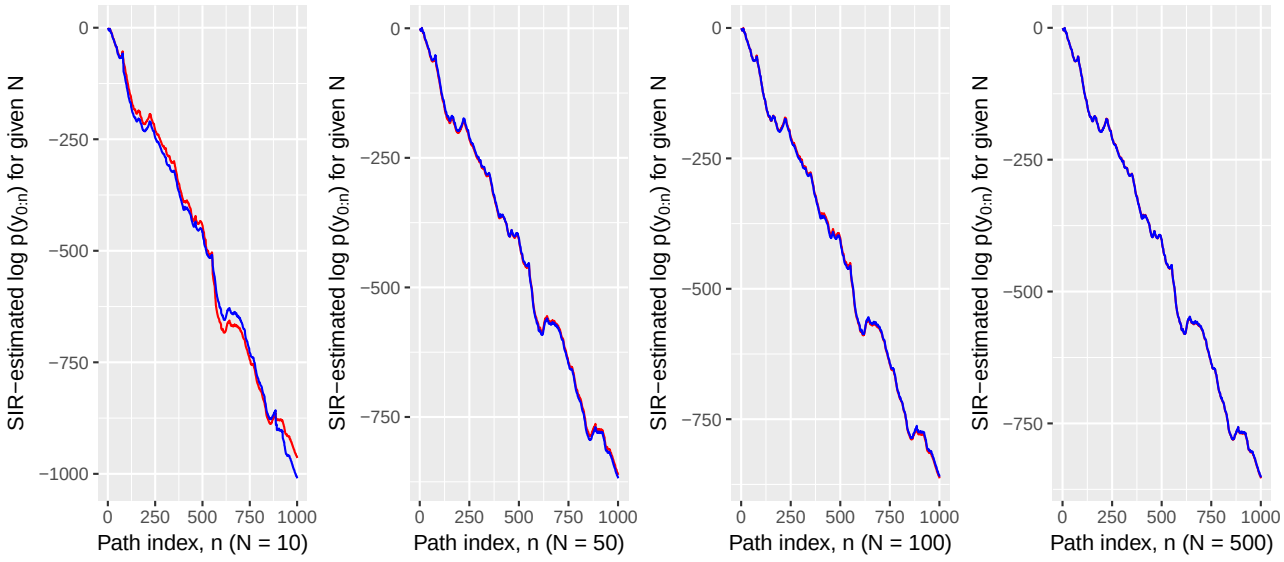

Figure 7: $\log p_\theta^N(y_{0:n})$), SIR estimate of $\log p_\theta(y_{0:n})$ as a function of n for various numbers of particles N. Each line corresponds to a distinct seed for the SIR.

Question 3 (i) (c)

Describe a procedure that could be used to approximate the Monte Carlo bias and variance of the estimator of the normalizing constant $p_\theta(y_{0:n})$.

The Monte Carlo bias and variance for the SIR estimator of the normalizing constant could be approximated as follows. The estimator is known to be unbiased, therefore we need only approximate its variance. Generate B length-n trajectories for the model specified. Evaluate the SIR marginal likelihood estimate $p_\theta^N(y_{0:n})$ for each trajectory. Evaluate the sample variance of these estimates to obtain an approximation of the Monte Carlo variance of the SIR estimator.

An issue with the above approach is that evaluating $p_\theta^N(y_{0:n})$ may not be possible for large N on account of limited numerical precision. This problem is unavoidable, since the magnitude of the variance of $p_\theta^N(y_{0:n})$ decreases with increasing n also. It might be more fruitful to consider the variance of a log-likelihood estimator or estimator for the relative variance.

Question 3 (i) (d)

Will $\log p_\theta^N(y_{0:n})$ *eventually become identical to* $\log p_\theta(y_{0:n})$ *as* N *increases? Explain your reasoning.*

Based on Equation 53 in the course notes, the relative variance of $p_\theta^N(y_{0:T})$ is proportional to $\frac{1}{N}$, which implies that $p_\theta^N(y_{0:n})$ is asymptotically consistent. If $p_\theta^N(y_{0:n})$ is asymptotically consistent in N, then it converges in probability to $p_\theta(y_{0:n})$. Since log is a Borel function that is continuous over its domain, this would imply that $\log p_\theta^N(y_{0:n})$ converges in probability to $\log p_\theta(y_{0:n})$.

Question 3 (ii) (a)

For two different independent runs of SIR plot $\log p_\theta^N(y_{0:n})$ *versus* n *for* $N = 50, 100, 500$. *In your code initialise every time your random number generator using a different seed.*

The requested plots, along with one for $N = 10$, are shown in Figure 7. Figure 8 presents the absolute difference between the graphs of Figure 7 and indicates that the Monte Carlo variance of $\log p_\theta^N(y_{0:N})$ decreases with increasing N.

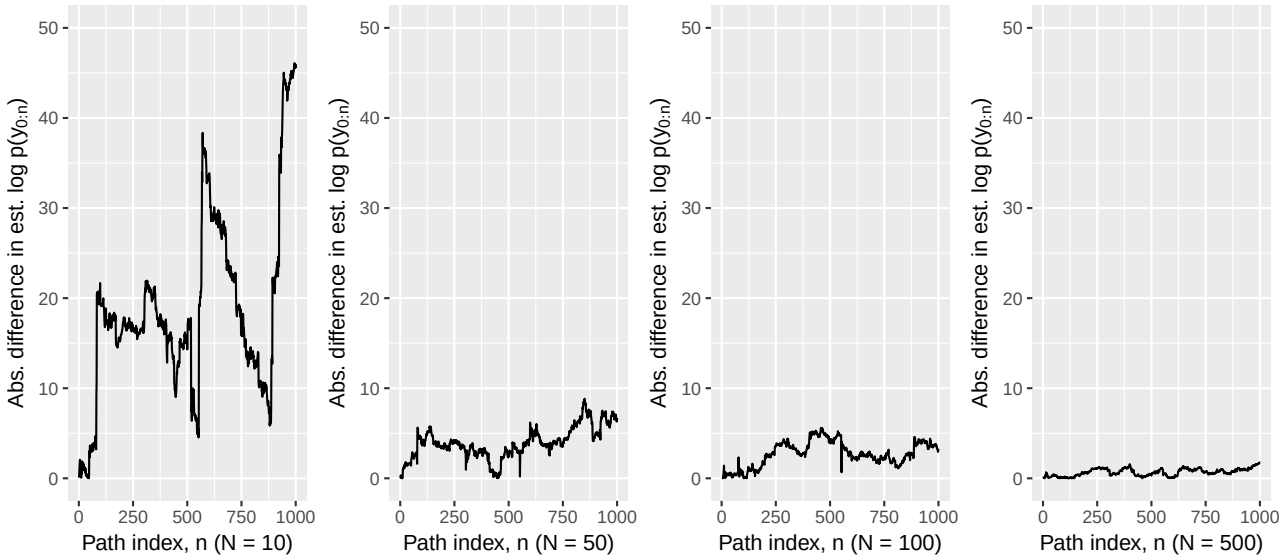

Figure 8: Absolute difference between the $p_\theta^N(y_{0:n})$) values shown in Figure 7. These results suggest that the variance of $p_\theta^N(y_{0:n})$ decreases with increaasing N.

Question 3 (ii) (b)

How will the plots change for a fixed N as T increases? There is no need to provide additional plots or numerical results, just provide some justification or intuition.

See Figure 8 for the absolute difference between a pair of SIR estimates for $\log p_\theta(y_{0:n})$. These plots indicate that the variance of $\log p_\theta^N(y_{0:n})$ increases with T. The course notes also state explicitly that the variance of $p_\theta^N(y_{0:n})$ increases linearly with T (at the top of page 41), suggesting that the variance of $\log p_\theta^N(y_{0:n})$ increases also.

Question 3 (iii) (a)

Assume just σ, β are known and set to their true values and only ρ is an unknown parameter. Construct a new smaller data-set using only the first 50 observations of your already simulated data-set. Also you can safely assume that $\rho \in [-1, 1]$.

For a grid of values that uses 50 points, plot $\log p_\rho^N(y_{0:50})$ versus ρ for $N = 50$. Suggest an estimate for the maximum likelihood estimator of ρ.

Figure 9 contains the results of the requested grid search. Based on this plot, an estimate of ρ of $\hat{\rho} \approx 0.6$ seems appropriate. This is not the $\hat{\rho}$-value that maximizes the observed likelihood estimates ($\hat{\rho} \approx 0.26$ does), but is the value that would maximize a (hypothetical) smooth function fit to the data. Notice that both of these values are quite different to the known value of ρ (0.91).

Question 3 (iii) (b)

Are there any potential problems in this plot that can affect the accuracy of the MLE estimate?

Our MLE estimate is based on estimates of the log-likelihood that have non-zero variance, which means that directly optimizing the estimated log-likelihood does not correspond to optimizing the log-likelihood. This is why in Question 3 (iii) (a), a smooth fit to the log-likelihood estimates informed our MLE estimate rather than the raw log-likelihood estimates themselves. Assuming that the log-likelihood is a continuous function of ρ may not always be justified also.

A further problem relates to the bias of the log-likelihood estimator $\log p_\theta^N(y_{0:n})$. We only know that this estimator is biased. We do not know whether the bias of is a function of ρ. If it is, then Figure 9 will misrepresent how the log-likelihood varies as a function of ρ. For example, if the bias is more

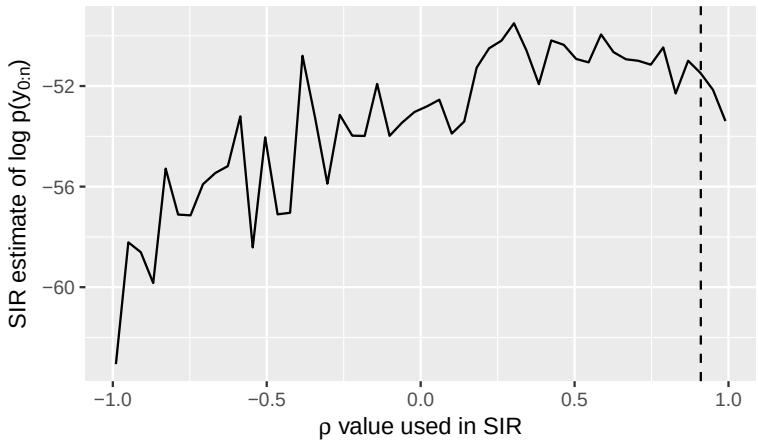

Figure 9: SIR estimates of $p_\theta(y_{0:50})$ for $N = 50$ with proposed ρ values in $(-1, 1)$. The dashed line corresponds to $\rho = 0.91$, the true value of ρ in the state-space model.

negative for values of ρ closer to 1 and is essentially zero for values of ρ near 0, then our MLE estimate of $\hat{\rho} = 0.6$ is likely to be an under-estimate.

Question 3 (iv)

We wish to improve the plot in Question 3 (ii) (a). Suggest two different ideas that could be used to improve the plot in question 2 (a) just above. For each of your suggestions comment how and why you think they will improve the plot. You are not required to implement your ideas for this question.

We could improve our estimate of the log-likelihood $\log p_\theta(y_{0:50})$ in a number of ways. First, we could improve the performance of our SIR algorithm. We might do this by using a proposal density that results in a lower weight variance. Improving our approximation of the $p(x_n|y_{0:n})$ would improve our approximation of $p_\theta(y_{0:50})$. The proposal density that minimizes the weight variance is:

$$q(x_n|y_n, x_{n-1}) = p(x_n|y_n, x_{n-1}) \propto g(y_n|x_n)f(x_n|x_{n-1}) \tag{53}$$

A closed-form expression for the distribution of $p(x_n|y_n, x_{n-1})$ is not accessible since Y_n is a nonlinear function of X_n. One way to circumvent this issue is to linearize our state-space expression for $Y_n = \phi(x_n, w_n)$ about $(\rho x_{n-1}, 0)$ and determine the parameters of the Gaussian for $Y_n|X_n$ under this approximation. We could then derive a Gaussian approximation to $p(x_n|y_n, x_{n-1})$, as is done on page 19 of the course notes.

Another way that we could enhance our estimate of the log-likelihood would be to correct its bias. This could be achieved by taking a second-order Taylor expansion of $\log Z = \log p_\theta^N(y_{0:50})$ around $Z = \mathrm{E}[p_\theta^N(y_{0:50})] = p_\theta(y_{0:50})$ then rearranging to obtain the approximately bias-corrected estimator:

$$\log p_\theta^N(y_{0:T}) + \frac{1}{2}\,\mathrm{Var}\left[\frac{p_\theta^N(y_{0:50})}{p_\theta(y_{0:50})}\right] \tag{54}$$

where the relative variance can be approximated in terms of the incremental weights according to the expression at the bottom of page 41 of the course notes.

Appendix

```
# Question 2
set.seed(60)
T_    <- 100 # n = 10, 50, 100
N     <- 500 # N = 10, 100, 500
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)

# 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)
}


# ! 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)

kf.EX  <- rep(NA, T_+1)
kf.EX2 <- rep(NA, T_+1)

A <- rho
B <- tau
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)

# Store the first and second moments
kf.EX[1]  <- kf.mu_nln[1]
kf.EX2[1] <- kf.Sigma_nln[1] + kf.mu_nln[1]^2

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]

  # Store the first and second moments
  kf.EX[t]  <- kf.mu_nln[t]
  kf.EX2[t] <- kf.Sigma_nln[t] + kf.mu_nln[t]^2
}


# ! Sequential importance sampler non-optimal ! #
# 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)
eta.d <- function(x_0) dnorm(x_0, mean=0, sd=1)
eta.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=rho*x_n_1, sd=tau)
f.r <- function(x_n_1) rnorm(1, mean=rho*x_n_1, sd=tau)

# Hidden state initial sampling density, q0(X_0|Y_0)
q0.d <- function(x0, y0) eta.d(x0) # f(X_0)
q0.r <- function() eta.r()

# Hidden state transition sampling density, q(X_n | X_{n-1}, Y_n)
q.d <- function(x_n, x_n_1) f.d(x_n, x_n_1) # f(X_n|X_{n-1})
q.r <- function(x_n_1) f.r(x_n_1)

# Initialize particle and weight matrix
sisf.x <- matrix(rep(NA, N*(T_+1)), nrow=(T_+1)) # Each column is a particle
sisf.w <- matrix(rep(NA, N*(T_+1)), nrow=(T_+1)) # Will be normalized
sisf.EX  <- rep(NA, T_+1)
sisf.EX2 <- rep(NA, T_+1)

# NB for q = f, we perform redundant calculations

# Perform the initial step
sisf.x[1, ] <- sapply(1:N, function(dmmy) q0.r())
sisf.w[1, ] <- sapply(sisf.x[1, ], eta.d)*sapply(sisf.x[1, ], function(xi0) g.d(y[1], xi0)) /
                sapply(sisf.x[1, ], function(xi0) q0.d(xi0, y[1]) )
sisf.w[1, ] <- sisf.w[1, ]/sum(sisf.w[1, ])

# Compute moment estimates
```

```r
sisf.EX[1]  <- sum(sisf.w[1, ]*sisf.x[1, ])
sisf.EX2[1] <- sum(sisf.w[1, ]*(sisf.x[1, ]^2))

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

  # Compute importance weights by updating previous weights
  sisf.w[t, ] <- sisf.w[t-1, ]*
                 apply(sisf.x[(t-1):t, ], 2, function(xi) f.d(xi[2], xi[1]))* # f(x_n/x_{n-1})
                 sapply(sisf.x[t, ], function(xi_t) g.d(x_n=xi_t, y_n=y[t]))/ # g(x_n/y_n)
                 apply(sisf.x[(t-1):t, ], 2, function(xi) q.d(xi[2], xi[1])) # q(x_n/x_{n-1})

  # Normalize weights
  sisf.w[t, ] <- sisf.w[t, ]/sum(sisf.w[t, ])

  # Compute moment estimates
  sisf.EX[t]  <- sum(sisf.w[t, ]*sisf.x[t, ])
  sisf.EX2[t] <- sum(sisf.w[t, ]*(sisf.x[t, ]^2))
}

# ! Sequential importance resampler non-optimal ! #
# Initialize particle and weight matrix
sirf.x <- matrix(rep(NA, N*(T_+1)), nrow=(T_+1)) # Each column is a particle
sirf.w <- matrix(rep(NA, N*(T_+1)), nrow=(T_+1)) # Will be normalized
sirf.EX  <- rep(NA, T_+1)
sirf.EX2 <- rep(NA, T_+1)

# Perform the initial step
sirf.x[1, ] <- sapply(1:N, function(dmmy) q0.r())
sirf.w[1, ] <- sapply(sirf.x[1, ], eta.d)*sapply(sirf.x[1, ], function(xi0) g.d(y[1], xi0)) /
  sapply(sirf.x[1, ], function(xi0) q0.d(xi0, y[1]) )
sirf.w[1, ] <- sirf.w[1, ]/sum(sirf.w[1, ])
sirf.x[1, ] <- sample(sirf.x[1, ], size=N, replace=T, prob=sirf.w[1, ])

# Compute moment estimates
sirf.EX[1]  <- sum(sirf.w[1, ]*sirf.x[1, ])
sirf.EX2[1] <- sum(sirf.w[1, ]*(sirf.x[1, ])^2)

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

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

  # Normalize weights
  sirf.w[t, ] <- sirf.w[t, ]/sum(sirf.w[t, ])

  # Compute moment estimates
  sirf.EX[t]  <- sum(sirf.w[t, ]*sirf.x[t, ])
```

```
  sirf.EX2[t] <- sum(sirf.w[t, ]*(sirf.x[t, ])^2)

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


# ! Optimal sequential importance sampler ! #

# Only need to alter the proposal mechanism and density

# 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)
eta.d <- function(x_0) dnorm(x_0, mean=0, sd=1)
eta.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=rho*x_n_1, sd=tau)
f.r <- function(x_n_1) rnorm(1, mean=rho*x_n_1, sd=tau)

# 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=(rho*x_n_1*sigma^2 + y_n*tau^2)/(tau^2 + sigm
                                       sd=sqrt(tau^2*sigma^2/(tau^2 + sigma^2)))
q.r <- function(x_n_1, y_n) rnorm(1, mean=(rho*x_n_1*sigma^2 + y_n*tau^2)/(tau^2 + sigma^2),
                                  sd=sqrt(tau^2*sigma^2/(tau^2 + sigma^2)))

# Initialize particle and weight matrix
siso.x <- matrix(rep(NA, N*(T_+1)), nrow=(T_+1)) # Each column is a particle
siso.w <- matrix(rep(NA, N*(T_+1)), nrow=(T_+1)) # Will be normalized
siso.EX  <- rep(NA, T_+1)
siso.EX2 <- rep(NA, T_+1)

# Perform the initial step
siso.x[1, ] <- sapply(1:N, function(dmmy) q0.r(y[1]))
siso.w[1, ] <- sapply(siso.x[1, ], eta.d)*sapply(siso.x[1, ], function(xi0) g.d(y[1], xi0)) /
               sapply(siso.x[1, ], function(xi0) q0.d(xi0, y[1]) )
siso.w[1, ] <- siso.w[1, ]/sum(siso.w[1, ])

# Compute moment estimates
siso.EX[1]  <- sum(siso.w[1, ]*siso.x[1, ])
siso.EX2[1] <- sum(siso.w[1, ]*(siso.x[1, ])^2)

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

  # Compute importance weights by updating previous weights
```

```
  siso.w[t, ] <- siso.w[t-1, ]*
    apply(siso.x[(t-1):t, ], 2, function(xi) f.d(xi[2], xi[1]))* # f(x_n/x_{n-1})
    sapply(siso.x[t, ], function(xi_t) g.d(x_n=xi_t, y_n=y[t]))/ # g(x_n/y_n)
    apply(siso.x[(t-1):t, ], 2, function(xi) q.d(xi[2], xi[1], y[t])) # q(x_n/x_{n-1})

  # Normalize weights
  siso.w[t, ] <- siso.w[t, ]/sum(siso.w[t, ])

  # Compute moment estimates
  siso.EX[t]  <- sum(siso.w[t, ]*siso.x[t, ])
  siso.EX2[t] <- sum(siso.w[t, ]*(siso.x[t, ])^2)
}


# ! Optimal sequential importance resampler ! #

# Initialize particle and weight matrix
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)) # Will be normalized
siro.EX  <- rep(NA, T_+1)
siro.EX2 <- rep(NA, T_+1)

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

# Compute moment estimates
siro.EX[1]  <- sum(siro.w[1, ]*siro.x[1, ])
siro.EX2[1] <- sum(siro.w[1, ]*(siro.x[1, ])^2)

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]))* # f(x_n/x_{n-1})
                 sapply(siro.x[t, ], function(xi_t) g.d(x_n=xi_t, y_n=y[t]))/ # g(x_n/y_n)
                 apply(siro.x[(t-1):t, ], 2, function(xi) q.d(xi[2], xi[1], y[t])) # q(x_n/x_{

  # Normalize weight increments
  siro.w[t, ] <- siro.w[t, ]/sum(siro.w[t, ])

  # Compute moment estimates
  siro.EX[t]  <- sum(siro.w[t, ]*siro.x[t, ])
  siro.EX2[t] <- sum(siro.w[t, ]*(siro.x[t, ])^2)

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

```
# Compute the trajectory of the expected value
plot(x.star, type='l')
lines(kf.mu_nln, col='red')
#lines(sis.x[, which.max(sis.w[T_+1, ])], col='purple', lty=2, lwd=1)
lines(rowSums(siso.w*siso.x), col='blue', lty=2)
lines(rowMeans(siro.x), col='green', lty=3)
lines(rowSums(sisf.w*sisf.x), col='purple', lty=2)
lines(rowMeans(sirf.x), col='orange', lty=2)


# Now evaluate the variance estimates for each model
# Split the plots by SIR/SIS, use both proposal densities on each
# Start with N = 500

# Compute variance estimates
sisf.VarX <- sisf.EX2 - sisf.EX^2
siso.VarX <- siso.EX2 - siso.EX^2
sirf.VarX <- sirf.EX2 - sirf.EX^2
siro.VarX <- siro.EX2 - siro.EX^2
kf.VarX   <- kf.Sigma_nln # To avoid ambiguity
```

```
# Question 3 (ii)
set.seed(69)

logpNSeed1 <- list()
logpNSeed2 <- list()

rho   <- 0.91
sigma <- 1
beta  <- 0.5
T_    <- 1000

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)
}


# For two different independent runs of SIR plot logpN(y0:n) versus n (n  T) for N = 50, 100,
# your code initialise every time your random number generator using a different seed.

# Code the SIR first,
# then implement the likelihood estimator.

# Data storage
for (N in c(10, 50, 100, 500)){
  sir.x <- matrix(rep(NA, N*(T_+1)), nrow=T_+1)
```

```
  sir.w <- matrix(rep(NA, N*(T_+1)), nrow=T_+1)

  # Initial sampling distribution (same as X0's)
  q0.r <- function() rnorm(1, mean=0, sd=sqrt(sigma^2/(1 - rho^2)))
  q0.d <- function(x) dnorm(x, mean=0, sd=sqrt(sigma^2/(1 - rho^2)))

  # Transition sampling distribution (f(x_n|x_{n-1}))
  q.r <- function(x_n_1) rnorm(1, mean=rho*x_n_1, sd=sigma)
  q.d <- function(x_n, x_n_1) dnorm(x_n, mean=rho*x_n_1, sd=sigma)

  # Likelihood and transition density functions (used to compute incremental weights)
  f0.d <- function(x0) dnorm(x0, mean=0, sd=sqrt(sigma^2/(1 - rho^2)))
  f.d <- function(x_n, x_n_1) dnorm(x_n, mean=x_n_1, sd=sigma)
  g.d <- function(y_n, x_n) dnorm(y_n, mean=0, sd=beta*exp(x_n/2))
  w0 <- function(x0, y0) f0.d(x0)*g.d(y0, x0)/q0.d(x0)
  w <- function(x_n, x_n_1, y_n) f.d(x_n, x_n_1)*g.d(y_n, x_n)/q.d(x_n, x_n_1)

  sir.logpN <- list(rep(NA, T_+1), rep(NA, T_+1))
  for (i in 1:2){
    set.seed(i)

    # Run the filter
    sir.x[1, ] <- replicate(N, q0.r())
    sir.w[1, ] <- sapply(sir.x[1, ], function(x0) w0(x0, y[1]))
    sir.x <- sir.x[ , sample(1:N, N, replace=T, prob=sir.w[1, ])]

    for (t in 2:(T_+1)) {
      sir.x[t, ] <- sapply(sir.x[t-1, ], q.r)
      sir.w[t, ] <- apply(sir.x[(t-1):t, ], 2, function(xt_1_xt) w(xt_1_xt[2], xt_1_xt[1], y[t
      sir.x <- sir.x[ , sample(1:N, N, replace=T, prob=sir.w[t, ])]
    }

    # Compute estimate of the log-likelihood
    sir.logpN[[i]] <- cumsum(log(rowMeans(sir.w)))
  }

  logpNSeed1[[as.character(N)]] <- sir.logpN[[1]]
  logpNSeed2[[as.character(N)]] <- sir.logpN[[2]]
}
```

```
# Question 3 (iii)
set.seed(69)

rho   <- 0.91
sigma <- 1
beta  <- 0.5
T_    <- 50

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)
```

```r
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)
}

# MLE for rho investigation
# rho_p is a proposed value for rho

# Set number of particles
N <- 100
rho_grid <- seq(-0.99, 0.99, length.out=50)
rhologpN <- c() # to store log-likelihoods for each proposed rho value

for (rho_p in rho_grid){
  # Configure the filter for rho_p
  sir.x <- matrix(rep(NA, N*(T_+1)), nrow=T_+1)
  sir.w <- matrix(rep(NA, N*(T_+1)), nrow=T_+1)

  # Initial sampling distribution (same as X0's)
  q0.r <- function() rnorm(1, mean=0, sd=sqrt(sigma^2/(1 - rho_p^2)))
  q0.d <- function(x) dnorm(x, mean=0, sd=sqrt(sigma^2/(1 - rho_p^2)))

  # Transition sampling distribution (f(x_n|x_{n-1}))
  q.r <- function(x_n_1) rnorm(1, mean=rho_p*x_n_1, sd=sigma)
  q.d <- function(x_n, x_n_1) dnorm(x_n, mean=rho_p*x_n_1, sd=sigma)

  # Likelihood and transition density functions (used to compute incremental weights)
  f0.d <- function(x0) dnorm(x0, mean=0, sd=sqrt(sigma^2/(1 - rho_p^2)))
  f.d <- function(x_n, x_n_1) dnorm(x_n, mean=x_n_1, sd=sigma)
  g.d <- function(y_n, x_n) dnorm(y_n, mean=0, sd=beta*exp(x_n/2))
  w0 <- function(x0, y0) f0.d(x0)*g.d(y0, x0)/q0.d(x0)
  w <- function(x_n, x_n_1, y_n) f.d(x_n, x_n_1)*g.d(y_n, x_n)/q.d(x_n, x_n_1)

  # Run the filter
  sir.x[1, ] <- replicate(N, q0.r())
  sir.w[1, ] <- sapply(sir.x[1, ], function(x0) w0(x0, y[1]))
  sir.x <- sir.x[ , sample(1:N, N, replace=T, prob=sir.w[1, ])]

  for (t in 2:(T_+1)) {
    sir.x[t, ] <- sapply(sir.x[t-1, ], q.r)
    sir.w[t, ] <- apply(sir.x[(t-1):t, ], 2, function(xt_1_xt) w(xt_1_xt[2], xt_1_xt[1], y[t]))
    sir.x <- sir.x[ , sample(1:N, N, replace=T, prob=sir.w[t, ])]
  }

  # Compute estimate of the log-likelihood
  rhologpN <- rbind(rhologpN, c(rho_p, sum(log(rowMeans(sir.w)))))
}
```