These problems constitute a part of the course Nonlinear system identification using sequential Monte Carlo methods, which is a part of the Summer school on foundations and advances in stochastic filtering (FASF 2015), given at Centre Tecnològic de Telecomunicacions de Catalunya (CTTC), Barcelona, Spain in June 2015.

Good luck!

1 [Bootstrap particle filter for LGSS] Consider the following scalar linear Gaussian state space (LGSS) model

\begin{align}
    x_{t+1} &= 0.7x_t + v_t, \quad v_t \sim \mathcal{N}(0, 0.1), \quad (1a) \\
    y_t &= 0.5x_t + e_t \quad e_t \sim \mathcal{N}(0, 0.1). \quad (1b)
\end{align}

Let the initial state be distributed according to \( x_1 \sim \mathcal{N}(0, 1) \).

(a) Write this model on the form

\begin{align}
    x_{t+1} | (x_t = x_t) &\sim f(x_{t+1} | x_t), \quad (2a) \\
    y_t | (x_t = x_t) &\sim g(y_t | x_t). \quad (2b)
\end{align}

In other words, find the probability density functions \( f(\cdot) \) and \( g(\cdot) \) in (2) corresponding to the model (1).

(b) Simulate the model (1) to produce \( T = 100 \) measurements \( y_{1:T} \). Based on these measurements compute the optimal (in the sense that it minimizes the mean square error) estimate of \( x_t | y_{1:t} \) for \( t = 1, \ldots, T \). Implement a bootstrap particle filter and compare to the optimal estimates. You can for example perform this comparison by plotting the root mean square estimate (RMSE) \( \varepsilon(N) \) as a function of the number of particles used in the particle filter (also plot the RMSE for the optimal estimator in the same figure). The RMSE is defined according to

\[
    \varepsilon(N) \triangleq \sqrt{\frac{1}{T} \sum_{t=1}^{T} (\hat{x}_{i,t}^N(N) - x_t)^2}, \quad (3)
\]

where \( \hat{x}_{i,t}^N(N) \) denotes the estimate of the state produced with a particle filter using \( N \) particles and \( x_t \) denotes the true state.


(c) **Optional** We are free to use more general proposal densities in the propagation step of the particle filter. Change the bootstrap particle filter to propose new particles according to

\[ x^i_t \sim p(x^i_t \mid \hat{x}^i_{t-1}, y_t), \] (4)

rather than using \( x^i_t \sim f(x_t \mid \hat{x}^i_{t-1}) \). Compare the resulting state estimates to the ones obtained in Problem (b). Does this provide better or worse estimates, explain why. *Hint:* Start by deriving an explicit expression for \( p(x_t \mid x_{t-1}, y_t) \) when the model is given by (1).

2 **[Stochastic volatility]** Consider the so-called stochastic volatility (SV) model

\[
\begin{align*}
    x_{t+1} &\sim N(x_{t+1}; \phi x_t, \sigma^2), \\
y_t &\sim N(y_t; 0, \beta^2 \exp(x_t)),
\end{align*}
\]

where the parameter vector is given by \( \theta = \{ \phi, \sigma, \beta \} \). Here, \( x_t \) denotes the underlying latent volatility (the variations in the asset price) and \( y_t \) denotes the observed scaled log-returns from some financial asset. The \( T = 500 \) observations that we consider in this task are log-returns from the NASDAQ OMX Stockholm 30 Index during a two-year period between January 2, 2012 and January 2, 2014. We have calculated the log-returns by \( y_t = 100 \log(s_t) - \log(s_{t-1}) \), where \( s_t \) denotes the closing price of the index at day \( t \). The data is found in seOMXlogreturns2012to2014.csv. For more details on stochastic volatility models, see e.g. [2, 6].

(a) Assume that the parameter vector is given by \( \theta = \{ 0.98, 0.16, 0.70 \} \). Estimate the marginal filtering distribution at each time index \( t = 1, \ldots, T \) using the particle filter with \( N = 500 \) particles. Plot the mean of the distribution at each time step and compare with the observations. Is the estimated volatility reasonable?

(b) Let us now view the particle filter as a way of estimating the JSD \( p(x_{1:T} \mid y_{1:T}) \). Plot the particles \( \{ x_{1:100} \}_{i=1}^{N} \) using \( N = 100 \) particles. How well is the JSD approximated?

(c) Compute the distribution of the error in the log-likelihood estimates by \( M = 500 \) Monte Carlo iterations (this can take a couple of minutes to compute). That is, compute the log-likelihood estimate on the same data \( M \) times and compute the histogram and kernel density estimate of the error given that the true log-likelihood is \(-695.62\). What distribution does the error have? Are the estimates biased? What happens when \( N \) increases? *Hint:* Make use of marginalization to compute the log-likelihood. \( p_\theta(y_{1:T}) = \int p_\theta(x_{1:T}, y_{1:T})dx_{1:T} \). For nonlinear state space models this results in that we can compute an estimate of the log-likelihood by making use of the particle weights. The estimator for the log-likelihood is given by

\[ \hat{\ell}(\theta) = \log \hat{p}_\theta(y_{1:T}) = \sum_{t=1}^{T} \log \left[ \sum_{i=1}^{N} \tilde{w}_i^t \right] - T \log N, \]

where \( \tilde{w}_i^t \) denotes the unnormalised weight for particle \( i \) at time \( t \).

(d) Implement an importance sampler (IS) to estimate the parameter posterior distribution given by

\[ p(\theta \mid y_{1:T}) \propto p(y_{1:T} \mid \theta) p(\theta), \]

where \( p(y_{1:T} \mid \theta) \) and \( p(\theta) \) denotes the likelihood function and the parameter prior, respectively. Here, we make use of an uniform prior over \([0, 1] \in \mathbb{R}\) for each of the three parameters, respectively.

Choose an appropriate proposal distribution for \( \phi \) and calculate the estimated parameter posterior for \( \phi \) (keeping the other parameters fixed to the values given above) using 500 samples from the log-likelihood and \( N = 500 \) particles. Present the result as a histogram of the posterior estimate together with a kernel density estimate and the proposal distribution.
(e) (Optional) Repeat the previous task for all the parameters \( \{\phi, \sigma, \beta\} \) with 2 000 samples and \( N = 1000 \) particles (this can take some time, i.e. an hour to compute). Check the number of weights that are non-zero. What problems might the use of IS lead to if the number of parameters are large?

(f) (Optional) If you know about the Metropolis-Hastings (HW) algorithm, it is fairly simple to implement the particle marginal MH (PMMH) algorithm for parameter inference in the SV model. The main problem is that the intractable likelihood is required for computing the acceptance probability given by

\[
\alpha(\theta'', \theta') = \min \left( 1, \frac{p_{\theta''}(y_{1:T}) p(\theta') q(\theta' | \theta'')}{p_{\theta'}(y_{1:T}) p(\theta'') q(\theta'' | \theta')} \right). \tag{6}
\]

However, it is possible to show that we can replace the analytically intractable likelihood with the estimate from the particle filter and still obtain a valid MH algorithm, which gives a Markov chain that converges to the target distribution (the parameter posterior) as its stationary distribution. To implement the PMMH, you need to code the following algorithm:

- Initialise with \( \theta_0 \) and compute \( \hat{p}_{\theta_0}(y_{1:T}) \) using the particle filter.

- FOR \( k = 1 \) to \( K \)
  - Propose a new parameter from a random walk proposal by sampling \( \theta' \sim \mathcal{N}(\theta_{k-1}, \epsilon^2) \).
  - Run the particle filter to obtain \( \hat{p}_{\theta'}(y_{1:T}) \).
  - Calculate the acceptance probability \( \alpha(\theta', \theta_{k-1}) \) in (6) by replacing \( p_{\theta''}(y_{1:T}) \) with \( \hat{p}_{\theta''}(y_{1:T}) \) and \( p_{\theta'}(y_{1:T}) \) with \( \hat{p}_{\theta_{k-1}}(y_{1:T}) \). Note that the random walk proposal is symmetric and the ratio between the proposal distributions cancels.
  - Make a standard accept/reject decision. If the parameter is accepted, set \( \{\theta_k, \hat{p}_{\theta_k}(y_{1:T})\} \leftarrow \{\theta', \hat{p}_{\theta'}(y_{1:T})\} \), otherwise keep the old parameter and likelihood estimate.

- END FOR

In practice, we usually compute the log-likelihood and compute the log of the acceptance probability for numerical stability. This is done by computing

\[
\alpha(\theta'', \theta') = \exp \left[ \hat{\ell}(\theta'') - \hat{\ell}(\theta') + \log p(\theta'') - \log p(\theta') \right].
\]

Implement the PMMH algorithm for inferring \( \phi \) in the same setting as above, i.e. using the same number of particles and prior distributions. Select a reasonable value for \( \epsilon \) and \( K \) and plot the resulting posterior estimate using a histogram and kernel density estimate.

**Hint:** It is numerically beneficial to work with the log-weights in the particle filter. That is compute the weight by \( v_i^t = \log \mathcal{N}(y_i; 0, \beta^2 \exp(x_i^t)) \) and use the following transformation to compute the corresponding particle weights: (i) compute \( v_{\max,t} = \max\{v_i^t\}_{i=1}^N \) and (ii) compute \( \tilde{w}_i^t = \exp(v_i^t - v_{\max,t}) \). The particle weights can now be normalised as usual. Also, make use of this procedure in the IS algorithm for parameter inference.

**Background:** The PMMH algorithms is one member of the Particle Markov chain Monte Carlo (PMCMC) family of algorithms introduced by [1]. See also Chapter 5 in [5] for an introduction to these algorithms.
3. **(An SMC sampler for localization)** We want to localize an object positioned at $x_0$ in our world $[-12, 12]^2$, as illustrated in Figure 1. We have access to a bunch of measurements of the position. As the measurements are corrupted by heavy-tailed noise from the exponential distribution, we are not really interested in just a point estimate of $x_0$, but instead we want the entire posterior distribution $p(x_0|Y)$ of its position, reflecting the uncertainty inherent in the problem.

![Figure 1: Illustration of our world $[-12, 12]^2$, with the true position $x_0$ indicated using a red circle.](image)

(a) Prepare code for simulating $T$ independent measurements from the following model

$$
y^1_t = x^1_0 + n^1_t b^1_t, \quad (7a)
$$

$$
y^2_t = x^2_0 + n^2_t b^2_t, \quad (7b)
$$

for $t = 1, \ldots, T$, where $n^1_t$ and $n^2_t$ are exponentially distributed with scale parameter 2, and $P(b^1_t = 1) = P(b^1_t = -1) = \frac{1}{2}$ and similarly for $b^2_t$. (Such a heavy tailed measurement model is useful in, e.g., modeling of ultra-wideband measurements [4]) In Matlab, you can use

$$
Y = \text{repmat}(x0, [1 T]) + \text{exprnd}(2, [2 T]) \ast (1 - 2 \ast (\text{rand}([2 T]) > 0.5));
$$

(b) Based on our background knowledge on the problem we know that a reasonable prior to be used for the position $x_0$ is the following,

$$
p(x_0) = \mathcal{N}(x_0 | \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 7 & 0 \\ 0 & 7 \end{bmatrix}). \quad (8)
$$

**Design a** (somewhat smooth) **transition** $\pi_1, \ldots, \pi_P$ from the prior $\pi_1(x_0) = p(x_0)$ to the posterior $\pi_P(x_0) = p(x_0|Y) \propto p(x_0) \prod_{t=1}^T p(y_t|x_0)$. Hints can be found in [3, Section 2.3.1].

(c) **Implement a** $\pi_n$-invariant Metropolis-Hastings (MH) kernel, i.e., the following algorithm: Assume you start with one particle (sample) $x_{n-1}$.

(a) Propose a new particle $x'$ using a proposal, e.g., a random walk, as $x' = x_{n-1} + v$, where $v \sim \mathcal{N}(0, H)$ is a multivariate normal distribution. (It is crucial for the performance of the algorithm to choose $H$ sensible. Here, a first attempt could be $H \approx I$, the identity matrix.)

(b) Compute the acceptance ratio $\alpha_i$ as $\alpha_i = \frac{\pi_n(x')}{\pi_n(x_{n-1})}$. (If the proposal is not symmetric, as the random walk, the expression becomes more involved.)

(c) Set $x_n = x'$ with probability $\min(\alpha_i, 1)$, otherwise set $x_n = x_{n-1}$.

(d) **Put everything together in an SMC sampler** according to the following algorithm:

1. Sample from prior
2. **for** $n = 1$ **to** $P$
3. Update particle weights*
4. Resample particles
5. Propagate particles using the MH kernel

---

*Note: The document contains some mathematical symbols and code snippets that are not rendered properly in this text format. For a full understanding, please refer to the original document or use a tool that supports advanced mathematical notation rendering.
6. end

*There are alternative ways to update the particle weights in an SMC sampler. The easiest is probably to set the weight of particle $x_{n-1}^i$ as $\pi_n(x_{n-1}^i)/\pi_{n-1}(x_{n-1}^i)$ [3, eq. (31)].

(e) Test your algorithm by making sure that it converges to something sensible (i.e., close to $x_0$) when a fairly large number of measurements and particles (say, e.g., $N = 100$ particles and $T = 50$ measurements) are used. An efficient way to debug particle methods (in 1 or 2 dimensions) is usually to plot the particles. In this small example, we can also plot the particles along with the ground truth as a contour plot of $\pi_n$ (obtained by evaluating $\pi_n$ in each pixel in your plot). It could look something like this when it is working:

![Particle plots](image)

The black dots are the particles, the red circle is the true position of $x_0$ and the contours are proportional to $\pi_n$, for $n = 1, \ldots, 10$.

(f) As you may have noticed, it is fairly easy to run into numerical problems with the particle weights already in this toy example (if you have not, try a small number of particles ($N \approx 5$) and a large number of measurements $T \approx 300$, and watch out for all particle weights being rounded off to 0). In many cases, it is therefore advisable to compute and work with the logarithms of the particle weights instead. Re-implement your algorithm using the logarithms of the weights instead.

(g) (Optional) You may also have noticed that the particles sometimes appears ‘to lag behind’ if the sequence $\pi_1, \ldots, \pi_P$ evolves ‘too fast’. A remedy is to repeat the number of MH steps for each $n$. That is, repeat Step 5 in the pseudo algorithm above $K$ times, within the bigger for-loop. Can you improve the performance of your algorithm? (A vague, but intuitive, measure of performance is simply to look at plots, similar to those above, and see how well the particles ‘follows’ to $\pi_n$.)

By setting $P$ to 1 (the number of transition steps) and increasing $K$, we get a standard MCMC algorithm (and not a SMC algorithm anymore). See if you can compare the performance by, e.g., plotting the mean square error as a function of computational time.

(h) (Optional) To make the problem we are solving more interesting, assume another measurement model. Instead of measuring the objects absolute position, we are now measuring its relative distance to some sensors $s_j$,

$$y_j = \|x_0 - s_j\| + n,$$

where $n$ still is exponentially distributed (but one-dimensional), and $s_j$ denotes the coordinates for sensor $j$. (Note that the distance between the sensor and the object is known (except for the noise), but not the angle.)

Update your code using this measurement model instead. What does the posterior look like for the case of only one sensor in the origin? What about multiple sensors with different locations? How well is the SMC sampler doing in each scenario, respectively?
References


