More Statistics for IIB Project

Name: Zhenhua Lin
Crsid: zhl24
College: Fitzwilliam College

February 1, 2024

Abstract

1 Understanding of the Big Story

In Michaelmas term (work documented in the previous report), what we did was building a framework for generating and inferring state space models driven by Levy process, NVM process as the most discussed example. In the inference part, we assumed that the exact driving process is known with full details, including the jump sizes and times, making the rest of the system Gaussian (hence called a conditionally Gaussian system), and we implemented a Kalman filter to infer in such case. In this work, we first narrow down the assumption of knowing the exact process driving the system with full details to knowing the exact generator for the driving process, and use it as the proposal for a bootstrap particle filter which can be used alone or combined with a Kalman filter to form the marginalized particle filter to infer in the system. After that, we probably would further narrow down the assumption to knowing the kind of process and infer the parameters? And eventually no prior knowledge?

2 Particle Filter

Inference in stochastic system only has closed form solution for the posterior in some very limited cases, including the simple linear Gaussian model (system with linear dynamic equations i.e. not quadratic or sinusoidal and driven by Gaussian noise) and the general discrete hidden Markov model. In the vast majority of cases, non-linearity or non-Gaussianity render an analytic solution intractable.

For the Gaussian systems with non-linear dynamics, one classic solution is the extended Kalman filter. The principle is simple, by using Taylor expansion to have a linear approximation of the non-linear dynamics that is compatible with Kalman filter. This method, obviously, fails if the system has substantial non-linearity, or the system is driven by highly non-Gaussian noise. There are some other approaches based on approximation of the non-Gaussian distribution, such as via mixture of Gaussians or second order Taylor series, but all of them have limitations to the form of the posterior distributions.

We hence want a method that can achieve non-linear filtering with no requirements in the form of posterior distribution. And methods based on Monte Carlo satisfy such needs, but with a significant cost in the computational power. However, it is still quite a popular choice because of its generality that allows inference of full posterior distributions in general state-space models.

2.1 General Bayesian Filtering Problem

The main reference for this session is the paper about particle filter tutorail.

Here we study the most important sampling approach for hidden states, particle filtering. Particle filter does similar functions as Kalman filter, but the difference is that the particle filter can be applied to both linear and non-linear dynamics. The filter enables representing state estimates with arbitrarily shaped probability distributions. The key questions lie in how to set up a particle filter for the inference problem, what would be the outputs and how would the outputs be fed to the system to provide effective inference.

As mentioned before, particle filter solves the problem of similar structure but compatible with non-linear dynamics:

1. Process model for state changes over time given noise and optional inputs:

$$\mathbf{x}_k = f_k(\mathbf{x}_{k-1}, \mathbf{u}_k, \mathbf{v}_{k-1}) \tag{1}$$

2. A measurement model of the form:

$$\mathbf{z}_k = \mathbf{h}_k(\mathbf{x}_k, \mathbf{u}_k, \mathbf{n}_k) \tag{2}$$

   The main difference is that both **f** and **h** can be arbitrarily shaped distributions.

The particle filter is a Bayesian filter i.e. estimation is performed using Bayesian theory, allowing for estimating a state by combining a statistical model for measurement(likelihood) with a prior probability. The estimation is solved in a similar form to Kalman filter that applies recursive predict-update cycle.

The main steps for such Bayesian filter have the same formulae (both Kalman and Particle filters belong to this class):

1. The **predict step** basically combines the posterior estimate of the previous state with the process model to estimate the current state without observation, via a marginalizing integral. The specific formula can be obtained via marginalizing the posterior estimate of the current state given previous observations. This essentially makes use of the hidden Markov structure.

$$p(\mathbf{x}_k|\mathbf{z}_{1:k-1}) = \int p(\mathbf{x}_k|\mathbf{x}_{k-1})p(\mathbf{x}_{k-1}|\mathbf{z}_{1:k-1})d\mathbf{x}_{k-1} \tag{3}$$

   This formula can be understood as the forward simulation from the previous state posterior prediction making use of the system hidden state transition dynamics. The weighted sum of the forward simulations varied in each initial hidden state is the final prediction of the next hidden state, i.e. Bayesian prediction.

2. The **update step** is basically to estimate the current state after a current observation by combining the prediction posterior and emission mechanism (simply **h**) and also a marginalisation constant via the Bayes theorem. Essentially negative feedback of the prediction using the observation at the corresponding time position.

$$p(\mathbf{x}_k|\mathbf{z}_{1:k}) = \frac{p(\mathbf{z}_k|\mathbf{x}_k)p(\mathbf{x}_k|\mathbf{z}_{1:k-1})}{p(\mathbf{z}_k|\mathbf{z}_{1:k-1})} \tag{4}$$

   Where $p(\mathbf{z}_k|\mathbf{x}_k) = \mathbf{h}_k(\mathbf{x}_k, \mathbf{u}_k, \mathbf{n}_k)$ and $p(\mathbf{x}_k|\mathbf{z}_{1:k-1})$ is just from the last step.

   The more difficult one is the denominator, the normalizing term, which is also updated in every update step by combining the predict posterior probability with the emission mechanism. This is important in some cases with dynamic structure e.g. noise parameters as the hidden states to be predicted.

$$p(\mathbf{z}_k|\mathbf{z}_{1:k-1}) = \int p(\mathbf{z}_k, \mathbf{x}_k)p(\mathbf{x}_k|\mathbf{z}_{1:k-1})d\mathbf{x}_k \tag{5}$$

   Pay attention to the time index in the formulae which provide very important intuition but may be messed up.

The integrals in the predict and update steps for Bayesian filters as shown in equation (3) and (5) are only analytically solvable under strong assumptions : finite dimensional discrete state variables or linear models with Gaussian pdfs. What a particle filter does is basically approximating the posterior pdf by a discrete pdf such that there are minimal restrictions in the models involved. The optimal Bayesian solution is approximated by a sum of weighted dirac delta samples. Particle filtering is essentially an approximation of the whole probability space via a combination of multiple pointwise evaluation carried by the offseted delta function corresponding to the true place.

$$p(\mathbf{x}_{0:k}|\mathbf{z}_{1:k}) \sim \sum_{i=1}^{N_s} w_k^i \delta(x_{0:k} - x_{0:k}^i) \tag{6}$$

The intuition behind particle filtering is that, we have multiple paths or particles each of which represents a specific evaluation in the whole probability space. Therefore, if we can have an infinite number of particles, we could ideally have a complete evaluation of the whole probability space. (This summation formula is not especially meaningful just to point out that this is an approximation for the true space).

Where $\{w_k^i, \mathbf{x}_{0:k}^i\}_{i=1s}^{N}$ is the set containing N_s sample sequences and weights, with each weight representing the relative importance of each of sample sequences, and the sum of weights is 1. Samples associated with high weights are believed to be closer to the true state sequence than samples associated with low weights. The integral of the sum of delta functions i.e. equation (6) is hence the summation of the weights and equal to one, thus a valid probability distribution approximation.

The discrete approximation of a continuous pdf turns intractable integrals into summations over N_s samples, which are often referred to as the particles.

2.2 Algorithms for Particle Filtering (The following is mainly just for the algorithms, specific derivation should check professor Godsill's overview paper)

The discrete approximation of a continuous pdf turns intractable integrals into summations over N_s samples. The samples are usually referred to as particles hence the name particle filter. The advantages of representing the posterior by a set of weighted particles include (i) the ability to represent arbitrarily shaped pdfs (assuming enough samples) and (ii) minimal restrictions on the process and measurement models. This combination of advantages is one of the main reasons for the popularity of the particle filter. The most obvious drawbacks are (i) the lack of expressiveness in case the number of particles is too low and (ii) the increased computational costs compared to the usual inference methods.

The sample-based approximation comes with an obvious challenge. The posterior pdf that must be estimated is unknown hence sampling from it is impossible. Samples must therefore be drawn from another distribution instead. This distribution, referred to as importance density (sometimes called proposal density) will be denoted q. The weights compensate for the fact that samples are drawn from the importance density q rather than the posterior pdf. Any function that is positive where the posterior is positive can be used as importance density (sequential importance sampling).

Particles are basically just samples with weights in the filtering scheme.

2.2.1 The Most Basic Particle Filter: Sequential Importance Sampling (SIS)

The simplest particle filter algorithm is hence just sequential importance sampling, and we assume q to be a generator function we have and p to be the desired distribution. In the filtering scheme, both q and p refer to the transition density function in the system.

1. We first define the number of particles we want for the algorithm: N_s, and we draw N_s samples or particles from the prior distribution $q(\mathbf{x}_0)$ and assign the initial weights for each sample as :

$$w_0^i \propto \frac{p(\mathbf{x}_0^i)}{q(\mathbf{x}_0^i)} \tag{7}$$

   The weights are always defined as the ratio of the posterior hidden states probability given observations (none for prior) to the importance probability of observing the data sequence from the generator probability.

2. Then in each step, we propagate the particles in last step: $\{x^i_{k-1},w^i_{k-1}\}^{N_s}_{i=1}$as the current observations $\mathbf{z}_k$ arrive, to obtain the new particles $\{x^i_k,w^i_k\}^{N_s}_{i=1}$.

   We first draw samples from the proposal distribution q, conditional on the last trajectory of the particle and the current observation (need Bayesian expansion to evaluate, note that this is not the transition density):
$$x^i_k \sim q(x^i_k|x^i_{k-1}, \mathbf{z}_k) \tag{8}$$

   Next we assign weights to the samples to make them particles, which equal to the ratio of the exact posterior distribution to the proposed one, but note that these are evaluated over the whole trajectory for each particle, not just the instantaneous values:
$$w^i_k \propto \frac{p(\mathbf{x}^i_{0:k}|\mathbf{z}_{1:k})}{q(\mathbf{x}^i_{0:k}|\mathbf{z}_{1:k})} \tag{9}$$

   But note that this equation is not trivial to evaluate directly, and instead, we solve it by using a similar recursive update form by rewriting the equation as:
$$w^i_k \propto w^i_{k-1}\frac{p(z_k|x^i_k)p(x^i_k|x^i_{k-1})}{q(x^i_k|x^i_{k-1}, z_k)} \tag{10}$$

   with the numerator terms being the emission and transition density functions, and the denominator term being the proposal density function that needs Bayesian expansion to evaluate (samples generated not just based on the last states but also the current observation, two inference paths). Use simple sum-to-one constraint to have the exact expression from the variation part.

   Then we have the particles for the current time step: $\{x^i_k,w^i_k\}^{N_s}_{i=1}$

3. The particles give us a probability distribution of the hidden states at the time step accoridng to formula (6) i.e.
$$p(\mathbf{x}_{0:k}|\mathbf{z}_{1:k}) \sim \sum_{i=1}^{N_s} w^i_k\delta(x_{0:k} - x^i_{0:k}) \tag{11}$$

   with the sum of weight history of a particle equal to the probability of the hidden states being the trajectory of a specific particle (discrete distribution approximation over some particular sequences). Then we could do something like ML estimate to infer the hidden states of the noisy signal using non-linear dynamics.

An additional thing to note is that we often use the normalized weights for the particles instead. So the weight of the ith particle now becomes:
$$\hat{w}^{(i)} = \frac{w^{(i)}}{\sum_j w^{(j)}} \tag{12}$$

This has a simple derivation:

1. Say the target of estimate is some random function h(x), such that we want to know the expectation:
$$E(h(x)) = \int h(x)p(x)dx \tag{13}$$

2. Again by importance sampling, we have the estimation formula:
$$E(h(x)) = \int h(x)p(x)\frac{q(x)}{q(x)}dx \sim \frac{1}{N}\sum_i \frac{p(x^{(i)})}{q(x^{(i)})}h(x^{(i)}) = \frac{1}{N}\sum w^{(i)}h(x^{(i)}) \tag{14}$$

3. To use the normalized weights, we need an additional relation:

$$1 = \int p(x)dx = \int p(x)\frac{q(x)}{q(x)}dx \sim \frac{1}{N}\sum_i \frac{p(x^{(i)})}{q(x^{(i)})} = \frac{1}{N}\sum_i w^{(i)} \tag{15}$$

   The key to note is that we have the similarity:

$$N \sim \sum_i w^{(i)} \tag{16}$$

4. Finally, we put the expression for N in (16) into (14) to obtain the estimation formula in the normalized weights:

$$E(h(x)) \sim \sum_i \frac{w^{(i)}}{N} h(x^{(i)}) = \sum_i \frac{w^{(i)}}{\sum_j w^{(j)}} h(x^{(i)}) = \sum_i \hat{w}^{(i)} h(x^{(i)}) \tag{17}$$

   Note that the important thing is that h can be anything we want to estimate. The most straight forward example is the mean E(x), which would be just the weighted sum of the particles, way more reasonable than the simple average.

2.2.2 Bootstrap Particle Filter: SIS with Resampling (For specific derivation, check the PF paper by Professor Godsill, the following is just the algorithm)

However, the simplest SIS algorithm has several problems and modifications are needed to make it an effective particle filter algorithm. The most famous problem is the weight degeneracy problem.Weight degeneracy often occurs in scenarios where the likelihood function sharply contrasts the prior or proposal distribution. In such cases, particles that are not close to the high probability regions of the likelihood function end up with negligible weights. Another common cause is the accumulation of errors over time in the estimated state, which leads to a situation where most particles fail to predict the observed data accurately, resulting in their weights becoming insignificant. Note that this doesn't have much relation to the MH problem of acceptance probability tending to 0.

The solution to that is by implementing a sequential importance sampling algorithm with resampling for the particles.

The problem is defined as below:

After a few iterations one particle weight will be very close to one and all other particle weights will be almost zero. The consequences of the degeneracy problem are (i) almost all computational effort will be put into computations related to particles that have negligible or no contribution to the overall estimate and (ii) the number of effective particles is only one. The latter greatly limits the performance and expressiveness of the filter. The expressiveness is limited since a single particle can only represent one point in the state space rather than pdfs of arbitrary shapes. Performance is poor since the particle filter will diverge. Divergence occurs in case state estimation errors increase over time and are unacceptably large. Once diverged, the filter fails to "follow" the true state or reduce estimation errors to acceptable values.

The solution to this problem is resampling. In the most basic resampling, say we have decided to have N trajectories, using the normalized weights as the probabilities of selection of the particles, we draw N particles with replacement. Therefore, trajectories with small importance weights are eliminated, whereas those with large importance weights are replicated.The normalized importance weights are set to 1/N after the resampling step.

Therefore we have an algorithm for the basic particle filter with resampling:

1. To start the algorithm, we need several inputs, including the number of trajectories we want N, initial particles $\{X_0\}$, observation data, emission function from samples to observation g, and the transition function for particles q (we denote f as the actual transition function). In bootstrap particle filter, f=q directly.

   Note also that we need to compute the initial weights for the initial set of particles as in (18) which shows the weight for the ith particle without normalization yet. Note that we are only interested in the contribution form the ith particle, so the condition is on the ith hidden state but not the whole hidden states in the beginning, where π_0 is the initial state distribution. We need the initial probabilities for the weights. Note that we need to normalize the weights after the computation. **Note what weights except the normalised weights should all be in the log domain! Normalisation should also be via logsumexp.**

$$w_0^{(i)} = \frac{g(y_0|x_0^{(i)})\pi_0(x_0^{(i)})}{q_0(x_0^{(i)}|y_0)} \tag{18}$$

The transition function should take the particles and return the propagated particles to the next time step through the system. The emission function should take the observation and particles at the same time step to return the corresponding emission probability.

The initial particles should be sampled form some initial prior.

2. Then we run the algorithm through each time step in the observation data to infer the hidden states. Note that SIS method applies to time series data.

3. In each iteration after initialization, we perform **resampling** first for the particles from the last step, based on the normalized weights $\{\hat{w}_{t-1}\}$ which are taken directly as the selection probabilities. We draw N particles with replacement according to the normalized weights. After resampling, we set the normalized weights of the resampled particles evenly to $\frac{1}{N}$.

   Then we perform the **propagation** step for the resampled particles based on the transition kernel function provided (return new proposed samples directly). For the ith trajectory (often no dependence on observation y, just the previous particles):

$$x_t^{(i)} \sim q(x_{t-1}^{(i)}, y_t) \tag{19}$$

   Finally we **update and normalize the weights**. The unnormalized weights are computed according to equation (20). Note that we need the kernel to compute the transition probability in the general case. Remember to normalize the weights. w_{t-1} is basically the uniform weights passed form the last step.

$$w_t^{(i)} = w_{t-1}^{(i)} \frac{g(y_t|x_t^{(i)})f(x_t^{(i)}|x_{t-1}^{(i)})}{q_t(x_t^{(i)}|x_{t-1}^{(i)}, y_t)} \tag{20}$$

   But for the **weight update in the bootstrap particle filter**, f=g and we need only the emission probability as shown in (21). Remember also to normalize the weights.

$$w_t^{(i)} = w_{t-1}^{(i)} g(y_t|x_t^{(i)}) \tag{21}$$

2.3 Particle Filter for Levy State Space Model

We first implement a basic bootstrap particle filter to the Levy state space model. Hence, what we need are the transition function for propagating the particles, and the emission function for computing the probability of a particle given an observation.

First, revise the formula for the Levy state space model driven by an NVM process, the Normal Gamma process to be more specific, as shown in equation (22).

$$\mathbf{X(t)} = \mathbf{e}^{\mathbf{A}(t-s)}\mathbf{X}(s) + \sum_{\frac{s}{T} \le V_i \le \frac{t}{T}} [\mu_W Z_i + \sigma_W \sqrt{Z_i} U_i] \mathbf{e}^{\mathbf{A}(t-V_i)}\mathbf{h} \tag{22}$$

We further evaluate it into the form that is convenient for simulation, with Δt being the simulation time step size, $\{Z_i, V_i\}$ being the jump sizes and times for the subordinator process, and σ_w μ_w being the NVM parameters. Note that such system has a fixed transition function given uniform simulation time step, but the system exponent in the noise has to be calculated for each jump and evaluation point. We compute the new sample by simulate the driving process first and then finidng the corresponding jumps in the interval using formula (22). **Note that in our simulation scheme, we divide the time axis into many dt intervals. So the jump times we obtained are essentially $\tau_i = V_i - t$, thus whats inside the bracket is just minus of the offseted jump time in the dt interval $-\tau_i$. Hence, we dont need to bother with the exact time point of the system t**.

$$X_{n+1} = e^{\mathbf{A}\Delta t} X_n + \sum_{\frac{t-\Delta t}{T} \leq V_i \leq \frac{t}{T}} [\mu_W Z_i + \sigma_W \sqrt{Z_i} U_i] \mathbf{e}^{\mathbf{A}(t-V_i)} \mathbf{h} \tag{23}$$

The observations are again the true hidden states covered by Gaussian noise of some variance.

$$\mathbf{Y}_n = \mathbf{X}_n + \sigma \mathbf{U}_n \tag{24}$$

Then, what we need to do is to draw samples from the transition function f($\mathbf{X}_{n+1}|\mathbf{X}_n$) for forward simulation or sampling and define the observation function g($\mathbf{Y}_n|\mathbf{X}_n$) i.e. the weight function to run a bootstrap particle filter. Note that the transition function here probably does not have a closed form, and there is no need to define it, because we just need to simulate forwards to have new samples proposed. That is for the transition function, all we need is the exact driving process generator to simulate forwards by dt to obtain the jump sizes and times.

We can therefore understand particle filtering scheme as simulating multiple paths to try to reconstruct the real one. It is just having multiple simulations from zero, and using weights as the measure for matching of a simulation to the real one (constrained by weight which is higher for more likely simulations given the observations). It is something like the weighted sum of all simulated paths. The observation probability is shown in equation (25). Note that since additional normalisation step would be carried out, and the normalisation constant is the same for all samples since it depends on the data only, and as mentioned in the paper, that would be constant for fixed data filtering scheme, there is no need to include it in the function. We just need a score for matching.

$$g(\mathbf{Y}_n|\mathbf{X}_n) = N(\mathbf{X}_n, \sigma^2\mathbf{I}) \sim exp((\mathbf{Y}_n - \mathbf{X}_n)^T(\mathbf{Y}_n - \mathbf{X}_n)/2\sigma^2) \tag{25}$$

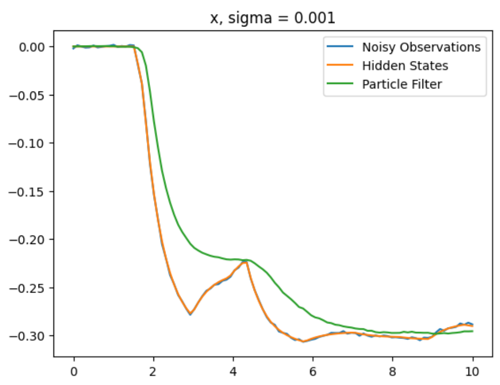
The main problem in building the particle filter transition function is that since it is just some random process generator, but we do not necessarily know the type of process or the exact parameters for such generator. For this case, we would just assume that we know exactly the type of process and parameters, by using the exact generator that generates the data, and leave such problem for latter discussion.

Note that in coding, always remember that sample generation always has the chance of returning empty array which could cause dimension mismatch.

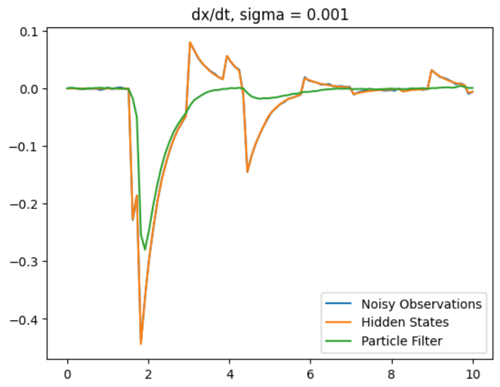
Hence, we have a simple implementation of particle filtering in the Levy state space model. Some example results are shown in figure 1 below.

3 Marginal Particle Filter for NVM State Space System

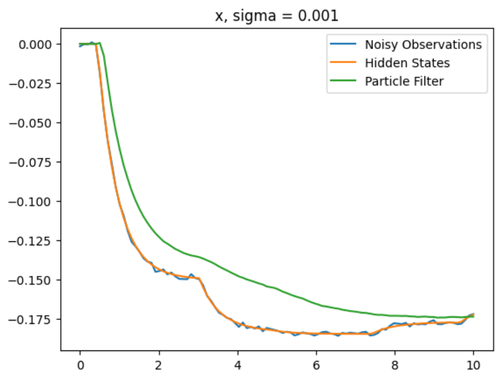
Marginal particle filter can be applied to problems where the linear and non-linear parts have clean separation. This is exactly the case of our system driven by NVM process, where if we condition on the non-linear states, the rest of the states are just simple Gaussian process. In marginal particle filter, instead of using particle filter to propose the complete dynamics of the system, we use it to propose only the non-linear part of states in each time step. And condition on the non-linear proposal, the rest of the states are just simple Gaussian which can be estimated by linear approach i.e. Kalman filter. Such clean separation is aligned with the nature of a system driven by the NVM process, which sequentially generates the non-linear subordinator process first and then generates the linear process conditional on the non-linear parts. The main technical difficulty lies in how to compute the weights of the particles in this case, and would be discussed in the following session with comparison with the previous pure particle filter case.



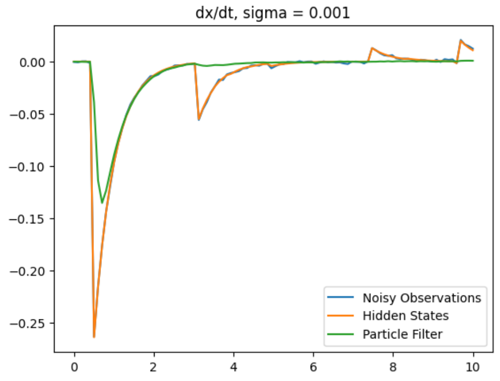
(a) Particle Filtering with 1000 Particles (x)



(b) Particle Filtering with 1000 Particles ($\frac{dx}{dt}$)



(c) Particle Filtering with 10000 Particles (x)



(d) Particle Filtering with 1000 Particles ($\frac{dx}{dt}$)

Figure 1: Particle Filtering Implementation Example Results

3.1 Derivation of Marginal Particle Filter Algorithm for Levy State Space System

We again begin by listing out the Levy state space dynamic model as shown in equation (26). Note that in the previous pure particle filtering case, the proposal generates the jump times and the NVM jump sizes (whats inside the bracket). The special thing is to note that that would include both the linear dynamics U_i and the non-linear dynamics Z_i. A seemingly appealing structure would be proposing the same NVM jump sizes again, and hence the weight function is p(Y|proposal) would be equivalent to p(Y|X), the emission probability again, since the NVM jump sizes and times form deterministic transformation to the hidden state, and then conditional on NVM jump sizes we know the subordinator jumps also (information subset) and can have the mean and covariance of the conditional system for us to run a Kalman filter to give the final output. But that would not be a valid marginal particle filter, and it is in fact doing more or less the same thing as the standard particle filter statistically, but only with an additional Kalman filtering step which introduces more uncertainty, because that proposal is still proposing the combined states.

$$X_{n+1} = e^{\mathbf{A}\Delta t} X_n + \sum_{\frac{t-\Delta t}{T} \leq V_i \leq \frac{t}{T}} [\mu_W Z_i + \sigma_W \sqrt{Z_i} U_i] \mathbf{e}^{\mathbf{A}(-\tau_i)} \mathbf{h} \tag{26}$$

Therefore, what a valid proposal should look like should be proposing only the subordinator jump sizes Z_i and times, leaving the Gaussian part in the NVM jump U_i to be inferred by a Kalman filter, such that we can make use of the nature of Kalman filter as the optimal inference solution to improve the inference results. **(The key thing to realize is that, the nature of the SDE system depends completely**

on the underlying driving process. The separation between linear and non-linear parts of the system lies in the separation property in the NVM process, but not the system.)

$$\mathbf{m} = \mu_W \sum_{\frac{s}{T} \leq V_i \leq \frac{t}{T}} Z_i \mathbf{e}^{\mathbf{A}(-\tau_i)}\mathbf{h} \tag{27}$$

$$\mathbf{S} = \sigma_W^2 \sum_{\frac{s}{T} \leq V_i \leq \frac{t}{T}} (\sqrt{Z_i}\mathbf{e}^{\mathbf{A}(-\tau_i)}\mathbf{h})(\sqrt{Z_i}\mathbf{e}^{\mathbf{A}(-\tau_i)}\mathbf{h})^T \tag{28}$$

Then, we can define the weights for particles using p(observation|proposal) = p(Y|Z_i, τ_i). Note that these conditions would be equivalent to the mean and variance in the conditionally Gaussian term, since the transformation between them is deterministic as shown in (27) and (28). The weight would hence be calculated by p(Y|m,S) and that is in fact the marginal likelihood in the conditional Gaussian system, and we can estimate it via Kalman filter directly. The conditionally Gaussian system structure we would be working on is in the form of standard hidden Markov state transition and emission structure, with $V_n \sim N(\mathbf{m}, \mathbf{S})$. Note that the transition noise is a biased Gaussian noise here, simply add the bias to the deterministic transition and the noise would become zero mean, and the system would be compatible with the classical zero mean noise Kalman structure again.

$$X_{n+1} = FX_n + V_n \tag{29}$$

$$Y_n = HX_n + W_n \tag{30}$$

The marginal weights according to the Kalman filter notes by professor Godsill should be calculated as shown in equation (31).

$$p(y_t|y_{1:t-1}) = N(\hat{X}_{t|t-1}, S_{x_t} + \sigma_n I) \tag{31}$$

This is in fact simply taking expectation wrt all terms using the simple hidden state emission linear equation. The particles here are just the conditional Gaussian parameters computed using the gamma jump sizes and times and the system structure. Inference of the hidden states is done by running a Kalman on the conditionally Gaussian system. σ_n is just the observation noise covariance matrix.

Therefore, a complete marginalised particle filtering algorithm would look like:

1. We first define the number of particles and the proposed system for generating the subordinator processes for particle filtering. Give also some initialisation to particle filter and Kalman filter.

2. In each iteration, we first use the proposed system to generate the non-linear states, subordinator jump sizes and times, as the particles. These could be transformed deterministically and directly into the Gaussian mean and variance in the conditionally Gaussian system, like in equation (27) and (28). Note that this generation is independent in different iterations. Also we don't have enough information to compute the weights just based on the non-linear states here.

3. Each particle would represent a conditionally Gaussian system (deterministic transformation between subordinator process and conditional Gaussian parameters). We then run a Kalman filter for each particle for one transition and correction step to infer the hidden states. The marginal probability can be computed directly in the Kalman correction step, which is exactly the weight for the corresponding particle. We then normalize the weights (always in the log domain!) for resampling.

4. Then, we can eventually do resampling of particles. Note that, not only just the particles need to be resampled, but also the inferred states by Kalman filter. We need to re-run the Kalman filter for each resampled particle to produce the new inference results (2 runs!). The new weights should also be calculated.

5. Finally, we can combined the inference results using the new samples. That should be done before resampling using the true weights and brand-new predictions, since that would correspond to the lowest Monte Carlo error, though after resampling using the new weights computed **(not the uniform weights re-assigned! Bad performances!)** still has quite good performances.

3.2 Extended State in Marginalised Particle Filter for Parameter Estimation

This is a method mentioned in the Levy state space model paper, that is particularly for the marginalised particle filtering method, based on the bias term of the Gaussian noise in the conditionally Gaussian system, which carries information about μ_w in an NVM process so we are essentially inferring the parameter while we are inferring the mean of the conditional Gaussian distribution.

We again start from the 3 important equations for derivation of such method in the NVM system. What this mainly implies to the non-linear state proposal is that, we now no longer have the full information to deterministically transform the jump sizes and times to the conditional Gaussian distribution mean parameter (equation (33) can no longer be used here) , and the variance can still be deduced in the same way as before, since now μ_w is taken away as a state.

This changes the hidden Markov structure, especially the key driving conditional Gaussian distribution, since the mean is now taken out to be used as an external state. The driving Gaussian is now a zero mean one with the same covariance structure as before.

$$X_{n+1} = e^{\mathbf{A}\Delta t}X_n + \sum_{\frac{t-\Delta t}{T} \leq V_i \leq \frac{t}{T}} [\mu_W Z_i + \sigma_W \sqrt{Z_i} U_i]\mathbf{e}^{\mathbf{A}(-\tau_i)}\mathbf{h} \tag{32}$$

$$\mathbf{m} = \mu_W \sum_{\frac{s}{T} \leq V_i \leq \frac{t}{T}} Z_i \mathbf{e}^{\mathbf{A}(-\tau_i)}\mathbf{h} \tag{33}$$

$$\mathbf{S} = \sigma_W^2 \sum_{\frac{s}{T} \leq V_i \leq \frac{t}{T}} (\sqrt{Z_i}\mathbf{e}^{\mathbf{A}(-\tau_i)}\mathbf{h})(\sqrt{Z_i}\mathbf{e}^{\mathbf{A}(-\tau_i)}\mathbf{h})^T \tag{34}$$

We can see how to include the extended state immediately once we formulate the conditionally Gaussian hidden Markov system with bias extracted to make the noise zero-mean, as shown in equation (35), with $V_n \sim N(0, \mathbf{S})$. The mean of the noise is extracted as the additional constant. This bias naturally carries the information about μ_w, and can hence be used for estimating μ_w in the NVM process.

$$X_{n+1} = e^{\mathbf{A}\Delta t}X_n + \mu_W \sum_{\frac{s}{T} \leq V_i \leq \frac{t}{T}} Z_i \mathbf{e}^{\mathbf{A}(-\tau_i)}\mathbf{h} + V_n \tag{35}$$

Since μ_w is no longer a part of the conditional Gaussian distribution, we need to define our new state space system with an extended state of μ_w. Hence the state vector is now shown in equation (36). Note that we need not go back to our exact differential equation to come up with a new $\mathbf{A}$, and what we need to do is just to define a new hidden Markov model with the new state vector.

$$\hat{\boldsymbol{X}} = \begin{bmatrix} \boldsymbol{X} \\ \mu_w \end{bmatrix} = \begin{bmatrix} x \\ \frac{dx}{dt} \\ \mu_w \end{bmatrix} \tag{36}$$

Here, we outline the marginalised particle filtering scheme with an augmented state.

1. First, we have our particle filter proposing the jump sizes and times of the subordinator process as usual. But, here we convert them into the summation term instead of the mean, and the noise covariance matrix predicted in the same way.

2. The hidden Markov transition model is hence as shown below. The extended state equation is just a simple equality, since it is some constant parameter with no evolution. We now rely on the particle filter to propose the summation term $\sum_{\frac{s}{T} \leq V_i \leq \frac{t}{T}} Z_i \mathbf{e}^{\mathbf{A}(-\tau_i)}\mathbf{h}$ instead of the mean, and the covariance matrix for V_n.

$$\hat{\boldsymbol{X}}_{n+1} = \begin{bmatrix} e^{\mathbf{A}\Delta t} & \sum_{\frac{s}{T} \leq V_i \leq \frac{t}{T}} Z_i \mathbf{e}^{\mathbf{A}(-\tau_i)}\mathbf{h} \\ 0 & 1 \end{bmatrix} \hat{\boldsymbol{X}}_n + \begin{bmatrix} V_n \\ 0 \end{bmatrix} \tag{37}$$

Or to be more detailed:

$$\begin{bmatrix} \boldsymbol{X}_{n+1} \\ \mu_w \end{bmatrix} = \begin{bmatrix} e^{\mathbf{A}\Delta t} & \sum_{\frac{s}{T} \leq V_i \leq \frac{t}{T}} Z_i \mathbf{e}^{\mathbf{A}(-\tau_i)}\mathbf{h} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{X}_n \\ \mu_w \end{bmatrix} + \begin{bmatrix} V_n \\ 0 \end{bmatrix} \tag{38}$$

Just equation (35) but in a different convention. Note that the combined noise term would have a covariance matrix as shown in equation (39). Since the augmented state μ_w is not masked by any noise in transition.

$$\begin{bmatrix} \Sigma_{V_n} & \mathbf{0} \\ \mathbf{0} & 0 \end{bmatrix} \tag{39}$$

3. Next is the emission model. That would be basically the same as before, since μ_w is not observed in this case (it is not like derivatives of x, so it does not have any embedded information from it, and hence no observation?). The original observation model is:

$$\mathbf{Y}_n = \mathbf{G}\mathbf{X}_n + W_n \tag{40}$$

The new emission model is just:

$$\mathbf{Y}_n = [\mathbf{G}_n, \mathbf{0}] \begin{bmatrix} \boldsymbol{X}_{n+1} \\ \mu_w \end{bmatrix} + W_n \tag{41}$$

With W being the same noise as before.

4. Another important thing is the particle weight, $p(\mathbf{Y}_n|\{Z_i\}, \{\tau_i\})$. This is slightly different from the previous case, since now the jump sizes and times can no longer transform deterministically to the conditional Gaussian mean and variance. Now, they are equivalent to the summation term in the hidden Markov transition model matrix and the covariance matrix in the HMM transition noise. Therefore, conditional on the jump sizes and times, the weight probability magically turns into the marginal probability of observation again, which is perfectly compatible with our original MPF strutcure.

3.3 The New Hidden Markov Model with Partial Observation

Now, we change our hidden Markov model of Langevin dynamics, such that in the observation model, instead of observing the full states masked by noises, we observe one of them only, such that the other one is effectively obtained by integration or differentiation of the other state. Here, we choose to observe the state x, following the convention in the Levy state space model paper, and infer the $\frac{dx}{dt}$ state, that's essentially differentiation. The model hence looks like:

$$\mathbf{dX(t)} = \mathbf{AX(t)}dt + \mathbf{h}dW(t) \tag{42}$$

The derivative relationship is contained in matrix $\mathbf{A}$, so there is no need to manually perform integration or differentiation, once we perform simulation or inference using the corresponding $\mathbf{A}$.

The solution to this SDE is again:

$$X_{n+1} = e^{\mathbf{A}\Delta t} X_n + \sum_{\frac{t-\Delta t}{T} \leq V_i \leq \frac{t}{T}} [\mu_W Z_i + \sigma_W \sqrt{Z_i} U_i] \mathbf{e}^{\mathbf{A}(-\tau_i)}\mathbf{h} \tag{43}$$

$$\mathbf{A} = \begin{bmatrix} 0 & 1 \\ 0 & \theta \end{bmatrix} \tag{44}$$

$$\mathbf{h} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{45}$$

The key difference is in the observation model. Instead of the classical identity matrix which means selecting all states, it is now a row vector to have dot product with the states to select a particular state here.

$$\mathbf{Y}_n = \mathbf{G}\mathbf{X}_n + \sigma_n U_n \tag{46}$$

And now, U_n is a 1D independent Gaussian noise. $\mathbf{G}$ is a row vector with all elements 0 except a 1 at the state selected.

$$\mathbf{G} = [1, 0] \tag{47}$$

The main change in such model is then in the Kalman correct function or other likelihood function (since we are using the marginalised particle filter here, the likelihood function is incorporated in the Kalman correction function, since the likelihood is the marginal likelihood computed in the Kalman correction step). We just change G and the Cholesky decomposition of the covariance matrix of the noise, which is just the scalar std in this case.

4 Particle MCMC

4.1 Comparison between Sequential Monte Carlo and Markov Chain Monte Carlo

Particle filtering is a **sequential Monte Carlo algorithm**, that samples from a state space system **with time evolution** like a Markov chain. **MCMC**, on the other hand, samples from a **static** distribution by using proposal to explore the sample space constrained by the acceptance probability (basically a measure of likelihood of the proposal compared to the current position). SMC samples based on the Markov structure of the state space system, whereas MCMC samples from a known distribution that is more similar to the distribution of variables at a particular time step in a Markov chain.

These are both Monte Carlo based algorithms, which implies that they can do general distributions, including those with non-linear dynamics, but they are both quite computationally expensive.

4.2 The Foundation for MCMC Algorithm: Metropolis Hasting (Reference from 4M24 Lectures)

Metropolis Hasting is an algorithm derived for sampling from the invariant distribution of a Markov chain, with the invariant distribution being the target distribution. More advanced algorithms such as Gibbs sampler and Langevin MCMC are just Metropolis Hasting using the exact proposal (acceptance probability equal to 1) and using Langevin dynamics for the proposal from a given state.

We always tend to sample from the stationary distribution of the Markov chain $\pi(dy)$. By detailed balance assuming the reversibility of the Markov Chain that:

$$\pi(x)p(x, dy) = \pi(dy)p(dy, x) \tag{48}$$

In practice, we don't always have the exact transition function p(,), and instead, we use a proposal function q(,), but the proposal transition kernel may not satisfy the equality if not perfectly match, leading to biased transition to one side. Metropolis Hasting is a solution to this problem by introducing an acceptance probability of transition $\alpha(,)$:

$$\pi(x)q(x, y)\alpha(x, y) = \pi(y)q(y, x) \tag{49}$$

However, this is subject to the condition that the transition from x to y is more frequent than that from y to x for the proposal kernel q(,), since α is a probability and takes value from the range [0,1]:

$$\pi(x)q(x, y) > \pi(y)q(y, x) \tag{50}$$

In such case, we have the expression for the acceptance probability that enforces balance in the bidirectional transitions:

$$\alpha(x, y) = \frac{\pi(y)q(y, x)}{\pi(x)q(x, y)} \tag{51}$$

In practice, we don't swap x and y if the tilt is the other way round, and instead we introduce a unity cap to the acceptance probability as follows:

$$\alpha(x,y) = min(\frac{\pi(y)q(y,x)}{\pi(x)q(x,y)}, 1) \tag{52}$$

Note that although the variables in $\pi()$ are different in the numerator and denominator, the functions are the same and hence the normalisation constants are the same. Therefore, in computing the acceptance probability, we don't need to know the normalisation constant for π, since the one at the top and bottom cancel. (But we need for q, since these would be completely different posterior functions)

Using the acceptance probability shown in equation (52), the full proposal in MH is:

$$q_{MH}(x,y) = q(x,y)\alpha(x,y) \tag{53}$$

Note that the definition of MH still needs only the proposal, as the proposal is also the only information needed to calculate the acceptance probability. In MH, the moves to the higher probability regions are always accepted with 100% probability, but those to the lower probability regions could be rejected.

The overall algorithm for Metropolis Hasting is hence simply:

1. Simulate a sample y from $q(x^j, .)$
2. Compute $\alpha(x^j, y)$. And with the probability α(this could be obtained by generating a sample from a uniform distribution [0,1] and see if the sample is greater than α), set the next sample x^{j+1}as y, otherwise set it as the same as x^j.
3. Repeat the process and end until enough samples have been obtained.

Eventually we are sampling from the invariant distribution of the Markov chain.

Gibbs sampling is just a special case of Metropolis Hasting with acceptance probability always being 1. Metropolis Hasting is the most general way to sample from a joint distribution. It also decomposes the joint distribution into a series of product of univariate conditionals which are calculated by the kernels. The only difference is that the transition could be rejected and samples may stay in the same place.