Simulation and Inference in Limit Order Books

Choon Kiat Lee

Contents

1 Introduction

Systematic trading of financial securities is a growing field, with just over half of the trading volume in S&P 500 options attributed to high frequency algorithmic trading.

In this project, we aim to develop a *momentum*-based trading strategy. *Momentum*-based trading strategies aim to identify an ongoing price trend and then take a market position in order to benefit from its continuation. We aim to do this by developing a model for the price dynamics of the traded commodity, then use bayesian filtering techniques to track the underlying trends in the price.

The underlying dynamics of financial asset values are not clear, despite extensive study of the behaviour of asset prices. Thus, the first part of our project focuses on building a model of the underlying asset price dynamics. Since our momentum-based trading strategy relies on its ability to determine a trend to follow, we focus on developing a model which can model time-varying trends in price.

The second part of our project focuses on applying Bayesian filtering techniques for performing state inference given a state space model of price dynamics obtained in the previous part. For inference, we have explored both the use of a simple bootstrap particle filter as

well as a more sophisticated Rao-Blackwellised particle filter based on a Scale Mixture of Normals (SMiN) representation of our proposed model. We have also performed parameter estimation, and have considered both maximum likelihood estimates of the mean reversion coefficient as well as fully bayesian approaches utilising state augmentation.

Lastly, we will apply our inference techniques to a real dataset of recent EURUSD exchange rates. We apply a basic trading strategy to our model, and show that it has the potential to generate excess returns in the market.

2 Literature Review (To Do)

3 Theory

3.1 Modelling

3.1.1 Basic State Space Model

We build upon the model proposed by Christensen et al. [1] and summarised in Equation 1. In the original Christensen model, the state of the traded commodity is modelled by 2 components: "value" x_1 and "trend" x_2. These components correspond to the the price and return of the traded commodity respectively.

In this model, the trend is specified as a mean-reverting random process subject to two different types of random innovations: Gaussian noise of constant volatility and random jumps. The value component(x_1) is obtained by integrating the trend process (x_2) with respect to time. Observed prices, y are modeled as observations of the value process x_1 corrupted by random Gaussian noise.

$$\begin{bmatrix} dx_{1,t} \\ dx_{2,t} \end{bmatrix} = \underbrace{\begin{bmatrix} 0 & 1 \\ 0 & \theta \end{bmatrix} \begin{bmatrix} x_{1,t} \\ x_{2,t} \end{bmatrix} dt}_{\text{Mean Reverting Returns}} + \underbrace{\begin{bmatrix} 0 \\ \sigma \end{bmatrix} dW_t}_{\text{Brownian Motion}} + \underbrace{\begin{bmatrix} 0 \\ 1 \end{bmatrix} dJ_t}_{\text{Finite-activity Gauss-Poisson jump}} \tag{1}$$

However, sharp trend changes can cause difficulty for momentum strategies. This model attempts to address this issue by allowing for a finite number of discrete jumps in the trend process. This allows for modelling of sharp changes of sentiment in the market, and allow the filtered predictions to reflect these changes of sentiment more quickly than models that simply smooth the price series.

3.1.2 Proposed State-Space Model

Statistical testing has found that infinite-activity jumps (rather than finite-activity jumps) are present in high frequency stock returns [2]. This might be due to significant price movements triggering stop-loss orders that prompt further trading, leading to a self-exciting series of high frequency small jumps [3].

Thus, we propose modelling a tradable commodity's price using Lévy-type processes with infinite jump activity as given by Equation 3 below.

$$\begin{bmatrix} dx_{1,t} \\ dx_{2,t} \end{bmatrix} = \underbrace{\begin{bmatrix} 0 & 1 \\ 0 & \theta \end{bmatrix} \begin{bmatrix} x_{1,t} \\ x_{2,t} \end{bmatrix} dt}_{\text{Mean Reverting Returns}} + \underbrace{\begin{bmatrix} 0 \\ 1 \end{bmatrix} dL_t}_{\text{Infinite-activity Lévy process}} \tag{2}$$

$$y_t = x_{1,t} + \sigma_{obs}\eta_t \tag{3}$$

where:

$$\eta_t \sim \mathcal{N}(0, 1)$$

3.1.3 Choice of Lévy process

We choose use an α-stable Lévy process to capture the characteristics of the noise terms of the basic state space model provided in Equation 1 – namely the Brownian motion term and the jump term. An α-stable Lévy process has both a continuous gaussian component and an infinite-activity jump process component, which allows us to capture the noise characteristics proposed in Equation 1 in a concise manner. This also allows us to leverage existing literature developed for α-stable Lévy proceses to tackle the inference problem.

The family of α-stable distributions forms a rich class of distributions which allow for asymmetry and heavy tails. We define a real-valued random variable X to follow an stable distribution $S_\alpha(\sigma, \beta, \mu)$ if and only if its characteristic function is given by:

$$\mathbb{E}[e^{itX}] = \begin{cases} \exp\left(-\sigma^\alpha |t|^\alpha [1 - i\beta \text{sign}(t) \tan(\frac{\alpha\pi}{2})] + i\mu t\right) & \alpha \neq 1 \\ \exp\left(-\sigma^\alpha |t| [1 + i\beta \frac{2}{\pi} \text{sign}(t) \ln t] + i\mu t\right) & \alpha \neq 1 \end{cases} \tag{4}$$

We can then define the scalar α-stable process L_t that we will use as the driving Lévy process in our state-space model as a process which has:

- $L_0 = 0$
- Independent α-stable increments: $L_t - L_s \sim S_\alpha((t-s)^{1/\alpha}, \beta, 0), t > s$

For $1 \leq \alpha < 2$, this is a pure jump plus gaussian drift process as desired.

3.1.4 Solution for the proposed model

We consider a one-dimensional version of our proposed model which only considers the returns component (x_2).

$$dx_{2,t} = \theta x_{2,t} dt + \sigma dL_t \tag{5}$$

For the scalar case, this can be solved using results based on [4] to give a tractable state transition density:

$$f(x_{2,t}|x_{2,s}) \sim S_\alpha(\sigma_{t-s}, \beta, \exp(\theta(t-s)x_{2,s}), \qquad \sigma_{\delta t} = \left(\sigma \frac{\exp(\alpha\theta\delta t) - 1}{\alpha\theta}\right)^{1/\alpha} \tag{6}$$

3.1.5 Formulation of the Inference Problem in Discrete Time

Using the solution to the one-dimensional version of the proposed model in Equation 6, we can formulate our original continuous-time state space model as a discrete-time state space model where the parameters of the discrete-time state matrices $\boldsymbol{A}(\delta t)$ and $\boldsymbol{b}(\delta t)$ vary according to the continuous time solution and the gap to the previous time step, δt.

To reflect the discretised nature of time used in the inference problem, we change the time indexes using $k = t + \delta t, k - 1 = t$. For brevity, we will also assume in all subsequent equations that $\boldsymbol{A}$ and $\boldsymbol{b}$ are implicitly dependent on δt, and will drop this explicit dependence.

This approach allows us to use established techniques used for discrete time inference.

$$\begin{aligned} \mathbf{x}_k &= \mathbf{A}(\delta t).\mathbf{x_{k-1}} + \mathbf{b}(\delta t).L_k \\ y_k &= \mathbf{C}.\mathbf{x}_k + \mathbf{d}.\eta_k \\ \text{where} \qquad \delta_t &= t_k - t_{k-1} \\ & L_k \sim S_{\alpha/2}(1, 1, 0) \text{ and } \eta_k \sim \mathcal{N}(0, 1) \end{aligned} \tag{7}$$

3.2 Notes on the tail approximations for alpha stable distribution

We take a brief segue to provide some theory into the tail approximation to the α-stable distribution that will be useful in the later sections.

When $\alpha < 2$, the tails of the α-stable distribution are are asymptotically equivalent to a Pareto law. More precisely, if λ is a standardized α-stable variable $\lambda \sim S_\alpha(\sigma, \beta, \mu)$ with $0 < \alpha < 2$, $\sigma = 1$, $\mu = 0$ then as $\lambda \to \infty$ [4]:

$$\lim_{x\to\infty} P(\lambda > x) = C_\alpha(1+\beta)x^{-\alpha} \tag{8}$$

where:

$$C_\alpha = \left(2\int_0^\infty x^{-\alpha}\sin x dx\right)^{-1} = \frac{1}{\pi}\Gamma(\alpha)\sin(\frac{\alpha\pi}{2})$$

We can then differentiate Equation 8 formally to obtain the probability distribution function (PDF) of the tail approximation:

$$\begin{aligned}\lim_{x\to\infty} S_\alpha(\lambda = x|\sigma = 1, \beta, \mu = 0) &= f_{pareto}(\lambda = x|\alpha, \beta) \\ &= \alpha C_\alpha(1+\beta)x^{-\alpha-1}\end{aligned} \tag{9}$$

Numerical accuracy of the tail approximation

We first define a metric for the absolute relative error in the tail approximation:

$$\epsilon_{rel}(x) = \left|\frac{S_\alpha(\lambda = x|1, \beta, 0) - f_{pareto}(\lambda = x|\alpha, \beta)}{S_\alpha(\lambda = x|1, \beta, 0)}\right| \tag{10}$$

We can then investigate the numerical accuracy of the tail approximation for $\beta = 1$ and $\alpha < 1$ using Figure 1. As expected, we note that the relative error in the tail approximation decreases monotically.

Note that for $\alpha < 1$, there is no clear dependence on the value of α and the rate of convergence of the α-stable distribution to the paretian tail approximation.

Sampling from the paretian tail

Using the paretian tail approximation to the α-stable distribution , we note that up to a certain scale parameter($\alpha(1+\beta)C_\alpha$), sampling from a left-truncated α-stable distribution can be approximated by sampling from a left-truncated pareto distribution.

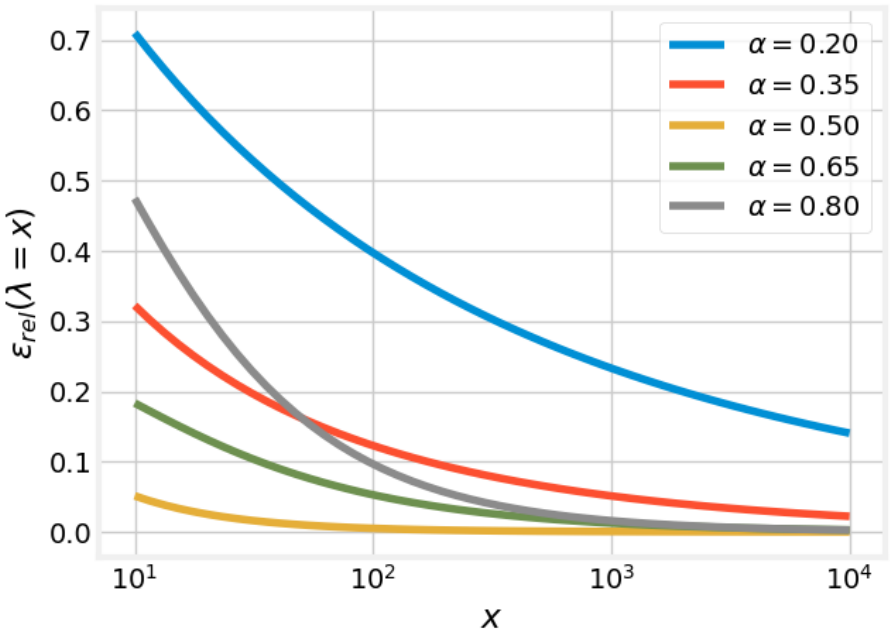


Figure 1: Relative error in the tail approximation $\epsilon_{rel}(x)$ for $\beta = 1$ and $\alpha < 1$

For a left-truncated pareto distribution $f_{pareto}(x|\alpha, \beta)$ lower bounded by $x \geq L$, the corresponding PDF $f_{bounded\ pareto}(x|\alpha, \beta, L)$ is given by Equation 11:

$$f_{bounded\ pareto}(x|\alpha, \beta, L) = \alpha(1+\beta)C_\alpha L^\alpha x^{-\alpha-1} \tag{11}$$

We can then sample from this left-truncated pareto distribution easily using the inverse transform method.

$$x = \left(\frac{1-U}{L^\alpha(1+\beta)C_\alpha}\right)^{-\frac{1}{\alpha}} \qquad \text{where: } U \sim \text{Unif}(0,1) \tag{12}$$

Following Equation 12, x will be left-truncated pareto distributed according to $f_{bounded\ pareto}(x|\alpha, \beta, L)$

3.3 Inference

3.3.1 Generic One-Dimensional Bootstrap Particle Filter

With the continuous-time probability density from Equation 6, we can perform inference using a simple bootstrap particle filter.

The bootstrap particle filter produces for each time step k a set of weighted samples $\left\{w_k^{(i)}, \boldsymbol{x}_k^{(i)}\right\}$ according to:

1. Draw new samples $\boldsymbol{x}_k^{(i)}$ from the importance distribution:

$$\boldsymbol{x}_k^{(i)} \sim \pi(\boldsymbol{x}_k|\boldsymbol{x}_{k-1}^{(i)}, y_{1:k}).i = 1, ..., N \tag{13}$$

   In the special case of the bootstrap particle filter, we choose:

$$\pi(\boldsymbol{x}_k|\boldsymbol{x}_{k-1}^{(i)}, y_{1:k}) = p(\boldsymbol{x}_k|\boldsymbol{x}_{k-1}^{(i)}) \tag{14}$$

2. Calculate new weights according to:

$$\begin{aligned} w_k^{(i)} &\propto w_{k-1}^{(i)} \frac{p(y_k|\boldsymbol{x}_k^{(i)})p(\boldsymbol{x}_k^{(i)}|\boldsymbol{x}_{k-1}^{(i)})}{\pi(\boldsymbol{x}_k|\boldsymbol{x}_{k-1}^{(i)}, y_{1:k})} \\ &= w_{k-1}^{(i)} p(y_k|\boldsymbol{x}_k^{(i)}) \end{aligned} \tag{15}$$

3. Perform resampling to increase the number of particles with effective weights by selecting $\boldsymbol{x}^{(i)}$ with probability $w_k^{(i)}$. This corresponds to approximating the posterior using a point mass distribution given by Equation 16,then selecting a sample from the approximated posterior distribution.

$$p(\boldsymbol{x}_{0:k}|y_{1:k}) \approx \sum_{i=1}^{N} w_k^{(i)} \delta(\boldsymbol{x}_{0:k} - \boldsymbol{x}_{0:k}^{(i)}) \tag{16}$$

The exact state transition density is only tractable for the single dimensional model given in Equation Equation 5. Thus, we first perform inference on the single dimensional returns(x_2) model, and generate the price (x_1) process through numerical integration. The observations provided to the model are the observed price differences $s_k = y_{k+1} - y_k$

This corresponds to the following one dimensional model:

$$\begin{aligned} x_{2,k} &= \underbrace{\exp\left(\theta(t-s)\right)}_{A} x_{2,k-1} + \underbrace{\sigma_{\delta t}}_{b} L_k \\ s_k &= x_{2,k} + \underbrace{\sigma_{obs}}_{d} \eta_k \\ x_{1,k} &= x_{1,k-1} + (\delta t) s_{k-1} \end{aligned} \tag{17}$$

While this simple method manages to remove noise significantly in the returns process, it suffers greatly from drift. Small errors in the inference of the returns process are compounded, leading to the inferred price process deviating from the ground truth (See subsubsection 5.1.1).

3.3.2 Drift-Corrected Bootstrap Particle Filter

To combat the problem of drift, we incorporate a discrete integration approximation to the price process (x_1) into our state space model (Equation 18). This allows us to use numerical integration to obtain an estimate of the current inferred value of the price process at every time step that can then be used to correct for price drift in the system. This drift-corrected state space model (Equation 18) forms the basis of all further explorations.

$$\underbrace{\begin{bmatrix} x_{1,k} \\ x_{2,k} \end{bmatrix}}_{\mathbf{x_k}} = \underbrace{\begin{bmatrix} 1 & \delta t \\ 0 & e^{\theta \delta t} \end{bmatrix}}_{\mathbf{A}} \underbrace{\begin{bmatrix} x_{1,k-1} \\ x_{2,k-1} \end{bmatrix}}_{\mathbf{S_{k-1}}} + \underbrace{\begin{bmatrix} 0 \\ \sigma_{\delta t} \end{bmatrix}}_{\mathbf{b}} L_k, \qquad L_k \sim S_\alpha(1, \beta, 0) \tag{18}$$

3.3.3 Generic Rao-Blackwellised Particle Filter (RBPF)

Scale Mixture Of Normals (SMiN) Representation

In foreign exchange markets such as the EUR/USD, market moves are widely regarded to be symmetric in nature [5]. This motivates the selection of the parameter $\beta = 0$. Instead of considering the general α-stable process, we can instead restrict ourselves to a more analytically tractable sub-class, the a Symmetric α-stable (SαS) process instead.

For $\beta = 0$, we have a convenient Scale Mixture of Normals (SMiN) representation based on the product property of α-stable distributions: If X and Y are independent random variables with $\lambda_k \sim S_{\alpha/2}(1, 1, 0)$ and $\eta_k \sim S_2(1, 0, 0) = \mathcal{N}(0, 1)$, then $\lambda_k \eta_k \sim S_\alpha(1, 0, 0)$.

We can then convert the discretised state space model specified in Equation 18 into SMiN form as specified in Equation 19 below.

$$\underbrace{\begin{bmatrix} x_{1,k} \\ x_{2,k} \end{bmatrix}}_{\mathbf{x_k}} = \underbrace{\begin{bmatrix} 1 & \delta t \\ 0 & e^{\theta \delta t} \end{bmatrix}}_{\mathbf{A}} \underbrace{\begin{bmatrix} x_{1,k-1} \\ x_{2,k-1} \end{bmatrix}}_{\mathbf{x_{k-1}}} + \underbrace{\begin{bmatrix} 0 \\ \sigma_{\delta t} \end{bmatrix}}_{\mathbf{b}} \sqrt{\lambda_k} \eta_k; \qquad \eta_k \sim \mathcal{N}(0, 1),\ \lambda_k \sim S_{\alpha/2}(1, 1, 0) \tag{19}$$

This means that conditional upon us observing λ_k, $\boldsymbol{x}_k$ is gaussian, making inference much more tractable. We exploit this fact by designing a Rao-Blackwellised Particle Filter to sample λ_k by using a simple particle filter, and propogating the states using a Kalman Filter conditioned upon the sampled λ_k of the particle.

We take a short detour to formulate the inference problem using the SMiN representation here. In the inference problem, we are seeking to infer the state variables $\mathbf{x}_k$ given the observations y_k. we assume model dynamics as follows:

$$\begin{aligned} \mathbf{x}_k &= \mathbf{A}\mathbf{x}_{k-1} + \mathbf{b}\sqrt{\lambda_k}\eta_k \\ y_k &= \mathbf{C}\mathbf{x}_k + d\epsilon_k \\ \lambda_k &\sim S_{\alpha/2}(1,1,0) \\ \eta_k, \epsilon_k &\sim \mathcal{N}(0,1) \end{aligned} \tag{20}$$

It will also be useful to use the following expression for y_k:

$$\begin{aligned} y_k &= \mathbf{C}\mathbf{A}\mathbf{x}_{k-1} + \underbrace{\begin{bmatrix}\mathbf{C}\mathbf{b}\sqrt{\lambda_k} & d\end{bmatrix}}_{\mathbf{e}_k} \underbrace{\begin{bmatrix}\eta_k \\ \epsilon_k\end{bmatrix}}_{\mathbf{n_k}} \\ \mathbf{n}_k &\sim \mathcal{N}(\mathbf{0}, \mathbf{I}) \\ \lambda_k &\sim S_{\alpha/2}(1,1,0) \end{aligned} \tag{21}$$

This transforms our state equation form into a (nearly?) α-stable sub-gaussian form. (This is NOT in α-stable sub-gaussian form, see: $<$ α-stable sub-gaussian definition $>$ for reference)

Generic Rao-Blackwellised Particle Filter (RBPF)

For Rao-Blackwellised Particle Filtering, we partition the state vector into gaussian and non-gaussian components. We can then use standard Kalman Filtering to obtain optimal estimates for the gaussian state components, after obtaining estimates for the non-gaussian state components.

At each time step k, the Rao-Blackwellised Particle Filter produces for each time step k a set of weighted samples $\left\{w_k^{(i)}, \lambda_k^{(i)}, \mu_k^{(i)}, \Sigma_k^{(i)} : i = 1, ..., N\right\}$ according to:

1. Draw new latent variables $\lambda_t^{(i)}$ for each particle in $i = 1, ..., N$ from the corresponding importance distribution:

$$\lambda_k^{(i)} \sim \pi(\lambda_k | \lambda_{0:k-1}^{(i)}, y_{1:k}) \tag{22}$$

For the generic RBPF, we choose the importance distribution:

$$\pi(\lambda_k | \lambda_{0:k-1}^{(i)}, y_{1:k}) = p(\lambda_k | \lambda_{0:k-1}^{(i)}) = S_{\alpha/2}(\lambda_k | 1, 1, 0) \tag{23}$$

2. Calculate new weights as follows:

$$w_k^{(i)} \propto w_{k-1}^{(i)} \frac{p(y_k | \lambda_{0:k}^{(i)}, y_{1:k-1}) p(\lambda_k^{(i)} | \lambda_{k-1}^{(i)})}{\pi(\lambda_k^{(i)} | s_{0:k-1}^{(i)}, y_{1:k})} \tag{24}$$

Here, the likelihood term $p(y_k | u_{0:k}^{(i)}, y_{1:k-1})$ is obtained using the predictive error decomposition from the Kalman Filter:

Kalman Filtering Prediction Step:

$$p(\boldsymbol{x}_{0:k} | \lambda_{0:k}^{(i)}, y_{1:k-1}) = \mathcal{N}(\boldsymbol{x}_k | \mu_k^{-(i)}, \Sigma_k^{-(i)}) \tag{25}$$

where:

$$\mu_k^{-(i)} = \boldsymbol{A} \mu_{k-1}^{(i)}$$
$$\Sigma_k^{-(i)} = \boldsymbol{A} \Sigma_{k-1}^{-(i)} \boldsymbol{A}^T + \boldsymbol{b}^T \boldsymbol{b} \lambda_k^{(i)}$$

Predictive Error Decomposition:

$$\begin{aligned} p(y_k | \lambda_{0:k}^{(i)}, y_{1:k-1}) &= \int p(y_k | \lambda_{0:k}^{(i)}, \boldsymbol{x}_{0:k}) p(\boldsymbol{x}_{0:k} | \lambda_{0:k}^{(i)}, y_{1:k-1}) d\boldsymbol{x}_{0:k} \\ &= \mathcal{N}(y_k | \boldsymbol{C} \mu_k^{-(i)}, \boldsymbol{C} \Sigma_k^{-(i)} \boldsymbol{C}^T + d^2) \end{aligned} \tag{26}$$

3. Perform Kalman Filter updates for each of the particles conditional on the drawn latent variables $\lambda_k^{(i)}$.

$$p(\boldsymbol{x}_{0:k}|\lambda_{0:k}^{(i)}, y_{1:k}) = \mathcal{N}(\boldsymbol{x}_k|\mu_k, \Sigma_k) \tag{27}$$

where:

$$\begin{aligned}
\boldsymbol{v}_k^{(i)} &= y_k - \boldsymbol{C}\mu_k^{(i)} \\
\boldsymbol{S}_k^{(i)} &= \boldsymbol{C}\Sigma_k^{-(i)}\boldsymbol{C}^T + d^2 \\
\boldsymbol{K}_k^{(i)} &= \Sigma_k^{-(i)}\boldsymbol{C}^T\boldsymbol{S}_k^{-1}
\end{aligned}$$

$$\begin{aligned}
\mu_k^{(i)} &= \mu_k^{-(i)} + \boldsymbol{K}_k^{(i)}\boldsymbol{v}_k^{(i)} \\
\Sigma_k^{(i)} &= \Sigma_k^{-(i)} - \boldsymbol{K}_k^{(i)}\boldsymbol{S}_k^{(i)}[\boldsymbol{K}_k^{(i)}]^T
\end{aligned}$$

4. Perform multinomial resampling to increase the number of effective particles.

Potential Problems with the RBPF

When $y_k - \boldsymbol{CA}\boldsymbol{\mu}_{k-1}^{(i)}$ is large, the RBPF is often unable to get a good importance sampling estimate for λ_k.

When $y_k - \boldsymbol{CA}\boldsymbol{\mu}_{k-1}^{(i)}$ is large, this implies that λ_k is likely to be large also. (See Figure 3, noting that $p(\lambda_k|y_k - \boldsymbol{CA}\boldsymbol{\mu}_{k-1}^{(i)}) \propto S_{\alpha/2}(\lambda_k)\mathcal{N}(y_k - \boldsymbol{CA}\boldsymbol{\mu}_{k-1}^{(i)}|\lambda_k))$. As a large λ_k lies in the low probability right tail of the particle proposal distribution given by Equation 23, very few particles $\lambda_k^{(i)}$ are generated from the particle proposal distribution which are close to the actual λ_k.

This problem is exacerbated by the fact that the α parameter of the proposal distribution is half of the original α. This causes the tails of the proposal distribution is to decay very slowly, increasing the number of particles needed to give a good importance sampling estimate.

This results in sample impoverishment, whereby there are only a few effective particles with non-negligible weights, which causes the performance of the RBPF to be slightly worse for very low numbers of particles.

One method of quantifying sample impoverishment in a particle filter (whilst adjusting for number of particles) is by measuring the entropy of the particle filter weights given in Equation 28.

$$H(w) = \sum_{i=1}^{N} w_i \log(w_i) \qquad \text{where: } \sum_{i=1}^{N} w_i = 1 \tag{28}$$

In order to compare the entropy of the particle weights across different number of particles, we instead use a normalised entropy measure given in Equation 29.This normalised entropy measures the change in entropy between a set of weights with uniform distribution (an ideal "optimized" set of weights) and a set of weights with a non-uniform distribution, and is scaled to be independent of the number of particles, as well as the base used in calculating the entropy.

$$H_n(w) = \frac{1}{\log_b(N)} \sum_{i=1}^{N} w_i \log_b(w_i) \qquad \text{where: } \sum_{i=1}^{N} w_i = 1 \tag{29}$$

We demonstrate the problem of sample impoverishment by simulating a single time step of the particle filter, for varying values of $y_k - \boldsymbol{CA\mu}_{k-1}^{(i)}$. Fixing $\mu_{k-1}^{(i)}$, $\Sigma_{k-1}^{(i)}$ whilst varying y_k and N, we simulate one time step of the RBPF update step described above and present the normalised entropy of the particle filter weights obtained in Figure 2. We see that the normalised entropy of the particle filter weights drops rapidly for large $y_k - \boldsymbol{CA\mu}_{k-1}^{(i)}$, and that the as N increases, the normalised entropy of the particle filter weights is more resistant to sample impoverishment at large values of $y_k - \boldsymbol{CA\mu}_{k-1}^{(i)}$ as expected.

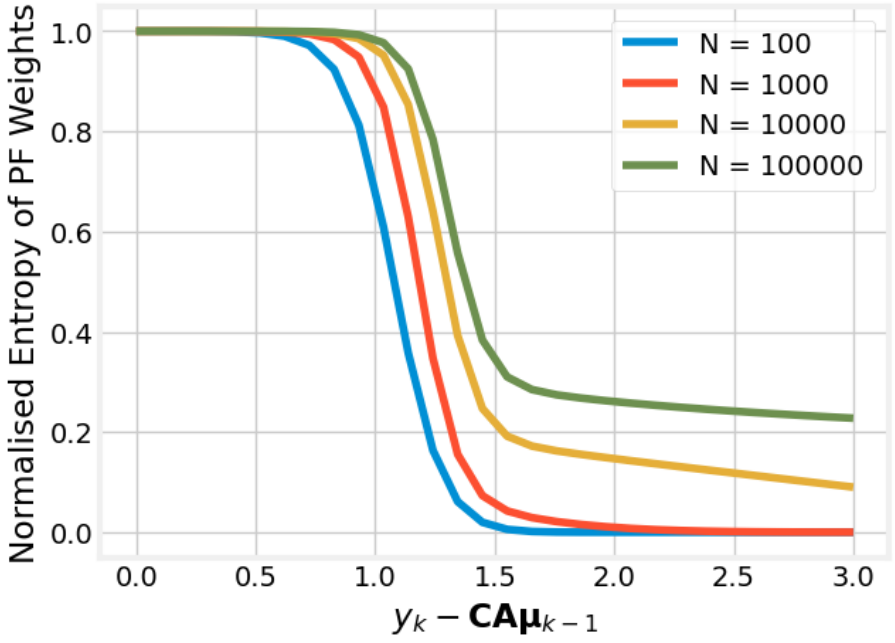


Figure 2: Change in normalised entropy of PF weights as $y_k - \boldsymbol{CA\mu}_{k-1}^{(i)}$ and N are varied

3.3.4 Dense Sampling Rao-Blackwellised Particle Filter (RBPF)

We note that the cause of the problem of sample impoverishment is due to the low number of particles being drawn from the tails of the proposal distribution, leading to low numbers of particles with effective weights when $y_k - \boldsymbol{CA\mu}_{k-1}^{(i)}$ is large

Thus, one method of reducing sample impoverishment in the particle filter would be to simply generate a larger number of particles from the tails of the proposal distribution. In order to preserve the required proposal distribution, we need to then scale the weights of the extra particles generated from the tails appropriately.

This leads to the following algorithm for sampling from the proposal distribution given in Equation 23:

- Define the hyperparameters ϵ_λ (controls the accuracy of the tail approximation) and m (a multiplier which controls the number of extra particles to draw from the tails of the proposal distribution). In this example, we select $\epsilon_\lambda = 1 \times 10^{-1}$ and $m = 3$.
- Calculate the number of extra particles (N_{extra}) to draw from the right tail using:

$$\begin{aligned} \epsilon_\lambda &> \epsilon_{rel}(\lambda_k = threshold) \\ \epsilon_\lambda &> \frac{f_{pareto}(\lambda_k = threshold|\alpha/2, 1) - S_{\alpha/2}(\lambda_k = threshold|1, 1, 0)}{S_{\alpha/2}(\lambda_k = threshold|1, 1, 0)} \\ N_{extra} &= P(\lambda > threshold) \times Nm \end{aligned} \tag{30}$$

- Draw N_{extra} particles from the right tail:

$$\lambda_k^{(i)} \sim f_{bounded\ pareto}(\lambda_k|\alpha = \alpha/2, \beta = 1, L = threshold) \qquad i = 1...N_{extra} \tag{31}$$

- Draw $N - N_{extra}$ particles from the truncated alpha stable distribution using a simple rejection sampler:
  - Draw $\lambda_k^{(i)} \sim S_{\alpha/2}(\lambda_k|1, 1, 0)$
  - Accept if $\lambda_k^{(i)} < threshold$
- Alter the weights of the particles drawn from the tails:

$$w_i = \frac{1}{m} w_i \qquad \text{for: } i \in [1, N_{extra}]$$

3.3.5 Adaptive Sampling Rao-Blackwellised Particle Filter (RBPF)

One way of attempting to improve the performance of the RBPF is to form a better importance sampling distribution. In this section, we seek to form a better importance sampling distribution for $\pi(\lambda_k|y_{1:t}, \lambda_{0:k-1})$.

For optimality, we want $\pi(\lambda_k|y_{1:t}\lambda_{0:k-1}^{(i)}) \approx p(\lambda_k|y_{1:t}, \lambda_{0:k-1}^{(i)})$.

It is hard to formulate an analytical form for this density directly. To get around this, we can instead form the analytical joint distribution for $p(y_k, \lambda_k|y_{1:k-1}, \lambda_{0:k-1}^{(i)})$.

We first augment the probability density given above with $x_{0:k-1}$. This allows us to use results from the previously-performed Kalman Filter step, $\mu_{k-1}^{(i)}$ and $\Sigma_{k-1}^{(i)}$.

$$
\begin{aligned}
& p(y_k, \lambda_k, x_{0:k-1}|y_{1:k-1}, \lambda_{0:k-1}^{(i)}) \\
\propto\ & p(y_k|y_{1:k-1}, \lambda_{0:k}^{(i)}, x_{0:k-1}) p(x_{0:k-1}|y_{1:k-1}, \lambda_{0:k}^{(i)}) p(\lambda_k^{(i)}|\lambda_{0:k-1}^{(i)}) \\
\approx\ & p(y_k|y_{1:k-1}, \lambda_{0:k}^{(i)}, x_{0:k-1}) p(x_{0:k-1}|y_{1:k-1}, \lambda_{0:k-1}^{(i)}) p(\lambda_k|\lambda_{0:k-1}) \\
=\ & \mathcal{N}(y_k|\mathbf{CA}\mathbf{x}_{k-1}, \mathbf{e}\mathbf{e}^T) \mathcal{N}(\mathbf{x}_{0:k-1}|\boldsymbol{\mu}_{k-1}^{(i)}, \boldsymbol{\Sigma}_{k-1}^{(i)}) S_{\alpha/2}(1,1,0) \\
=\ & \mathcal{N}\left(\begin{bmatrix} y_k \\ \boldsymbol{x}_{k-1} \end{bmatrix} | \begin{bmatrix} \boldsymbol{CA}\boldsymbol{\mu}_{k-1}^{(i)} \\ \boldsymbol{\mu}_{k-1}^{(i)} \end{bmatrix}, \begin{bmatrix} \boldsymbol{e}\boldsymbol{e}^T + (\boldsymbol{CA})\boldsymbol{\Sigma}_{k-1}^{(i)}(\boldsymbol{CA})^T & \boldsymbol{CA}\boldsymbol{\Sigma}_{k-1}^{(i)} \\ \boldsymbol{\Sigma}_{k-1}^{(i)}(\mathbf{CA})^T & \boldsymbol{\Sigma}_{k-1}^{(i)} \end{bmatrix} \right) S_{\alpha/2}(1,1,0)
\end{aligned}
$$

We can then obtain the required density $p(y_k, \lambda_k^{(i)}|y_{1:k-1}, \lambda_{0:k-1})$ by marginalising out $x_{0:k-1}$:

$$
\begin{aligned}
p(y_k, \lambda_k|y_{1:k-1}, \lambda_{0:k-1}^{(i)}) &= \int p(y_k, \lambda_k, x_{0:k-1}|y_{1:k-1}, \lambda_{0:k-1}^{(i)}) dx_{0:k-1} \\
&= \mathcal{N}(y_k|\mathbf{CA}\mu_{k-1}^{(i)}, (\mathbf{Cb})(\mathbf{Cb})^T\lambda_k + d^2 + (\mathbf{CA})\boldsymbol{\Sigma}_{k-1}^{(i)}(\mathbf{CA})^T) S_{\alpha/2}(\lambda_k|1,1,0) \\
&= \mathcal{N}(y_k|\mathbf{CA}\mu_{k-1}^{(i)}, \sigma_\lambda\lambda_k + \mu_\lambda) S_{\alpha/2}(\lambda_k|1,1,0) \qquad (32)
\end{aligned}
$$

where:

$$
\begin{aligned}
\sigma_\lambda &= (\mathbf{Cb})(\mathbf{Cb})^T \\
\mu_\lambda &= d^2 + (\mathbf{CA})\boldsymbol{\Sigma}_k^{(i)}(\mathbf{CA})^T
\end{aligned}
$$

By fixing the value of y_k in Equation 32, we can obtain the desired distribution $p(\lambda_k|y_{1:k}, \lambda_{0:k-1})$.

This formulation of the joint distribution also gives us an alternative interpretation for the sampling density $p(\lambda_k|y_{1:k}, \lambda_{0:k-1})$.

We can reinterpret this as finding the posterior density of λ_k given the observations $y_k - \boldsymbol{CA\mu}_{k-1}$. The prior on λ_k is the α-stable proposal distribution $S_{\alpha/2}(\lambda_k|1,1,0)$ while the likelihood is the unknown variance of a normal distribution $(\mathcal{N}(y_k|\boldsymbol{CA\mu}_{k-1}, \sigma_\lambda\lambda_k + \mu_\lambda))$.

Rejection Sampling

In general, there is no closed form expression for this distribution. However, we can utilise the fact that we can draw samples from the prior distribution easily using the Chamber-Mallow-Stuck method [6]. This can be used as a suitable proposal function for rejection sampling.

We adapt the methods of [7] to draw samples from this distribution.

The target distribution for the rejection sampling scheme is given by:

$$\begin{aligned} f(y_k, \lambda_k|y_{1:k-1}, \lambda^{(i)}_{0:k-1}) &= \mathcal{N}(y_k|\boldsymbol{CA\mu}^{(i)}_{k-1}, \sigma_\lambda\lambda_k + \mu_\lambda) S_{\alpha/2}(\lambda_k|1,1,0) \\ &= \mathcal{N}(y_k - \boldsymbol{CA\mu}^{(i)}_{k-1}|, \sigma_\lambda\lambda_k + \mu_\lambda) S_{\alpha/2}(\lambda_k|1,1,0) \end{aligned}$$

Using a proposal distribution $g(\lambda_k|y_{1:k}, \lambda_{0:k-1}) = S_{\alpha/2}(1,1,0)$, we can use the likelihood as a valid rejection function as it is bounded from the above:

$$\begin{aligned} \mathcal{N}(y_k - \boldsymbol{CA\mu}^{(i)}_{k-1}|0, \sigma_\lambda\lambda_k + \mu_\lambda) &\leq \frac{f(y_k, \lambda_k|y_{1:k-1}, \lambda_{0:k-1})}{g(y_k, \lambda_k|y_{1:k-1}, \lambda_{0:k-1})} \\ &= M \end{aligned}$$

$$M = \begin{cases} \frac{1}{\sqrt{2\pi(y_k - \boldsymbol{CA\mu}^{(i)}_{k-1})^2}} \exp(-0.5) & (y_k - \boldsymbol{CA\mu}^{(i)}_{k-1})^2 \geq \mu_\lambda \\ \frac{1}{\sqrt{2\pi d^2}} \exp\left(-\frac{(y_k - \boldsymbol{CA\mu}^{(i)}_{k-1})^2}{2d^2}\right) & (y_k - \boldsymbol{CA\mu}^{(i)}_{k-1})^2 \leq \mu_\lambda \end{cases} \tag{33}$$

This gives a suitable rejection sampler as:

1. Draw $\lambda_k \sim S_{\alpha_2}(1,1,0)$
2. Draw $u \sim U(0, M)$

3. If $u \geq \mathcal{N}\left(y_k - \boldsymbol{CA}\boldsymbol{\mu}_{k-1}^{(i)} | 0, \sigma_\lambda \lambda_k + \mu_\lambda\right)$, reject λ_k and go to Step (1)

Improvements on the rejection sampler:

The rejection sampler suffers from poor acceptance rates when $y_k - \boldsymbol{CA}\boldsymbol{\mu}_{k-1}^{(i)}$ is very large. Most of the samples generated by the prior fall into the left tail of the likelihood, and have extremely low probabilities of being accepted (Figure 3).

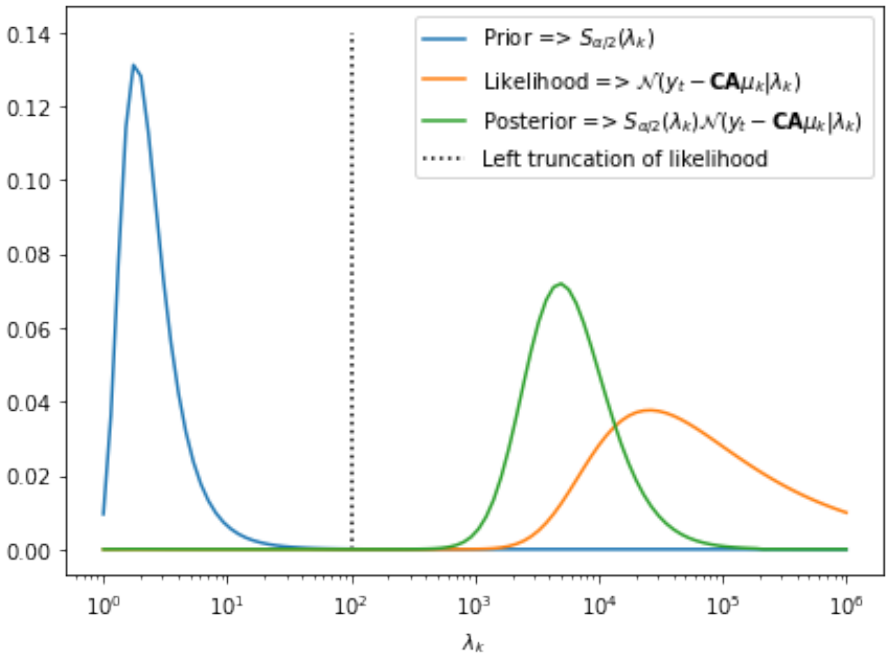


Figure 3: Illustration of the poor acceptance rate regime of prior/likelihood interaction

Due to the additional observation noise term (d), we are unable to use the Inverse Gamma approximation to the full posterior as detailed in [7]. Instead, we attempt to improve the sampling efficiency by truncating the left tail of the likelihood distribution.

Due to the intractability of calculating the Cumulative Distribution Function (CDF) of the likelihood analytically, we left-truncate the likelihood where it is smaller than a chosen epsilon. In this project, this small epsilon was chosen to be $\epsilon_{trunc} = 1 \times 10^{-50}$. Further checks are also performed to ensure that this truncation only occurs well into the right tail of the prior.

This allows us to generate samples from a left-truncated form of the α-stable prior instead of the full prior. These generated samples are much more likely to be in the high probability regions of the likelihood, thus improving our sampling efficiency greatly.

We sample from the left truncated α-stable distribution by applying a paretian tail approximation to the α-stable distribution , then sampling from the paretian tail approximation using Equation 12.

This improved rejection sampler can be described as:

1. Define hyperparameter ϵ_{trunc}
2. Calculate 95th percentile of the prior ($\lambda_{0.95}$) where: $\int_0^{\lambda_{0.95}} S_{\alpha/2}(\lambda_k|1,1,0) = 0.95$
3. Draw proposals $\lambda_k^{(i)}$
   - Calculate a possible truncation value $\lambda_{k,trunc}$ where $\mathcal{N}(y_k - \boldsymbol{CA}\boldsymbol{\mu}_{k-1}^{(i)}|\lambda_{k,trunc}) = \epsilon_{trunc}$.
   - If this truncation value also lies in the right tail of the prior (i.e. $\lambda_{k,trunc} > \lambda_{0.95}$) then sample particle from the paretian tail approximation

$$\lambda_k^{(i)} \sim f_{bounded\ pareto}(x|\alpha/2, \beta = 1, \lambda_{k,trunc})$$

   - Otherwise, sample particle from the original prior:

$$\lambda_k^{(i)} \sim S_{\alpha/2}(1,1,0)$$

4. Draw $u \sim U(0, M)$
5. If $u \geq \mathcal{N}\left(y_k - \boldsymbol{CA}\boldsymbol{\mu}_{k-1}^{(i)}|0, \sigma_\lambda\lambda_k + \mu_\lambda\right)$, reject $\lambda_k^{(i)}$ and go to Step (3)

3.4 Parameter Estimation

In this section, we describe how to apply Bayesian parameter estimation for the Rao-Blackwellized Particle Filters developed in subsubsection 3.3.3 to subsubsection 3.3.5 using a Metropolis-within-Gibbs sampling scheme.

We build upon the discrete-time conditionally gaussian state space models given by Equation 19, with the extension that the matrices governing the model dynamics $\boldsymbol{A}(\boldsymbol{\theta})$, $\boldsymbol{b}(\boldsymbol{\theta})$, $\boldsymbol{C}(\boldsymbol{\theta})$ and $\boldsymbol{d}(\boldsymbol{\theta})$ all depend on some parameter vector $\boldsymbol{\theta}$ of fixed dimension.

In the context of our project, we define the parameter estimation problem as estimating the parameter vector $\boldsymbol{\theta} = \{\theta, \sigma, \sigma_{obs}\}$, given observations up to time T (i.e. $y_{1:T}$)

$$p(\boldsymbol{\theta}, \lambda_{0:k}|y_{1:k}) \tag{34}$$

We construct a Markov Chain to target the joint distribution of the sample parameters $\boldsymbol{\theta}$ and RBPF latent states $\lambda_{1:T}^{(i)}$ given by **??** using Gibbs Sampling by iteratively performing the following steps:

1. We draw $\boldsymbol{\theta}$ iteratively using a Metropolis Hastings step to target the required marginal distribution for θ_j given by Equation 35.

$$\theta_j \sim p(\theta_j|\theta_{-j}, \lambda_{0:T}, y_{1:T}) \propto p(y_{1:T}|\boldsymbol{\theta}, \lambda_{0:T}^*)p(\lambda_{0:T}^*|\boldsymbol{\theta})p(\theta_j|\theta_{-j}) \tag{35}$$

   Here, we denote $\theta_{-j} = \boldsymbol{\theta} \setminus \theta_j$ and using $\lambda_{0:T}^*$ to make explicit the notion that a single realisation of $\lambda_{0:T}$ is being used.

   We can sample from this probability distribution using a Metropolis-Hastings step, with proposal density $q(\theta_j^*|\theta_i')$, where θ_j' is the current sample of parameter θ_j and θ_j^* is the current proposal for θ_j.

   For the Metropolis-Hastings step, we accept the current proposal θ_j^* with acceptance probability $\min(1, \alpha)$, with α given by:

$$\alpha = \frac{p(y_{1:T}|\theta_{-j}, \theta_j^*, \lambda_{0:T}^*)p(\lambda_{0:T}^*|\theta_{-j}, \theta_j^*)p(\theta_j^*|\theta_{-j})q(\theta_j'|\theta_j^*)}{p(y_{1:T}|\theta_{-j}, \theta_j', \lambda_{0:T}^*)p(\lambda_{0:T}^*|\theta_{-j}, \theta_j')p(\theta_j'|\theta_{-j})q(\theta_j^*|\theta_j')} \tag{36}$$

2. With the accepted parameter vector $\boldsymbol{\theta_{accepted}}$ from step 1, we can then look to sample another realisation of $\lambda_{0:T}^*$ by first drawing a set of weighted samples $\left\{w_T^{(i)}, \lambda_{0:T}^{(i)}\right\}$ from the RBPF, denoted by denoted by Equation 37.

$$\left\{w_T^{(i)}, \lambda_{0:T}^{(i)}\right\} \sim p(\lambda_{0:T}|y_{1:T}, \boldsymbol{\theta_{accepted}}) \tag{37}$$

   Using the weighted samples $\left\{w_T^{(i)}, \lambda_{0:T}^{(i)}\right\}$ as an importance-sampling estimate of $p(\lambda_{0:T}|y_{1:T}, \boldsymbol{\theta_{accepted}})$, we then draw a particular realisation $\lambda_{0:k}^*$ using multinomial sampling by selecting $\lambda_{0:T}^{(i)}$ with probability $w_T^{(i)}$

For each time step, we thus obtain joint samples $\{\boldsymbol{\theta}, \lambda_{0:T}^*\}$, which can be marginalised to obtain samples from the posterior distribution of parameters.

4 Experimental Design

4.1 Simulated Data

In order to objectively evaluate the performance of our inference algorithms, we simulate the price and returns of a tradable commodity using the 2 dimensional discrete time state space model described in Equation 18. This provides us with the ground truth state that we can use to compare with the inferred state estimates from our particle filter.

This also allows us to check that the model has the appropriate characteristics that we are looking for.

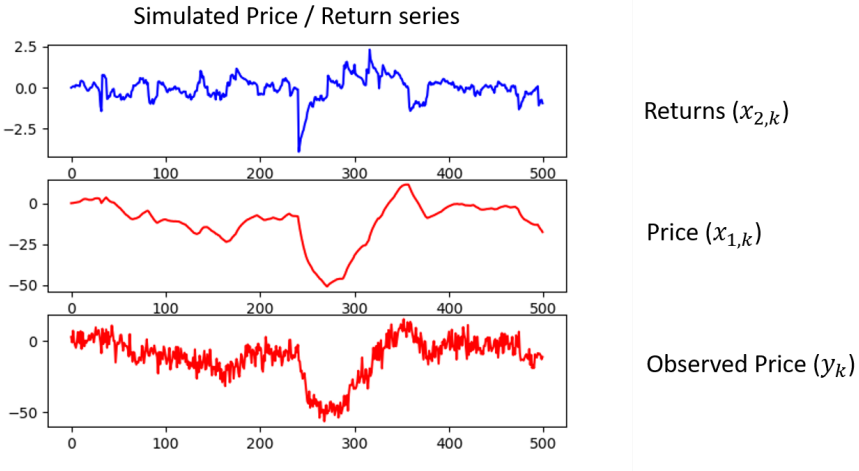


Figure 4: Simulation Results for 500 iterations

Figure 4 shows a simulated price(x_1) / returns(x_2) series, together with the observed price series corrupted by gaussian noise(y).

We note that the returns series has the desired jump activity, together with a gaussian drift back to zero mean. The gaussian drift back to zero mean allows for a predictable trend to be captured in the price process, while the jump activity allows for sharp changes in the trend.

This shows that the state-space model described in Equation 18 is able to capture both a predictable trend in asset prices, as well as sharp changes in these trends, both extremely useful for a momentum-based strategy.

4.2 EUR / USD Exchange Rate Data

In this project, we will focus mainly on applying the inference techniques developed in this report to the foreign exchange (*forex*) market. Forex is considered to be world's largest

and most liquid financial markets, and trades 24 hours a day, five days a week. It is also famous for being fast-paced and volatile, which dovetails well with high frequency trading as it provides plenty of trading opportunities and the volatility required for making significant profits.

The main dataset used for testing has been obtained through the Swiss online foreign exchange bank, Dukascopy [8]. We focus mainly on the highly liquid EUR/USD currency pair, which has a high trading frequency.

We use 2 different types of foreign exchange data for testing: tick level data and fixed snapshot data.

Tick level data is a high frequency stream-like list of transactions, with new events arriving over 100 times per second. For each transaction, we receive the transaction price together with the next closest buy (bid) and sell (ask) prices to the transaction price. An example of this data is provided in Figure **??**

On the other hand, the snapshot data we obtain provides the current closest bid and ask prices at regular intervals.

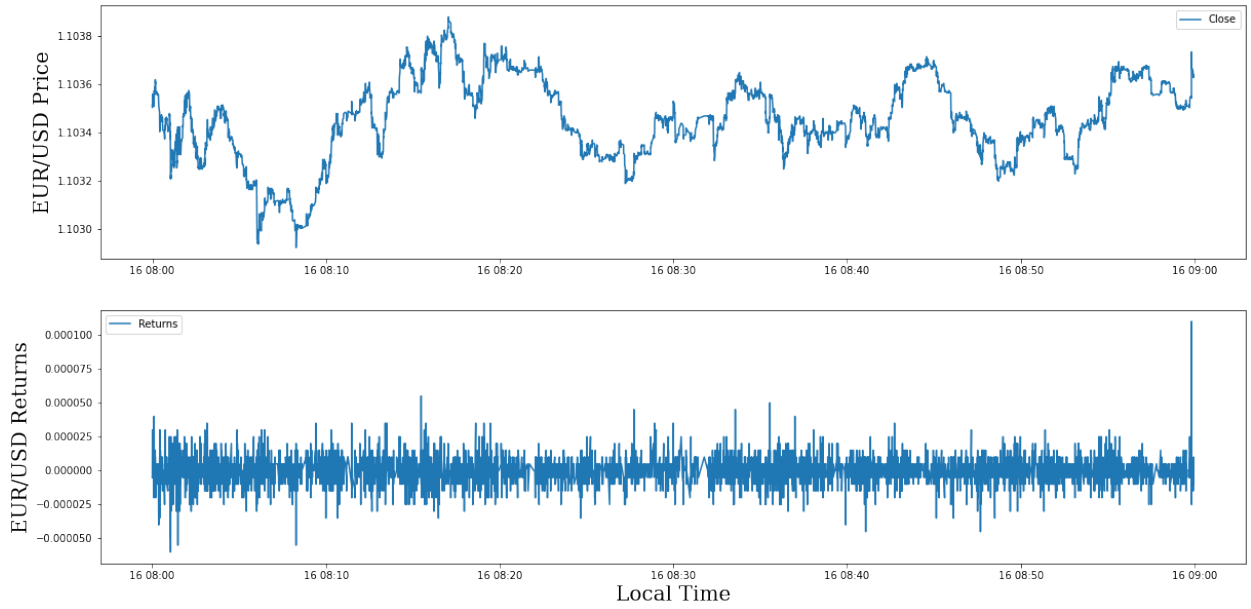
As we are most interested in high-frequency analysis of the EUR/USD data, we focus most on the tick level data. Snapshot data is used to allow us to investigate the impact of observation frequency on our trading performance.

Next, we will examine in greater detail some key characteristics of the EUR/USD dataset that we are using to highlight any potential problems that might occur as a result of working on real exchange rates rather than simulated data.

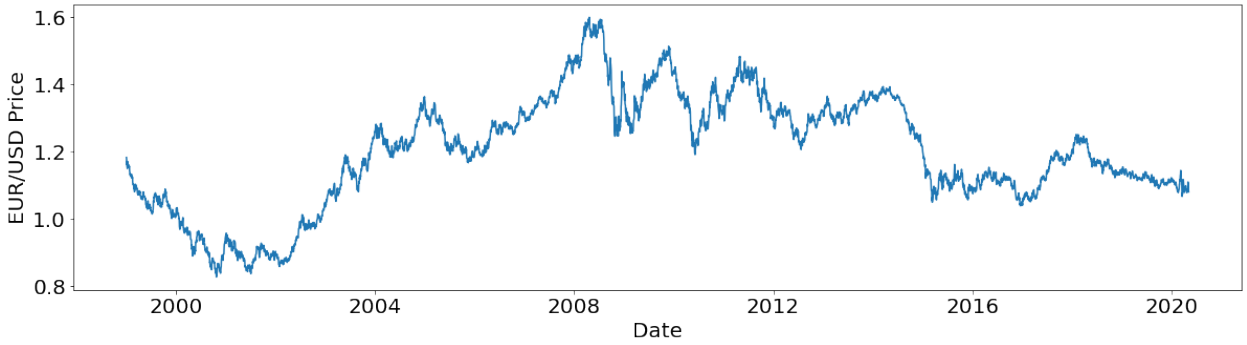
4.2.1 Quantisation

The high frequency tick data that we obtain is often *quantized*, and consists of only a few price levels. This is typical, as exchanges do not allow arbitrary prices to be transacted. In this case, the minimum price change possible is $5 \times 10^{-}6$ EUR/USD.

This is a violation of one of the core assumptions of the stochastic modelling: that the price levels are continuous. As the minimum return ($5 \times 10^{-}6$) is small compared to the standard deviation of the returns, we ignore this problem in the project. However, we note that this could be a problem in extremely high frequency trading, where the price does not fluctuate much, hence causing the minimum return to be significant compared to the standard deviation of returns.



(a) Example of EUR / USD Tick Level Data, 1600 - 1700 on 10/02/2019



(b) Example of EUR / USD Daily Close Data, 1999 - 2020

4.2.2 Intra-day volatility

Unlike the stock market, the forex market is decentralised and open for trading 24 hours a day, 5 days a week. This means that there can be significant intra-day variability in the volatility of the forex data that we receive.

Figure 5a shows a striking example of this intra-day volatility over a 3 day period. It is readily apparent that the volatility of this particular forex exchange is high closer to European trading hours (0600 - 1800), and more muted once that time period is over. Other than this, alternative intra-day patterns have also been empirically shown to exist in many financial markets [9], [10]

However, in our models we assume a constant observation noise σ_{obs}, which is directly contra-

dicted by the presence of intra-day volatility. One way of accounting for intra-day volatility is to employ a stochastic volatility term in the state space models used to model the dynamics of the market.

Instead of resorting to that complexity, we instead employ a simpler method. We estimate the volatility of the EUR/USD exchange rate using a GARCH(1,1) model [11], then dividing the observed returns by its estimated volatility. The intuition behind this idea is that this helps to normalise the volatility of the returns across all time periods, hence reducing the intra-day volatility observed.

Figure 5b shows the resultant price and volatility series obtained by applying this method, qualitatively demonstrating significant intra-day volatility reduction.

4.3 Performance Metrics

4.3.1 Statistical Metrics

Mean Square Error

The mean square error (MSE) is used to compare forecasts with the real future observed state. This allows us to assess the predictive power of the inference algorithms that we are testing.

The mean square error is defined over all time points K as:

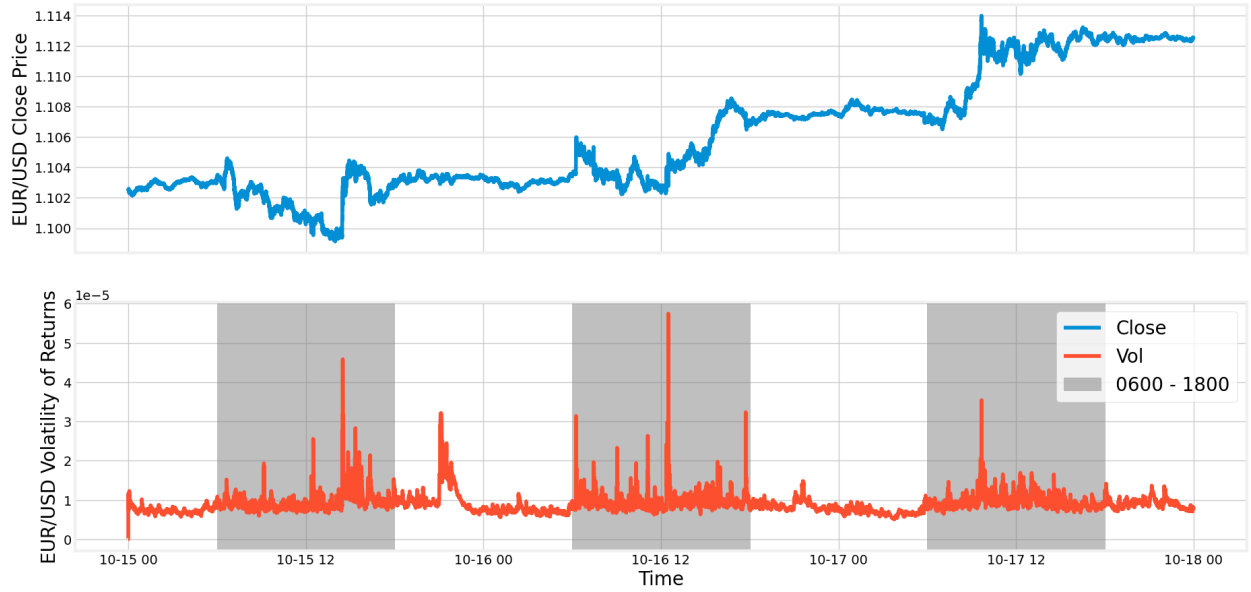
$$MSE = \mathbb{E}(e^2) = \frac{1}{K}\sum_{i=1}^{K}(y_i - \hat{y}_i)^2 \tag{38}$$

4.3.2 Actual Trading Performance Metrics

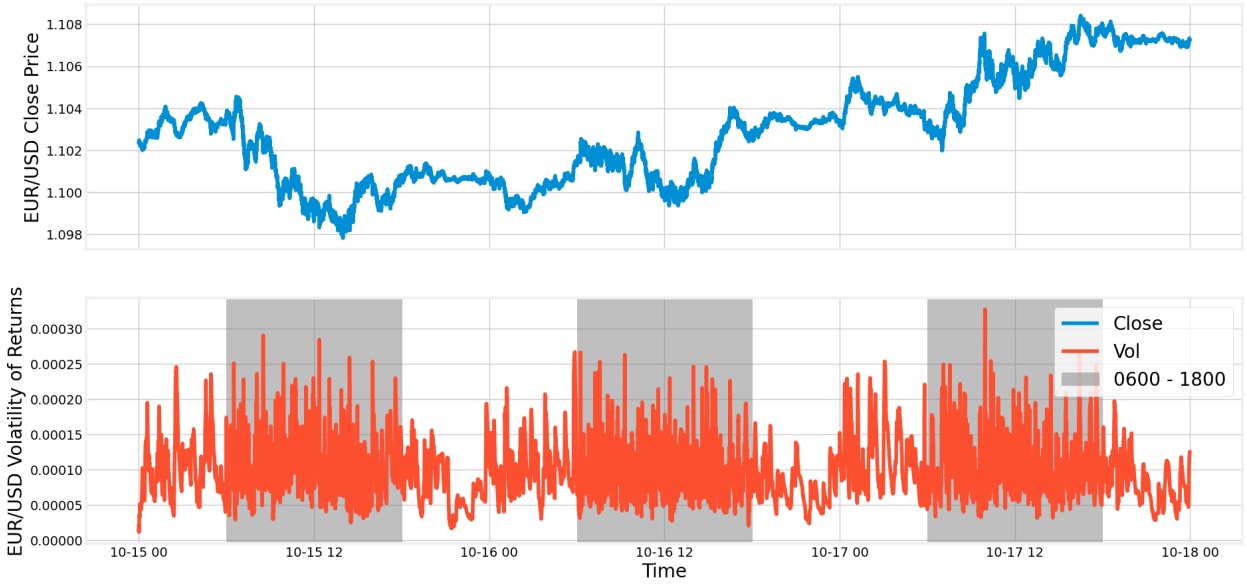
As we are interested in optimising the inference algorithms described in this paper for practical use, we want to select a performance metric that more closely reflects the actual results achieved if the filtering model is used as the backend for a real trading system.

Binary Prediction Error

One rudimentary trading algorithm that could be employed to transform the filtered state into actual trading signals simply invests a set amount of money if the predicted return is positive, and shorts the same amount if the predicted return is negative.



(a) Example of Intra-day Volatility for EUR/USD forex data



(b) EUR/USD forex data after intra-day volatility correction

This simple trading algorithm will generate profits if the sign of the predicted returns matches that the sign of the actual returns.

Thus, one metric that we could use to measure the performance of the trading system looks to see how many times the sign of the predicted returns matches the sign of the actual returns (Binary Prediction Error).

This provides a more realistic measure of the profitability of this particular trading system.

A binary prediction error $BPE < 0.5$ indicates that the filtered signal is predicting correct trade decisions more often than not.

We define the binary prediction error formally as:

$$BPE = \frac{1}{N}\sum_{i=1}^{N} b_i$$
$$where: \quad b_i = \begin{cases} 0 & \text{sign}(y_i) = \text{sign}(\hat{y}_i) \\ 1 & \text{otherwise} \end{cases} \tag{39}$$

Sharpe Ratio

Instead of the simple trading algorithm detailed above, we might instead want to use more sophisticated ways of converting the filtered signal into an actual trading signal.

We can quantify the trading profitability of this actual trading signal by running the trading algorithm and obtaining the actual backtested realised returns. Using these realised profits, we can use the *sharpe ratio* of the actual realised profits to objectively evaluate the performance of the algorithm.

The sharpe ratio measures the average return earned per unit of risk(volatility) undertaken, and is defined by:

$$\text{Sharpe Ratio} = \frac{\text{Mean of Portfolio Returns}}{\text{Standard Deviation of Portfolio Returns}} \tag{40}$$

Because the sharpe ratio is defined in units of risk, it can help to explain if the returns obtained are due to a profitable trading signal or simply as a result of taking on too much risk. The greater the sharpe ratio, the better the trading strategy's risk-adjusted performance, hence indicating that the trading signal is more profitable.

4.4 Hyperparameter selection

4.5 Trading Algorithm

5 Experimental Results and Discussion

5.1 Comparison of inference algorithms

We have implemented a total of 5 methods for conducting inference as described in subsection 3.3.These models can be conceptually divided into 2 classes: bootstrap particle filtering

based models (1,2) and rao-blackwellized particle filtering based models (3,4,5). In this section, we will compare and discuss the results obtained by the 5 inference algorithms.

1. A generic one-dimensional bootstrap particle filter (PF) which inferred only the returns (x_2) state.
2. A drift-corrected bootstrap particle filter which infers both price (x_1) and returns (x_2) states simultaneously.
3. A Rao-Blackwellized Particle Filter (RBPF) for the case of a symmetrical α-stable distribution ($\beta = 0$)
4. Dense Sampling improvement for the RBPF to reduce sample impoverishment
5. Adaptive Sampling improvement for the RBPF to create a fully adapted RBPF

Simulated Data Generation

Unless explicitly stated, simulated data (with the simulation process described in subsection 4.1 has been used to obtain the results in this section. This provides us with both ground truth data for the states, as well as the actual parameters used, allowing us to test the effectiveness of the inference algorithms without the complications of parameter inference. The hyper-parameters used for generation of this simulated data is shown in Table 1, and an example of the simulated data generated using these hyper-parameters is shown in Figure 5.

Parameter	Description	Value
α	α-stable distribution tail exponent	1.2
β	α-stable distribution skewness parameter	0
σ	α-stable distribution volatility	2×10^{-4}
σ_{obs}	observation noise	2×10^{-1}
θ	mean reversion coefficient	-0.05
δt	time gap between observations	1
T	Number of time steps	1500

Table 1: Hyper-parameters used for simulated data

Overall Comparison

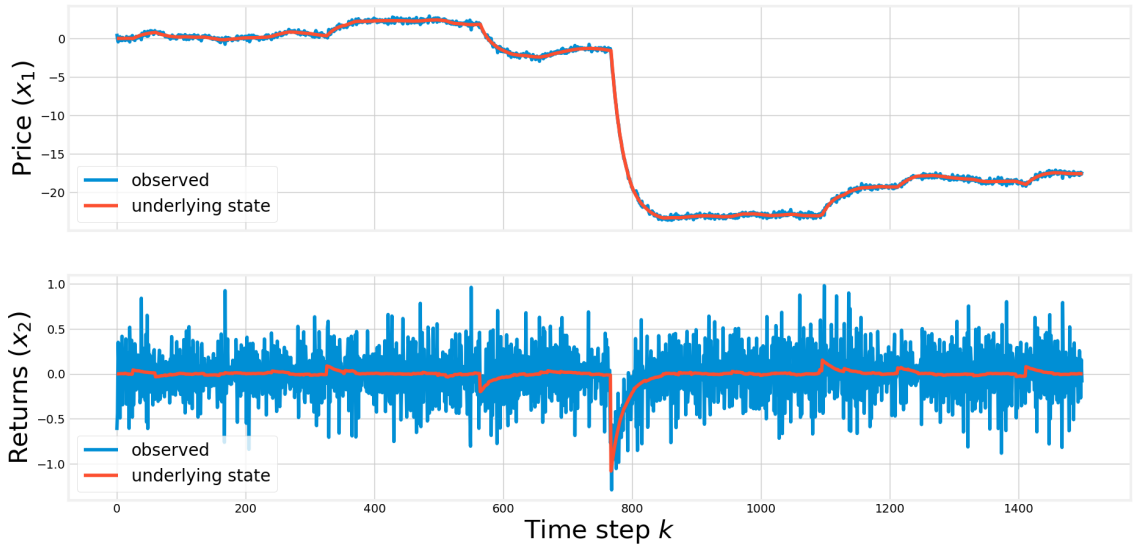
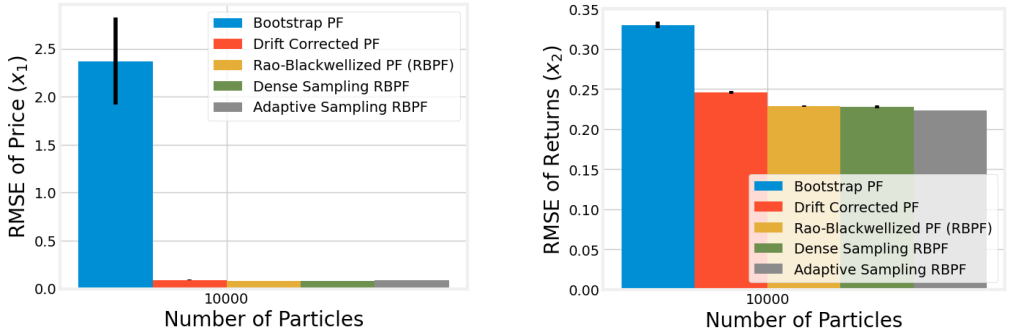


Figure 5: Example of simulated data

As a brief comparison of all the 5 inference algorithms, Figure 6 shows the performance of all 5 inference algorithms using a large number of particles ($N = 10000$). We can observe that the RBPF based particle filtering based algorithms perform better in general than the bootstrap particle filtering based algorithms in terms of the Root Mean Square Error (RMSE) of both the underlying price and returns state. This continues to hold true even for lower number of particles. We will now compare the performance of the 2 classes of inference algorithms separately to allow for a more detailed comparison of the inference performance of each algorithm.



(a) Root Mean Square Error (RMSE) of the price(x_1) process

(b) Root Mean Square Error (RMSE) of the returns(x_2) process

Figure 6: Comparison of all inference algorithms, $N = 10000$ particles

Comparison of Bootstrap Particle Filters

Firstly, we note that the generic one dimensional bootstrap particle filter performs very badly in tracking the underlying price state, as can be seen from Figure 7a. With a larger number of particles, the drift-corrected bootstrap particle filter is significantly better at tracking the underlying price state.

However, both algorithms perform almost just as well at inferring the underlying returns state, although the RMSE of the drift-corrected particle filter when tracking the underlying returns state exhibits large variability for low number of particles. This is likely due to the drift-corrected particle filter being very sensitive to sample impoverishment in the same manner as described in **??**.

For low N and large $y_k - \boldsymbol{CA}\boldsymbol{\mu}_{k-1}^{(i)}$, the samples of $\boldsymbol{x}_k$ obtained from the particle filter proposal distribution are likely to have very low acceptance weights, resulting in a lack of particles with effective weights. If there are no particles with effective weights generated by the particle filter, the tracking performance of the particle filter can be severely diminished. The larger the observed price outlier ($y_k - \boldsymbol{CA}\boldsymbol{\mu}_{k-1}^{(i)}$) and the lower the number of particles N, the greater the impact of this problem and the less likely that particles with effective weights will be generated.

Due to sample impoverishment, we expect the performance of the bootstrap particle filters to fall rapidly as the number of particles drops. Furthermore, we also expect the RMSE achieved by the bootstrap particle filters to be more variable at low numbers of particles. This effect can be seen in Figure 7a and 7c.

Lastly, we note that the Binary Prediction Error (BPE) of the returns process is much less affected by low numbers of particles as compared to the RMSE. Optimising for BPE only requires the particle filter to predict the correct *direction* of returns, and is hence a less challenging task compared to accurately forecasting the underlying return state (measured by RMSE). This is encouraging, as it implies that the actual trading performance of the particle filters is likely to be less affected by random initialisations of the particles filter, and hence is likely to be more robust to outliers in the data.

For $N = 1000$ and $N = 10000$, we also note that the BPE (0.25) is much lower than a BPE based solely on change (0.5), indicating that the bootstrap particle filters are frequently able to correctly predict the direction of the underlying return state. This is encouraging, and suggests that trading algorithms based on these filters will be profitable.

Comparison of Rao-Blackwellized Particle Filters

Figures 7b, 7d and 7f show how the performance of the 2 bootstrap particles filters varies as the number of particles, N is changed.

Firstly, we note that the performance of the 3 RBPF-based particle filters is in general better than the bootstrap particle filters across all performance metrics. This suggests that the ability of the RBPF-based particle filters to marginalise out the gaussian components of the hidden state is very useful in improving the inference performance of the particle filter.

Secondly, we note that in general, the performance of the dense sampling RBPF and the adaptive sampling RBPF are both slightly better than that of the generic RBPF, suggesting that the dense sampling and adaptive sampling improvements are both useful in improving the inference performance of the generic RBPF. Figures 7b and 7d indicate that the performance of the generic RBPF degrades significantly as the number of particles (N) used drops, likely due to sample impoverishment. However, the dense sampling RBPF and adaptive sampling RBPF are both more resistant to this degradation in performance, indicating that they are likely less affected by sample impoverishment.

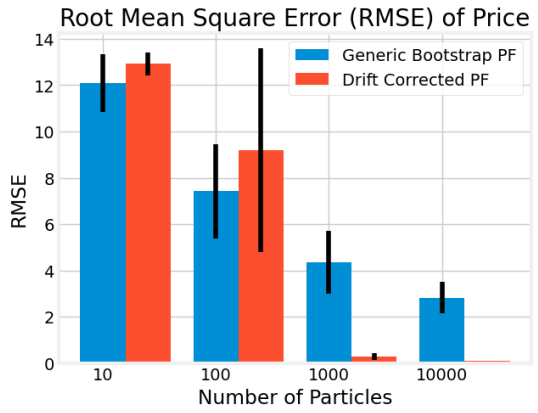
In particular, the adaptive sampling RBPF is extremely resistant to the degradation in performance as the number of particles drops, with good inference performance even with as low as 10 particles.

Lastly, we note that the binary prediction error of the inferred underlying returns state remains very stable and low for almost all values of N. Similarly as above, this is likely due to the more forgiving nature of using BPE as a metric. We are able to achieve BPEs of (~ 0.22) which is very low, suggesting that our particle filtering algorithms are able to infer the direction of the underlying returns state correctly over 70% of the time.

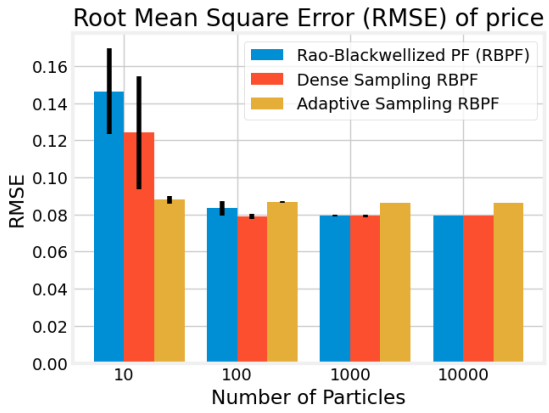
5.1.1 Bootstrap Particle Filters

To take a deeper look at why the generic one dimensional bootstrap particle filter of subsubsection 3.3.1 performs badly at tracking the underlying price state, we use a dataset consisting of 60 minutes of high frequency tick data for the EUR/USD currency pair from the 16th of October 2019, totalling 4196 discrete price observations.

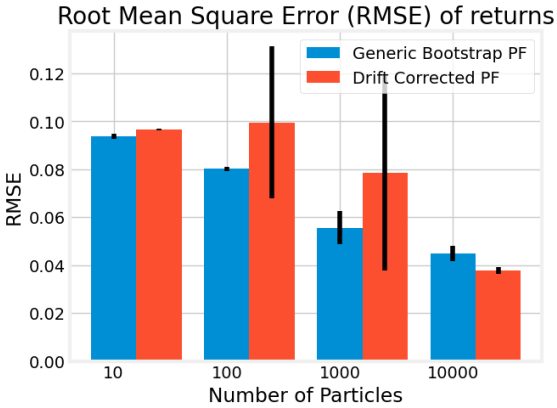
Figure 8 gives both the observed and inferred returns and price process for the simple bootstrap particle filter. We can clearly see the effectiveness of the noise removal in the inferred returns process. However, we note that the inferred price process suffers greatly from drift. Small errors in the inference of the returns process are compounded, leading to



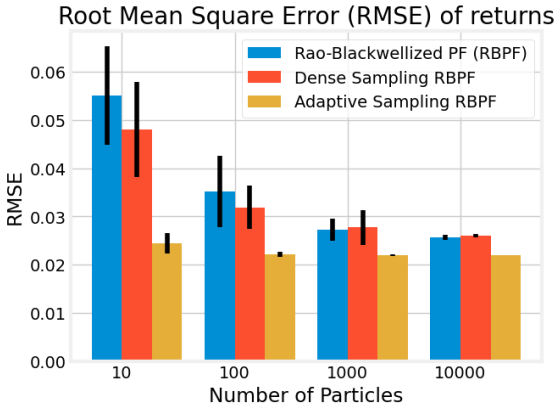
(a) RMSE of price(x_1) state



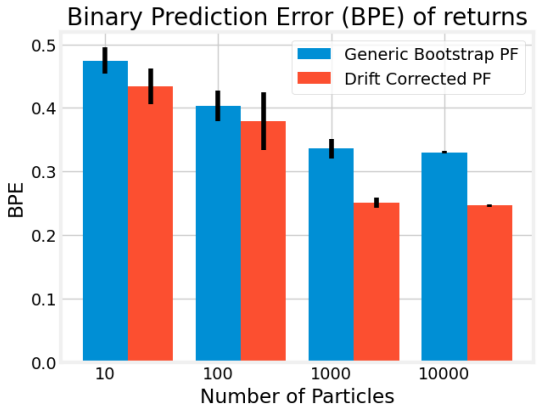
(b) RMSE of price(x_1) state



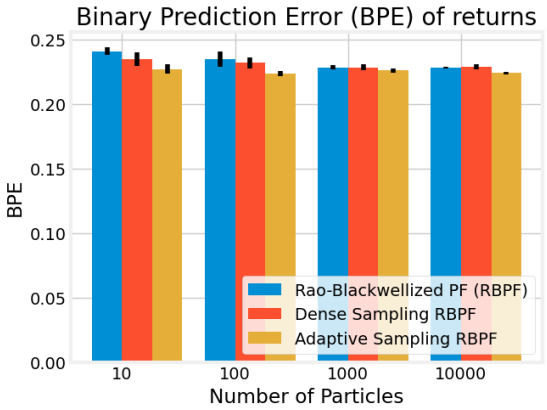
(c) RMSE of returns(x_2) state



(d) RMSE of returns(x_2) state



(e) BPE of returns(x_2) state

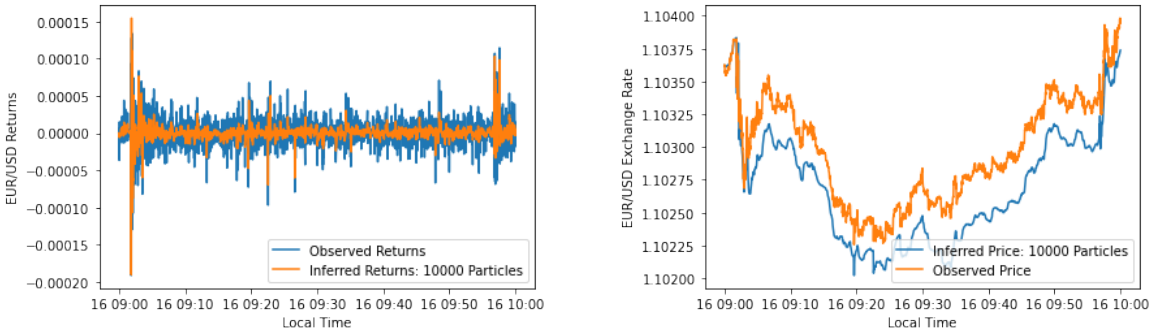


(f) BPE of returns(x_2) state

Figure 7: Comparison of inference performance between the 5 inference algorithms as N varies

the inferred price process deviating from the observed price.

This causes the poor inference results when the generic particle filter is used to infer the underlying price process of the simulated signal in Figure 6a.

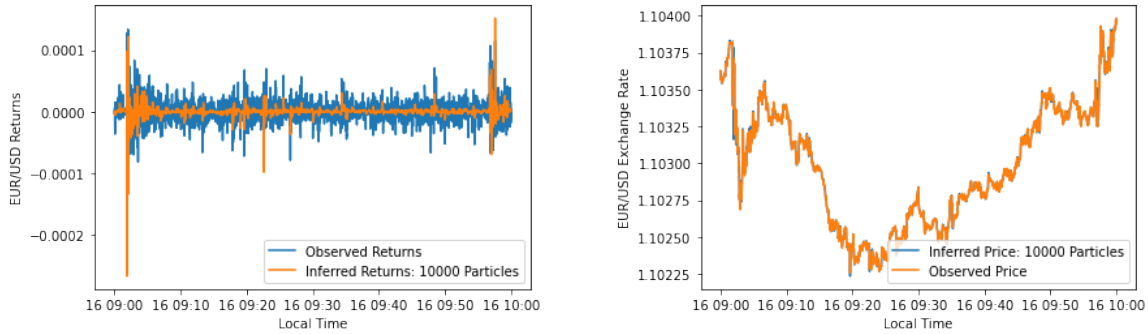


(a) Inferred Returns Process (b) Inferred Price Process

Figure 8: Simple Bootstrap Particle Filter, $N = 10000$ particles

On the other hand, the drift-corrected particle filter attempts to correct for price drift in the system by using numerical integration to obtain an estimate of the current inferred value of the price process at every time step.

Figure 9a gives both the observed and inferred returns and price process for the drift-corrected particle filter. We note qualitatively that the inferred returns process shows much greater noise reduction, and the inferred price process no longer suffers from price drift.



(a) Inferred Returns Process (b) Inferred Price Process

Figure 9: Drift-Corrected Particle Filter, $N = 10000$ particles

5.1.2 Dense Sampling Rao-Blackwellized Particle Filter

We now examine the reduction in sample impoverishment due to the dense sampling improvement described in **??**.

Similarly to **??**, we demonstrate the problem of sample impoverishment by simulating a single time step of the particle filter, for varying values of $y_k - \boldsymbol{CA}\boldsymbol{\mu}_{k-1}^{(i)}$. Fixing $\mu_k^{(i)}$, $\Sigma_k^{(i)}$ whilst varying y_k and N, we simulate one time step of the RBPF update step described above and present the normalised entropy of the RBPF particle weights with and without the dense sampling improvement in Figure 2.

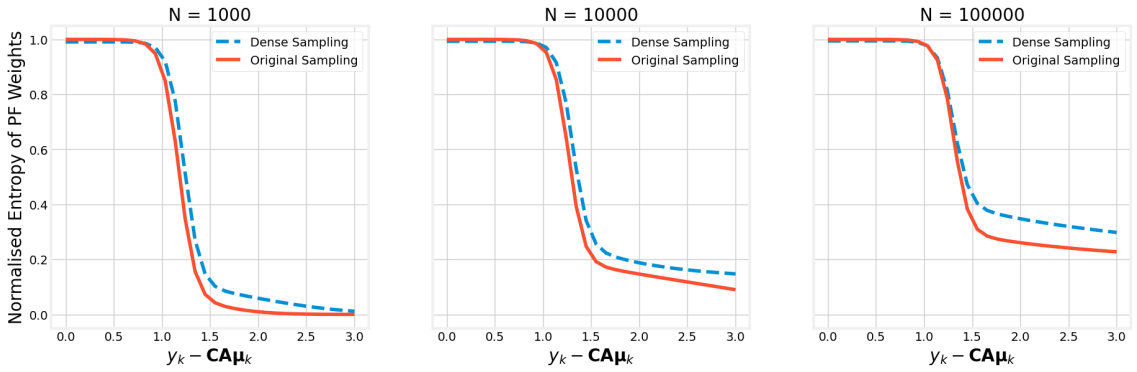


Figure 10: Improvement in normalised entropy of RBPF particle weights using dense sampling with hyperparameters: ($\epsilon_\lambda = 0.1$, $m = 2$)

We can see from Figure 2 that the dense sampling improvement causes the normalised entropy of the RBPF particle weights to increase for larger price outliers $y_k - \boldsymbol{CA}\boldsymbol{\mu}_{k-1}^{(i)}$. A higher normalised entropy indicates that the weights of the RBPF particles are more evenly distributed and hence that there are a larger number of particles with effective weights under the dense sampling improvement. This helps to reduce sample impoverishment, and can improve the performance of the generic RBPF.

5.1.3 Adaptive Sampling Rao-Blackwellized Particle Filter

Figure 11 shows the distribution of samples $\lambda_k^{(i)}$ from the Adaptive Sampling RBPF as well as the ground truth λ_k used to generate the simulation data. Figure 12 shows a similar figure for the generic RBPF *after* the resampling step.

Comparing Figures 11 and 12, we can see qualitatively that the distribution of samples from the Adaptive Sampling RBPF is distributed more closely around the ground truth λ_k,

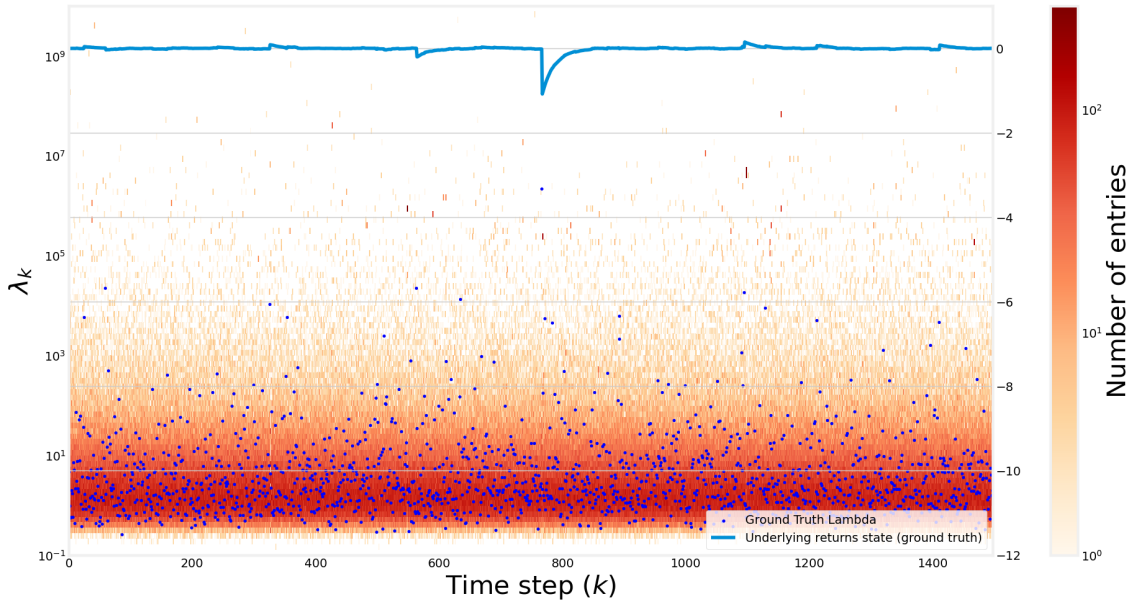


Figure 11: Distribution of samples $\lambda_k^{(i)}$ from the Generic RBPF

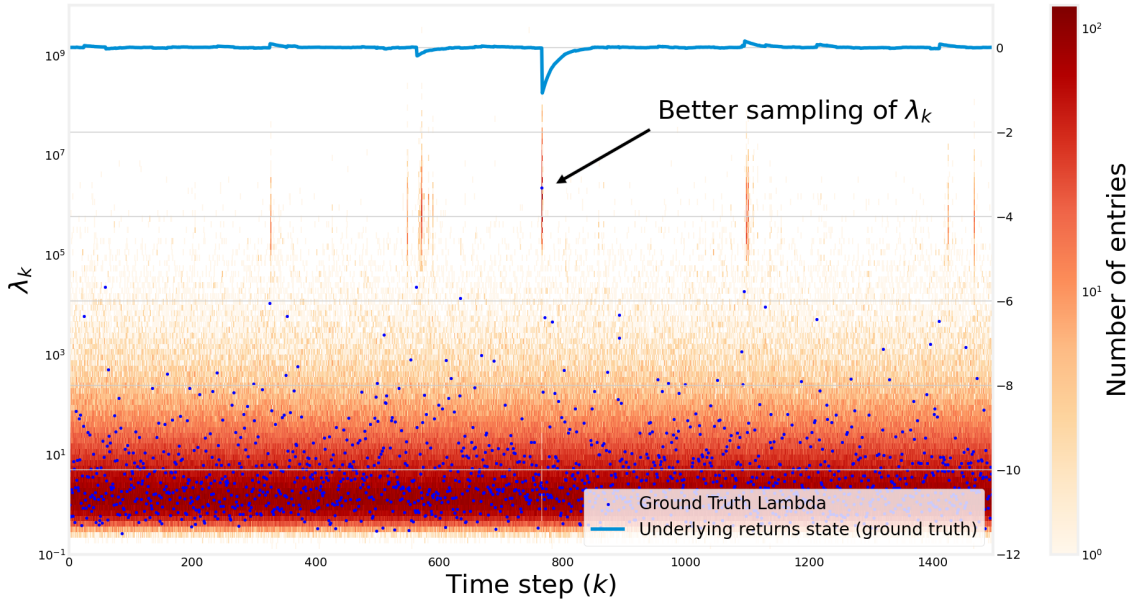


Figure 12: Distribution of samples $\lambda_k^{(i)}$ from the Adaptive Sampling RBPF

resulting in a many more effective samples being generated using the Adaptive Sampling RBPF compared to the generic RBPF.

Note the extremely challenging nature of forming a good importance sampling estimate of the posterior $p(\lambda_k|y_k, \mu_{k-1}, \Sigma_{k-1})$ using only samples from the prior as λ_k ranges over 7 orders of magnitude (10^{-1} to 10^7) in this example. This makes it very difficult to form a good discrete

importance sampling estimate, which would require a huge amount of particles to cover the entire possible range of λ_k (which theoretically has infinite support). With the adaptive sampling improvement, we are able to use the information from the current observation to sample directly from the approximate posterior, which increases the sampling efficiency of the particle filter greatly.

We next investigate how few particles we can use with the new adaptive sampling particle filter before performance degrades unacceptably. Figure 13 shows how the RMSE of the inferred underlying returns process changes for very low values of N. Interestingly, we note that the adaptive sampling RBPF is still capable of good inference performance even with extremely low $N = 4$, unlike the generic RBPF, which suffers from degradation of inference performance when using a low number of particles.

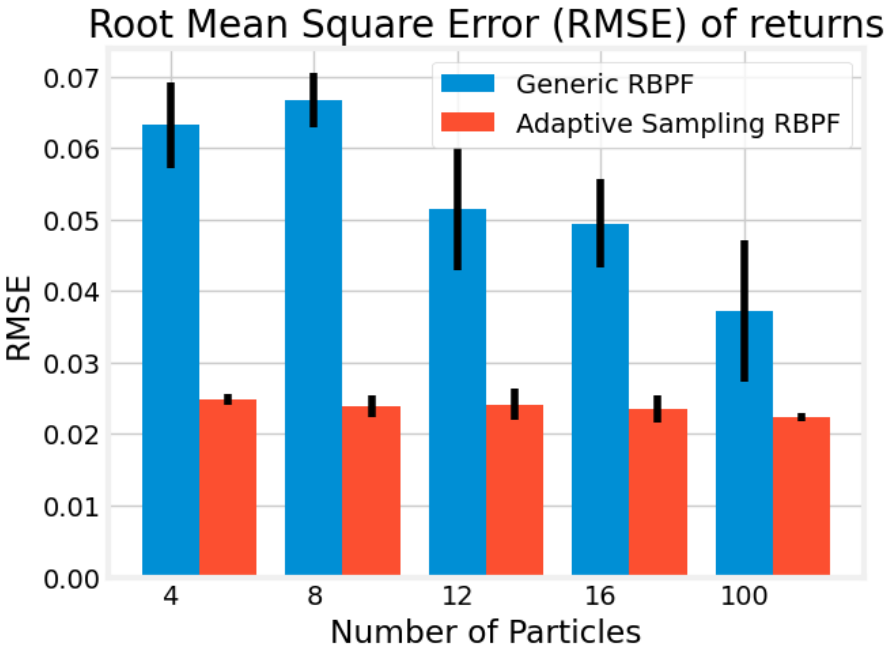


Figure 13: Performance of the RBPF-based particle filters for very low N

Lastly, we investigate the accuracy and sampling efficiency of our approximate improved rejection sampling method (described in subsubsection 3.3.5) that was employed in the adaptive sampling RBPF.

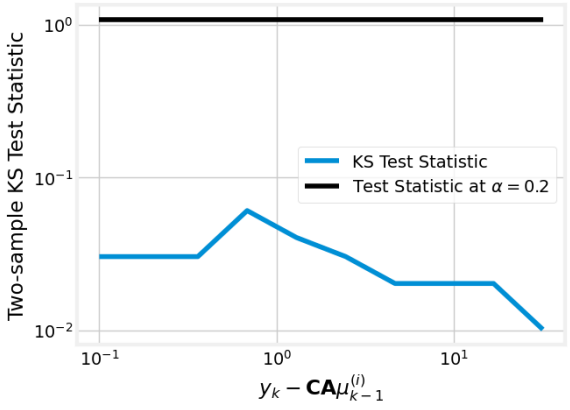
To investigate the accuracy of the approximate improved rejection sampler, we use the two-sample Kolmogorov–Smirnov(KS) test. The KS test is a two-sided statistical test for the null hypothesis that 2 independent sets of samples are drawn from the same continuous distribution. The KS test statistic quantifies the distance between the empirical distribu-

tion function of 2 sets of samples. The smaller the test statistic, the closer the empirical distribution function of these 2 sets of particles.

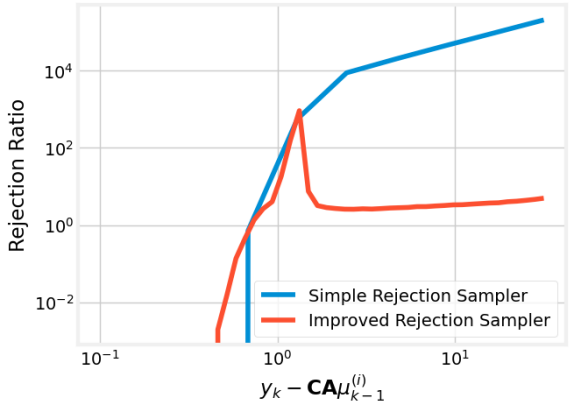
Figure 14a shows the two-sample KS test statistic obtained when comparing 10000 samples from each of the simple and improved rejection samplers as $y_k - \boldsymbol{CA}\boldsymbol{\mu}_{k-1}^{(i)}$ varies. We note that the KS test statistic is in general very small, and that even at a significance level of $\alpha = 0.2$, we are unable to reject the null hypothesis that the 2 sets of samples are different in distribution, demonstrating that our approximate improved rejection sampler is able to produce samples which are close in distribution to the exact simple rejection sampler.

Figure 14b shows the rejection ratio of each of the 2 rejection samplers when used to generate 10000 samples. The rejection ratio, defined in Equation 41, provides a measure of the sampling efficiency of a rejection sampler by giving an indication of the expected number of iterations needed to draw a single sample from the rejection sampler. For small $y_k - \boldsymbol{CA}\boldsymbol{\mu}_{k-1}^{(i)}$, the improved rejection sampler is equivalent to the original rejection sampler, resulting in roughly similar rejection ratios. However, at large $y_k - \boldsymbol{CA}\boldsymbol{\mu}_{k-1}^{(i)}$, the improved rejection sampler is much more efficient than the simple rejection sampler, and provides large efficiency gains of over 2 orders of magnitude when $y_k - \boldsymbol{CA}\boldsymbol{\mu}_{k-1}^{(i)}$ is large.

$$\text{rejection ratio} = \frac{\text{num samples rejected}}{\text{num samples to be generated through rejection sampling}} \tag{41}$$



(a) Two-sample Kolmogorov–Smirnov(KS) Test between simple and improved rejection samplers



(b) Rejection Ratio

Figure 14: Accuracy and sampling efficiency of the improved rejection sampler

5.1.4 Computational Cost Comparison

5.2 Parameter Estimation

5.3 Application to EUR / USD Exchange Rates

In this section, we evaluate the performance of the trading signals generated by our particle filter when applied to EUR/USD exchange rates (For the month of Nov 2019). Figure X shows the profit and loss generated by a trading strategy employing the generic RBPF with $N = 1000$ particles and hyperparmeters described in subsection 4.4.

5.3.1 Trading Performance

In this section, we evaluate the performance of the trading signals generated by our particle filter when applied to EUR/USD exchange rates (For the month of Nov 2019).

Figure 15 shows the profit and loss generated by a trading strategy employing the generic RBPF with $N = 1000$ particles and hyperparmeters described in subsection 4.4. The performance of the fund constructed based on the inference models developed in this project look to be very promising, with a annualized sharpe ratio of 1.10.

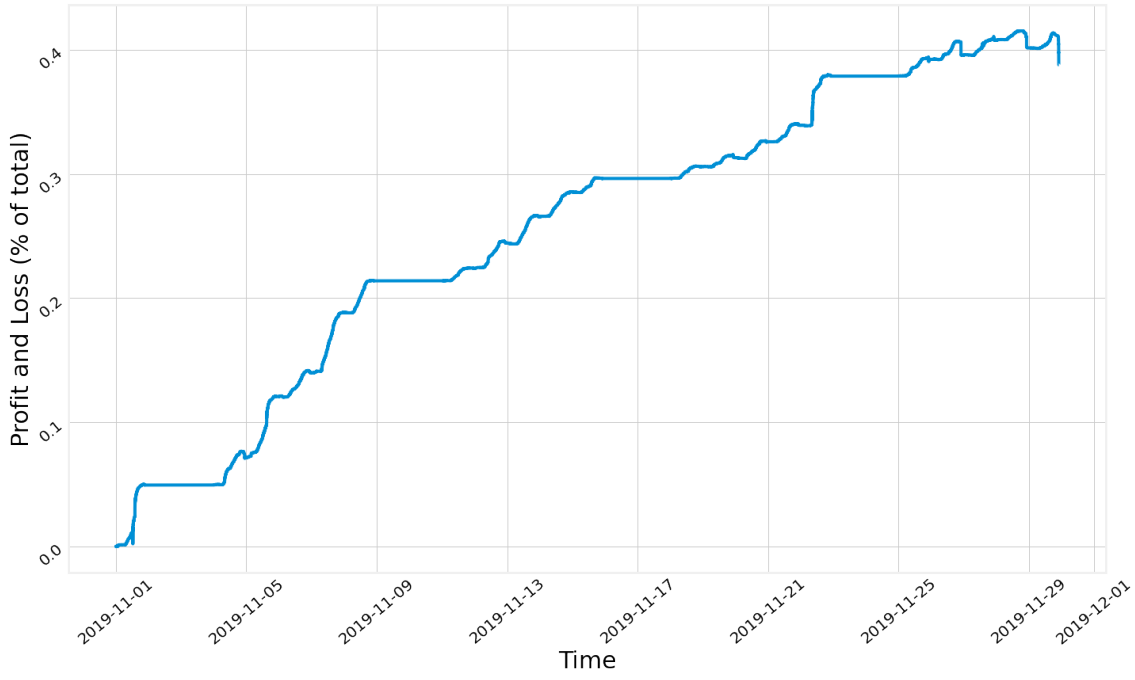


Figure 15: Profit and Loss (PnL) in terms of % of the total fund

5.3.2 Time-scale exploration

In this section, we evaluate the performance of the trading signals generated by our particle filter when applied to EUR/USD exchange rates (For the month of Nov 2019). Figure X shows the profit and loss generated by a trading strategy employing the generic RBPF with $N = 1000$ particles and hyperparmeters described in subsection 4.4.

6 Conclusion

References

[1] Hugh L Christensen, James Murphy, and Simon J Godsill. "Forecasting high-frequency futures returns using online Langevin dynamics". In: *IEEE Journal of Selected Topics in Signal Processing* 6.4 (2012), pp. 366–380.

[2] Yacine At-Sahalia, Jean Jacod, et al. "Testing whether jumps have finite or infinite activity". In: *the Annals of Statistics* 39.3 (2011), pp. 1689–1719.

[3] Carol L Osler. "Stop-loss orders and price cascades in currency markets". In: *Journal of international Money and Finance* 24.2 (2005), pp. 219–241.

[4] Gennady Samoradnitsky. *Stable non-Gaussian random processes: stochastic models with infinite variance*. Routledge, 2017.

[5] Peter Tankov. *Financial modelling with jump processes*. Chapman and Hall/CRC, 2003.

[6] John M Chambers, Colin L Mallows, and BW Stuck. "A method for simulating stable random variables". In: *Journal of the american statistical association* 71.354 (1976), pp. 340–344.

[7] Simon Godsill and Ercan E Kuruoglu. "Bayesian inference for time series with heavy-tailed symmetric α-stable noise processes". In: *Proc. Applications of heavy tailed distributions in economics, engineering and statistics* (1999).

[8] *Historical Data Feed*. URL: `https://www.dukascopy.com/swiss/english/marketwatch/historical/` (visited on 05/12/2020).

[9] Yan-Leung Cheung et al. "Intraday stock return volatility: The Hong Kong evidence". In: *Pacific-Basin Finance Journal* 2.2-3 (1994), pp. 261–276.

[10] Torben G Andersen, Martin Thyrsgaard, and Viktor Todorov. "Time-varying periodicity in intraday volatility". In: *Journal of the American Statistical Association* (2019), pp. 1–26.

[11] Paolo Zaffaroni. "Large-scale volatility models: theoretical properties of professionals' practice". In: *Journal of Time Series Analysis* 29.3 (2008), pp. 581–599.

List of Figures

List of Tables