UNIVERSITY OF CAMBRIDGE
Department of Engineering

Simulation and inference for limit order books in high-frequency finance

Author Name: Oliver Dunkley
Supervisor: Professor Simon J. Godsill

Date: 30 May 2018

Simulation and inference for limit order books in high-frequency finance

Oliver Dunkley, Pembroke College*

30 May 2018

Technical Abstract

This project aims to modify and improve filtering and prediction methods used for analysing time-series financial data, in order to achieve superior filtering performance compared with the previous state of the art. Improvements to these methods are beneficial not only to trading algorithms but across engineering, in problems as wide as robotics and forecasting. The focus here, however, is on the limit order book: the flow of buy and sell orders that drives the market.

A stochastic model for generating limit order book data is derived, where trade data is treated as a hidden Markov model with additional nonlinear Poisson-distributed jumps present. The Kalman and particle filters are extended to utilise additional information from the limit order book, in order to most accurately detect these jumps and the price momentum. Modifications to the original particle filter, and to previous work incorporating order volume data, include: an additional trend term in the state equation; resetting the trend means and covariances on jumps; exploring weighted functions of order volumes; and combining these methods into an overall best model.

This model is implemented, and tests conducted on low-frequency and high-frequency trade data to assess both empirical and statistical metrics. These include mean square error, binary prediction accuracy, log-likelihood and inverse CDF sampling over multiple time frames, to best approximate performance in a real trade system. Consideration is also given to the challenges faced in applying these methods to very high-frequency data, and how they may be addressed. Methods are explored for optimising hyperparameters against changing real-time data, and for implementing these algorithms in an efficient and scalable way conducive to real-world use.

Performance improvements are demonstrated with the new combined model over previous models, and its versatility is shown by its ability to track jumps in high-frequency data which exhibit complex nonlinear dynamics. These results are promising for future applications of these methods in work on limit order book inference and more generally in time series analysis.

*Supervised by Professor Simon J. Godsill, Signal Processing and Communications Laboratory.

Contents

Chapter 1
Introduction

The real-world application of statistical models often presents unforeseen challenges to researchers, and this is certainly the case with the particle filter and its applications in high-frequency trading. The field requires overcoming unique problems: data and trends are changing constantly, there is intense competition and time frames are becoming increasingly short. As a result, advancements to the particle filter that incorporate more information from the limit order book can be highly useful; they can also contribute to other aspects of engineering that rely on time series analysis. In this report, several new particle filter models are presented, and their improved performance is shown against real data at varying order update frequencies. The impact of these results in the wider field of order book inference is also considered.

1.1 Motivation

Trading firms and fund managers rely on accurate models of their equities and derivatives to provide liquidity to the market, and to make profitable trades. Due to the tiny margins that these firms often operate with, any improvement in the performance of these models can provide a significant increase in profits. A more stable market with less risk of extreme price swings is another benefit when demand for assets can be more reliably predicted by better models. In the high-frequency domain, with order execution times now measured in microseconds (Cont, 2011), trades are usually made entirely algorithmically, and strategies typically use statistical methods such as those presented in this report.

There is a wider application of the research undertaken, due to the fact that noisy, hard to predict financial data displays many of the same characteristics as tracking data observed in physical control applications. For example, time series analysis of sensor data is increasingly important in autonomous vehicles and other areas of robotics (Pendleton et al., 2017). On a wider level, areas as broad as weather forecasting make use of the underlying Kalman filter and similar Bayesian methods (Galanis et al., 2006). Advancements in the algorithms used in financial trades, then, may also find beneficial uses in physical engineering challenges and other important industries, even where the limit order book is not used.

1.2 Literature review

There is an extensive body of literature on financial time series analysis. Research areas range from probabilistic models that treat incoming order events as exponentially distributed random variables (Cont, Stoikov, and Talreja, 2010), to neural networks including long short-term memory modules that apply deep learning to the problem (Bao, Yue, and Rao, 2017). The field of time series prediction in general is rich: other advances in areas such as Gaussian processes (Frigola-Alcalde, 2015) are a demonstration of the wide range and versatility of tools that can be applied. While these methods have proved promising in increasing trading performance in some circumstances, they are generally highly compute-intensive and, especially in the case of deep networks, are typically black-box-type algorithms with little transparency in their relation to market fundamentals. The lack of revealing uncertainty information in deep learning models has typically prevented them from being applied to many financial problems. While work is ongoing to change this and introduce more useful uncertainty to many deep learning algorithms (Gal, 2016), there are still many useful applications of more classical statistical methods like those presented in this work.

In finance, statistical models typically used for investment decisions have included GARCH (reviewed in Cont, 2011) and Black-Scholes (reviewed in Papanicolaou, 2015), while momentum strategies are also popular in high-frequency trading. Their success might appear to contradict the efficient market hypothesis, which states that it is impossible to predict future price movements because all information about an asset is reflected in its current price. However, market inefficiencies do exist and are driven mainly by investor psychology, so the economic assumption that financial data follows a martingale process is relaxed. Here, we will consider specifically the concept of modelling high-frequency trade data as a jump-diffusion process, on which inference will be performed with Bayesian filtering algorithms including the Kalman and particle filters (also known as sequential Monte Carlo).

Since their introduction in 1993 (Gordon, Salmond, and Smith, 1993), particle filters have been studied and developed extensively, due to their flexibility and performance when applied to problems to which typical linear filters could not adapt. Work on the use of Rao-Blackwellisation (Rao, 1945; Blackwell, 1947) to enable the combination of linear Gaussian models and nonlinear jumps has been ongoing for over a decade (reviewed in Godsill, 2007), and was more recently applied specifically to the financial tracking domain by Christensen, Murphy, and Godsill (2012). Different resampling schemes have also been proposed, which can significantly change the performance characteristics of a particle filtering algorithm (reviewed in Turner and Sherlock, 2013).

The issue of online optimisation, especially of hyperparameters, has been tackled outside the financial domain, with methods such as online expectation-maximisation (Cappé, 2011). In finance, it has also been shown (Andersen and Bollerslev, 1997) that various markets including foreign exchange and equities display strong intraday periodicity. This implies that a model which runs continually must have some built-in method for accounting for these dynamics and updating its hyperparameters over time. We will consider that aim later in this report.

1.3 Aims and objectives

The primary goals of the project were to investigate new models for improving the accuracy of price forecasting, and to make use of the full order book state with all high-frequency tick data to see if performance improvements could be made over previous state-of-the-art methods. This would specifically focus on extensions and modifications to the particle filter and Kalman filter. Trade-offs must be made in a practical sense between performance and speed, as it was desired to allow the algorithms to run online in real-time.

Performance would be compared when evaluated across multiple future time frames, using several performance metrics that highlight the different strengths and weaknesses of the models. New methods for incorporating high-frequency tick data were also to be investigated, as existing methods were found to perform poorly with the different shape of state evolution seen in event-based data compared to lower-frequency snapshot data.

Chapter 2 presents the theory, derivation and mathematical reasoning behind the improved models. Chapter 3 explores the datasets used for the experimental analysis and compares some of their important properties. Chapter 4 outlines the experimental method and implementation of the algorithms, how they apply to the chosen datasets and how performance can be compared between them. Chapter 5 gives the important results and discusses the improvements of these methods over previous models, and Chapter 6 gives concluding statements and looks to how the research here may be implemented in a practical setting in the future.

Chapter 2

Theory

In this chapter, we explore the theory behind the methods used in this area of high-frequency financial analysis. Notable algorithms are the Kalman filter and particle filter, and these are presented in the context of limit order book inference. New ways of using these methods on limit order book data are derived, and methods of achieving optimum performance are considered.

2.1 Bayesian filters

Bayesian filters include the well-known Kalman and particle filters, and provide the basis for the work hereafter. *Filtering* means computing the conditional probability density function $p(x_t|y_0, ..., y_t)$, i.e. estimating the hidden state of a Markov process using observations up to and including the current time. In contrast, *prediction* is the estimation of some future state x_{t+m} for $m > 0$, based on the same observations. The Kalman filter is an algorithm for computing the distribution over x_t, using a *prediction* step followed by an *update* step.

The particle filter (otherwise known as sequential Monte Carlo), is a way of modelling the distribution of x_t using a set of *particles* – points containing some state distribution – each with an associated weight w_i that is sequentially updated by importance sampling and resampling. During each step a similar prediction-update step occurs for each particle, but particles with the highest weights are then propagated forward, while those with the lowest weights are removed. The benefit of the particle filter is that it can model nonlinear states, i.e. cases where the state transition equation is not just a multiplication by some transition matrix; we will use this to model the Poisson-distributed jump times in our model.

Monte Carlo methods in general involve the random sampling of points used to estimate an integral. The particle filter is an example of a Monte Carlo algorithm because each particle's state x_{t+1} is sampled from the state-space model's nonlinear probabilistic distribution, one that cannot be computed exactly. This necessitates the use of a large number of particles and their random samples to approximate the true distribution by the central limit theorem. The parallel nature of the particles means the algorithm benefits from increased computing power to allow more particles to be sampled concurrently. This is a computing challenge which we will return to later.

2.2 The context of order book inference

The limit order book (LOB) contains the state of all open limit orders for a traded commodity, at one point in time. It consists of bids and asks: requests to buy and sell at prices below and above the mid-price, respectively. As orders are entered, cancelled or fulfilled, the order book evolves with the mid-price maintained as the mean of the best bid and ask. Ordinarily, the state of the order book at any point in time cannot tell us about the history or trend of the asset's price, so useful features are typically computed and encoded into a state vector for use with a state-space model.

2.2.1 A stochastic model for continuous-time order arrivals

Consider that incoming orders may be modelled according to independent Poisson processes (Cont, Stoikov, and Talreja, 2010), such that order events arrive at exponentially distributed times. Limit buy or sell orders arrive at a distance i away from the best bid or ask price, respectively, with rate $\lambda(i)$; market buy or sell orders arrive with rate μ; and cancellations at a distance i away from the best bid or ask price arrive with rate $\theta(i)x$, where x is the existing volume of open orders at that price level. One possible function for λ is a power law:

$$\lambda(i) = \frac{k}{i^\alpha} \ . \tag{2.1}$$

Under these constraints, the state of the order book X at any moment in time can therefore be expressed as a continuous-time Markov chain. The number of open orders x at any price level has transition probabilities proportional to the distance of i from the ask price p_A, or the bid price p_B. This simple model of order flow provides a theoretical and empirical justification for the more complex state-space model that follows, as it was found by Cont, Stoikov, and Talreja (2010) that real data can be well-fitted to this model.

2.2.2 A state-space model for order book evolution

We model the evolution of the order book state, and its associated mid-price, as a jump-diffusion process, meaning it can be seen as a series of random jumps that are Gaussian distributed in size and Poisson distributed in arrival time, among a steady Gaussian trend with some Gaussian noise. The state is encoded in such a way that this can be considered a hidden Markov model, with standard state transition and output equations. The governing stochastic differential equation can be expressed as

$$dX_t = AX_t\,dt + b\,dW_t + c\,dJ_t \tag{2.2}$$

and the output equation is

$$y_i = x_{1,t} + v_t \tag{2.3}$$

where

$$v_t \sim \mathcal{N}(0, \sigma_{\text{obs}}^2)\ . \tag{2.4}$$

Here, X_t is the state vector, A is used to derive the state transition matrix, W_t is a Gaussian random variable and J_t is a Gauss-Poisson random variable representing the jump process. In the standard model proposed by Christensen, Murphy, and Godsill (2012), the state evolution is defined with

$$\begin{aligned} X_t &= \begin{bmatrix} x_{1,t} \\ x_{2,t} \end{bmatrix}, \qquad A = \begin{bmatrix} 0 & 1 \\ 0 & \theta_0 \end{bmatrix}, \\ b &= \begin{bmatrix} 0 \\ \sigma_0 \end{bmatrix}, \qquad c = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \end{aligned} \tag{2.5}$$

which represents a mean-reverting stochastic process with $x_{1,t}$ as the mean of the price variable at time t, and $x_{2,t}$ the mean of the trend variable. The mean-reversion coefficient is θ_0, σ_0 is the trend noise and σ_{obs} is the observation noise parameter.

By excluding the jump term J_t for now, and integrating over the range $t = S \to T$, eq. (2.2) is solved as follows (with derivation from Christensen, Murphy, and Godsill, 2012):

$$e^{-At}X_t = \int_0^t e^{-A\tau} b\,dW_\tau + k\ , \tag{2.6}$$

$$X_T = e^{A(T-S)}\left[X_S + \int_S^T e^{-A\tau} b\,dW_\tau\right]\ . \tag{2.7}$$

With the solution for X_T above, we see that X_T must be Gaussian distributed since X_S is Gaussian, and so is the stochastic integral. The expectation of the integral is also zero, so the expectation of X_T is

$$\mathbb{E}[X_T] = e^{A(T-S)}\mathbb{E}[X_S] \tag{2.8}$$

and its covariance is

$$\text{cov}(X_T) = e^{A(T-S)}\Big[Q(S,T) + \text{cov}(X_S)\Big](e^{A(T-S)})' \tag{2.9}$$

where

$$Q(r,s) \triangleq \int_r^s e^{-At} bb'(e^{-At})'\,dt\ . \tag{2.10}$$

The use of $Q(\cdot,\cdot)$ follows as a result of the Itô isometry, and a detailed proof is given in Christensen, Murphy, and Godsill (2012). In practice, Särkkä's method is used for efficiently computing an approximation to $Q(\cdot,\cdot)$ (Särkkä, 2006, which builds on work by Van Loan, 1978). It is summarised here for this implementation as follows.

We can express $Q(\cdot,\cdot)$ as

$$Q(r,s) = q(s) - q(r) \tag{2.11}$$

where

$$q(t) = e^{-At}P(t)(e^{-At})' \tag{2.12}$$

and

$$P(t) = e^{At}\sigma_J^2(e^{(At})' + \int_0^t e^{A(t-\tau)}bIb'(e^{A(t-\tau)})'d\tau \tag{2.13}$$

where the σ_J^2 term is zero if no jump is detected in that time interval, or non-zero otherwise. Then, using the method of matrix fraction decomposition (Särkkä, 2006, Remark 2.2), this is solved with

$$\begin{bmatrix} C(t) \\ D(t) \end{bmatrix} = \exp\left\{ \begin{bmatrix} A & bIb' \\ 0 & -A' \end{bmatrix} t \right\} \begin{bmatrix} I \\ 0 \end{bmatrix} , \tag{2.14}$$

and the final solution for $Q(\cdot,\cdot)$ is found with

$$P(t) = C(t)D(t)^{-1} . \tag{2.15}$$

This allows the integral in the iteration steps of the Kalman filter to be computed efficiently as the product of a decomposed matrix exponential, whose complexity scales with the dimension of A.

The above results for the state's first and second moments are used with the Kalman filter to infer the most likely state, dependent on all previous observations. However, because the jump term J_t is not a purely Gaussian process (jump arrival times are Poisson distributed), it requires an additional method to be used. The particle filter enables the unknown jump times to be sampled, propagating forward the particles most likely to generate the observed sequence and discarding the others at each step.

2.2.3 Inference with the Rao-Blackwellised particle filter

We will use a Rao-Blackwellised particle filter to propagate forward state estimates given the observed order book data: this method allows the use of the Kalman filter for efficient computation of the linear parts of the transition function, augmented with a particle filter for the nonlinear parts. The Kalman part of the filter is defined at time t_i with states X

and observations y given by

$$X_{t_i} = F_i X_{t_{i-1}} + u_i \ , \quad u_i \sim \mathcal{N}(0, C_{u_i}) \ , \tag{2.16}$$
$$y_i = G_i X_{t_i} + v_i \ , \qquad v_i \sim \mathcal{N}(0, C_{v_i}) \ , \tag{2.17}$$

with transition and observation matrices, F_i and G_i respectively, given by

$$F_i = e^{A(t_i - t_{i-1})} \ , \tag{2.18}$$
$$G_i = \begin{bmatrix} 1 & 0 \end{bmatrix} \ , \tag{2.19}$$

and transition and observation covariances C_{u_i} and C_{v_i} defined as

$$C_{u_i} = \mathrm{cov}(X_{t_i} \,|\, \tau_{t_{i-1}:t_i}, X_{t_{i-1}}) \ , \tag{2.20}$$
$$C_{v_i} = \sigma^2_{\text{obs}} \ . \tag{2.21}$$

Let $\tau_{a:b}$ define the location of the set of jumps within the time from a to b. Then, the prediction error decomposition,

$$p(y_i \,|\, y_{1:i-1}, \tau_{t_0:t_i}) \sim \mathcal{N}(y_i \,|\, \mu_{y_i}, C_{y_i}) \ , \tag{2.22}$$

allows us to sequentially calculate the observation likelihood, which is useful for later statistical analysis. The mean and covariance of the filtering distribution at the relevant times are given by

$$\mu_{y_i} = G_i \mu_{i\,|\,0:i-1} \ , \tag{2.23}$$
$$C_{y_i} = G_i C_{i\,|\,0:i-1} G_i' + C_{v_i} \ , \tag{2.24}$$

where

$$\mu_{i\,|\,0:i-1} = F_i \mu_{i-1\,|\,0:i-1} \ , \tag{2.25}$$
$$C_{i\,|\,0:i-1} = F_i C_{i-1\,|\,0:i-1} F_i' + C_{u_i} \ , \tag{2.26}$$
$$K_i = C_{i\,|\,0:i-1} G_i' (G_i C_{i\,|\,0:i-1} G_i' + C_{v_i})^{-1} \ , \tag{2.27}$$
$$\mu_{i\,|\,0:i} = \mu_{i\,|\,0:i-1} + K_i (y_i - G_i \mu_{i\,|\,0:i-1}) \ , \tag{2.28}$$
$$C_{i\,|\,0:i} = (I - K_i G_i) C_{i\,|\,0:i-1} \ . \tag{2.29}$$

Summing the weighted particle means for particles p with weights W_p and means $\mu_j^{(p)}$ provides the expected state value at time t_j,

$$\hat{X}_{t_j} = \mathbb{E}[X_{t_j} \,|\, y_{1:j}] \approx \sum_{p \in P_{t_j}} W_p t_j \mu^{(p)}_{j\,|\,0:j} \ . \tag{2.30}$$

During an iteration of the particle filter at time j, the number of child particles $M_j^{(p)}$ for each parent p is given by

$$M_j^{(p)} = \max(1, \lfloor M W_p(t_{j-1}) \rfloor) \ , \tag{2.31}$$

where M is some offspring parameter, and each child particle q has weights defined by

$$W_q(t_{j-1}) = \frac{W_p(t_{j-1})}{M_j^{(p)}} \ . \tag{2.32}$$

One or more proposed jump times, τ, for each new particle is sampled from

$$\tau_*^{(q)} \sim p_{\text{jump-time}}(\tau_* \,|\, t_{j-1}, \tau_{t_0:t_{j-1}}^{(p)}) \ , \tag{2.33}$$

with particles where these times fall before the current observation time defined as *jumping* particles. Therefore, using eq. (2.22), the particle weight updates are given by

$$W_q(t_j) \propto W_q(t_{j-1}) p(y_j \,|\, y_{1:j-1}, \tau_{t_0:t_j}^{(q)}) \ . \tag{2.34}$$

Full details of the variable rate particle filter algorithm are given by Christensen, Murphy, and Godsill (2012), but this completes the derivation of the equations used in implementing the models herein.

2.3 Extending the particle filter

Since the focus of the project was on improving the performance of this inference algorithm, several modifications to the original particle filter for the jump-diffusion process detailed above were implemented. These are derived in the following sections, before details of the model combining all of the modifications are given.

2.3.1 Extended trend model

Firstly, the extended trend state model uses the idea that significant impacts to the market spur larger, sharper changes to the price trend and jump distribution than general volatility does. It adds an additional state parameter, allowing trends at different time scales to be maintained. For this, we add the additional term $d\,dV_t$ to eq. (2.2), where V is another Gaussian process scaled by noise parameter σ_e, and the additional mean reversion coefficient term θ_e adds another dimension to the model. The new state equation is

$$dX_t = AX_t\,dt + b\,dW_t + c\,dJ_t + d\,dV_t \tag{2.35}$$

and the state vectors are now

$$X_t = \begin{bmatrix} x_{1,t} \\ x_{2,t} \\ x_{3,t} \end{bmatrix} , \quad A = \begin{bmatrix} 0 & 1 & 0 \\ 0 & \theta_0 & -\theta_0 \\ 0 & 0 & \theta_e \end{bmatrix} ,$$

$$b = \begin{bmatrix} 0 \\ \sigma_0 \\ 0 \end{bmatrix} , \quad c = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} , \quad d = \begin{bmatrix} 0 \\ 0 \\ \sigma_e \end{bmatrix} . \tag{2.36}$$

This is solved in a similar way to eq. (2.2), resulting in:

$$X_T = e^{A(T-S)}\left[X_S + \int_S^T e^{-A\tau} b \, dW_\tau + \int_S^T e^{-A\tau} d \, dV_\tau\right] \tag{2.37}$$

over the range $t = S \to T$. As before, X_T must be Gaussian since X_S is Gaussian and so are the two stochastic integrals. Now, similarly to before,

$$Q(r,s) = \int_r^s e^{-At} bb' (e^{-At})' \, dt + \int_r^s e^{-At} dd' (e^{-At})' \, dt \; . \tag{2.38}$$

Thus, the only modifications to the Kalman filter algorithm in Section 2.2.2 are to replace (bb') with $(bb' + dd')$ when calculating $Q(r,s)$, and to use the new state vectors in eq. (2.36).

2.3.2 Trend reset model

For the second proposed modification, the trend resetting jump model considers that large jumps in price trend or order flow may be best modelled by a reset of one or more of the state parameter means and their covariances, setting them to zero or some fixed prior. This follows from the theory that significant events causing abrupt price changes may cause an existing trend to disappear just as abruptly. In this model, instead of child particles inheriting their parents' state mean and covariance on jumps, the children's trend terms are reset to zero while keeping their price terms. This is achieved by setting the state mean and covariance of every particle that is sampled with a jump at time point i to

$$\mu_{i|0:i-1} = \begin{bmatrix} [\mu_{i|0:i-1}]_0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} , \tag{2.39}$$

$$C_{i|0:i-1} = \begin{bmatrix} [C_0]_{0,0} & 0 & \cdots & 0 \\ 0 & & & \\ \vdots & & \text{\huge 0} & \\ 0 & & & \end{bmatrix} . \tag{2.40}$$

2.3.3 Volumetric model

Thirdly, a model is also outlined utilising the order book's volume data. A version of this model was developed previously by Jerjian (2017), which includes additional elements in the state transition matrix for the volume of orders $v_{i,t}$ at the closest n price levels around the mid-price, with additional noise parameters σ_v and mean reversion coefficients θ_v. The model then infers the additional state parameters for these, with the state equation:

$$\begin{bmatrix} dx_{1,t} \\ dx_{2,t} \\ d\phi_{-n,t} \\ \vdots \\ d\phi_{-1,t} \\ d\phi_{1,t} \\ \vdots \\ d\phi_{n,t} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\ 0 & \theta_0 & v_{-n,t} & \cdots & v_{-1,t} & v_{1,t} & \cdots & v_{n,t} \\ 0 & 0 & \theta_v & & & & & \\ \vdots & \vdots & & \ddots & & \mathbf{0} & & \\ \vdots & \vdots & & & \ddots & & & \\ \vdots & \vdots & & \mathbf{0} & & \ddots & & \\ \vdots & \vdots & & & & & \ddots & \\ 0 & 0 & & & & & & \theta_v \end{bmatrix} \begin{bmatrix} x_{1,t} \\ x_{2,t} \\ \phi_{-n,t} \\ \vdots \\ \phi_{-1,t} \\ \phi_{1,t} \\ \vdots \\ \phi_{n,t} \end{bmatrix} dt + \begin{bmatrix} 0 \\ \sigma_0 \\ \sigma_v \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \sigma_v \end{bmatrix} dW_t + \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ 0 \end{bmatrix} dJ_t \ . \tag{2.41}$$

2.3.4 Functions of order book volume

Alternative models utilising the volume data in other ways can also be implemented. Instead of adding extra parameters for volumes at each price level around the mid-price, some function of the volume data can be computed and used instead. This has two effects: first, it allows for the impact of nonlinear functions of the volumes, that cannot be modelled by the linear state equation above, to be considered. Second, it reduces the number of extra dimensions added to the size of the transition matrix. Each extra dimension adds significant computational expense – scaling with at least $\mathcal{O}(n^2)$ for n state parameters – with possibly little benefit, and also has the potential of leading to overfitting to the data.

Two such functions are: a simple sum of all order volumes on each side of the mid-price, and a weighted sum, where the volumes are multiplied by an exponential factor depending on their distance from the mid-price. Both of these reduce the number of extra dimensions added to the transition matrix to just two. With a two-parameter volume function, the state equation from eq. (2.41) becomes the much simpler equation:

$$\begin{bmatrix} dx_{1,t} \\ dx_{2,t} \\ d\phi_{B,t} \\ d\phi_{A,t} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & \theta_0 & V_{B,t} & V_{A,t} \\ 0 & 0 & \theta_v & 0 \\ 0 & 0 & 0 & \theta_v \end{bmatrix} \begin{bmatrix} x_{1,t} \\ x_{2,t} \\ \phi_{B,t} \\ \phi_{A,t} \end{bmatrix} dt + \begin{bmatrix} 0 \\ \sigma_0 \\ \sigma_v \\ \sigma_v \end{bmatrix} dW_t + \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} dJ_t \ . \tag{2.42}$$

A simple imbalance sum was the first function implemented, which adds the total weight of all open orders on each side of the mid-price. The log of this value is used to emphasize the relative orders of magnitude, and this function leads to the parameters:

$$V_A = \log\left(\sum_{n=1}^{N_A} v_n\right) , \tag{2.43}$$

$$V_B = \log\left(\sum_{n=1}^{N_B} v_{-n}\right) , \tag{2.44}$$

where v_i is the volume of open orders at the price level i, for levels where $i > 0$ means prices above the mid-price for open ask orders, and $i < 0$ means prices below the mid-price for open bid orders. N_A and N_B are the number of price levels where open ask and bid orders sit, respectively. Two separate values, and therefore parameters, were used here instead of a single imbalance number, to capture the overall size of all open orders as well as their predominant direction.

In addition, to test the theory that order volumes closer to the mid-price are more likely to be an active part of upcoming events, and therefore should be weighted more, an exponential weight was applied to each volume. This mirrors the exponential order book shape proposed in Section 2.2.1, and leads to a second function:

$$V_A = \log\left(\sum_{n=1}^{N_A} e^{-cn} v_n\right) , \tag{2.45}$$

$$V_B = \log\left(\sum_{n=1}^{N_B} e^{-cn} v_{-n}\right) , \tag{2.46}$$

where $c > 0$ is some exponential weight shape constant. This allows tuning c to change the expected order book shape, and thus control the model performance.

2.3.5 Combined model

The final model that was used to generate the results given later is a combined model that includes the three main optimisations detailed above. It is assumed that extra order book volume information supplements the trend detection and also follows the jump-reset theory, so that the models can be combined constructively.

The final state equation is

$$dX_t = AX_t\, dt + b\, dW_t + c\, dJ_t + d\, dV_t \tag{2.47}$$

and the state vectors are

$$X_t = \begin{bmatrix} x_{1,t} \\ x_{2,t} \\ x_{3,t} \\ \phi_{-n,t} \\ \vdots \\ \phi_{-1,t} \\ \phi_{1,t} \\ \vdots \\ \phi_{n,t} \end{bmatrix}, \quad A = \begin{bmatrix} 0 & 1 & 0 & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\ 0 & \theta_0 & -\theta_0 & v_{-n,t} & \cdots & v_{-1,t} & v_{1,t} & \cdots & v_{n,t} \\ 0 & 0 & \theta_e & 0 & & & & & \\ 0 & 0 & 0 & \theta_v & & & & & \\ \vdots & \vdots & & & \ddots & & \mathbf{0} & & \\ \vdots & \vdots & & & & \ddots & & & \\ \vdots & \vdots & & \mathbf{0} & & & \ddots & & \\ \vdots & \vdots & & & & & & \ddots & \\ 0 & 0 & & & & & & & \theta_v \end{bmatrix},$$

$$b = \begin{bmatrix} 0 \\ \sigma_0 \\ 0 \\ \sigma_v \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \sigma_v \end{bmatrix}, \quad c = \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ 0 \end{bmatrix}, \quad d = \begin{bmatrix} 0 \\ 0 \\ \sigma_e \\ 0 \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ 0 \end{bmatrix} \tag{2.48}$$

in the case of the most complex individual volume level option. The matrix A simplifies if functions of the order book volume are used instead, as in eq. (2.42). This state equation is solved using the standard Rao-Blackwellised particle filter, with the exception of the particle states when a jump is placed. They are set for that iteration such that the price mean is maintained, but the covariance is reset to the prior and all trend terms set to 0:

$$\mu_{i|0:i-1} = \begin{bmatrix} [\mu_{i|0:i-1}]_0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \tag{2.49}$$

$$C_{i|0:i-1} = \begin{bmatrix} [C_0]_{0,0} & 0 & \cdots & 0 \\ 0 & & & \\ \vdots & & \mathbf{0} & \\ 0 & & & \end{bmatrix}. \tag{2.50}$$

This completes the derivation of the models used in the project. They are implemented in Chapter 4 and tested in Chapter 5, but first we will look at the properties of the data used in testing, and at the viability of these methods from a theoretical perspective.

Chapter 3

Exploring order books

The datasets used are of vital importance to the success of any experimental inference project. This chapter summarises the analysis of the various data sources available and compares the real and synthetic order book data. Properties of this data are presented which are important for the later optimisation of the filtering algorithms, and the different types of data available are discussed.

3.1 Real order book data

The focus of these experiments will be on foreign exchange market or *forex* data. The large, well-distributed forex market, with its popular trade pairs and easy accessibility, is a popular choice for high-frequency traders and thus a good place to focus our analysis. The main dataset used for testing is supplied by Cambridge Capital, and contains order book data for the EUR/USD currency pair. Due to market opening hours, all timestamps are given in terms of the *trade date*, which is the following day's calendar date if the time is after 5:00 PM, and the current day's if before.

Three different types of data are provided for testing: best bid and offer (*BBO*), snapshots and ticks. The BBO data includes only the closest buy (*bid*) and sell (*ask*) price to the mid-price, along with the volume of open orders at those levels, at specific timestamps which arrive roughly one second apart. Snapshot data gives the full order book at these time points, including volumes for every price point either side of the mid-price. The tick data is most complete: it comprises a high-frequency stream-like list of orders, with new events arriving over 100 times per second. This requires further transformation from raw order events into a snapshot-like form before it is useful for analysis by our algorithms, but provides the most granularity of all the data formats and is therefore most useful for high-frequency analysis.

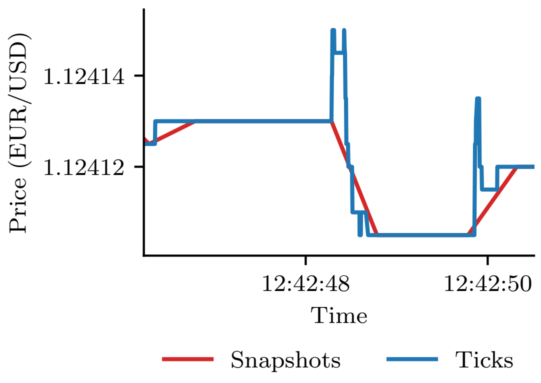

Figure 3.1. Example of snapshot and tick mid-price observations for the same trade period.

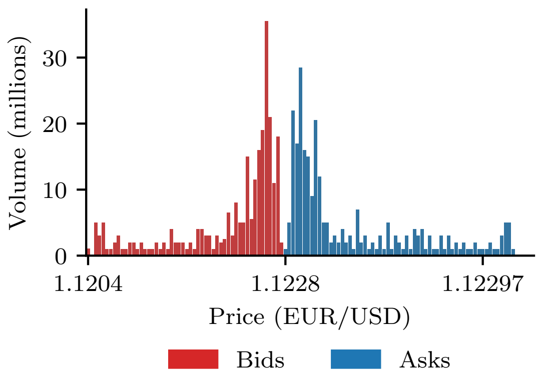

Figure 3.2. Example of open order volume distribution either side of the mid-price, at one moment in time.

3.1.1 Ticks and snapshots

In comparing the tick and snapshot data, we see in Figure 3.1 that the order event ticks corresponding to a certain price movement contain significantly more information than the snapshot data for that same time period. For example, we see that the tick data often contains spikes from orders that are created and then immediately cancelled. The stepped decrease of the tick line in the middle of the chart also demonstrates how large price movements do not happen as a result of a single order, and are instead made of a series of cascading events very close together in time. This means a model that only considers the difference between consecutive prices will not correctly detect large price jumps. However, the particle filter used in this project works despite this pattern because it relies on price movement over a set time, so is not affected by this characteristic of the tick data.

3.1.2 Order event types

Each bid or ask order in the streaming tick data is given a unique identifier, and contains an order type along with a price point and volume (the amount requested to be traded). Events can be a *new* order, an order *cancel* request, or an order *modify* request, where the outstanding volume for that order identifier is changed. Each new order can either be a *market* or a *limit* order. Market orders immediately match with the best priced outstanding order or orders on the opposite side, and limit orders add to the outstanding volume at their respective price, waiting to match. Orders may be cancelled at any time before they are matched (*fulfilled*).

It is found from the tick data that the vast majority of orders opened are cancelled before being fulfilled, often in a very short time: over the hour 00:00 to 01:00 on trade day 2015-09-02, exactly 50% of all new order events are cancellations. This is to be expected since methods involving rapid posting and cancelling of orders, including *spoofing* and *quote stuffing*, are common tactics by high-frequency traders who attempt to gain an edge over the market (Prewitt, 2012). This suggests we might be able to make inferences from cancelled orders that otherwise have no impact on the market mid-price. Since these orders are encoded into the order book volumes shown in Figure 3.2, models that incorporate these volumes, including those used in this work, should make use of this information. Although it may be beneficial to consider other ways of interpreting the noisy spikes in mid-price exhibited in Figure 3.1, these are outside the scope of this project, and are explored as possible future modifications in Section 6.2.

3.2 Synthetic order book data

In addition to the real dataset used, it is possible to generate synthetic order book data. It is theorised that this might be beneficial because it allows careful control over the order book properties, which might then be used to more efficiently optimise the models that follow. The continuous-time stochastic model proposed by Cont, Stoikov, and Talreja (2010) and detailed in Section 2.2.1, that uses independent Poisson processes to simulate order events, was implemented to generate examples of these synthetic order books. Portions of synthetic data were generated using parameters optimised for similarity with real order books. Below, we will compare these with real data.

3.2.1 Comparing synthetic and real data

Figures 3.3 and 3.4 show samples of real and synthetic data plotted as heat maps of open order volume. The real data, shown in Figure 3.3, exhibits properties typical of order-driven markets: order volume is highest narrowly above and below the mid-price, where orders are likely to be fulfilled fastest if the mid-price moves in that direction. Volume then tails off gradually for a short time, before remaining much more stable further away. Longer-lasting open orders sit in more regular, prominent levels at round price points (prices with decimals ending in multiples of 10) further away from the mid-price. Higher volumes are seen at these round price levels, an effect of investor psychology that serves to resist or delay price movements through these points, as clusters of orders typically gather around them. The overall price trend of the data shows small, stochastic movements as well as larger, more rapid jumps, which gives credence to the mixed approach of tracking jumps as well as trend diffusion in our model.

The portion of synthetic data generated from the stochastic model, shown in Figure 3.4, appears to approximate well the order volumes of the real data around the mid-price, with a similar decaying volume peak on each side. This data also shows a similar random-walk type movement, although due to the model's simplicity it is unable to model longer-term

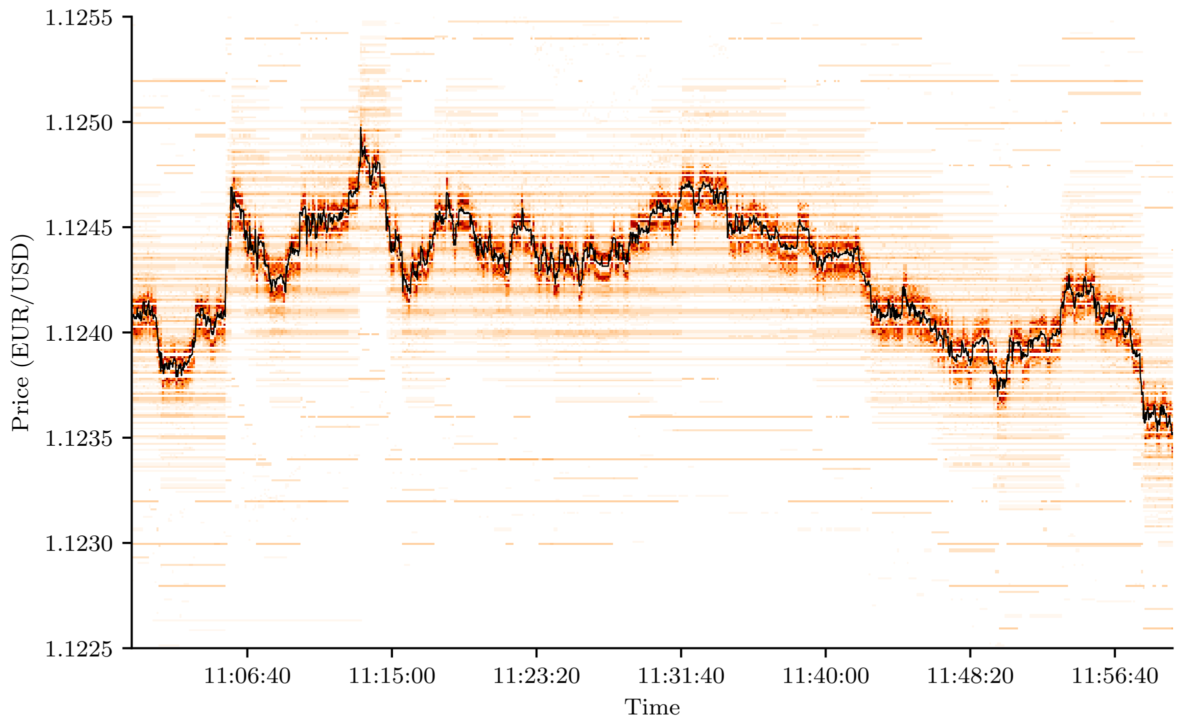

Figure 3.3. Real data: mid-price (black) and heat map showing volume of open orders above and below the mid-price over a one-hour period.

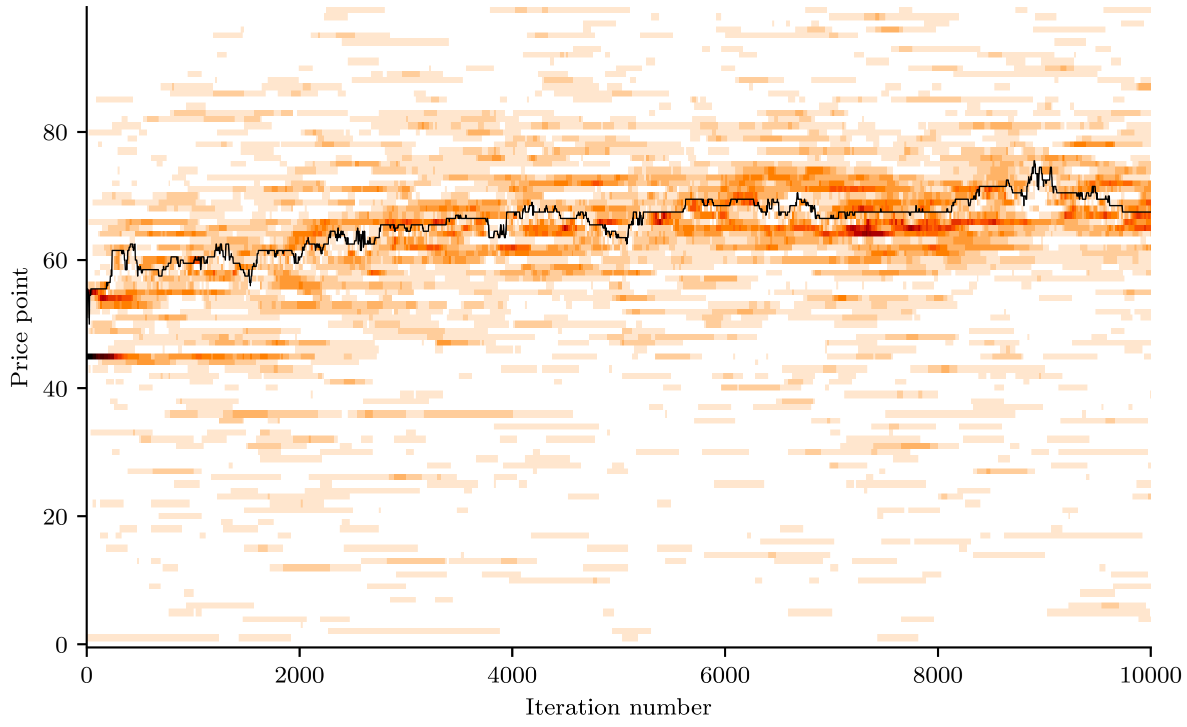

Figure 3.4. Synthetic data generated with 10,000 iterations. Mid-price (black) and heat map showing volume of open orders above and below the mid-price.

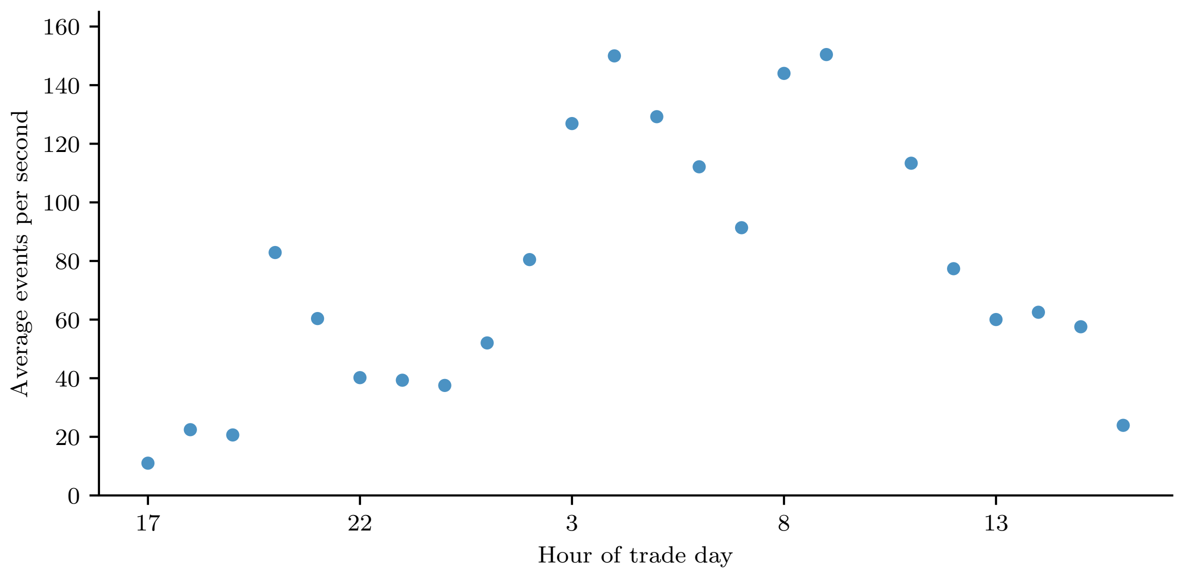

Figure 3.5. Order event rate (fulfilled or cancelled orders) per second, for each hour of a sample trade day (*EUR/USD 2015-09-02*).

trends or any larger jumps. The synthetic model also has no knowledge of human tendency towards round price points, which shows up visibly in the real model. This is important as longer-lasting orders around these points may play different roles in price movement to the higher-frequency orders closer to the mid-price. Therefore, while this synthetic model is useful for validating the exponential shape of the order book (which was considered when designing volume weighting functions), its simplicity means we will not use it for further analysis.

3.3 Limit order book properties

3.3.1 Order rate

The rate of incoming orders has several effects on the market that must be accounted for, and we hypothesise that the intraday variance of the order rate, which may be correlated with market opening hours, will necessitate adjusting the jump frequency accordingly in order to achieve the most accurate model. Figure 3.5 shows the hourly average order rate over a sample trade day.

We see that the most active hours are in the early morning, from 3:00 to 11:00 AM, and that there is a factor of 10 increase in market activity in peak hours, compared to the off-peak. Note that this graph was generated using tick data, so it includes not just new orders but also cancellations and modifications. Therefore, it is not necessarily the case that there is an increase in *orders* during these hours, but merely an increase in market activity that will also include an increase in other types of order-related events.

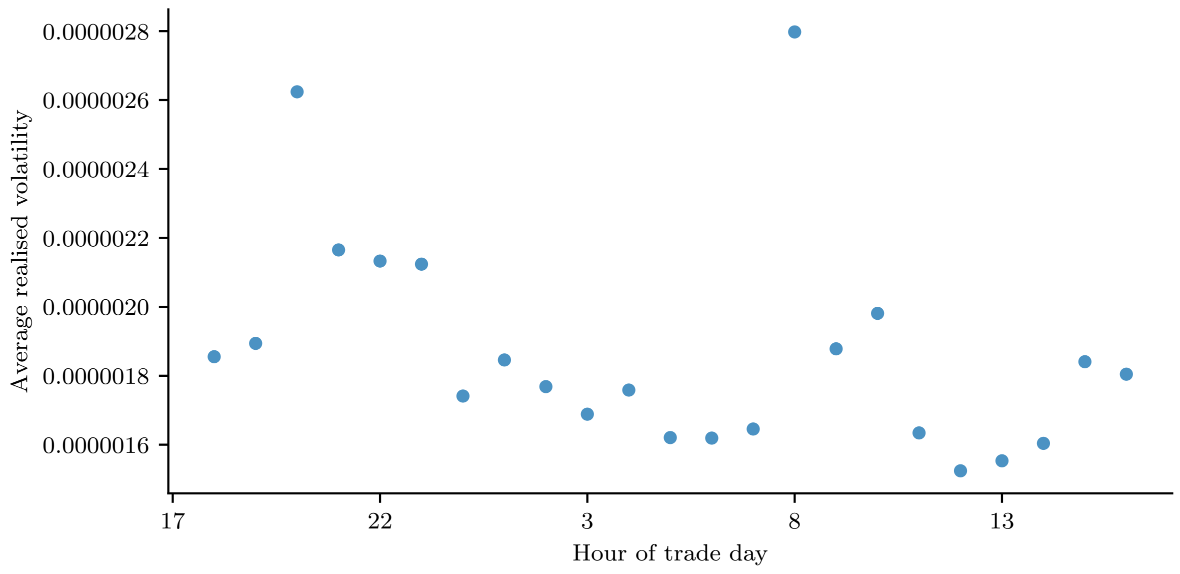

Figure 3.6. Average volatility for each hour of a sample trade day (*EUR/USD 2015-09-02*).

3.3.2 Volatility

An asset's volatility, along with its derived indicators like the volatility ratio, is an important property of any market data. High volatility leads to increased risk in trading an asset, so it is valuable information in a high-frequency trading scenario. It was hypothesised that knowing the volatility might be beneficial in calibrating the filtering hyperparameters. Figure 3.6 shows the hourly realised volatility over a day of trade data, defined as

$$V_n = \sqrt{\sum_{i=1}^{n} \left(\log \left(\frac{p_{i+1}}{p_i} \right) \right)^2} \tag{3.1}$$

over the set of mid-prices $\{p_i\}_{i=1}^n$ of the measured portion of a sample trade day.

We see from comparing Figures 3.5 and 3.6 that there appears to be some inverse relationship between market activity and volatility, which suggests that a higher fraction of the incoming orders during peak market hours have a smaller effect on the mid-price: they are either cancelled or of smaller relative size than orders during off-peak hours. However, this is not a convincing trend as there are multiple outlying hours. As a result, the observation of poor correlation between volatility and other order book characteristics led to the decision to not directly use these valuations in numerically optimising the filter. Both the volatility and order rate remain useful, however, as quick calculations to get a broad look at a large amount of order book data, and as a precursor to further analysis.

Given this holistic analysis of the experimental dataset, we now move onto examining the implementation of the filtering and prediction algorithms, in order to assess their performance on this data.

Chapter 4

Experimental design

This chapter summarises the main features of the software that was written to perform the filtering and prediction, along with the performance metrics and optimisation techniques used to balance prediction accuracy with practical compute time. Particular attention was paid to ensuring all algorithms work in an online manner, so that they could be easily adapted for use in a real-world environment with incoming real-time data. Due to the limited computing power available during this research, the pre-formatted order book data were used in these experiments. However, the code was written so that all simulations could equally be run with streaming tick data, and would give the same results.

4.1 Test platform development

All computing and simulation on the order book data was performed with Python and the SciPy numerical computing library (Jones et al., 2001); see Appendix A for a detailed discussion of the programming considerations when designing the filtering code.

The combined model in Section 2.3.5 was implemented, along with the baseline model and separate models for each modification. Classes were written to fulfil the following requirements: building usable order books from the supplied tick data; generating synthetic order book data; implementing the filtering and prediction algorithms; performing automatic hyperparameter optimisation; analysing performance using various metrics; and generating appropriate graphs. In total, over 1,300 lines of code were written to provide a high performance implementation of the algorithms, with over 2,000 more for supporting tests, data parsing and visualisation routines.

The simulation algorithms were designed to take a snapshot at any point in continuous time and use its included timestamp to calculate the appropriate state propagation update. A burn-in time was used to discard the initial period of results for measurement purposes, as the state and covariance matrices take a short time to settle down from their initialised values.

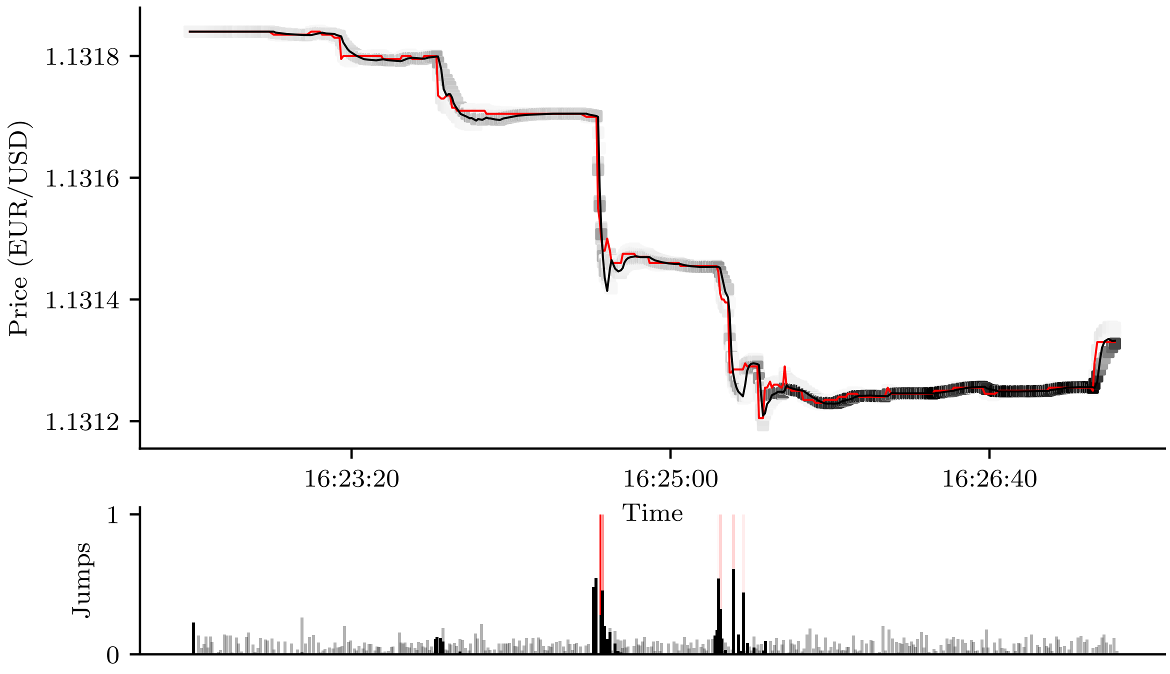

Figure 4.1. Jump detection with baseline model over 5-minute period. Top chart shows observed mid-prices (red), filtered price (black) and particle concentration at that location (grey). Bottom chart shows percentage of particles that jump at that time with lag 0 (grey) and lag 10 (black), along with jump size (brightness of red bars). Reflects the similar chart in Christensen, Murphy, and Godsill (2012).

4.1.1 Jump detection

Before continuing with the implementation of the modifications, the standard jump-enabled model was tested. Figure 4.1 gives a qualitative look at performance of the baseline jump-enabled algorithm over a short period, and shows the ability of the jump detector to react quickly to rapid price changes. There is some over-shoot in the tracking line during jumps, but otherwise the algorithm appears to work well at picking out jumps. Charts like this were used during model construction to assess the sensitivity of the jump detector to what could be deemed actual jumps in price, and to help reduce this overshoot with the extended models. Further plots for the optimised models are shown in Section 5.3.

4.1.2 Effective sample size

One enhancement to the standard particle filtering algorithm is the use of an effective sample size measure. The ability to determine when it is necessary to resample, rather than to do so on every update iteration, saves computing time and reduces the variance added with each step (Doucet and Johansen, 2008). The effective sample size is defined, at time n, as

$$\mathrm{ESS} = \Big(\sum_{i=1}^{N} (W_n^{(i)})^2 \Big)^{-1} \ . \tag{4.1}$$

Although the effective sample size was implemented, and is documented here for completeness, it was found that it was not necessary to use it when generating the final results due to the relatively low number of particles used in testing. If many more particles were used, the ESS could be beneficial to reduce the number of resampling steps, which may be a bottleneck a highly parallel implementation.

4.2 Performance metrics

Performance of the filtered observation data was assessed using both probabilistic and empirical methods. This is essential for both characterising different trends in market data for analysis purposes, and for optimising the models for best performance in a hypothetical real-world trade scenario. The following functions were implemented in a rolling window fashion, as well as cumulatively for measurement over certain pre-defined time periods. This allows the metrics to be useful over long periods of time, and in a real-world trade scenario where more recent results are of greater value than very old results.

4.2.1 Log-likelihood

The use of log-likelihood is theoretically motivated by its presence in the derivation of the particle filter, but it has the issue that the priors must be carefully specified to avoid encouraging over-fitted models.

The log-likelihood of the estimated particle states is used to assess how likely it is that the observed data was generated by the model under consideration. Using the prediction error decomposition given previously in eq. (2.22), the Kalman filter log-likelihood for each observation at time point i, given all previous observations, is

$$\mathcal{L}_K(\theta) = \sum_{i=1}^{N} \mathcal{N}(y_i \mid \mu_{y_i}, C_{y_i}) \ . \tag{4.2}$$

With multiple particles and jumps enabled, the overall log-likelihood is then found by the weighted sum of each individual particle's log-likelihood, multiplied by its normalised weight:

$$\mathcal{L}(\theta) = \sum_{q=1}^{Q} \Big(W_q \sum_{i=1}^{N} p(y_i^{(q)} \mid y_{1:i-1}^{(q)}, \tau_{t_0:t_i}^{(q)}) \Big) \ , \tag{4.3}$$

where $p(\cdot \mid \cdot)$ is the importance sample value that incorporates the nonlinear jump times τ.

4.2.2 Mean square error

The mean square error (*MSE*) is also used to compare forecasts with their real future observed state. Mean square error can never be reduced below the observation noise variance σ^2_{obs}, but it is useful for assessing the robustness of the model. It allows for comparisons when predicting the state at arbitrary future times, which are found by propagating the Kalman filter forward using the current state given the most recent observation. The mean square error is defined over all time points N, when the current state is projected ahead by j time points (which defines the *lag*), by

$$\text{MSE}_j \triangleq \mathbb{E}[e_j^2] = \frac{1}{N-j} \sum_{i=1}^{N-j} (y_{i+j} - \hat{y}_{i+j})^2 \tag{4.4}$$

where $\hat{y}_{i+j}$ is the future predicted price, equal to $G_i \mu_{i+j \,|\, 0:i}$. Both mean square error and the binary prediction accuracy (which follows) are useful because it is possible to change the time that the state is propagated into the future, to optimise the model for performance at different future time frames. This allows analysis of the most useful time scales to be considered, depending on the data source and market conditions. In the experimental results that follow in Chapter 5, this is done for between one and 10 seconds into the future, simulating a moderately high-frequency trade system. Measuring the mean square error on predicted prices rather than directly on filtered prices also helps to prevent overfitting of the model, and encourages generalisation.

4.2.3 Binary prediction accuracy

The goal of using performance metrics is to assist in optimising the algorithm for its practical use, and the metrics given above are designed to indirectly approximate, as closely as possible, the actual performance of the model. Our filtering algorithms do not provide any kind of trading signals, so the performance metrics above can only abstractly measure the results we are interested in from a financial perspective. Since they are so far removed from any real trade algorithms used in practice, we are motivated to design a more realistic method of performance measurement.

One such metric, designed to more closely approximate the results that might be achieved if the filtering model was used as the backend for a real trading system, is the binary prediction accuracy. While this project is not focused on designing actual trading algorithms, one rudimentary algorithm looks at whether the direction of the current trend, when propagated forward, matches that of the most recently observed mid-price at some lag of j time points. It should give some insight as to whether our other performance results could also generate returns in real-world scenarios. Let the binary accuracy score be defined as

$$B_j \triangleq \frac{1}{N}\sum_{i=1}^{N} b_i \tag{4.5}$$

where

$$b_i = \begin{cases} 1\ , & \text{if } (y_{i+j} > y_i \text{ and } \hat{y}_{i+j} > y_i) \text{ or } (y_{i+j} < y_i \text{ and } \hat{y}_{i+j} < y_i)\ , \\ 0\ , & \text{otherwise}\ . \end{cases} \tag{4.6}$$

While this does not provide any kind of confidence metric, the objective should be to achieve $B > 50\%$, suggesting correct trade decisions would be made more often than not.

4.2.4 Inverse CDF method

The inverse cumulative density function method allows us to perform model validation (Pitt and Walker, 1998). This is useful for assessing how well the particle filter is distributing its predictions according to the observation inputs. The method works as follows: first, samples are drawn uniformly from the filter's mixture of Gaussians (using each particle p's mean and variance, and its normalised weight W) at time i. This state distribution is given by

$$\hat{y}_i \sim \sum_p W_p \mathcal{N}(\hat{y}_i \mid \mu_{y_i}^{(p)}, C_{y_i}^{(p)})\ . \tag{4.7}$$

There is no closed form expression for the inverse CDF of this distribution, but since the CDF of a Gaussian mixture is simply the sum of its weighted component CDFs,

$$\text{CDF}\Big(\sum_i \pi_i f_i(\cdot)\Big) = \sum_i \pi_i F_i(\cdot)\ , \tag{4.8}$$

where $F(\cdot)$ is the CDF of $f(\cdot)$, we are able to find the inverse CDF of eq. (4.7) numerically. In practice, this can be done with a simple bisection search method (Dagpunar, 1989).

With a sample drawn using a uniform seed over $[0, 1]$ as an input to this inverse CDF, we can now obtain the cumulative probability for all values less than that sample, using a different Gaussian distribution centered on the observed price y_i and using the observation standard deviation σ_{obs}. That is, with the uniformly sampled price values $\hat{y}_i$ from the particle distribution, we find the cumulative probability of $\hat{y}_i$ according to the distribution $\mathcal{N}(\hat{y}_i \mid y_i, \sigma_{\text{obs}}^2)$, which is

$$\Phi\Big(\frac{\hat{y}_i - y_i}{\sigma_{\text{obs}}}\Big) \tag{4.9}$$

where $\Phi(z)$ is the standard normal CDF.

By sampling these values using the inverse CDF method at each time point, we can then plot them as a histogram, which we expect to be roughly uniform (i.e. flat between 0 and 1) if the model distribution does in fact approach the true distribution of the price observation's underlying state.

Table 4.1. Hyperparameters used across all models, and their typical approximate values.

	Description	**Relationship to data**	**Value**
σ_{obs}	Observation noise	Std. dev. of observed/moving average error	$\sim 10^{-5}$
σ_0	Trend (x_2) diffusion volatility	Some fraction of σ_{obs}	$\sim 10^{-5}$
θ_0	Trend (x_2) mean reversion coefficient	Manually selected	~ -0.02
σ_v	Vol. (ϕ_i) diffusion volatility	Some fraction of σ_{obs}	$\sim 10^{-9}$
θ_v	Vol. (ϕ_i) mean reversion coefficient	Manually selected	~ -0.01
σ_e	Extra trend (x_3) diffusion volatility	Some fraction of σ_{obs}	$\sim 10^{-6}$
θ_e	Extra trend (x_3) mean reversion coefficient	Manually selected	~ -0.02
μ_J	Jump size mean	Assume unbiased price movement	0
σ_J	Jump size std. dev.	Some fraction of σ_{obs}	$\sim 10^{-5}$
λ_J	Jump rate (seconds^{-1})	Depends on desired performance	~ 0.1
N	Number of particles	0 for Kalman, > 0 for particle filter	50
k	Scale parameter for initial covariance	$>$ average movement between observations	$\sim 10^{-8}$
T	Number of threads	Manually selected	3

4.3 Hyperparameter selection

Tests were conducted with different hyperparameters, which are summarised in Table 4.1 along with their relationship to the trade data. The values given are order-of-magnitude and vary slightly according to the data, as discussed below. In order to accurately account for changing volatility and order volume as market conditions change over time, tests were run with the parameters computed automatically and updated regularly to keep up with changing conditions. This necessitates using additional hyperparameters to control the rolling window size of the performance metrics. These would be set manually in practice, but for simplicity in the experiments the data was separated into hourly blocks, each analysed as a whole unit.

When providing an objective function to the optimisation routines, experiments were run with this objective being to minimise either the mean square error, the negative binary accuracy score, or the product of both, MSE $\times(1 - \text{BIN})$, as a combined objective. This proved to be easier to optimise than the log-likelihood in practice.

4.3.1 Tree-structured Parzen estimators

The goal of constant-state hyperparameter optimisation is to discover the combination of parameters for any given set of data that maximises the performance of the filtering and prediction algorithms. One method for achieving this is a simple grid search, where various combinations of parameters are tested over a fixed grid until the optimum set is found. However, a more efficient method, especially for large numbers of parameters as we are dealing with, is the tree-structured Parzen estimator. It uses a stochastic search method that has been shown to find the optimum set more efficiently than grid search

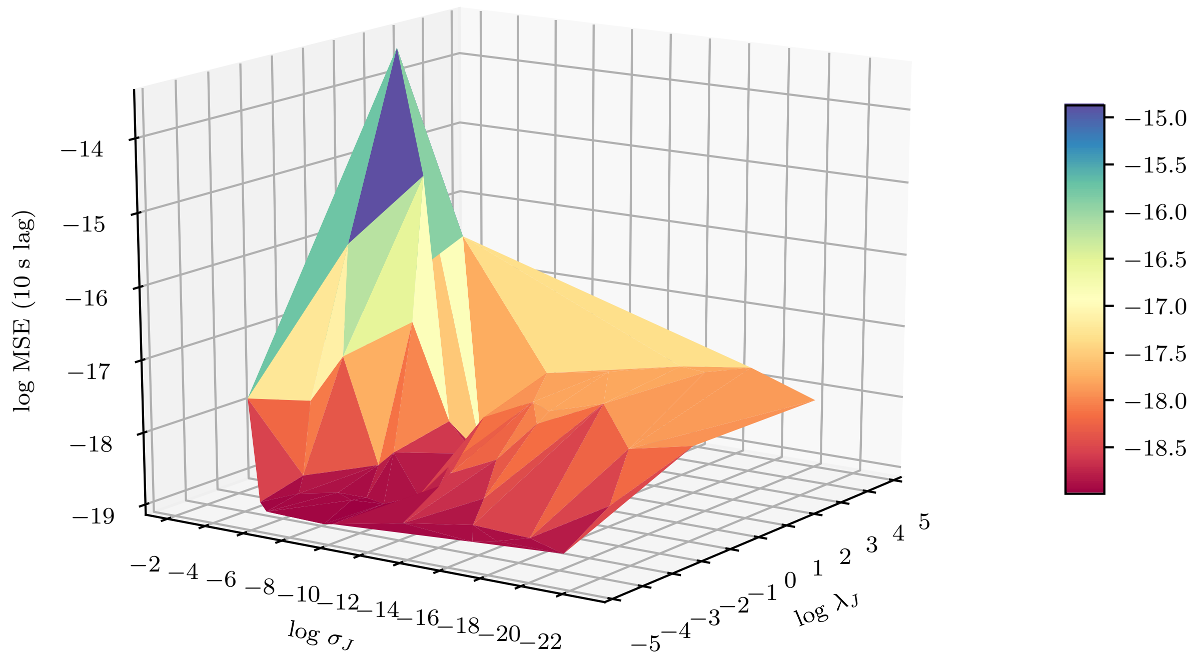

Figure 4.2. Parameter optimisation: log-log plot of results from varying the jump size standard deviation σ_J, and jump rate λ_J, compared to mean square error with 10 s lag.

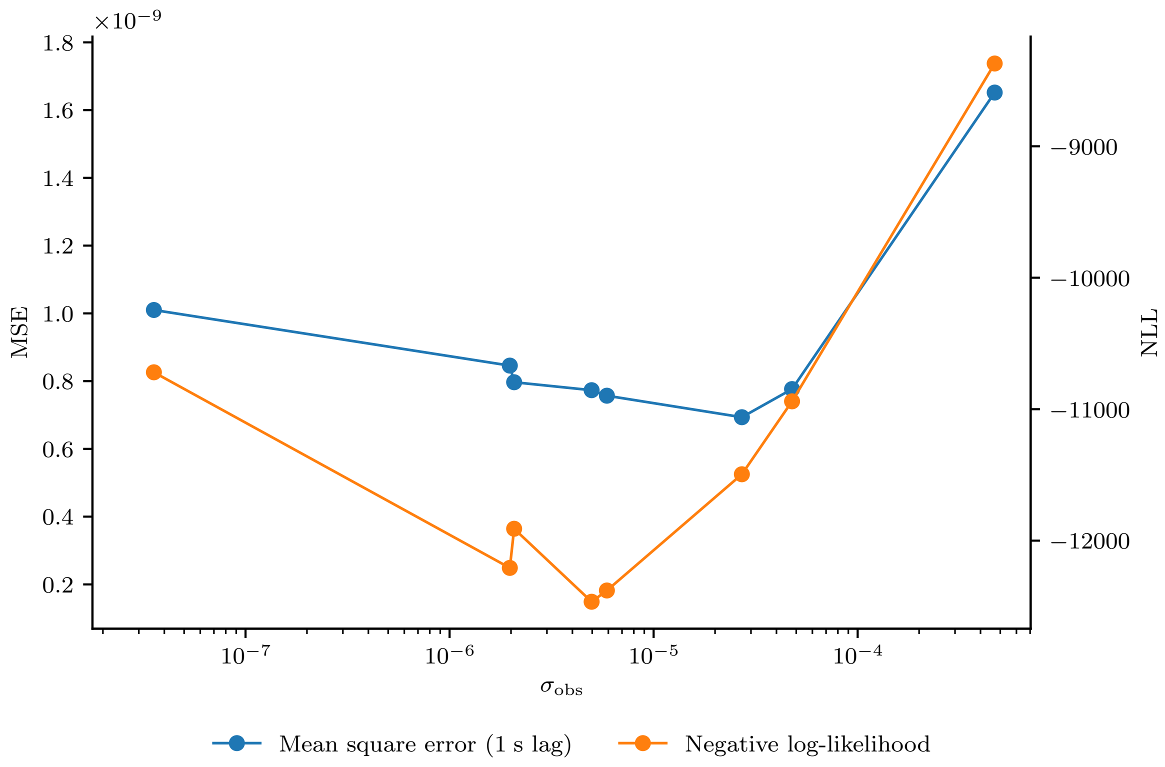

Figure 4.3. Parameter optimisation: semi-log plot of results from varying the observation noise parameter σ_{obs}, compared to both mean square error and negative log-likelihood.

(Bergstra et al., 2011).

Figures 4.2 and 4.3 show some examples of the results from optimising hyperparameters using tree-structured Parzen estimators. Figure 4.2 optimises σ_J and λ_J together as a multidimensional surface against the mean square error at a 10-second lag, while Figure 4.3 shows the optimisation of σ_{obs} and has a minimum at $\sigma_{\text{obs}} \approx 10^{-5}$. The fact that both the log-likelihood and mean square error plots are approximately convex over this range, and are both minimised by roughly the same hyperparameter value, is encouraging for the use of lagged mean square error as a complementary performance metric to log-likelihood.

4.3.2 Adapting to changing market conditions

Since the filtering algorithm is designed to run in an online manner, hyperparameters should also be able to update over time to adapt to changing market conditions. In practice, this could be achieved by running the Parzen tree estimator detailed above on small blocks of order book data, using the previous block's results to update the hyperparameters for the ongoing algorithm. This would require very little additional computing power, so would likely be an appropriate and straightforward solution.

To demonstrate this, the above method was applied to the optimisation of all relevant hyperparameters at a one-second lag, for successive one-hour blocks of trade data. Plots of these optimisations are omitted for brevity; see Appendix B for tables listing the results of this optimisation, over the Pareto frontier combining different hyperparameters. This highlights their change over time as market conditions vary.

4.3.3 Controlling update frequency

As shown in Figure 3.5, the raw trade data used in testing can average upwards of 100 events per second and, in a real-world environment running on a larger system for a popular market pair, this may increase even further into the thousands. As such, it is important to be able to maintain practical speed levels of the trade algorithms, even during periods of unusually high market activity. Therefore, included in testing was a way to limit the update frequency of the state-space model to an arbitrary value. Tests were run to find how far this frequency could be reduced while still maintaining appropriate accuracy, in combination with the other variables; results are discussed in the next chapter.

All model features have now been outlined and implemented in code, and hyperparameters have been optimised using the methods described above. We now proceed to highlighting the key results that arise from analysing their performance.

Chapter 5

Results and discussion

This chapter considers the effects of the modifications proposed in Chapter 4 to the full combined model that was detailed in Section 2.3.5. First, results are presented using the previously defined performance metrics for the model with and without jumps enabled, and with different numbers of particles used. Results are then given for the different functions of order volume included in the state transition matrix, and also with and without the update frequency limited to some fraction of the performance time frame. Finally, several more extensive tests are performed on the final optimised model with all of these modifications in place. These results are discussed in the context of likely real-world trade performance, compared with the unoptimised standard model.

Note that, for reproducibility, and to ensure clarity in qualitative plots, all of these results are generated with the manually selected 10-second optimised hyperparameters listed in Table B.3.

5.1 Particle filter optimisation

Several significant results of the experiments are highlighted below, and details are given for many of the avenues explored while analysing the new particle filters that were given in Chapter 2. Unless otherwise specified, all results were generated from the previously introduced EUR/USD tick dataset.

5.1.1 Discussion

Figure 5.1 shows the results for testing with jumps enabled and disabled, and with different numbers of particles being used to estimate the jumps. These results demonstrate a small but consistent improvement in log-likelihood with the enabling of the particle filter in addition to the standard linear Kalman filter, which is corroborated by a similar improvement in mean square error. This is to be expected since we saw in Figure 4.1 that the filter is able to react quickly to sudden changes in price direction with the use

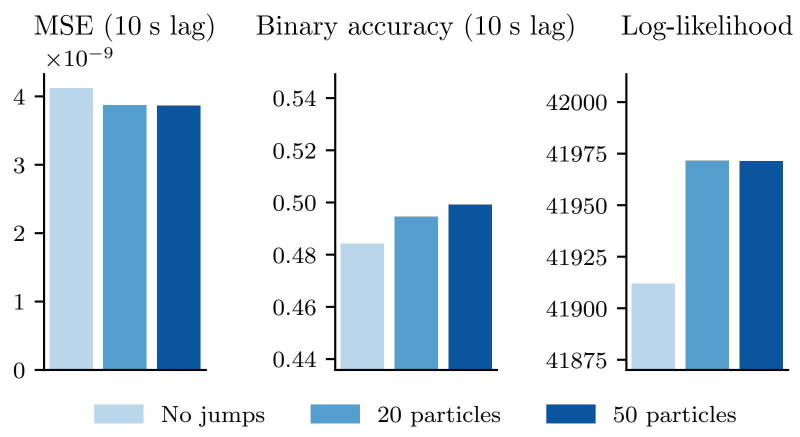

Figure 5.1. Performance comparison: models with jumps disabled or enabled with different particle numbers, measured over one-hour trade period (*EUR/USD 2015-09-02 12:00:00-13:00:00*).

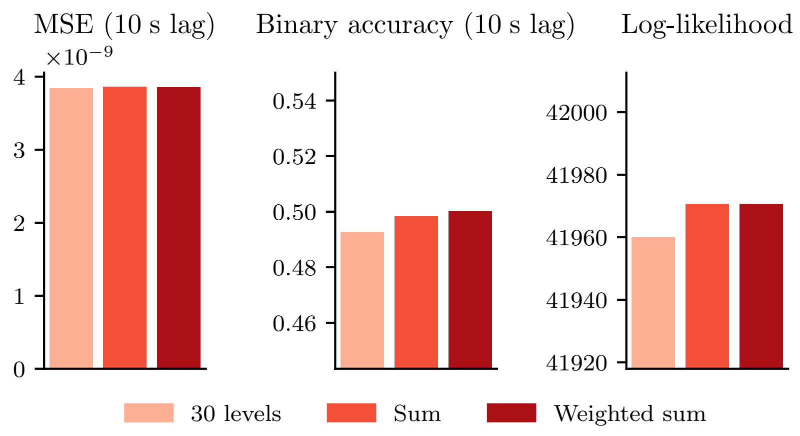

Figure 5.2. Performance comparison: models with different volume functions, measured over one-hour trade period (*EUR/USD 2015-09-02 12:00:00-13:00:00*).

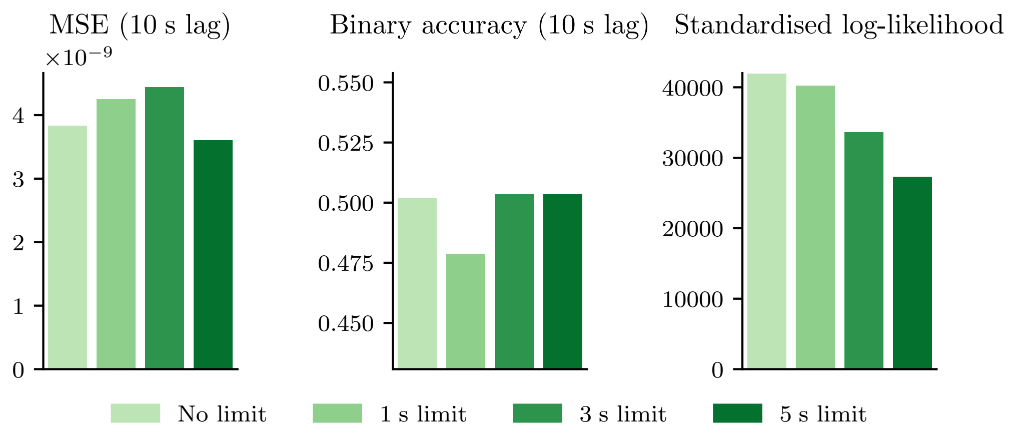

Figure 5.3. Performance comparison: models with different update frequency limits, measured over one-hour trade period (*EUR/USD 2015-09-02 12:00:00-13:00:00*). Note that log-likelihood here is scaled to account for the different numbers of iterations.

of the particle filter. There is no significant difference in log-likelihood or mean square error between using 20 and 50 particles, suggesting that at this level of accuracy there is no benefit to higher particle counts within the same order of magnitude, and that 20 particles are enough to explore all relevant times at this update rate. However, the binary accuracy result does improve slightly with the addition of extra particles, as it does when going from the Kalman filter to the particle filter. This difference is small, though, and given the lack of evidence that higher particle numbers improve performance against the other metrics, we will continue to use 20 particles as a good middle ground between speed and accuracy.

Figure 5.2 shows the results for testing of different functions of the order volumes at price levels near the mid-price (30 levels was found to be the optimum number in testing for the linear model with n levels). It shows a marginal improvement, across every performance metric, from using the more simple sum functions instead of the separate volume levels. This suggests that using individual levels not only adds unnecessary computing expense, but also causes overfitting, and is harder to tune compared to models that use fewer parameters to account for this data. Using the exponential weighted sum results in marginally better performance on binary accuracy than the standard sum, but is no better on mean square error and log-likelihood. Therefore, we will continue with the simple sum as it eliminates the extra parameter c that would require additional tuning (for this result, the best performance was found with $c = 0.035$).

Figure 5.3 shows the results for testing of different update frequency settings, with the goal of eliminating unnecessary iterations and improving resource efficiency. The standardised log-likelihood decreases with each increase in the frequency update time. This is expected since log-likelihood considers only the fit of the filtered data, and fewer updates means a worse fit on average. However, the other metrics evaluated at a 10-second lag show interesting results for one, three and five-second update intervals. The one-second limit appears to be worse than the immediate update model, but the five-second limit returns to the original performance level, and the three-second limit is worse in mean square error but equal in binary accuracy. These conflicting results are inconclusive, and suggest that the information present in the full snapshot updates (which are updated on the order of every one second anyway) is likely to be the most consistent for further analysis. Therefore, we proceed without using the rate limit on snapshot data, but will return to the consideration of update frequency when analysing the high-frequency tick data in Section 5.4.

5.2 Simulation speed

It is important to assess the trade-off in accuracy and compute time required for each different modification of the filtering algorithm. Results for measurements of the algorithm's

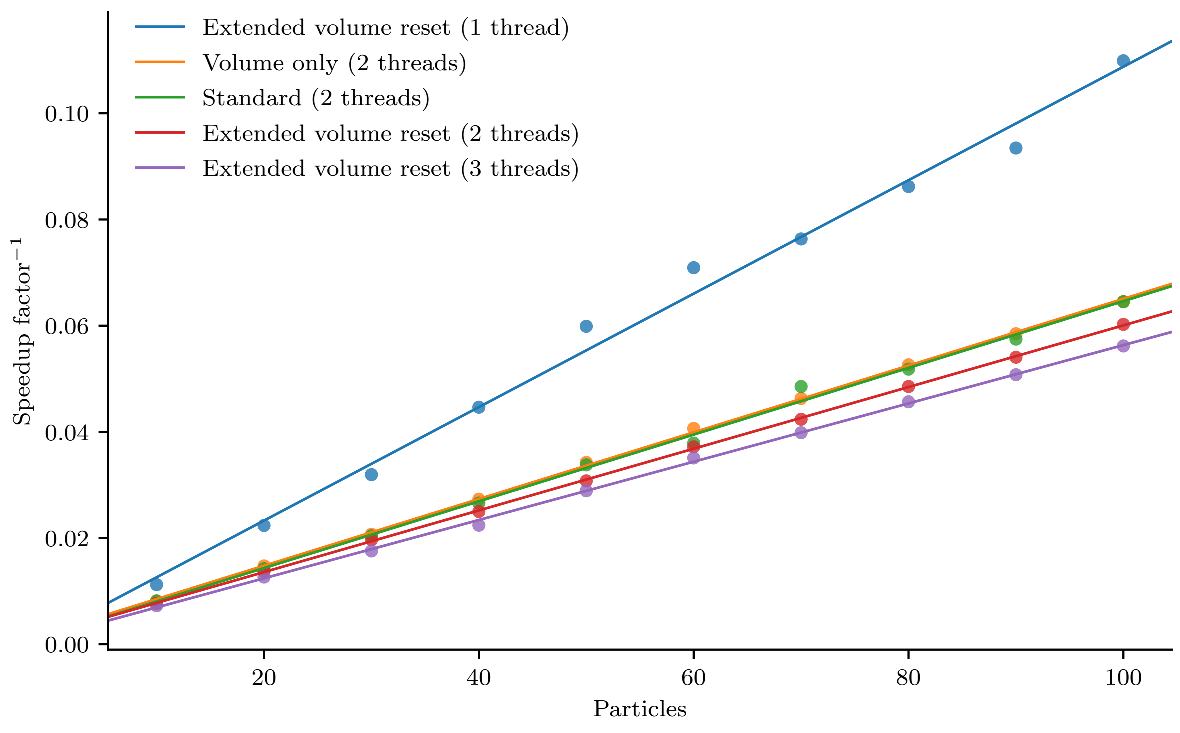

Figure 5.4. Simulation speed for filtering algorithm only, with increasing numbers of particles (run on *EUR/USD 2015-09-02 17:00:00-18:00:00*). The y-axis is the inverse of the runtime divided by the length of the actual period of trade data used.

runtime with different options are shown in Figure 5.4.

5.2.1 Discussion

As expected, we see a linear relationship between the number of particles used and filtering runtime for each model type used, when the number of threads is kept constant. The addition of extra dimensions to the state matrices with the *volume* model makes very little difference to runtime over the *standard* model, because only two variables are used to store the weighted sum of the volumes (as described in Section 2.3.4). The *extended volume reset* model is slightly faster than both the *standard* and *volume* models across all particle numbers, due to the state reset modification reducing the average computational load when calculating the matrix exponentials on each iteration.

Increasing the thread count results in a corresponding speed increase, although we see quickly diminishing returns above two threads due to the tests being run on a dual-core CPU, and the new particle threads competing with the main controller thread. Based on these experiments it was decided that the *extended volume reset* model with 2 or 3 threads is appropriately fast for further use in testing.

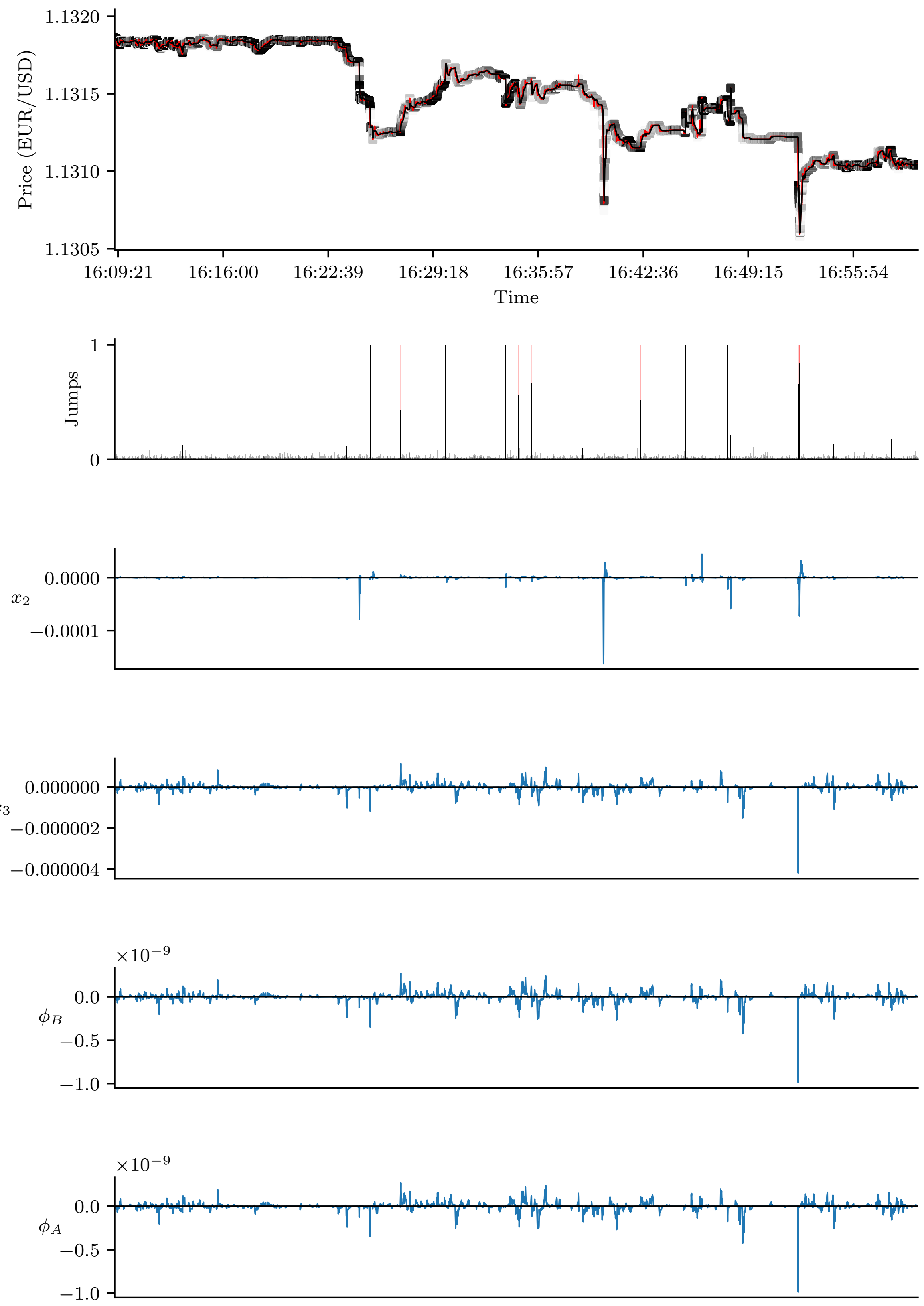

Figure 5.5. Full tracking plot with jumps and state means over a one-hour trade period (*EUR/USD 2015-09-02 16:00:00-17:00:00*), showing particle weights (grey), observed price (red) and filtered price (black). The x_2 plot is the first trend term, x_3 is the extended trend term, and ϕ_B and ϕ_A are the volume terms below and above the mid-price, respectively.

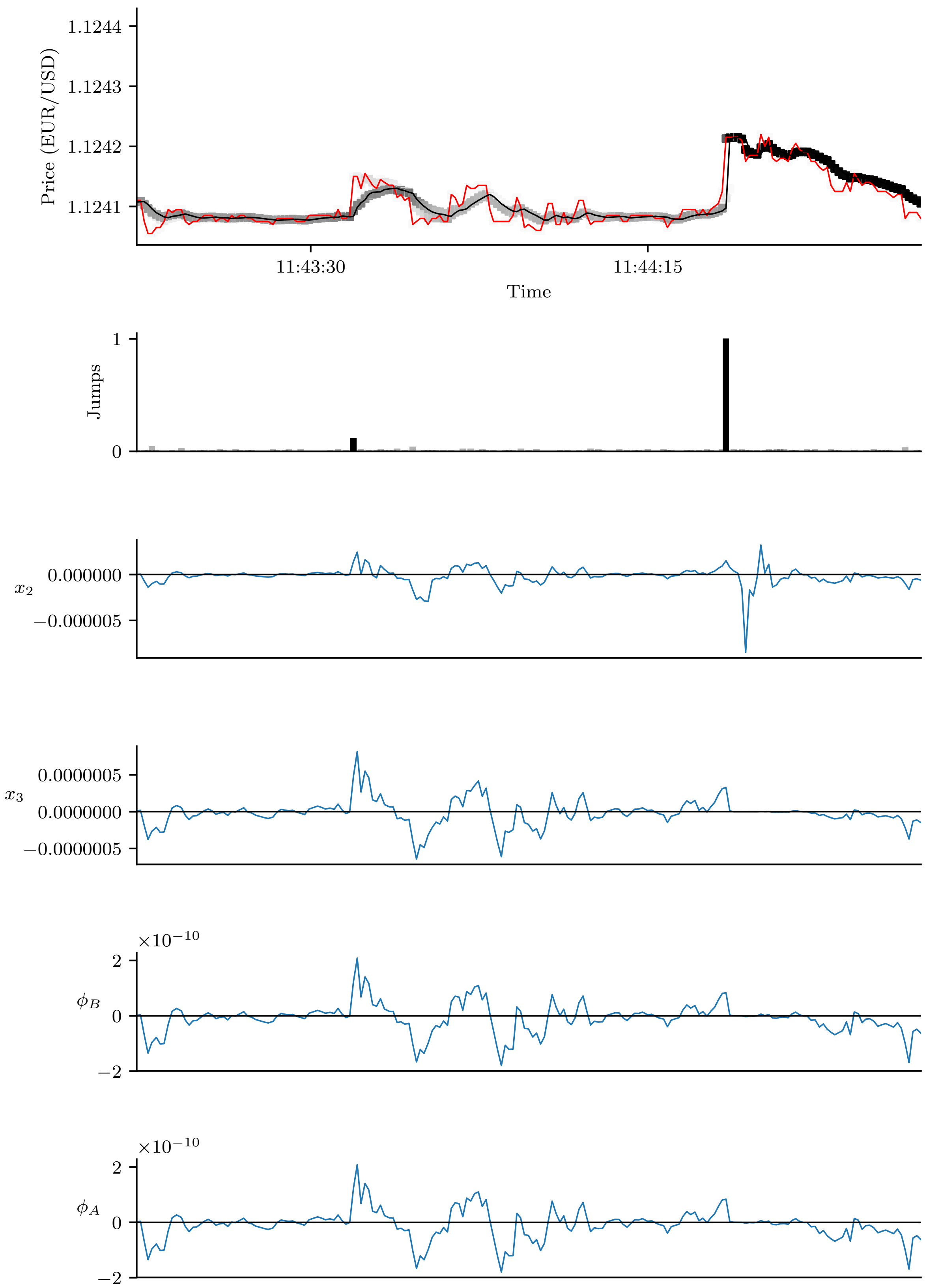

Figure 5.6. Full tracking plot for a short time with the combined model, showing tracking characteristics of each state parameter during times of small volatility and a sharper price jump. The axes, hyperparameters and data/trade date are identical to Figure 5.5.

5.3 Assessing the combined model

The final model was chosen based on the results considered above. It uses the full combined extended particle filter with the summed order book volumes as additional parameters, and the state reset on jumps enabled. Figures 5.5 and 5.6 show the qualitative performance of the best model on one hour of trade data, with the jump locations and the means of each state element during tracking. For a quantitative comparison of results achieved by this fully optimised model with the other particle filter algorithms, see the charts in Figure 5.7. These were generated using the models described above for a full day of one-hour trade periods, and represent the mean best performance with a 10-second lag for the mean square error, binary accuracy and log-likelihood metrics. Results from another metric, the plotting of an inverse CDF histogram, are shown in Figure 5.8.

5.3.1 Discussion

Jump tracking, shown in the qualitative results in Figure 5.5, enables the filter mean to model the sharp changes in price direction, with the colour of the bars accounting for the magnitudes of the jumps. The additional axes show the trend, extended trend and volume parameter means above and below the mid-price. The first trend term has clear maxima in the jump positions, and the peak magnitude decreases from the first trend term through to the volume terms. This suggests that the first trend term makes up the majority of the predicted price movement, but the fact that the later terms are non-zero means that they account for additional dynamics in the order book evolution, and serve to increase overall accuracy compared to the standard single trend model. This is also shown by the larger and more consistent non-zero areas of the extended trend and volume terms. These results support the theory that smaller movements are well-modelled by order book volume and the extended trend term, and larger movements around jumps are best modelled with the first trend term.

Figure 5.6 highlights the difference between the tracking states over a smaller time scale. Here we see more clearly that the first trend term is only significantly non-zero around the price jump, and is small for all other price movements. In contrast, the extended trend term and both the volume terms track the smaller price movements and not the jump. They are generally similar to each other, although the volume states are emphasized compared to the extended trend state in some places, such as at the very beginning and end of the displayed time period. These differences support the idea that the volume data provides some benefit to tracking over just the state trend terms.

Indeed, the most complete model, which combines the state extension, jump resetting and volume data, is shown to be the best performing across all three performance metrics assessed in Figure 5.7. For the mean square error, the extended reset model improves

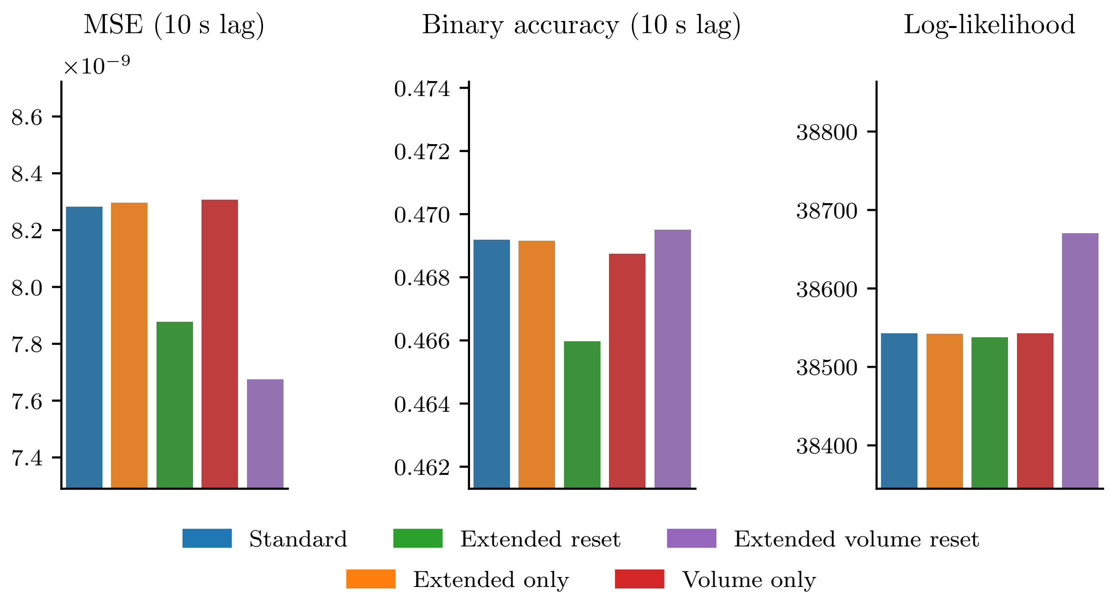

Figure 5.7. Results showing a comparison of filtering and prediction over one full day of snapshot data (*EUR/USD 2015-09-02*) for models with parameters optimised for 10-second lag (all with jumps enabled, and sum volume function where applicable).

significantly over the standard, extended and volume models on their own, and the combined extended reset model with volume improves this further. With the volume-only model, it was possible to replicate the results of previous work (Jerjian, 2017) but the performance gains were only evident within a small range of parameter values, and the model displayed poor robustness to different parameter choices that otherwise improved performance for the other models. However, when combined with the extended state and jump reset modifications, performance improved significantly. This suggests that there is useful information to be gained from the volume levels, but that it must be carefully implemented to avoid overfitting to the data.

The combined model also gives the best performance in binary accuracy and log-likelihood. Note that while all models give a binary accuracy of less than 50%, there is still a useful comparison to be drawn between the different models, and this validates the results of the other metrics. Although the improvements are smaller, the combined model displays an increase of 0.07% in binary accuracy and 0.33% in log-likelihood relative to the baseline standard model, in addition to the 7.34% improvement over the standard model in mean square error. We can be confident in these results as they were generated over an entire trade day – approximately 150,000 snapshot updates – and still display a notable improvement when averaged over this many trades.

Figure 5.8 shows the histogram generated using the inverse CDF method detailed in Section 4.2.4, with each time point from an hour of trade data used to generate samples according to the particle distribution at that time. We expect the histogram to be roughly

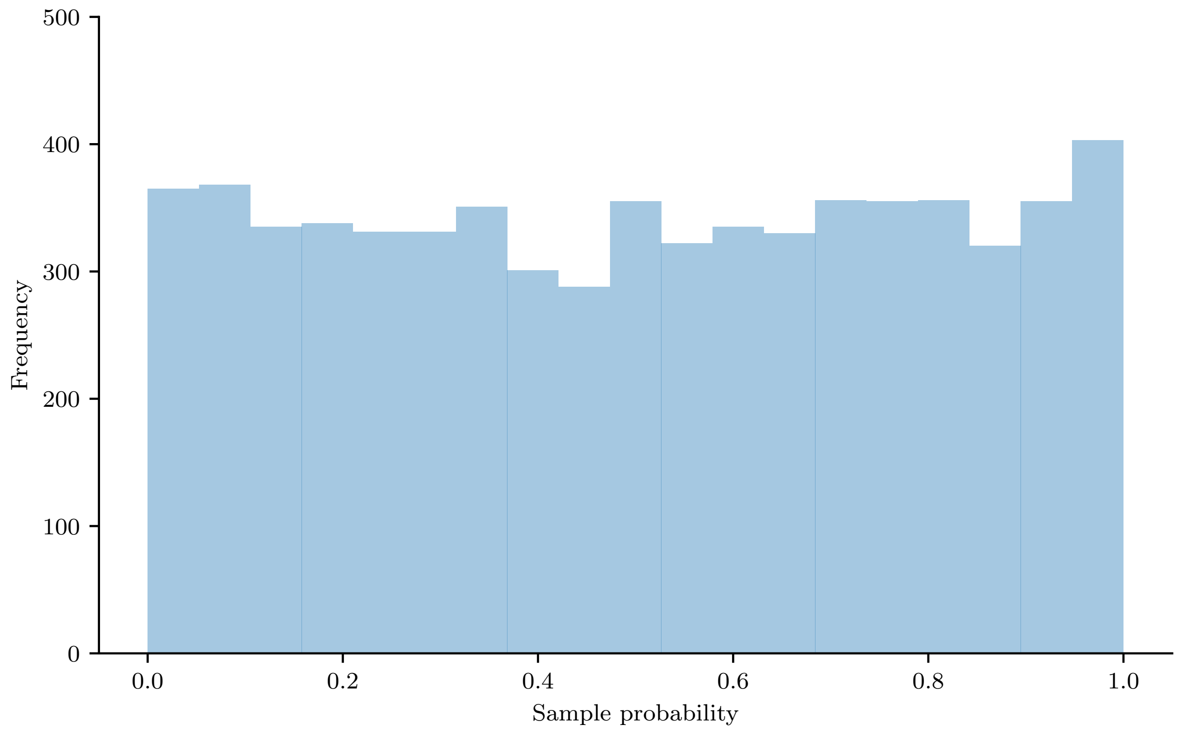

Figure 5.8. Inverse cumulative density function histogram (produced with snapshot data for *EUR/USD 2015-09-02 12:00:00-13:00:00*).

flat, and indeed it does show fairly good uniformity, suggesting the modified particle filter is still well suited to modelling the price variation. There is a small increase in the number of points with probability close to 0 or 1, indicating that the particle filter is introducing some additional volatility in its model of the price variance. This result is still encouraging, however, and shows that the particle filter is performing as expected even with the modifications introduced.

5.4 Performance on high-frequency tick data

The results above were generated using snapshot data, in order to prioritise testing as many hours of data as possible, rather than generating very slow results on fewer hours of tick data. However, it was shown in Figure 3.1 that the tick data contains many more high-frequency movements that are not present in the snapshots. Therefore, we conclude this results chapter by looking at how these algorithms scale for higher-frequency data, and consider a direct comparison in performance between tick and snapshot data. Consider Figures 5.9 and 5.10 which show particle tracking performance of the optimised combined filtering algorithm on two small sections of order book data.

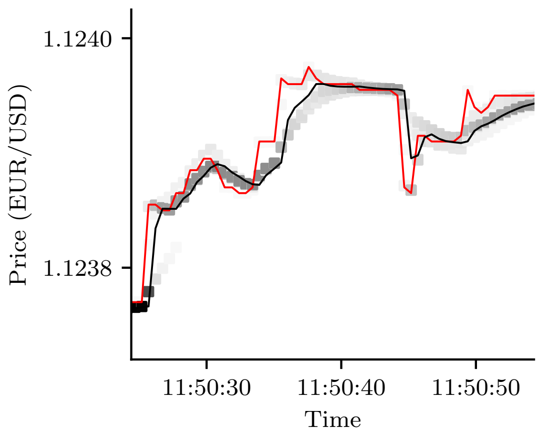

Figure 5.9. Particles (darker colour for higher weight), filter means (black line) and observed snapshot data (red line) for ∼1 s update time.

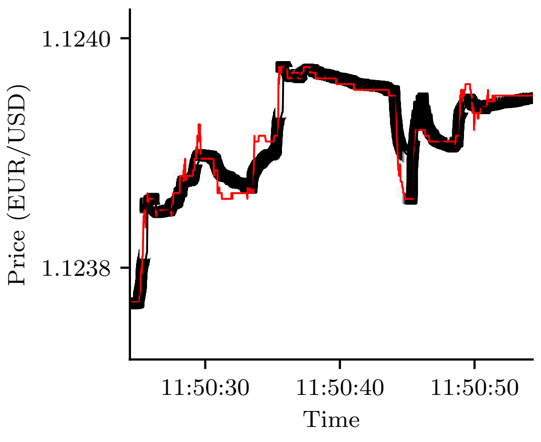

Figure 5.10. Particles (darker colour for higher weight), filter means (black line) and observed tick data for full rate incoming order events.

5.4.1 Discussion

The two figures show the contrast in accuracy between using the lower-frequency snapshot and higher-frequency tick data for filtering. Due to there being fewer updates available, the filter in figure Figure 5.9 shows visibly fewer particles present, and this demonstrates the issue of the resulting increase in uncertainty. The number of different paths taken by the particles here exemplifies the multimodality that can be captured by the particle filter. This also means that, although we see some particles tracking the correct (red) line, they are light grey in colour meaning the filter has assigned them a low weight, and the majority of the weight is concentrated in particles that take the smoother (black) line with poor tracking. Therefore, the jump detection which the particle filter is designed to perform is not working correctly, due to a lack of sufficiently many sequential updates. As a result, the high-frequency price movements are lost in the filtered information.

If we wish to make use of these high-frequency movements, then using the raw tick data is essential. When given the entire stream of tick data, Figure 5.10 shows much more accurate tracking of this information. The particles are darker in colour, demonstrating increased confidence in the price location and allowing these jumps to be detected more reliably. If we were attempting prediction at one second ahead, the filter on the right would perform better with its shorter inter-update time compared to the approximately one-second arrival time of the snapshots on the left.

This concludes the experimental results. We have seen the testing of all previously detailed enhancements to the particle filter, and their associated performance improvements, as well as a qualitative look at the characteristics of these filters on different types of real data.

Chapter 6
Conclusions

6.1 Main findings

In this work, the Kalman and particle filters were successfully implemented to replicate the results of Christensen, Murphy, and Godsill (2012), and the volumetric model given by Jerjian (2017). It was found that it was difficult to achieve good results with the separate volume elements used originally. Instead, the use of simpler sum functions of the volumes was found to be more robust and gave better results in almost all circumstances.

Derivations were completed for the extended trend state model and the trend resetting jump model. A combined model was produced using the best performing elements of each of these, which demonstrated improved tracking accuracy compared to previous best results. This was confirmed over several different performance metrics, both quantitative and qualitative. Analysis of the different types of order book data was conducted, and it was also found that using high-frequency tick data can allow more price jumps to be detected, especially at shorter time scales.

In addition to the model implementation, consideration was given to the importance of hyperparameter optimisation for adequate performance of the algorithms. The variation in optimal hyperparameter values over time was shown, motivating a need for continually assessing and updating these values in a real-world situation. The tree-structured Parzen estimator was shown to be an appropriate algorithm for finding these hyperparameters.

During simulations, the extended reset models were found to be faster compared to the standard model, and a speed increase was also seen when using multiple threads for processing. These results were generated with a comprehensive particle filter software library, developed in Python.

The results demonstrated in this work are a promising look at the flexibility of particle filter-based models, and of Bayesian statistical inference more generally in extracting diverse, useful signals from trade data and limit order books. While we looked at the theoretical benefits in detail, results also demonstrated the importance of considering the trade-offs that must be made between improved accuracy and compute time. These are likely to become increasingly important as more activity enters the markets at ever-shorter time frames, and the drive for more accurate filtering and prediction methods continues.

6.2 Limitations and future work

This work is limited in that it was not possible to test using many different sources of data. A goal of future work extending this research could be to examine the versatility of these models, not just to foreign exchange trade data, but to other types of limit order books, including stocks and derivatives. It would also be beneficial to implement a more comprehensive mock trading algorithm to backtest the filters in a more realistic scenario, although this would make it necessary to consider more complex trading effects like slippage that are absent from any simulated test.

Considering the model itself, further analysis of the complex patterns arising from order posts and cancellations in the high-frequency tick data, and how this affects the price evolution, would be useful for future research. While high-frequency effects were considered here, it is possible that methods for tracking order modifications and cancellations could also be incorporated directly into the particle filter, rather than indirectly measured through the state of the order book. For example, each order in the tick data has a unique identifier that is not currently used. Tracking of this identifier would enable linking order cancellations with the original order time, something which has not yet been considered. This would, however, likely result in a significant increase in computing power required to conduct this more complex analysis in real time.

Another area that could be explored is looking into different resampling schemes and the use of the effective sample size if large numbers of particles are used. These methods include multinomial resampling, stratified resampling, systematic resampling and residual resampling (Hol, Schön, and Gustafsson, 2006). Different extensions to the particle filter, like the auxiliary particle filter (Pitt and Shephard, 1999), could also be examined for their performance relative to the variable rate particle filter used here.

The field of high-frequency trading is changing rapidly, and it is possible that the market inefficiencies that enable exploiting these results for profit have been reduced over time such that these methods would no longer be feasible to use in a real-world scenario. Nonetheless, these filtering methods remain important in their ability to simplify, extract statistics and remove noise from various different kinds of data. It is likely that doing so with the ideas developed here – multiple trend terms, resetting trends on jumps, and incorporating non-standard data sources into the transition function – can still provide benefits in other areas, even where the exact form of the limit order book is not used.

References

Andersen, T. G. and T. Bollerslev (1997). “Intraday periodicity and volatility persistence in financial markets”. In: *Journal of Empirical Finance* 4.2-3, pp. 115–158.

Bao, W., J. Yue, and Y. Rao (2017). “A deep learning framework for financial time series using stacked autoencoders and long-short term memory”. In: *PLoS ONE* 12.7.

Bergstra, J. et al. (2011). “Algorithms for Hyper-Parameter Optimization”. In: *Advances in Neural Information Processing Systems (NIPS)*, pp. 2546–2554.

Blackwell, D. (Mar. 1947). “Conditional Expectation and Unbiased Sequential Estimation”. In: *The Annals of Mathematical Statistics* 18.1, pp. 105–110.

Cappé, O. (2011). “Online Expectation-Maximisation”. In: *Mixtures: Estimation and Applications*, pp. 1–53.

Christensen, H. L., J. Murphy, and S. J. Godsill (Aug. 2012). “Forecasting High-Frequency Futures Returns Using Online Langevin Dynamics”. In: *IEEE Journal of Selected Topics in Signal Processing* 6.4, pp. 366–380.

Cont, R. (2011). “Statistical Modeling of High Frequency Financial Data: Facts, Models and Challenges”. In: *IEEE Signal Processing* 28.5, pp. 16–25.

Cont, R., S. Stoikov, and R. Talreja (2010). “A Stochastic Model for Order Book Dynamics”. In: *Operations Research* 58.3, pp. 549–563.

Dagpunar, J. S. (1989). “An Easily Implemented Generalised Inverse Gaussian Generator”. In: *Communications in Statistics - Simulation and Computation* 18.2, pp. 703–710.

Doucet, A. and A. M. Johansen (2008). “A Tutorial on Particle Filtering and Smoothing: Fifteen years later”. In: *Oxford Handbook of Nonlinear Filtering* 4.

Frigola-Alcalde, R. (2015). “Bayesian Time Series Learning with Gaussian Processes”. PhD thesis.

Gal, Y. (2016). “Uncertainty in Deep Learning”. PhD thesis.

Galanis, G. et al. (2006). “Applications of Kalman filters based on non-linear functions to numerical weather predictions”. In: *Annales Geophysicae* 24.10, pp. 2451–2460.

Godsill, S. J. (2007). “Particle Filters for Continuous-time Jump Models in Tracking Applications”. In: *ESAIM: Proceedings* 19, pp. 39–52.

Gordon, N., D. Salmond, and A. Smith (1993). “Novel approach to nonlinear/non-Gaussian Bayesian state estimation”. In: *IEE Proceedings F Radar and Signal Processing* 140.2, pp. 107–113.

Hol, J. D., T. B. Schön, and F. Gustafsson (Sept. 2006). “On resampling algorithms for particle filters”. In: *NSSPW - Nonlinear Statistical Signal Processing Workshop 2006*. IEEE, pp. 79–82.

Intel (2007). “Intel ® Math Kernel Library Reference Manual”. In: *Numerical Algorithms* March, p. 3056.

Jerjian, M. (2017). “Simulation and Inference for Limit Order Books in High-Frequency Finance”. Master’s project.

Jones, E. et al. (2001). *SciPy: Open Source Scientific Tools for Python.*

Papanicolaou, A. (Apr. 2015). *Introduction to Stochastic Differential Equations (SDEs) for Finance.*

Pendleton, S. et al. (2017). “Perception, Planning, Control, and Coordination for Autonomous Vehicles”. In: *Machines* 5.6.

Pitt, M. K. and N. Shephard (1999). “Filtering Via Simulation : Auxiliary Particle Filters”. In: *Journal of the American Statistical Association* 94.446, pp. 590–599.

Pitt, M. and S. Walker (1998). *Marginal Construction of Time Series With Application To Volatility Models.*

Planitz, M. and E. Anderson (1995). “LAPACK Users Guide”. In: *The Mathematical Gazette* 79.484, p. 210.

Prewitt, M. (2012). “High-frequency trading: should regulators do more?” In: *Michigan Telecommunications and Technology Law Review* 19.1, pp. 131–161.

Rao, C. R. (1945). “Information and accuracy attainable in the estimation of statistical parameters”. In: *Bulletin of the Calcutta Mathematical Society* 37, pp. 81–91.

Särkkä, S. T. (2006). “Recursive bayesian inference on stochastic differential equations”. PhD thesis, pp. 1–248.

Turner, L. and C. Sherlock (2013). *An Introduction to Particle Filtering.*

Van Loan, C. F. (1978). “Computing Integrals Involving the Matrix Exponential”. In: *IEEE Transactions on Automatic Control* 23.3, pp. 395–404.

Appendix A
Programming considerations

A significant part of the project was concerned with the issue of writing code that ran fast enough across the large volumes of data used. Since an important result was that extending the state-space model to further dimensions can improve performance, it was problematic that this improvement also vastly increased the time the analysis took to run. The goal throughout was to produce code that could easily be used in a production environment, which means memory-safe, thread-aware code that uses efficient data structures and makes optimal use of expensive compute time.

Python was chosen for all code, with the open-source SciPy stack used because it is a mature and extensive scientific computing environment. The `numpy` and `scipy` libraries provide more versatility than packages like MATLAB in terms of interfacing with other systems and code, and faster prototyping than lower-level languages like C++.

A general overview of the object-oriented class layout of the project's main filter module is as follows: a `filter_wrapper` class provides the interface to the algorithms from external code; its `filter` class implements the general control steps of the Kalman and particle filters, and a `numerics` file implements the numerical computing functions for iteration updates. Other files, `stats` and `graphs`, contain functions that handle producing statistics and figures of results, and are called from the wrapper class.

All numerical algorithms were chosen for their computational efficiency. Profiling the code revealed that matrix exponentiation contributed to the vast majority of compute time, so the function to approximate it was carefully selected. The Taylor expansion, Padé approximation and eigen-decomposition were tested, and it was found that a truncated Taylor series was fastest, and could be reduced to the 5th order without noticeably affecting numerical accuracy. Thus, the calculation of e^X, where X is a matrix, is computed with

$$e^X \approx \sum_{k=0}^{n} \frac{1}{k!} X^k \tag{A.1}$$

where n is the order of the series, above which to truncate higher-order terms, and set at $n = 5$ in practice.

One rather obscure optimisation was to change the low-level matrix computation library from the default BLAS/LAPACK linear algebra libraries (Planitz and Anderson, 1995) to Intel's Math Kernel Library (MKL) (Intel, 2007), which is optimised for Intel CPUs. This provided at least a factor of two speed increase when running the filtering algorithms. If we were to use the project's algorithms as the basis for an online trading system, it would likely be beneficial to rewrite these numerically intensive functions in a compiled language like C++, and keep the existing Python wrapper code around it.

The use of the particle filter can be termed *embarrassingly parallel* because each particle's update step can be quite easily farmed out to a threadpool and executed concurrently. This is made straightforward with Python's `multiprocessing` library. Although modest improvements were seen when using multithreading with 4 processes on a standard consumer CPU, improvements tailed off rapidly above this. One could expect the possibility of a large performance increases if the code was modified to use, say, OpenCL on a GPU, with each particle given its own core. This would enable an increase in the number of particles used from our typical number of 20-50 to hundreds, with little or no performance cost if implemented carefully.

A further programming challenge encountered was that of parsing the order event tick data at a fast enough rate to keep up with the filtering algorithm. Each update event requires a complete order book including volume information for all outstanding orders, so these must all be updated in real time with incoming order events. This necessitates the use of appropriate data structures to reduce redundant copy instructions and maintain good use of allocated memory. The process proved so compute-intensive that the entire order stream was converted into snapshot data ahead of time to enable practical runtimes when filtering on the consumer laptop hardware used during testing.

Additional Python libraries that were essential to this work include: `hyperopt` for tree of Parzen optimisation; `pandas` for storing and operating on order book data in a usable format; and `matplotlib` and `seaborn` for producing figures.

Appendix B

Additional data

Table B.1. Optimised hyperparameters for full combined model, trained against one-second lagged MSE $\times$ $(1 - \text{BIN})$ objective (*EUR/USD 2015-09-02* – note that trade day begins at 17:00). Parameters with * were chosen manually.

Parameter	**Value for hour**					
	20:00	**00:00**	**04:00**	**08:00**	**12:00**	**16:00**
N*	10	10	10	10	10	10
λ_J*	0.1	0.1	0.1	0.1	0.1	0.1
σ_J	1.105×10^{-6}	4.502×10^{-6}	6.238×10^{-5}	3.424×10^{-5}	1.928×10^{-6}	8.731×10^{-6}
k	6.135×10^{-8}	4.434×10^{-8}	1.360×10^{-8}	3.536×10^{-8}	8.046×10^{-8}	6.574×10^{-8}
θ_e	-0.00650	-0.0231	-0.00208	-0.0204	-0.0714	-0.0777
θ_0	-0.00168	-0.00336	-0.00747	-0.00267	-0.00331	-0.0119
θ_v	-0.00302	-0.0156	-0.00106	-0.00893	-0.0147	-0.00147
σ_e	1.371×10^{-5}	3.344×10^{-7}	1.507×10^{-6}	4.718×10^{-7}	1.937×10^{-6}	2.451×10^{-6}
σ_{obs}	7.601×10^{-5}	3.373×10^{-6}	3.269×10^{-5}	8.737×10^{-6}	6.518×10^{-5}	5.570×10^{-5}
σ_0	6.532×10^{-5}	2.358×10^{-5}	5.458×10^{-5}	5.213×10^{-5}	2.597×10^{-5}	8.809×10^{-5}
σ_v	1.283×10^{-10}	1.005×10^{-8}	1.033×10^{-9}	5.383×10^{-8}	2.146×10^{-8}	1.739×10^{-10}
T*	3	3	3	3	3	3
μ_J*	0	0	0	0	0	0

Table B.2. Mean of one-second optimised hyperparameters for full combined model, averaged over the six one-hour periods shown in Table B.1.

N	λ_J	σ_J	k	θ_e	θ_0	θ_v
10	0.1	1.88×10^{-5}	5.01×10^{-8}	-3.36×10^{-2}	-5.07×10^{-3}	-7.48×10^{-3}
σ_e	σ_{obs}	σ_0	σ_v	T	μ_J	
3.40×10^{-6}	4.03×10^{-5}	5.16×10^{-5}	1.45×10^{-8}	3	0	

Table B.3. Hyperparameters used with all models for comparative testing over 10-second time frame (manually selected).

N	λ_J	σ_J	k	θ_e	θ_0	θ_v
20	0.1	4×10^{-5}	5×10^{-9}	-0.02	-0.05	-0.01
σ_e	σ_{obs}	σ_0	σ_v	T	μ_J	
1×10^{-6}	6×10^{-4}	1×10^{-5}	1×10^{-10}	3	0	

Appendix C
Risk assessment retrospective

In the risk assessment that was submitted at the start of the project in accordance with department regulations, potential hazards were identified that are typical of a computer-based office workspace, chiefly ergonomic issues. These include eye strain from improperly set up monitors and too much use without breaks, the possibility of repetitive strain injury from keyboard and mouse use, and back and leg pain from poor posture or seating equipment.

Steps were taken to mitigate the chances of these issues arising, including taking regular breaks (and occasional changes of scenery) during development and experimentation. No significant issues were encountered in the course of the project, and this was due to the consideration and prevention of these issues before they occurred.