Exploring the Applicability of Time Series Analysis to Astrophysical Simulations

Bachelor Thesis

University of Basel
Faculty of Science
Department of Mathematics and Computer Science
High Performance Computing Group

Advisor and Examiner: Prof. Dr. Florina M. Ciorba
Supervisor: Dr. Osman Seckin Simsek

Author: Lev Maznichenko
Email: lev.maznichenko@stud.unibas.ch
21-751-821

November 28, 2024

Acknowledgments

I would like to thank my supervisor Dr. Osman Seckin Simsek for accompanying me with patience and desire to help. I would like to also express my gratitude to Prof. Dr. Florina M. Ciorba for giving me the opportunity to work with topic of my choice and providing me with all the necessities.

I would like to also thank all my friends and family who supported me during this thesis, gave their opinion and asked valuable question. I want to express my gratitude towards the whole University of Basel and Departament of Mathematics and Computer science for the great opportunities and warm atmosphere.

Finally, I would like to thank Ivan Sulima for his feedback on the final draft of this thesis, for his opinion and for keen eye for details.

Abstract

Astrophysical simulations are widely used for getting insights into highly complex processes of the universe, such as star formation, the evolution of galaxies, or the generation of the cosmic structures. This thesis explores the application of classical and advanced time series analysis methods for the SPH-EXA (Smoothed Particle Hydrodynamics at Exascale) framework, which is one of the state-of-the-art astrophysical simulations, in order to predict system state without calculating through the simulation. Specifically, we focus on 2 scenarios - Sedov blast, which is a type of shock-wave phenomenon, and on subsonic turbulence, which includes chaotic and unpredictable dynamic movement of particles. Both of said scenarios pose a unique and complex challenges due to their nature and high computational demand.

We aim to find out if specialized methods such as Bayesian filtering and sequential online prediction can effectively reduce computational loads while accurately modeling particle movements in astrophysical contexts. This thesis presents Sequential Online Prediction with Particle Importance Resampling (SOPPIR) algorithm, which combines the approach of sequential online prediction with particle filtering.

After conducting experiments, we evaluate the performance of SOPPIR using Normalized Root Mean Square Error (NRMSE) as a metric for prediction accuracy. Results demonstrate that SOPPIR can achieve NRMSE values of approximately 2.56% for the subsonic turbulence case and 4.35% for the Sedov blast case, thus, it could significantly capture the essential dynamics of the system despite missing high volume of data. These results explicitly demonstrate, that time series analysis can provide valuable insights into particle behavior in astrophysical simulations, potentially enhancing existing approaches or offering supporting solutions.

Finally, this thesis clarifies the advantages of adapting time series techniques into existing simulation frameworks for real-time predictions and enhancing data analysis capabilities. Addressing the challenges posed by high-dimensional data and irregular time steps as well as missing the observations, we aim to contribute to the ongoing efforts to optimize astrophysical simulations for high-performance computing environments.

Contents

Chapter 1

Introduction

Astrophysical simulations provide important insights into complex processes in the universe such as star formation, galaxy evolution, and the creation of cosmic structures. Such simulations model the behavior of matter and energy over various spatial and temporal scales, requiring integration of various physical and mathematical models.

For a long time, developing such simulations with sufficient precision posed a complex challenge due to the computational intensity, which exceeded the capacity of most available computing systems. This challenge arises from the following key reasons:

1. **Multi-scale problem** - Described phenomena occur over a wide range of scales, for example, in our case we mostly discuss particles which move with subsonic velocity, but are limited by incredibly small, relatively to velocity, spatial limits.

2. **Multi-physics problem** - The said processes require the creation of such a model that could take into account both microscopic and macroscopic processes, including the laws of gravitation, hydrodynamics, thermodynamics and electromagnetism. Combining these physical forces inevitably leads to a significant increase in computational complexity.

3. **High computational demand** - Achieving precise and reliable results requires the use of a significantly large number of particles or grid points in simulations. These numbers can scale to millions or even billions, leading to substantial computational demands. This requires the use of highly optimized algorithms and cutting-edge hardware to manage the associated processing load effectively.

4. **Complex Nonlinear Behavior** - Astrophysical systems typically exhibit nonlinear dynamics, meaning small changes can lead to disproportionately significant outcomes over time. Capturing these behaviors accurately requires sophisticated numerical techniques and finely detailed temporal resolution, further intensifying the computational demand.

To address these issues, advanced computational frameworks such as SPH-EXA (Smoothed Particle Hydrodynamics at Exascale) have been developed. SPH-EXA leverages the Smoothed Particle Hydrodynamics (SPH) method, a particle-based approach well-suited for simulating fluid-like systems and managing complex boundary conditions. This makes it particularly effective for scenarios

like shockwave dynamics (e.g., Sedov blasts) and turbulence. Nevertheless, the scope and approach of SPH-EXA goes far beyond, its code was built with performance and scalability as the main defining elements of its structure, using the latest technologies in software engineering, big data and high performance computing.

This thesis focuses on the mentioned SPH-EXA framework which is designed to simulate astrophysical phenomena using SPH. We aim to investigate whether time series analysis methods can provide meaningful insights, such as predicting the system state without performing manual calculations, into SPH-EXA simulations, specifically for Sedov blast and subsonic turbulence cases. Albeit being highly-optimized and efficient, SPH-EXA still relies heavily on supercomputers, therefore, there is an interest in developing methods, that could potentially relax or remove this requirement altogether.

Although classical time series methods, such as ARMA, ARIMA, and the like, are not intended to be applicable to highly non-linear systems and irregular time steps, there are other, more specialized methods, in particular Bayesian filters (and successors) and sequential online prediction, which offer promising alternatives. Hence, in this study we examine if these advanced, less conventional and more specialized methods can contribute to reducing computational loads or filling the gaps in the data by predicting system state without performing manual calculations. We will focus on two particular SPH-EXA simulations, namely Sedov blast and Subsonic turbulence.

Thus, the goal of this thesis is to explore to which extent time series analysis and prediction methods can model or predict states of the system within SPH-EXA simulations, and to evaluate their effectiveness in capturing the dynamics of complex astrophysical phenomena. By evaluating these methods, we are be able to determine, whether they can offer computational or analytical benefits, potentially providing an alternative to or enhancing existing approaches.

Essentially, in this thesis we answer following research questions:

1. Is it possible to predict the movement of particles without calculating their movement through simulation? If so, how? What is the error?

2. If the simulation is already finished, is it possible to insert additional time steps between the existing? If so, how? What is the error?

We develop an algorithm, which we call Sequential Online Prediction with Particle Importance Resampling (SOPPIR), in order to answer posed questions. This algorithm combines 2 time series approaches, which are widely used for signal processing and state estimation for dynamic non-linear systems, in particular - particle filter and sequential online prediction.
SOPPIR shows excellent results in predicting the movement of the particles - while missing 25% of the data for subsonic turbulence case, it predicts the positions of the particles at each time step with NRMSE consistently lower than 3% and for Sedov blast case with 80% of missing observations it predicts the positions with NRMSE consistently lower than 5%.

Due to storage constraints, the data from simulation is often stored with quite sparse time steps, which leads to uncertainty about events between said time steps. One of the use cases of our algorithm can involve post-hoc recreating the movement of particles by making these steps more

frequent, resulting in more dense data leading to ability to create high-quality and more smooth visualization of the astrophysical phenomena.

The rest of this thesis is organized as follows:
In chapter 2 we provide the background information regarding Smoothed Particle Hydrodynamics (SPH) and its application in the SPH-EXA framework. We then present an overview of time series analysis methods, starting with AR and finishing with ARIMA, as well their limitations and why they can't be applied to our task. In this chapter we also cover various filtering techniques, like Bayesian filters and its successors, and their properties. Then we conclude with overview of sequential online prediction framework, which can be applied to our system in combination with refined filtering techniques. In Chapter 3 we introduce Sequential Online Prediction with Particle Importance Resampling (SOPPIR), providing an overview of the algorithm, detailed steps, and pseudocode. We also discuss nuances and time step splitting in the context of the algorithm. In Chapter 4 we present and discuss the results of the study, including the evaluation methodology and analysis of findings related to particle movement prediction and possible time step insertion. In Chapter 5 we conclude the thesis, summarizing key findings and discussing implications for astrophysical simulations. We also outline potential areas for future work and improvements. In Appendix at the end of the thesis we include additional relevant information.

Chapter 2

Background

2.1 Smoothed Particle Hydrodynamics

The smoothed particle hydrodynamics (SPH) method works by dividing the fluid into discrete elements called particles[18]. Each of these particles has a spatial distance (referred to as the 'smoothing length, which is often represented in equations as **h** over which their properties are 'smoothed' (therefore the name) by a kernel function [32]. It means that each physical quantity of each particle can be obtained by summing up the corresponding quantities of all particles which lie within two smoothed lengths. For example, the mass at the point **r** depends on the mass of all particles at distance **2h** from **r**.
The effect of each particle on the characteristics is evaluated according to its density and the distance to desired particle of interest. Mathematically, this is described by the kernel function, which is denoted by **W**. The Gaussian (normal distribution) function or the cubic spline [34] [1] are usually used as the kernel function. The latter function is zero for particles further away than **2h** contrary to the Gaussian function, where there is a slight influence at any finite distance. It saves computational resources by excluding the relatively small influence of distant particles.

The value of any physical characteristic **A** at the point **r** is given by the formula:

$$A(\mathbf{r}) = \sum_i \frac{m_i}{\rho_i} A_i W(\mathbf{r} - \mathbf{r}_i, h)$$

where:

1. m_i is the mass of particle i,
2. A_i is the value of quantity A for particle i,
3. ρ_i is the density associated with particle i,
4. $W(\mathbf{r} - \mathbf{r}_i, h)$ is the smoothing kernel function centered on particle i with smoothing length h.

[1] A cubic spline is a smooth function whose domain of definition is divided into a finite number of segments, at each of which it coincides with some cubic polynomial.

For example, the density ρ_i at some given particle i can be calculated by summing the contributions from all neighboring particles using the smoothing kernel [37]:

$$\rho_i = \rho(\mathbf{r}_i) = \sum_j m_j W(\mathbf{r}_i - \mathbf{r}_j, h)$$

This formula sums up the mass contributions from each of the particles within the smoothing length h, providing the local density at particle i's position.

Similarly, the spatial gradient of a quantity A at point $\mathbf{r}$ can be approximated in SPH by differentiating the smoothing kernel:

$$\nabla A(\mathbf{r}) = \sum_i \frac{m_i}{\rho_i} A_i \nabla W(\mathbf{r} - \mathbf{r}_i, h)$$

where ∇W is the gradient of the smoothing kernel function with respect to $\mathbf{r}$.

The smoothing length in SPH can be either be fixed or allowed to vary over time [42]. In case when each particle has it's own smoothing length which is adjusted over time, it will allow the simulation to effectively adapt to local conditions [18]. For example, in a dense regions with bunch of particles close together, the smoothing length can be set relatively small, which results in high spatial resolution. Contrary, in low density regions, where there are particles which are distanced from each other and resolution is low, the smoothing length can be larger, resulting in optimal computations for said regions. When combined with equation of state [2] and integrator [3], SPH can effectively simulate hydrodynamic flows [34]. However, the traditional artificial viscosity formulation, which is used in SPH, tends to blur shock waves and contact discontinuities to a much more than modern mesh-based methods.
The adaptability of SPH based on the Lagrangian approach is similar to adaptive mesh refinement which is used in state-of-art mesh-based codes, albeit latter case can be refined using any given criterion, since the hydrodynamics of smoothed particles are Lagrangian per se, it is limited in its refinement parameters to solely density ρ [40].
Often in simulations there is a need to model gravity as well in addition to hydrodynamics. The essence of SPH which is particle-based makes it a perfect choice to combibe with a gravity handler which is particle-based as well[38].

2.2 Smoothed Particle Hydrodynamics at Exascale

Smoothed Particle Hydrodynamics at Exascale (SPH-EXA) is an advanced framework designed to perform exascale astrophysical simulations. The term 'exascale' refers to computing systems capable of performing at least 10^{18} floating point operations per second (FLOPS) [21]. In the context of the SPH-EXA framework, this refers to a commitment to efficiently utilise such high-performance systems, enabling researchers to perform extremely complex astrophysical simulations

[2] Equation of state is a relation reflecting, for a particular class of thermodynamic systems, the relationship between macroscopic physical quantities characterizing it, such as temperature, pressure, volume, chemical potential, entropy, internal energy, enthalpy and others [8]

[3] An integrator is a technical element whose output signal (output value, output parameter) is proportional to the integral, usually in time, of the input signal [16]

with outstanding accuracy, and performance [28].
SPH-EXA stands noteworthy for its ability to adapt to a wide range of astrophysical scenarios, including shock waves and turbulence [11]. The platform is designed to dynamically adjust the smoothing length for each particle, allowing the resolution to adapt to local conditions, as it was discussed above. This capability is critical to accurately capture complex phenomena at a variety of spatial and temporal scales.
SPH-EXA builds on the strengths of existing smoothed partial hydrodynamics (SPH) approaches such as SPHYNX, ChaNGa and SPH-flow [7]. By integrating these methodologies, SPH-EXA provides an optimised codebase tailored for exascale computing environments. The platform can handle simulations containing billions of particles, allowing researchers to examine complicated events in great detail.
The SPH-EXA code is developed in C++20 and uses a hybrid parallelization strategy combining Message Passing Interface (MPI), OpenMP, CUDA and HIP [11]. This multifaceted approach ensures efficient utilisation of modern high-performance computing architectures, including multi-core processors and accelerators, thereby improving the performance and scalability of simulations.

2.3 System overview

The nature of our system creates a few limitations, and, as a result, we try to overcome such limitation by applying time series. Our system is the SPH-EXA simulation itself which models two cases - Sedov blast (shockwave) and subsonic turbulence[21]. The limitation of the system comes from the fact, that system does not automatically saves all the data, since for normal order simulations, say a billion particles, it may take up to several gigabytes. Thus, before the start of the simulation we need to set a time step which will be used for writing the data to the file. It is important to mention, that the simulation is performed until the determined time step is reached. Therefore we need to set up the parameters of the simulation in a way, that we can observe it without missing much data, but at the same time without writing too much of unnecessary data. The data is stored in HDF5 format which we then use for any manipulation with results of our simulation. Since we observe the simulation is performing indefinitely of the fact that we observe it, we have [7]:

1. **SPH iteration** - events, when all particles change their physical fields and move in different directions [34].

2. **Time** - Physical time which is has been simulated between two time steps.

3. **Time step** - events, largest amount of physical time the simulation can simulate in each SPH iteration depending on the setup.

As mentioned, there is no possibility to artificially map iterations 1:1 to time steps, therefore, we can only try to manipulate the parameters of the system in such a way, so this mapping will be close enough without missing much data. The parameters may include amount of particles, frequency of time step, scenario of simulation etc. Nevertheless, excluding cases where we want to track **each** change and therefore set the time step at a very low number (e.g. 0.0001 seconds), which will result in a proportionally large dataset with too much excessive data, we will always miss some observations.
Iterations occur for 2 reasons:

1. Frequency dependent on basic system characteristics - such iterations occur quite regular, with some minor time noise (e.g. each 0.1 $\pm$0.001s) [7]

2. Events which require instant system state recalculation, for example particles being close to each other in a higher resolution discretization in which case the movements of the particles must be recalculated in smaller physical time difference. In such cases it is obvious that particle can't keep moving in the same direction, therefore simulation need to redefine their velocities [7].[4]

The nature of missing data varies on the type of scenario, as shockwave is more linear and predictable in it's nature, there is no big harm in missing quite big chunk of the data, since the movement of particles in this scenario can be *mostly* defined by simple (which are not as simple as it may seem as we will show later) physical models. Contrary, due to chaotic and unpredictable nature [45, 48], subsonic turbulence is very hard to predict and each missing iteration can be very important due to drastic changes which are coming from subsonic speed of each particle. Therefore, for observer the instability of the system and its dynamics comes not only from chaotic nature of the phenomena per se, but also from the irregular events of iteration and slight time noise of the time steps.

2.3.1 Concrete cases

We have two test scenarios we work on:

1. **turb-50** case - scenario in which the subsonic turbulence for 50^3 (for each dimension) particles is simulated. Subsonic turbulence can be described as highly-chaotic and unpredictable phenomena which poses one of the most complex problems in classical physics [13]. We record the state of the system each 0.01 second for 10 seconds which results in 1000 time steps. After performing this simulation, it results in $\sim$ 1250 iterations. Keeping in mind the fact that we have 1000 time steps, it results in the fact that we miss 20% of iterations, hence we are not able to see the same amount of changes of the system state [7].

2. **sedov-50** case - scenario in which the sedov shockwave for 50^3 (for each dimension) particles is simulated. Contraty to subsonic turbulence, it is much more linear and predictable phenomena, which is compensated by more frequent iterations of the system, resulting in the fact, that observation with same frequency as for turb-50 case, are not that indicative. We record the state of the system each 0.01 second for 10 seconds which results in 1000 time steps. After performing this simulation, it results in $\sim$ 5000 iterations. Likewise, it means that we miss tremendous 80% of iterations, hence we don't see most of the changes of the system state [7].

However, we can't compare two different scenarios only by the amount of missing data. As it was explained above, said simulations have very different nature and missing a few observations for a shockwave scenario is not that tragic, given the fact of its linear-like behavior. As we will later see in the results, both of these cases are comparable in terms of prediction complexity.

[4] In the perfect SPH simulation particles cannot collide due to their nature - particles act on each other like unipolar magnets - they cannot collide and repel each other as they approach each other, therefore in cases when there are a lot of particles close, system automatically starts recalculating their velocities more often, which leads to more frequent iterations. [34].

2.3.2 Explanation of scenarios and structure of datasets

Our scenarios have an additional constraint, which comes from its simulated nature. In particular, our simulations are always done in 3-dimensional box with some preset boundaries, contrary to real-world phenomena, where there is clearly no physical boundaries. Particularly in our case, these boundaries are defined as $[-0.5, 0.5]$ for each axis, which results in 3-dimensional cubic box with each face be a length of exactly 1.

Datasets which are the result of performing the simulation for both scenarios have the following structure [7]:

1. **Time Steps**: The data is organized into a series of time steps, labeled sequentially (e.g., Step#0, Step#1, ..., Step#999).
2. **SPH iteration**: iteration of the simulation, when particles' physical quantities are recalculated due to set constraints or due to some simulation's event (e.g. to particles becoming very close to each other)
3. **Particles**: At each time step, the simulation records information for all particles of the dataset (e.g., 125,000 particles):
   (a) **Position**: The spatial coordinates of the particle in three dimensions (x, y, z).
   (b) **Velocity**: The velocity components of the particle along each axis (vx, vy, vz). The velocity vector is recalculated at each iteration by the simulation itself.
   (c) **ID**: A unique identifier for each particle.
   (d) **Density**: ρ associated with each particle, see Smoothed Particle Hydrodynamics.

In order to answer questions posed in Introduction, we need an approach which allows precise particle movement prediction, since, if such approach exists, it will be able to also insert an additional time step, since the nature of such process will be the same as for prediction. Taking into account the complexity and dynamic nature of said processes, we need to take a glance into advanced time series prediction techniques which would allow us to achieve the desired goal of accurately predicting the position and movement of particles. In the following sections, the theoretical foundations for such a method will be outlined, as well as explained why some of the classical methods are not applicable for our case.

2.4 Time series

A time series X_t is a sequence of discrete data observed at time points $t = 1, 2, \ldots, n$. Usually the time points t are regular [5], hence most of the algorithms are intended to work with such constraints [25]. The main difference between time series and any simple data sample is that in case of time series analysis, when working with data, we assume that there is a relationship between measurements over time, and it's not just the statistical diversity and statistical characteristics of the said sample. The most popular and widely-used (usually in finance, weather forecasting, demographics etc) time series methods are based on Autoregression and moving average [4].

[5] In case of time series analysis regularity stands for equally spaced points of time, so $t_1 = t_2 = ... = t_n$

2.4.1 Autoregression

An autoregressive model AR(p) defines the current value of a time series X_t as a linear combination of several previous values of said series, called lags, and a random error (noise) ϵ_t [5].

For a model of order p, the autoregressive model is written as follows:

$$X_t = c + \sum_{i=1}^{p} \varphi_i X_{t-i} + \epsilon_t$$

where:

1. X_t is the current value of the time series,
2. c is a constant or average value of the series (in some cases may be absent),
3. φ_i is autoregressive coefficients, which show the degree of influence of each lag X_{t-i} on the current value,
4. p is the order of the model, i.e. the number of previous values (lags) that are taken into account,
5. ϵ_t is a random[6] error or noise, assumed to be white noise with zero mean and constant variance.

As an example, for a first-order model AR(1), the current value of the series depends on only one previous value:

$$X_t = c + \varphi_1 X_{t-1} + \epsilon_t$$

where φ_1 is the autoregressive coefficient for one lag. If φ_1 is close to 1, it means high autocorrelation, i.e. the current value strongly depends on the previous one.

This model is often used in various fields, such as financial markets, economics indicators, and population dynamics:

1. **Financial markets** - AR models can capture short-term dependencies in stock prices, volatility forecasting and used in development of trading algorithms [44, 36].
2. **Economics indicators** - AR models are often used to predict GDP growth (or decline) rate, as well as inflation and unemployment rate [35, 44].
3. **Population dynamics** - AR models can also be used to make forecast species populations and spread of invasive species [3, 17].

2.4.2 Moving Average

The moving average model MA(q) describes the current value of a time series as a linear combination of the current and several previous values of random error (noise). Unlike AR, here the current

[6] Usually is random Gaussian variable

value does not depend on the past values of the time series, but on the previous forecast errors [22]. For a model of order q, the moving average model is written as follows:

$$X_t = \mu + \sum_{j=1}^{q} \theta_j \epsilon_{t-j} + \epsilon_t$$

where:

1. X_t is the current value of the time series,
2. μ is some constant or mean of the series,
3. θ_j is moving average coefficients, which show the influence of past errors on the current value,
4. q is the order of the model, i.e. the number of previous errors that are taken into account,
5. ϵ_{t-j} is the error (noise) at the j-th step back.

As an example, in the first-order model MA(1), the current value depends on only one previous error:

$$X_t = \mu + \epsilon_t + \theta_1 \epsilon_{t-1}$$

where θ_1 is the coefficient determining the impact of the error one step back.

This model is usually used together with AR, hence the application involves same fields such as financial markets, population dynamics, economic indicators and others [35, 44, 36]

2.4.3 Autoregressive moving average

The model ARMA(p, q) combines autoregression AR(p) and moving average MA(q) [6], where: - p is the order of the autoregression, - q is the order of the moving average.

Hence, ARMA(p, q) can be defined as following::

$$X_t = c + \sum_{i=1}^{p} \varphi_i X_{t-i} + \epsilon_t + \sum_{j=1}^{q} \theta_j \epsilon_{t-j}$$

where:

1. c is a constant,
2. φ_i is autoregressive coefficients,
3. θ_j is moving average coefficients,
4. ϵ_t is a white noise with zero mean and constant variance.

As example, an ARMA(2, 2) can serve:

$$X_t = c + \phi_1 X_{t-1} + \phi_2 X_{t-2} + \epsilon_t + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2}$$

This model accounts for the two prior values and two errors, hence it allows us to investigate for more complex dependencies and variations in the data.

Key characteristics of ARMA are [44]:

1. **Stationarity** - In order for ARMA model to be applicable, time series muse be stationary, which means that its mean and variance (and sometimes other values in extended models) must be constant over time, hence autocorrelation depends on the time lag and not on the time itself. For the AR component to be stationary, the coefficientsϕ_i must satisfy certain conditions (e.g., in the AR(1) model, stationarity is ensured when $|(|\varphi_1| < 1)$.

2. **Autocorrelation** - The autocorrelation function (ACF) and partial autocorrelation function (PACF) of the ARMA muse decay according to an exponential or oscillating law, and this is the key difference between ARMA and pure AR or MA.

3. **Lags** - The parameters p and q allow the ARMA model to account for both long-run autocorrelation (AR component) and short-run random noise effects (MA component).

4. **Parameter estimation** - The coefficients φ_i and θ_j of the ARMA model are estimated by various numerical methods, including least squares, maximum likelihood etc. These parameters determine the influence of past values and errors on the current value of the time series.

2.4.4 Autoregressive integrated moving average

On the base of ARMA model there was developed a more advanced ARIMA model, where I stands for "Integrated". This model is applied, when the time series data is non-stationary [5].
The ARIMA(p, d, q), where d is the order of integration, indicating the number of differentiations to achieve stationarity, can be defined as following:

$$\Delta^d X_t = c + \sum_{i=1}^{p} \varphi_i \Delta^d X_{t-i} + \epsilon_t + \sum_{j=1}^{q} \theta_j \epsilon_{t-j}$$

where $\Delta^d X_t$ stands for d-fold differentiation:

$$\Delta X_t = X_t - X_{t-1}$$

2.4.5 Limitations of classical methods

There are countless improvements and alternative methods (for example, Exponential smoothing) that try to bring something new and make classical methods more universal, but they are not suitable for our case for the following reasons:

1. Our dynamic system is too complex and chaotic.

2. Quite often only the previous observation is relevant (pops out from the previous point).

3. Nonlinearity of most parameters.

4. Impossibility, and consequently absence of need to attempt long-range predictions, again due to the chaotic nature of the system.

Hence, in our case, it is natural for us turn to more specific, tailored methods for similar scenarios. More specifically, we are talking about signal processing theory and its digital application - stochastic filtering, Bayesian filtering, and their successors.

2.5 Filtering

2.5.1 Stochastic Filtering

Stochastic filtering describes a problem where we need to determine the state of the system from either incomplete or noisy set of observations (often both).

In many real-world systems, the inner states cannot be directly observable because of some noise, limitations, flaws of sensors etc. Stochastic filters give an opportunity to make some estimations that are close to the true state of said hidden states via incorporating both the system dynamics and some uncertainties which are inherent in both the measurements (e.g. sensor) and process itself (e.g. movement of some object) [47].

Let's consider a discrete-time[7] stochastic process $\{X_k\}$ representing the hidden state and an observation process $\{Y_k\}$. The state evolution and observation models are given by:

1. **State Transition Model**:
$$X_k = f_{k-1}(X_{k-1}, V_{k-1}),$$
where V_{k-1} is the process noise with known statistics.

2. **Observation Model**:
$$Y_k = h_k(X_k, N_k),$$
where N_k is the observation noise with known statistics.

The goal is to compute the posterior distribution of the state given all available observations up to time k:
$$p(X_k \mid Y_{1:k}).$$

So, to be clear, stochastic filtering operates recursively through two main steps:

1. **Prediction**: Propagate the current state estimate forward in time using the system model to obtain a prior estimate.

2. **Update**: Incorporate the new observation to correct the prior estimate, accounting for uncertainties in both the model and measurements.

This recursive approach allows for real-time estimation as new data becomes available [26]. Said approach can be expanded by numerous methods, one of the most used is Bayesian techniques, which we discuss further in 2.5.3 Bayesian Filtering.

An important assumption in stochastic filtering applied to time series contrary to classical time series methods is so-called Markov property. The Markov property states that the future state of a system depends only on its current state, not on the sequence of events that happened before it [20]. In other words, the current state encapsulates all the information needed to predict the future state, making the system "memoryless" with respect to its past states. Mathematically, for a stochastic process X_t, the Markov property can be expressed as:
$$P(X_{t+1}|X_t, X_{t-1}, ..., X_0) = P(X_{t+1}|X_t)$$

[7] There are continuous-time versions of said problem, but we talk solely about discrete-time

This property significantly simplifies the computational complexity of filtering algorithms, as they don't need to store or process the entire history of states. It allows the filter to operate recursively, using only the most recent state estimate and the current observation to update its belief about the system state

2.5.2 Monte Carlo

Monte Carlo methods are computational algorithms that rely on random sampling to obtain numerical results. Usually they are used for tasks like approximating integrals or solving problems with probabilistic interpretations [39].

Given an integral of the form:

$$I = \int f(x)\,dx,$$

Monte Carlo integration approximates I using random samples x_i:

$$I \approx \frac{1}{N}\sum_{i=1}^{N} f(x_i),$$

where x_i are independent and identically distributed samples drawn from the probability distribution of x.

In stochastic filtering, Monte Carlo methods are used to approximate the posterior distribution $p(X_k \mid Y_{1:k})$ when analytical solutions are infeasible for some reason. Particle filters, the most popular application of sequential Monte Carlo method, use a set of weighted particles to represent and update said distribution [12].

To improve efficiency, variance reduction techniques are applied, for example, **Importance Sampling** - samples are drawn from a proposal distribution $q(x)$, and weights correct for the difference between $q(x)$ and the target distribution $p(x)$:

$$I \approx \frac{1}{N}\sum_{i=1}^{N} \frac{p(x_i)}{q(x_i)} f(x_i).$$

2.5.3 Bayesian Filtering

Bayesian filtering and stochastic filtering are often interchangeable terms, especially in the context of state-space models where Bayesian methods[8] are prevalent [10, 43]. However, to be precise, Bayesian filtering explicitly uses Bayes' theorem to update probability distributions of the system state as new observations become available. It focuses on computing the posterior distribution of the state given all available observations.

Bayesian filtering provides a probabilistic framework for sequential state estimation, combining prior knowledge with new observations, also Bayesian filters systematically incorporate all sources of uncertainty, making them optimal under the Bayesian paradigm. They offer a principled way to update beliefs about the system's state as new data arrives.

[8] such as Kalman or Particle filter

Same as in stochastic filtering (whose successor Bayesian filtering is), the state evolution and observation models are given by:

1. **State Transition Model**:
$$X_k = f_{k-1}(X_{k-1}, V_{k-1}),$$
where V_{k-1} is the process noise with known statistics.

2. **Observation Model**:
$$Y_k = h_k(X_k, N_k),$$
where N_k is the observation noise with known statistics.

Recursive Bayesian Estimation involves:

1. **Prediction Step**:
$$p(X_k \mid Y_{1:k-1}) = \int p(X_k \mid X_{k-1}) p(X_{k-1} \mid Y_{1:k-1})\, dX_{k-1}.$$
This step uses the system's dynamics to predict the state distribution at time k.

2. **Update Step**:
$$p(X_k \mid Y_{1:k}) = \frac{p(Y_k \mid X_k) p(X_k \mid Y_{1:k-1})}{p(Y_k \mid Y_{1:k-1})},$$
where $p(Y_k \mid Y_{1:k-1})$ ensures normalization.

The problem is, that these integrals are often intractable in closed form, particularly for nonlinear and non-Gaussian systems, necessitating approximate methods like the Kalman filter, particle filter, or ensemble methods, which are discussed further.

2.5.4 Classical Kalman Filter

Kalman Filter is a successor of Bayesian filtering, hence it's a recursive data processing (filter) algorithm which is used to estimate the state of dynamic system from noisy or incomplete observations [26]. It predicts the state of the system at the next time step and then updates this prediction based on the measurement. The general assumption is that the system can be described by linear stochastic equations and that both measurement noise and process are Gaussian per se. As it was mentioned above, we are considering solely discrete-time dynamic system, hence the state evolution and observation models are given by:

1. **State Evolution Model**:
$$\mathbf{x}_k = \mathbf{A}_{k-1}\mathbf{x}_{k-1} + \mathbf{B}_{k-1}\mathbf{u}_{k-1} + \mathbf{w}_{k-1},$$
where:
   (a) $\mathbf{x}_k$ is the state vector at time k.
   (b) $\mathbf{A}_{k-1}$ is the state transition matrix.
   (c) $\mathbf{B}_{k-1}$ is the control input matrix.

(d) $\mathbf{u}_{k-1}$ is the control input vector.

(e) $\mathbf{w}_{k-1}$ is the process noise, assumed to be Gaussian with zero mean and covariance $\mathbf{Q}_{k-1}$, i.e., $\mathbf{w}_{k-1} \sim \mathcal{N}(\mathbf{0}, \mathbf{Q}_{k-1})$.

2. **Observation Model**:

$$\mathbf{y}_k = \mathbf{H}_k \mathbf{x}_k + \mathbf{v}_k,$$

where:

(a) $\mathbf{y}_k$ is the observation vector at time k.

(b) $\mathbf{H}_k$ is the observation matrix.

(c) $\mathbf{v}_k$ is the measurement noise, assumed to be Gaussian with zero mean and covariance $\mathbf{R}_k$, i.e., $\mathbf{v}_k \sim \mathcal{N}(\mathbf{0}, \mathbf{R}_k)$.

The main algorithm of Kalman filter is to recursively proceed through the following steps:

1. **Initialization** - we set the initial state estimate $\hat{\mathbf{x}}_{0|0}$ and initial error covariance $\mathbf{P}_{0|0}$.

2. **Prediction Step** - we estimate predicted state using following equation:

$$\hat{\mathbf{x}}_{k|k-1} = \mathbf{A}_{k-1} \hat{\mathbf{x}}_{k-1|k-1} + \mathbf{B}_{k-1} \mathbf{u}_{k-1}.$$

and calculate predicted error covariance as following:

$$\mathbf{P}_{k|k-1} = \mathbf{A}_{k-1} \mathbf{P}_{k-1|k-1} \mathbf{A}_{k-1}^{\top} + \mathbf{Q}_{k-1}.$$

3. Then we proceed with **Update Step**:
We first calculate innovation or measurement residual:

$$\tilde{\mathbf{y}}_k = \mathbf{y}_k - \mathbf{H}_k \hat{\mathbf{x}}_{k|k-1}.$$

then we calculate innovation covariance:

$$\mathbf{S}_k = \mathbf{H}_k \mathbf{P}_{k|k-1} \mathbf{H}_k^{\top} + \mathbf{R}_k.$$

then we proceed with so-called Kalman Gain:

$$\mathbf{K}_k = \mathbf{P}_{k|k-1} \mathbf{H}_k^{\top} \mathbf{S}_k^{-1}.$$

finally, updated state estimate:

$$\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k \tilde{\mathbf{y}}_k.$$

and updated error covariance:

$$\mathbf{P}_{k|k} = (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k) \mathbf{P}_{k|k-1}.$$

Kalman filter recursion can be defined in these 2 steps - prediction and update [26]. Based on the previous state estimate, which is defined as combination of prediction and measurement, and some known inputs, say velocity, filter predicts the current state and some associated uncertainty. Then in update step filter receives a new measurement and computes the innovation, which is the divergence between the predicted measurement and actual[9]. The Kalman gain determines how much the prediction should be correction based on this innovation and some relative uncertainties of the said prediction and measurement. In the case of Kalman filter, we always assume that we have linearity and Gaussian noise, hence, the Kalman filter is optimal for system with said characteristics. In non-linear system the performance of Kalman filter is often expected to degrade [24].

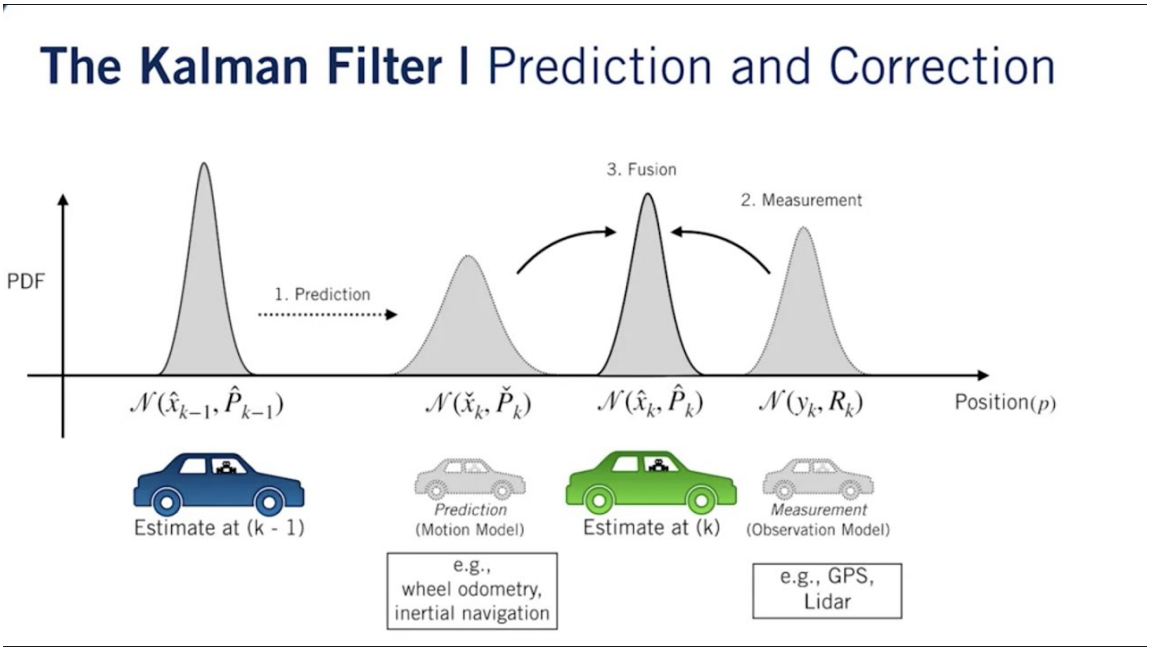


Figure 2.1: Visualization of the Kalman Filter method from [33]

Figure 2.1 shows the visualization of Kalman filter applied to the movement of a vehicle. Having the data at time step $(k-1)$, one can estimate the position of the vehicle at time step (k) using observation data from the sensor, which is flawed for one reason or another, and predicting the movement using some motion model

2.5.5 Successors of the Kalman Filter

To address the limitations of the standard Kalman filter in handling nonlinear systems, there were extensions like the **Extended Kalman Filter (EKF)**, the **Unscented Kalman Filter (UKF)** and **Ensemble Kalman Filter (EnKF)** [24, 15] developed.

[9]Which is still supposed to be flawed, hence we never treat is as "ground truth"

Extended Kalman Filter (EKF)

The EKF linearizes the nonlinear system (contrary to classical Kalman filter which can't work with nonlinear systems) around the current estimate using a first-order Taylor series expansion, effectively applying the Kalman filter to a linear approximation of the nonlinear system [41]. Essence of the Kalman filter is the same:

1. **State Transition Model**:
$$\mathbf{x}_k = \mathbf{f}(\mathbf{x}_{k-1}, \mathbf{u}_{k-1}) + \mathbf{w}_{k-1},$$
where $\mathbf{f}(\cdot)$ is a nonlinear function.
2. **Observation Model**:
$$\mathbf{y}_k = \mathbf{h}(\mathbf{x}_k) + \mathbf{v}_k,$$
where $\mathbf{h}(\cdot)$ is a nonlinear function.

On top of which we layer the EKF algorithm:

1. **Initialization** - Set initial state estimate $\hat{\mathbf{x}}_{0|0}$ and covariance $\mathbf{P}_{0|0}$.
2. **Prediction Step**:
   (a) **Predicted State Estimate**:
   $$\hat{\mathbf{x}}_{k|k-1} = \mathbf{f}(\hat{\mathbf{x}}_{k-1|k-1}, \mathbf{u}_{k-1}).$$
   (b) **Jacobian of the State Transition Function**:
   $$\mathbf{F}_{k-1} = \left.\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\right|_{\hat{\mathbf{x}}_{k-1|k-1}, \mathbf{u}_{k-1}}.$$
   (c) **Predicted Error Covariance**:
   $$\mathbf{P}_{k|k-1} = \mathbf{F}_{k-1}\mathbf{P}_{k-1|k-1}\mathbf{F}_{k-1}^\top + \mathbf{Q}_{k-1}.$$
3. **Update Step**:
   (a) **Predicted Measurement**:
   $$\hat{\mathbf{y}}_{k|k-1} = \mathbf{h}(\hat{\mathbf{x}}_{k|k-1}).$$
   (b) **Jacobian of the Observation Function**:
   $$\mathbf{H}_k = \left.\frac{\partial \mathbf{h}}{\partial \mathbf{x}}\right|_{\hat{\mathbf{x}}_{k|k-1}}.$$
   (c) **Innovation or Measurement Residual**:
   $$\tilde{\mathbf{y}}_k = \mathbf{y}_k - \hat{\mathbf{y}}_{k|k-1}.$$

(d) **Innovation Covariance**:

$$\mathbf{S}_k = \mathbf{H}_k \mathbf{P}_{k|k-1} \mathbf{H}_k^\top + \mathbf{R}_k.$$

(e) **Kalman Gain**:

$$\mathbf{K}_k = \mathbf{P}_{k|k-1} \mathbf{H}_k^\top \mathbf{S}_k^{-1}.$$

(f) **Updated State Estimate**:

$$\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k \tilde{\mathbf{y}}_k.$$

(g) **Updated Error Covariance**:

$$\mathbf{P}_{k|k} = (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k) \mathbf{P}_{k|k-1}.$$

So, essentially the EKF approximates the nonlinear functions $\mathbf{f}(\cdot)$ and $\mathbf{h}(\cdot)$ by linearizing them around the current estimate using the Jacobian matrices $\mathbf{F}_{k-1}$ and $\mathbf{H}_k$. Then it applies the classical Kalman filter equations to said linear approximations [41].

Unscented Kalman Filter (UKF)

The UKF addresses the limitations of the EKF by using the Unscented Transform (UT) to more accurately capture the mean and covariance of a nonlinear transformation of a Gaussian random variable without performing said linearization [24].

Unscented Transform can be defined as following:
Given a random variable $\mathbf{x}$ with mean $\bar{\mathbf{x}}$ and covariance $\mathbf{P}_x$, the UT approximates the statistics of a nonlinear transformation $\mathbf{y} = \mathbf{g}(\mathbf{x})$ by propagating a carefully chosen set of sample points (sigma points) through the nonlinear function. Hence, the general UKF algorithm can be defined as following:

1. **Initialization** - Set initial state estimate $\hat{\mathbf{x}}_{0|0}$ and covariance $\mathbf{P}_{0|0}$.
2. **Sigma Point Generation**:
   (a) **Compute Sigma Points** - For a state vector of dimension n, compute $2n + 1$ sigma points:

$$\chi_0^{(0)} = \hat{\mathbf{x}}_{k-1|k-1},$$

$$\chi_0^{(i)} = \hat{\mathbf{x}}_{k-1|k-1} + \left(\sqrt{(n+\lambda)\mathbf{P}_{k-1|k-1}}\right)_i, \quad i = 1, \ldots, n,$$

$$\chi_0^{(i)} = \hat{\mathbf{x}}_{k-1|k-1} - \left(\sqrt{(n+\lambda)\mathbf{P}_{k-1|k-1}}\right)_i, \quad i = n+1, \ldots, 2n,$$

   where λ is a scaling parameter, and $\left(\sqrt{(n+\lambda)\mathbf{P}_{k-1|k-1}}\right)_i$ denotes the i-th column of the matrix square root.

(b) **Assign Weights**:

$$W_m^{(0)} = \frac{\lambda}{n+\lambda},$$

$$W_c^{(0)} = \frac{\lambda}{n+\lambda} + (1-\alpha^2+\beta),$$

$$W_m^{(i)} = W_c^{(i)} = \frac{1}{2(n+\lambda)}, \quad i = 1, \ldots, 2n,$$

where α determines the spread of the sigma points, and β incorporates prior knowledge of the distribution ($\beta = 2$ is optimal for Gaussian distributions).

3. **Prediction Step**:

(a) **Propagate Sigma Points through the State Transition Function** :

$$\chi_{k|k-1}^{(i)} = \mathbf{f}(\chi_{k-1}^{(i)}, \mathbf{u}_{k-1}).$$

(b) **Predicted State Estimate**:

$$\hat{\mathbf{x}}_{k|k-1} = \sum_{i=0}^{2n} W_m^{(i)} \chi_{k|k-1}^{(i)}.$$

(c) **Predicted Error Covariance**:

$$\mathbf{P}_{k|k-1} = \sum_{i=0}^{2n} W_c^{(i)} \left(\chi_{k|k-1}^{(i)} - \hat{\mathbf{x}}_{k|k-1}\right) \left(\chi_{k|k-1}^{(i)} - \hat{\mathbf{x}}_{k|k-1}\right)^\top + \mathbf{Q}_{k-1}.$$

4. **Update Step**:

(a) **Propagate Sigma Points through the Observation Function**:

$$\gamma_k^{(i)} = \mathbf{h}(\chi_{k|k-1}^{(i)}).$$

(b) **Predicted Measurement**

$$\hat{\mathbf{y}}_{k|k-1} = \sum_{i=0}^{2n} W_m^{(i)} \gamma_k^{(i)}.$$

(c) **Innovation Covariance**:

$$\mathbf{S}_k = \sum_{i=0}^{2n} W_c^{(i)} \left(\gamma_k^{(i)} - \hat{\mathbf{y}}_{k|k-1}\right) \left(\gamma_k^{(i)} - \hat{\mathbf{y}}_{k|k-1}\right)^\top + \mathbf{R}_k.$$

(d) **Cross-Covariance Matrix**:

$$\mathbf{P}_{xy} = \sum_{i=0}^{2n} W_c^{(i)} \left(\chi_{k|k-1}^{(i)} - \hat{\mathbf{x}}_{k|k-1}\right) \left(\gamma_k^{(i)} - \hat{\mathbf{y}}_{k|k-1}\right)^\top.$$

(e) **Kalman Gain**:

$$\mathbf{K}_k = \mathbf{P}_{xy}\mathbf{S}_k^{-1}.$$

5. **Updated State Estimate**:

$$\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k \left(\mathbf{y}_k - \hat{\mathbf{y}}_{k|k-1}\right).$$

6. **Updated Error Covariance**:

$$\mathbf{P}_{k|k} = \mathbf{P}_{k|k-1} - \mathbf{K}_k\mathbf{S}_k\mathbf{K}_k^\top.$$

So, generally, the UKF algorithm uses a deterministic sampling approach to select a minimal set of sample points [10] which capture the true mean and covariance of the state distribution. Said points are propagated through the nonlinear functions, and the posterior mean and covariance are then reconstructed from the transformed points.

Ensemble Kalman Filter (EnKF)

The Ensemble Kalman Filter (EnKF) is a recursive data assimilation technique designed for high-dimensional state estimation problems. It uses an ensemble of model states to approximate the probability distribution of the system state and update these states based on new observational data. More specifically, it utilizes an ensemble of state vectors to represent the state distribution, updating the ensemble members using the Kalman filter equations applied to the ensemble statistics, therefore it can handle nonlinear models via propagating the ensemble through the nonlinear system dynamics, it reduces the computational complexity comparing to particle filter [11] and, what is quite important, it assumes that the errors are approximately Gaussian and that the covariance between the state variables can be estimated from the ensemble [15].

Considering a discrete-time nonlinear dynamic system:

1. **State Transition Model**:

$$\mathbf{x}_k = \mathbf{f}(\mathbf{x}_{k-1}) + \mathbf{w}_{k-1},$$

where:

(a) $\mathbf{x}_k \in \mathbf{R}^n$ is the state vector at time k.

(b) $\mathbf{f}(\cdot)$ is a nonlinear state transition function.

(c) $\mathbf{w}_{k-1} \sim \mathcal{N}(\mathbf{0}, \mathbf{Q}_{k-1})$ is the process noise.

2. **Observation Model**:

$$\mathbf{y}_k = \mathbf{h}(\mathbf{x}_k) + \mathbf{v}_k,$$

where:

(a) $\mathbf{y}_k \in \mathbf{R}^m$ is the observation vector at time k.

(b) $\mathbf{h}(\cdot)$ is a nonlinear observation function.

(c) $\mathbf{v}_k \sim \mathcal{N}(\mathbf{0}, \mathbf{R}_k)$ is the observation noise.

[10] so-called sigma points

[11] Particle filter algorithm is discussed in the next subsection

Then, the general EnKf algorithm can be described as following:

The EnKF operates by maintaining an ensemble of N state vectors $\{\mathbf{x}_k^{(i)}\}_{i=1}^N$ that represent the state distribution at time k.

1. **Initialization** - we generate an initial ensemble $\{\mathbf{x}_0^{(i)}\}_{i=1}^N$ by sampling from the initial state distribution $p(\mathbf{x}_0)$.
2. **For each time step k, perform the following steps**:
   (a) **Forecast (Prediction) Step**:
      i. **Propagate Each Ensemble Member Through the Model**:

$$\mathbf{x}_{k|k-1}^{(i)} = \mathbf{f}(\mathbf{x}_{k-1|k-1}^{(i)}) + \mathbf{w}_{k-1}^{(i)}, \quad i = 1, 2, \ldots, N,$$

      where $\mathbf{w}_{k-1}^{(i)}$ are independent samples from the process noise distribution $\mathcal{N}(\mathbf{0}, \mathbf{Q}_{k-1})$.
      ii. **Compute the Forecast Ensemble Mean and Covariance**: **Ensemble Mean**:

$$\bar{\mathbf{x}}_{k|k-1} = \frac{1}{N} \sum_{i=1}^{N} \mathbf{x}_{k|k-1}^{(i)}.$$

      Ensemble Covariance:

$$\mathbf{P}_{k|k-1} = \frac{1}{N-1} \sum_{i=1}^{N} \left(\mathbf{x}_{k|k-1}^{(i)} - \bar{\mathbf{x}}_{k|k-1}\right) \left(\mathbf{x}_{k|k-1}^{(i)} - \bar{\mathbf{x}}_{k|k-1}\right)^\top.$$

   (b) **Analysis (Update) Step**:
      i. **Perturb Observations (Optional)**: To account for the observational error in the ensemble, perturb the observations:

$$\mathbf{y}_k^{(i)} = \mathbf{y}_k + \mathbf{v}_k^{(i)}, \quad \mathbf{v}_k^{(i)} \sim \mathcal{N}(\mathbf{0}, \mathbf{R}_k).$$

      ii. **Compute the Kalman Gain Using the Ensemble Covariances**:
         A. **Observation Ensemble**: Propagate the forecast ensemble through the observation function:

$$\mathbf{y}_{k|k-1}^{(i)} = \mathbf{h}(\mathbf{x}_{k|k-1}^{(i)}).$$

         B. **Ensemble Observation Mean**:

$$\bar{\mathbf{y}}_{k|k-1} = \frac{1}{N} \sum_{i=1}^{N} \mathbf{y}_{k|k-1}^{(i)}.$$

         C. **Innovation Covariance**:

$$\mathbf{P}_{yy} = \frac{1}{N-1} \sum_{i=1}^{N} \left(\mathbf{y}_{k|k-1}^{(i)} - \bar{\mathbf{y}}_{k|k-1}\right) \left(\mathbf{y}_{k|k-1}^{(i)} - \bar{\mathbf{y}}_{k|k-1}\right)^\top + \mathbf{R}_k.$$

D. **Cross-Covariance Matrix**:

$$\mathbf{P}_{xy} = \frac{1}{N-1} \sum_{i=1}^{N} \left(\mathbf{x}_{k|k-1}^{(i)} - \bar{\mathbf{x}}_{k|k-1} \right) \left(\mathbf{y}_{k|k-1}^{(i)} - \bar{\mathbf{y}}_{k|k-1} \right)^{\top}.$$

E. **Kalman Gain**:

$$\mathbf{K}_k = \mathbf{P}_{xy} \mathbf{P}_{yy}^{-1}.$$

iii. **Update Each Ensemble Member**: Without observation perturbations:

$$\mathbf{x}_{k|k}^{(i)} = \mathbf{x}_{k|k-1}^{(i)} + \mathbf{K}_k \left(\mathbf{y}_k - \mathbf{y}_{k|k-1}^{(i)} \right).$$

With observation perturbations:

$$\mathbf{x}_{k|k}^{(i)} = \mathbf{x}_{k|k-1}^{(i)} + \mathbf{K}_k \left(\mathbf{y}_k^{(i)} - \mathbf{y}_{k|k-1}^{(i)} \right).$$

3. **Repeat for the Next Time Step**

Basically, the whole concept of ensemble Kalman filter can be defined in few simple steps [15]:

1. **Forecast Step** - Each ensemble member propagates through a nonlinear state transition function including process noise, and then the prediction ensemble contains uncertainty in the state prediction due to model dynamics and process noise.

2. **Analysis Step** - The predicted set is transformed into an observation space using a non-linear observation function, and Kalman gain is calculated based on sample covariances that reflect the relationships between the state variables and observations. Each member of the ensemble is then updated by shifting towards the observed data with a weight given by the Kalman gain. Perturbations of the observations can be added to preserve the correct statistical properties of the set, especially when the observation operator is linear.

Essentially, the disadvantage of Kalman filter and its successor lays in the fact that in case of highly nonlinear dynamical system and with system where the noise is non-Gaussian the performance of said algorithm is far from perfect. As we remember from 2.1, one of our astrophysical scenarios is subsonic turbulence, which has both of mentioned above characteristics. For similar system the algorithm, which is described below, was developed.

2.5.6 Particle Filter

The Particle Filter (PF), also known as the Sequential Monte Carlo (SMC) method, is a simulation-based algorithm used to estimate the posterior distribution of a system's state when dealing with nonlinear and non-Gaussian processes [12]. Particle filters represent the posterior distribution using a set of particles (hypotheses) with associated weights. These hypotheses are propagated over time using the system's dynamics, and their weights are updated based on the likelihood of the observations. In this paper, from now on we refer to the weighted particles used in the particle filter as 'hypotheses' to avoid confusion with the physical particles discussed in earlier section 2.1.

As in EnKF, we consider the discrete-time nonlinear dynamic system, however we consider errors to be non-Gaussian (e.g. Laplace distribution):

1. **State Transition Model**:
$$\mathbf{x}_k = \mathbf{f}(\mathbf{x}_{k-1}, \mathbf{v}_{k-1}),$$
where $\mathbf{v}_{k-1}$ is the process noise.

2. **Observation Model**:
$$\mathbf{y}_k = \mathbf{h}(\mathbf{x}_k, \mathbf{n}_k),$$
where $\mathbf{n}_k$ is the observation noise.

Hence, the Particle Filter Algorithm can be defined as following:

1. **Initialization** - Generate N hypotheses $\{\mathbf{x}_0^{(i)}\}_{i=1}^N$ from the initial state distribution $p(\mathbf{x}_0)$. Then set initial weights $w_0^{(i)} = \frac{1}{N}$.

2. **For each time step** k:

   (a) **Prediction Step**:
   Sample hypotheses:
   For each hypothesis i:
   $$\mathbf{x}_k^{(i)} \sim p(\mathbf{x}_k \,|\, \mathbf{x}_{k-1}^{(i)}).$$

   (b) **Update Step**:

      i. **Compute Importance Weights**:
      $$w_k^{(i)} = w_{k-1}^{(i)} \cdot p(\mathbf{y}_k \,|\, \mathbf{x}_k^{(i)}).$$

      ii. **Normalize Weights**:
      $$\tilde{w}_k^{(i)} = \frac{w_k^{(i)}}{\sum_{j=1}^N w_k^{(j)}}.$$

   (c) **Resampling Step (Importance Resampling)**:

      i. Resample N hypotheses from the current set $\{\mathbf{x}_k^{(i)}\}$ according to the normalized weights $\tilde{w}_k^{(i)}$.

      ii. After resampling, reset the weights:
      $$w_k^{(i)} = \frac{1}{N}.$$

The PF approximates the posterior distribution $p(\mathbf{x}_k \,|\, \mathbf{y}_{1:k})$ by a set of hypotheses and their associated weights [41].

Basically, the whole concept of particle filtering lays in 4 simple steps [12, 1]:

1. **Initialization** - this step is semi-optional, since often there is a lack of information which would help the algorithm make a correct initialization, but if there is, usually hypotheses are initialized across some started position

2. **Prediction** - each hypothesis represents some possible state of the system. Hypotheses are propagated according to predefines system dynamics, capturing the evolution of the state over time.

3. **Update** - the importance weights are updated based on the likelihood of the observed measurement given the hypothesis's state.

4. **Resampling (Importance Resampling)** - for all versions of Kalman filter there is a looming issue which is called "filter degeneracy". The concept of filter degeneracy refers to a situation where the filter's performance deteriorates over time. Same is applicable to hypotheses of particle filter and is expressed in a situation, when after several iterations, most hypotheses may have negligible weights, so the estimate is realistically based on a few hypotheses from the whole set. Therefore we need some mechanism which would prevent that, and this mechanism can be defined as following:
   Hypotheses with higher weights are more likely to be selected, allowing the filter to focus computational resources on the more probable regions of the state space. There are several techniques, which will be specifically discussed in (((placeholder for chapter 3 link))).

The proposal distribution is often chosen as the state transition model $p(\mathbf{x}_k \mid \mathbf{x}_{k-1})$, and the importance weights correct for the discrepancy between the proposal distribution and the true posterior.

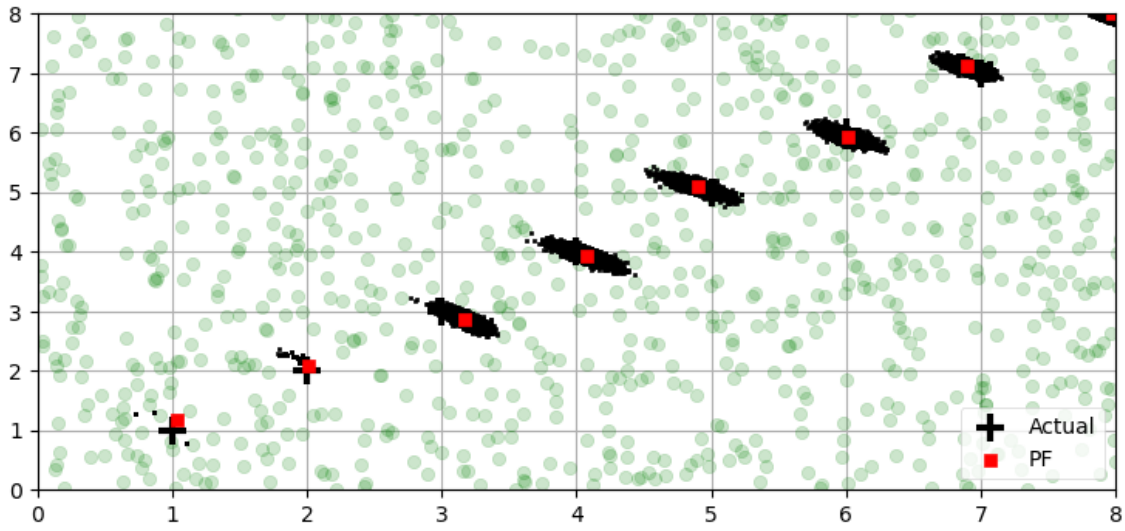


Figure 2.2: Visualization of the Particle filter method. Green dots represent original random distribution of points, black pixels represent the hypotheses of particle filter after the resampling step, black cross represents the actual position of a body and red square represents the estimated position using particle filter. Visualization is from [29]

Figure 2.2 shows the merged plot of 8 time steps into one (hence 8 actual values), where the object has a velocity of $(1, 1)$ in 2-Dimensional space starting from $(1, 1)$. At each time step one can see hypotheses of particle filter and it is easy to notice that it seems that in the first step there

are only 3 black dots, however, there are 5000 hypotheses of the position of the object. Then why would one see only 3? The reason for that is that during the resampling step these 3 positions were chosen multiple times. As we can see in further steps, there is more diversity, since more hypotheses have enough weight to be chosen during the resampling step.

2.5.7 Summary

In this section, modern filtering methods have been reviewed, each oriented towards solving specific problems of nonlinear dynamical systems with uncertain data [2], which is particularly relevant in the context of astrophysical simulations . Stochastic filtering, Monte Carlo methods, Bayesian filtering, and their further developments such as the Kalman filter, the extended and unweighted Kalman filters, and the ensemble Kalman filter are considered. All of them allow the estimation of the system state under conditions of significant noise and incomplete data.

Each of the methods is based on the principle of recursive prediction and updating of the system state. Depending on the level of nonlinearity, the type of error distribution and the available computational resources, one or the other method may be preferred. The particle filter, in particular, has demonstrated its effectiveness in the context of non-Gaussian and highly nonlinear systems [23], as in the case of SPH-EXA simulations, due to its adaptability and robustness to the problem of hypothesis degeneracy.

However, these filters are not a perfect answer to the questions we posed in the introduction because they are, suddenly, filters [27]. In our case, we get error-free and accurate information from the simulation. All the mentioned filters are designed to estimate the current hidden state of the system, but our goal is different, it is to try to predict its future state. That is why these methods can be the basis for our solution, but not the solution itself because the objective and constraints we have are different from the objectives and constraints of said methods.

2.6 Sequential Online Prediction

Sequential online prediction is a method which allows us to make real-time predictions about future state of the dynamic system as soon as new data becomes available [30]. This method is perfect for dynamic systems which evolve over time and for time-series based data, even in the presence of outliers and change points [31], hence, it is more than suitable for our case.

2.6.1 Definition

Consider a dynamic system described by a state-space model:

1. **State Transition Model**:
$$x_k = f(x_{k-1}, v_{k-1}),$$
where x_k is the hidden state at time k, $f(\cdot)$ is a possibly nonlinear state transition function, and v_{k-1} is the process noise.

2. **Observation Model**:
$$y_k = h(x_k, n_k),$$

where y_k is the observation at time k, $h(\cdot)$ is a possibly nonlinear observation function, and n_k is the observation noise.

At each time step k, the goal is to:

1. Estimate the current state: Compute $p(x_k|y_{1:k})$
2. Predict future states or observations: Compute $p(x_{k+1}|y_{1:k})$ or $p(y_{k+1}|y_{1:k})$

2.6.2 Particle Filters in Sequential Online Prediction

Naturally, Particle filters are a popular method for sequential online prediction, especially for non-linear and non-Gaussian systems. As it was described in 2.5.6 they approximate the posterior distribution of the state using a set of weighted particles [12, 14].

In the context of sequential online prediction:

1. **State Estimation**: Hypotheses represent possible current states of the system.
2. **Prediction**: Hypotheses are propagated through the state transition model to predict future states.
3. **Update**: Hypotheses weights are updated based on new observations.
4. **Resampling**: Hypotheses are resampled to prevent degeneracy.

2.6.3 Prediction with Ground Truth Observations

In our case, where we have access to ground truth observations, the sequential online prediction process can be enhanced. The approximate algorithm for the case when one has ground truth observations can look like this:

1. **Prediction**: At each time step, we use our current model (e.g., particle filter) to make a prediction for the next state or observation.
2. **Comparison**: We compare our prediction with the ground truth observation which comes from the simulation.
3. **Error Calculation**: We compute the prediction error, so we know can compute the weights for importance resampling
4. **Model Update**: We update hypotheses weights based on the likelihood of the ground truth observation and then we do importance resampling
5. **Performance Metrics**: We also calculate the error of out prediction, so we can evaluate our model and compare.

2.7 Conclusion

It is clear, that there is no perfect method which would be perfectly suitable for our case of multidimensional highly-dynamic non-linear system. Bottleneck is that it has ground truth values, hence eliminating proper use case of filtering algorithms. However, combining different methods and approaches in order to leverage their strengths and avoid their limitations is possible, which we are essentially trying achieve. In the next chapter our algorithm is explained using all the background information described in this chapter.

Chapter 3

Sequential Online Prediction with Particle Importance Resampling

3.1 Introduction

In this section, we present a detailed methodology of our proposed approach, named **Sequential Online Prediction with Particle Importance Resampling (SOPPIR)**. We design SOPPIR in a way that it combines the sequential online prediction with particle (hypothesis) importance resampling techniques, hence the name, in order to predict the future states of particles in a dynamic system. SOPPIR estimates predictions before incorporating new observations, allowing for real-time forecasting in systems where ground truth observations are available after prediction.

3.2 Overview of SOPPIR

The SOPPIR algorithm operates through the following steps:

1. **Initialization**: We generate an initial set of hypotheses around the some initial positions of the particle, incorporating uncertainty through noise.
2. **Prediction**: We propagate the hypotheses forward in time using a simplified physical model without incorporating the new observation yet.
3. **Estimation**: We estimate the predicted positions by computing the mean of all hypotheses.
4. **Observation**: We obtain the ground truth positions from the simulation.
5. **Weight Update**: We update the weights of the hypotheses based on their distances to the ground truth.
6. **Resampling**: We perform the resampling algorithm of hypotheses based on the updated weights using importance resampling.
7. **Repeat**: At the end of the day, we iterate the process across each particle for each time step.

This sequence emphasizes making predictions before incorporating ground truth observations, differing from standard particle filter implementations. It is important to notice, that contrary to traditional particle filter algorithm we first estimate and only then proceed with weight update and resampling. Similarly to Bayesian filtering methods, SOPPIR relies on Markov property, for more detailed explanation see 2.5.3.

3.3 Detailed Algorithm Steps

3.3.1 Initialization

Generate an initial set of hypotheses around the initial positions, introducing uncertainty to account for possible deviations:

1. For each particle in the simulation, let $\mathbf{p}_0$ represent its initial position.
2. Generate N hypotheses by adding random noise to $\mathbf{p}_0$:

$$\mathbf{x}_0^{(i)} = \mathbf{p}_0 + \boldsymbol{\epsilon}_0^{(i)}, \quad i = 1, 2, \ldots, N,$$

where $\boldsymbol{\epsilon}_0^{(i)}$ is sampled from a noise distribution (e.g., Gaussian or Laplace) [1] to represent initial uncertainty.

(a) **Gaussian Noise**:

$$\epsilon^{(i)} \sim \mathcal{N}(\mathbf{0}, \sigma_{\text{init}}^2 \mathbf{I}),$$

(b) **Laplace Noise**:

$$\epsilon^{(i)} \sim \text{Laplace}(\mathbf{0}, b\mathbf{I}), \quad b = \frac{\sigma_{\text{init}}}{\sqrt{2}}.$$

3.3.2 Prediction

Propagate the hypotheses forward using a simplified physical model to predict their next positions without considering new observations:

1. At each time step k, for each hypothesis i, update its position based on the model:

$$\mathbf{x}_k^{(i)} = \mathbf{x}_{k-1}^{(i)} + \mathbf{v}_{k-1}\Delta t_k + \boldsymbol{\eta}_{k-1}^{(i)},$$

where:

- $\mathbf{v}_{k-1}$ is the known velocity at time $k-1$.
- Δt_k is the time interval between steps $k-1$ and k.
- $\boldsymbol{\eta}_{k-1}^{(i)}$ is the process noise, sampled from a noise distribution to account for model uncertainty, which is again can be defined using either Gaussian or Laplacian distribution:

  (a) **Gaussian Noise**:

$$\epsilon^{(i)} \sim \mathcal{N}(\mathbf{0}, \sigma_{\text{init}}^2 \mathbf{I}),$$

  (b) **Laplace Noise**:

$$\epsilon^{(i)} \sim \text{Laplace}(\mathbf{0}, b\mathbf{I}), \quad b = \frac{\sigma_{\text{init}}}{\sqrt{2}}.$$

[1] Gaussian noise is often referred to as "normal" and Laplace noise is a type of heavy-tailed noise

3.3.3 Estimation

Compute the mean of all hypotheses to estimate the predicted position:

$$\hat{\mathbf{x}}_k = \frac{1}{N} \sum_{i=1}^{N} \mathbf{x}_k^{(i)}.$$

3.3.4 Observation

Obtain the actual position $\mathbf{p}_k$ of the particle from the simulation data at time k.

3.3.5 Weight Update

Update the weights of the hypotheses based on their agreement with the ground truth, emphasizing hypotheses closer to the actual position:

1. For each hypothesis i, compute its weight based on the distance to the ground truth:

$$w_k^{(i)} = \exp\left(-\frac{\|\mathbf{x}_k^{(i)} - \mathbf{p}_k\|^2}{2\sigma_o^2} \right),$$

   where σ_o^2 is the observation noise variance representing the confidence in the observations.

2. Normalize the weights to ensure they sum to one:

$$\tilde{w}_k^{(i)} = \frac{w_k^{(i)}}{\sum_{j=1}^{N} w_k^{(j)}}.$$

3.3.6 Resampling

Essentially, after the weight update step, where each hypothesis has an updated and normalized weight reflecting how far it is from the ground truth observation, we proceed to resample the hypotheses. This resampling step ensures that hypotheses with higher weights (i.e., those closer to the actual observed state) are more likely to be chosen to stay, while those with lower weights are discarded. This is the key step of our algorithm which prevents the degeneracy of our model and allows to incorporate all the dynmacist of the system. In our algorithm, we implement two resampling methods: **multinomial resampling** and **residual resampling**. These algorithm were chosen because of their complexity and because they allows us to reach our goal the best.

Random Resampling (Multinomial Resampling)

Multinomial resampling is a simple method where each hypothesis is resampled according to its normalized weight, hence the probability of selecting a specific hypothesis is directly proportional to its weight. Hypotheses with higher weights are more likely to be selected multiple times, while those with lower weights may not be selected at all.

Algorithm 1 Multinomial Resampling

Require: Hypotheses $\{\mathbf{x}_k^{(i)}\}_{i=1}^N$, normalized weights $\{\tilde{w}_k^{(i)}\}_{i=1}^N$
Ensure: Resampled hypotheses $\{\mathbf{x}_k^{(i)}\}_{\text{resampled}}$ with equal weights

Compute cumulative sum of weights:
$C_0 \leftarrow 0$
for $j = 1$ to N **do**
    $C_j \leftarrow C_{j-1} + \tilde{w}_k^{(j)}$
end for

Resampling:
for $i = 1$ to N **do**
    Generate random number u uniformly distributed in $[0, 1)$
    Find the smallest j such that $C_j \geq u$
    Assign $\mathbf{x}_k^{(i)} \leftarrow \mathbf{x}_k^{(j)}$
end for

Reset weights:
for $i = 1$ to N **do**
    $\tilde{w}_k^{(i)} \leftarrow \frac{1}{N}$
end for

Residual Resampling

Residual resampling reduces the variance introduced by resampling by first deterministically selecting integer copies of hypotheses based on their weights and then randomly distributing the residual (fractional) weights. This method ensures that said hypotheses with significant weights are retained proportionally before randomness is introduced.

In the essence of our algorithm we are free to choose any of described algorithms, the choice depends on the task and available resources. In our code we incorporate both approaches, which allows to compare them.

Algorithm 2 Residual Resampling

Require: Hypotheses $\{\mathbf{x}_k^{(i)}\}_{i=1}^N$, normalized weights $\{\tilde{w}_k^{(i)}\}_{i=1}^N$
Ensure: Resampled hypotheses $\{\mathbf{x}_k^{(i)}\}_{\text{resampled}}$ with equal weights

Compute expected counts:
for $i = 1$ to N **do**
    $N_i \leftarrow N \times \tilde{w}_k^{(i)}$
end for

Compute integer parts:
for $i = 1$ to N **do**
    $n_i \leftarrow \text{floor}(N_i)$
end for

Initialize resampled hypotheses list `ResampledHypotheses` as empty

Deterministic Assignment:
for $i = 1$ to N **do**
    for $j = 1$ to n_i **do**
        Add $x_k^{(i)}$ to `ResampledHypotheses`
    end for
end for

Compute residual weights:
for $i = 1$ to N **do**
    $r_i \leftarrow N_i - n_i$
end for
$R \leftarrow$ sum of all r_i

Normalize residual weights:
for $i = 1$ to N **do**
    $\tilde{r}_i \leftarrow \frac{r_i}{R}$
end for

Compute cumulative sum of residual weights:
$C_{\text{res},0} \leftarrow 0$
for $j = 1$ to N **do**
    $C_{\text{res},j} \leftarrow C_{\text{res},j-1} + \tilde{r}_j$
end for

Compute number of residual samples:
$N_{\text{res}} \leftarrow N -$ sum of all n_i

Residual Sampling:
for $i = 1$ to N_{res} **do**
    Generate random number u uniformly distributed in $[0, 1)$
    Find smallest j such that $C_{\text{res},j} \geq u$
    Add $x_k^{(j)}$ to `ResampledHypotheses`
end for

Reset weights:
for $i = 1$ to N **do**
    $\tilde{w}_k^{(i)} \leftarrow \frac{1}{N}$
end for

3.3.7 Repeat

Continue the process for subsequent time steps to iteratively predict and refine the hypotheses: For each time step $k = 1, 2, \ldots, T$ repeat the prediction, estimation, observation, weight update, and resampling steps.

SOPPIR effectively combines the advantages of sequential prediction and importance resampling, allowing for accurate and real-time forecasting in dynamic systems with available ground truth observations.

3.4 Algorithm Summary pseudocode

Finally, our algorithm can be summarized as the following pseudocode, as exception, here we will refer to our hypotheses as particles:

Algorithm 3 Sequential Online Prediction with Particle Importance Resampling (SOPPIR)

Require:
- Initial position p_0
- Velocities $\{v_k\}_{k=0}^{T-1}$
- Number of particles N
- Number of time steps T
- Time intervals $\{\Delta t_k\}_{k=1}^{T}$
- Process noise standard deviation σ_p
- Observation noise standard deviation σ_o

Ensure: Predicted positions $\{\hat{x}_k\}_{k=1}^{T}$

Initialization:
for $i = 1$ to N **do**
    Sample initial noise $\epsilon_0^{(i)}$ from the initial noise distribution
    Set $x_0^{(i)} \leftarrow p_0 + \epsilon_0^{(i)}$
end for
Set initial weights $\tilde{w}_0^{(i)} \leftarrow 1/N$ for all i

for $k = 1$ to T **do** ▷ Sequentially process each time step
    Prediction Step:
    for $i = 1$ to N **do**
        Scale process noise standard deviation by $\sqrt{\Delta t_k}$:

$$\sigma_p' \leftarrow \sigma_p \cdot \sqrt{\Delta t_k}$$

        Sample process noise $\eta_{k-1}^{(i)}$ from either normal or Laplace distribution with standard deviation σ_p'
        Update particle position $x_k^{(i)} \leftarrow x_{k-1}^{(i)} + v_{k-1} \cdot \Delta t_k + \eta_{k-1}^{(i)}$
    end for

    Estimation Step:
    Compute predicted position $\hat{x}_k$ as the average of all $x_k^{(i)}$:

$$\hat{x}_k \leftarrow \text{average of } x_k^{(i)} \text{ for } i = 1 \text{ to } N$$

    Estimation Step:
    Compute predicted position $\hat{x}_k$ as the average of all $x_k^{(i)}$:

$$\hat{x}_k \leftarrow \text{average of } x_k^{(i)} \text{ for } i = 1 \text{ to } N$$

    Observation Step:
    Obtain ground truth position p_k

Weight Update Step:
for $i = 1$ to N **do**
    Compute weight $w_k^{(i)}$ based on the Gaussian likelihood:

$$w_k^{(i)} \leftarrow \exp\left(-\text{square of } (x_k^{(i)} - p_k) \text{ divided by } (2 \cdot \sigma_o^2)\right) + \epsilon$$

    We need to add a small value $\epsilon = 1e\text{-}300$ to avoid division by zero
end for
Normalize weights:
Compute the sum of all $w_k^{(i)}$ for $i = 1$ to N
for $i = 1$ to N **do**
    Set $\tilde{w}_k^{(i)} \leftarrow w_k^{(i)}$divided by total sum of weights
end for

Resampling Step
if Using Multinomial Resampling **then**
    Compute cumulative sum of weights:
    Initialize $C_0 \leftarrow 0$
    for $j = 1$ to N **do**
        Set $C_j \leftarrow C_{j-1} + \tilde{w}_k^{(j)}$
    end for
    for $i = 1$ to N **do**
        Generate a random number u uniformly distributed in $[0, 1)$
        Find the smallest j such that $C_j \geq u$
        Assign $x_k^{(i)} \leftarrow x_k^{(j)}$
    end for

else if Using Residual Resampling **then**
    Compute expected counts for each particle:
    for $i = 1$ to N **do**
        Set $N_i \leftarrow N \cdot \tilde{w}_k^{(i)}$
    end for
    Compute integer parts of expected counts:
    for $i = 1$ to N **do**
        Set $n_i \leftarrow \text{floor}(N_i)$
    end for
    Initialize an empty list `ResampledSet`
    for $i = 1$ to N **do**
        for $l = 1$ to n_i **do**
            Add $x_k^{(i)}$ to `ResampledSet`
        end for
    end for
    Compute residual weights:
    for $i = 1$ to N **do**
        Set $r_i \leftarrow N_i - n_i$
    end for
    Normalize residual weights:
    Compute the sum of all residual weights
    for $i = 1$ to N **do**
        Set $\tilde{r}_i \leftarrow r_i$ divided by the total sum of residual weights
    end for
    Compute cumulative sum of normalized residual weights:
    Initialize $C_{\text{res},0} \leftarrow 0$
    for $j = 1$ to N **do**
        Set $C_{\text{res},j} \leftarrow C_{\text{res},j-1} + \tilde{r}_j$
    end for
    Compute the number of residual samples:
    Set $N_{\text{res}} \leftarrow N$ minus the sum of all n_i
    for $i = 1$ to N_{res} **do**
        Generate a random number u uniformly distributed in $[0, 1)$
        Find the smallest j such that $C_{\text{res},j} \geq u$
        Add $x_k^{(j)}$ to `ResampledSet`
    end for
    Replace particles:
    Set $\{x_k^{(i)}\}_{i=1}^N \leftarrow$ `ResampledSet`
end if
Reset weights:
for $i = 1$ to N **do**
    Set $\tilde{w}_k^{(i)} \leftarrow 1/N$
end for
end for ▷ End of sequential processing for all time steps

It is important to note, that in this pseudocode we do not include 3 things:

1. **Multiprocessing** - in our code we have incorporated general multiprocessing techniques. Since we have 2 versions of the code - GPU and CPU, hence the approach is slightly different and it does not change the overall algorithm.

2. **3D-space and boundaries** - in our case we work with 3-dimensional space and our particles are bounded in [-0.5, 0.5] box for each of three dimensions, but, again, it does not change the overall algorithm, as we just apply the modulo operation at every initialization and prediction step.

3. **Output** - in our code we save the results in HDF5 file, since it's one of the constraints.

3.5 Nuances and splitting the time steps

It is easy to notice that in our pseudocode we mention *all time steps*. We do it since we modified it in order to work with already saved data which mimics online obtainment of the system state, however, in real online prediction application, there is no need in processing all previous steps due to Markov Property and one can start working with next steps right away. Same applies to velocities.

As one can recall, one of two goals which we wanted to achieve was to make our time steps more frequent, e.g. knowing the data for each $\Delta t_k = 1, 2, \ldots, T$ we would like to know the positions at time steps $\Delta t_k = 1.5, 2.5, \ldots, T$ or any other arbitrary split, which can help researches to create more smooth visualizations. In our case it's easy to achieve, since we just can split Δt_k into few smaller Δt_k^i and propagate our algorithm on these steps. It is obvious that we will accumulate any error, but, as we will see in the results chapter, order of error is quite small, hence we can do it without significant deteriorations of results from visualization point of few - it will be almost impossible to see that some body is slightly off it's true position, taking into account the magnitude of phenomena.

3.6 Conclusion

The SOPPIR method is our approach of using the sequential online prediction for a dynamic system. By hypothesizing several predictions before any observations were made and subsequently improving the hypotheses in accordance with what was observed, SOPPIR manages to provide ways of ensuring that immediate forecasts are made and new information is used for improvements to predictions. The application of particle importance resampling also makes it possible to direct most computation resources towards the most likely participants which enhances both the performance and efficiency of the system.

Chapter 4

Results

4.1 Experimental Design

Table 4.1: Design of factorial experiments, total of 48 experiments.

Factors	**Values**	**Properties**
SPH-EXA TestCases	-n 125000 particles -s 10.0	Sedov Blast simulated for 10.0 seconds physical time for ? time-steps
	-n 125000 particles -s 10.0	Subsonic Turbulence simulated for 10.0 seconds physical time for ? time-step
Prediction techniques	Sequential Online Prediction with Particle Importance Resampling	200 — 500 — 1000 particles (hypotheses), Random — Residual resampling, Gaussian — Laplacian noise
Computing nodes	miniHPC Cascade Lake	2 x 28 Core Intel Xeon Gold 6258R CPU with 1.5 TB Memory 2 x Nvidia A100-PCIE with 40GB Memory
Metrics	NRMSE	Root mean square error of the model, RMSE = $\sqrt{\frac{1}{m \cdot n} \sum_{j=1}^{m} \sum_{i=1}^{n} \left(X_{i,j,\text{actual}} - X_{i,j,\text{predicted}}\right)^2}$ Normalized RMSE of the model, NRMSE = $\frac{\text{RMSE}}{\text{Range}} \times 100$
	Time	Total time taken to run the algorithm

Each experiment tests a specific combination of three parameters: type of noise, resampling algorithm, and the number of hypotheses. These combinations are applied to both Sedov blast and subsonic turbulence cases using both CPU and GPU implementations.

As can be seen in table 4.1, we have a total of 48 experiments, 24 of which are performed on GPU and 24 on CPU. We have NRMSE of the model and time it takes to run the algorithm as our metrics and 3 main parameters of our algorithm:

1. **Type of noise** - Gaussian represents the normal distribution of the noise and Laplacian represents heavy-tailed noise, which should perform better due to non-Gaussian nature of our phenomena.

2. **Resampling algorithm** - As we defined in Resampling, we have 2 algorithms of resampling which are slightly different and we expect residual residual resampling to perform better than random resampling.

3. **Number of hypotheses** - Essentially the number of hypotheses which we use to predict the state of the system, as it is defined in Particle Filter and Sequential Online Prediction with

Particle Importance Resampling. General assumption is that the more hypotheses we have, the more precise our prediction should be, but, of course, with some limit.

4.1.1 Metrics

To evaluate our method we need some metric. In our case we've chosen Root mean square error (RMSE) of prediction, which is quite common metric for many similar problems, and which is a well-suited choice in our case as it would represent how far is our prediction from the ground truth position[19, 30, 46, 9].
RMSE can be calculated as following:

$$\text{RMSE} = \sqrt{\frac{1}{m \cdot n} \sum_{j=1}^{m} \sum_{i=1}^{n} \left(X_{i,j,\text{actual}} - X_{i,j,\text{predicted}}\right)^2}$$

where

1. m is the number of particles.
2. n is the number of time steps.
3. $X_{i,j,\text{actual}}$ is the actual value of particle j at time step i.
4. $X_{i,j,\text{predicted}}$ is the predicted value of particle j at time step i

To make the results even more clear, we will calculate Normalized RMSE which is RMSE of the model divided by the range of our values. As it was mentioned in 2.3, our particles are moving in the 3-Dimensional cubic box with values for each dimensions in the range of $[-0.5, 0.5]$, hence, our range is exactly 1.0. Therefore, to convert our RMSE into NRMSE which can be expressed in percentage, we need to do the following calculations:

$$\text{NRMSE} = \frac{\text{RMSE}}{\text{Range}} \times 100$$

As a basis we can see how our algorithms performs without implying the noise, but with particle importance resampling. For this reason, in the code there is a *null-noise* option, where we don't incorporate any noise neither for initialization of hypotheses, we refer to this model as null-noise model and use it as a reference to evaluate our algorithm. This model results with NRMSE of 24% for turb-50 case and NRMSE of 29% for sedov-50 case. Both this number are natural, since in turb-50 case we are missing around 20% of the data and in case of sedov we miss more, but the nature of shockwave phenomena is much more linear and less chaotic, hence we won't miss as much changes as in turb-50 case. We can use both of this number to see how actually good our prediction is.

It is also important to remember that, since our prediction are online, we need to care about time of execution of our code. Hence, we also need to compare amount of time spent on the executing different methods (e.g. calculations for residual resampling are more complex than for random resampling) to understand if it's effective.

4.2 Experimental Evaluation

4.2.1 Evaluation of Accuracy

Our algorithm shows significant improvement comparing to null-noise model. As we can see from resulting tables, for sedov-50 NRMSE is in the range of $[4.35 - 4.64]\%$ and for turb-50 it stays in the range of $[2.56 - 2.78]\%$ comparing to NRMSE for null-noise models of 29% and 24% respectfully. Since the codes are fundamentally similar (because they represent the same algorithm), there is no significant difference between NRMSE performance of CPU and GPU versions of the code and the existing difference can be explained by the fact that we needed to write our own functions for random for GPU code, since numba does not support numpy.random.

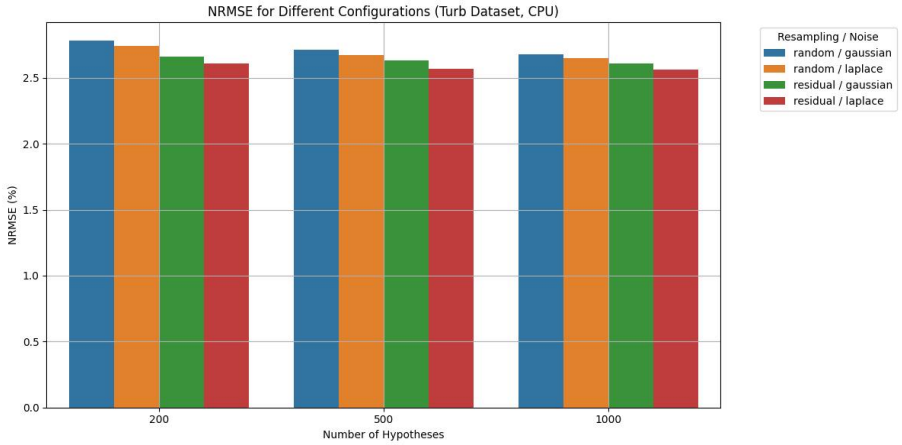


Figure 4.1: Comparison of NRMSE for different configurations without null-noise model for turb-50

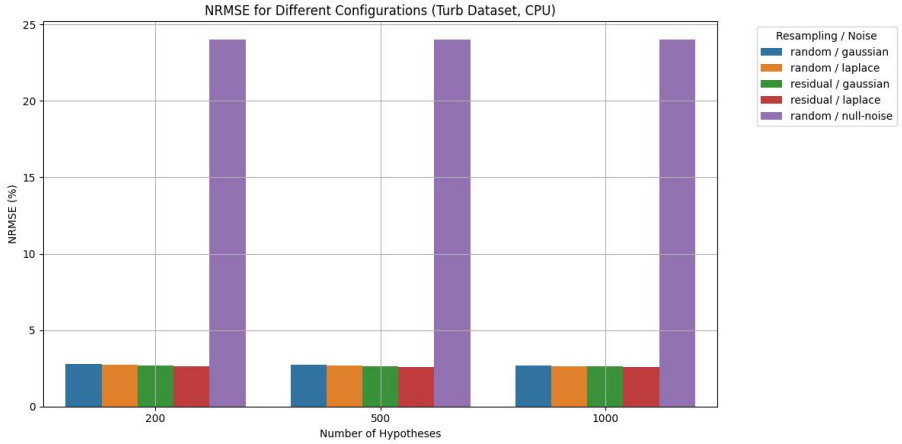


Figure 4.2: Comparison of NRMSE for different configurations with null-noise model for turb-50

From the NRMSE perspective for subsonic turbulence, as we can see in Figure 4.1 and 4.2, there is no significant difference between various parameters. As we expected, the more hypotheses we have, the lower NRMSE it is, same applies to type of noise and resampling algorithm - Laplacian performs better than Gaussian and residual performs better than random. However, the difference is not as significant as one would expect, hence this moment deserves additional investigation in future work.

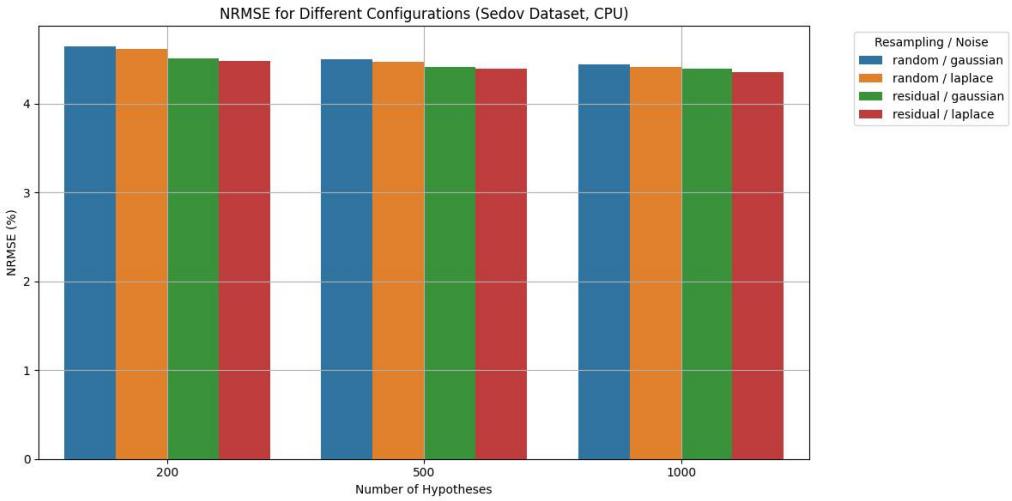


Figure 4.3: Comparison of NRMSE for different configurations without null-noise model for sedov-50

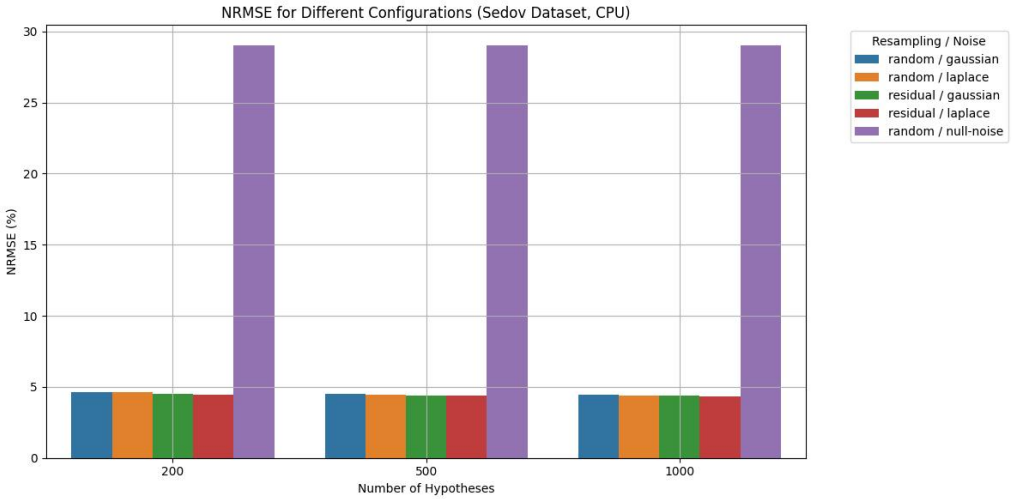


Figure 4.4: Comparison of NRMSE for different configurations with null-noise model for sedov-50

From the NRMSE perspective for Sedov blast, as we can see in Figure 4.3 and 4.4, general trend is the same as for subsonic turbulence with one difference - NRMSE is higher overall. It is quite clear, that our algorithm performs worse in the case where we miss 80% of the observations and improvement was slightly worse - in subsonic turbulence case we reduced the NRMSE by almost 90% - from 24% to 2.56% and in sedov blast we reduced NRMSE by 85% - from 29% to 4.45%. Overall trend with differences between parameters stays the same - residual resampling, Laplacian

noise and bigger number of hypotheses show better results than their counterparts.

It it important to note that even for specific particles which were the most difficult to predict, the NRMSE did not exceed 9% for sedov-50 and 7% for turb-50 dataset, hence, given the fact that NRMSE across all particles is around 4.35% and 2.56% respectfully, we can confidently say that said particles are outliers and that our algorithm works well even for the very complex cases of said simulations.

4.2.2 Evaluation of Performance

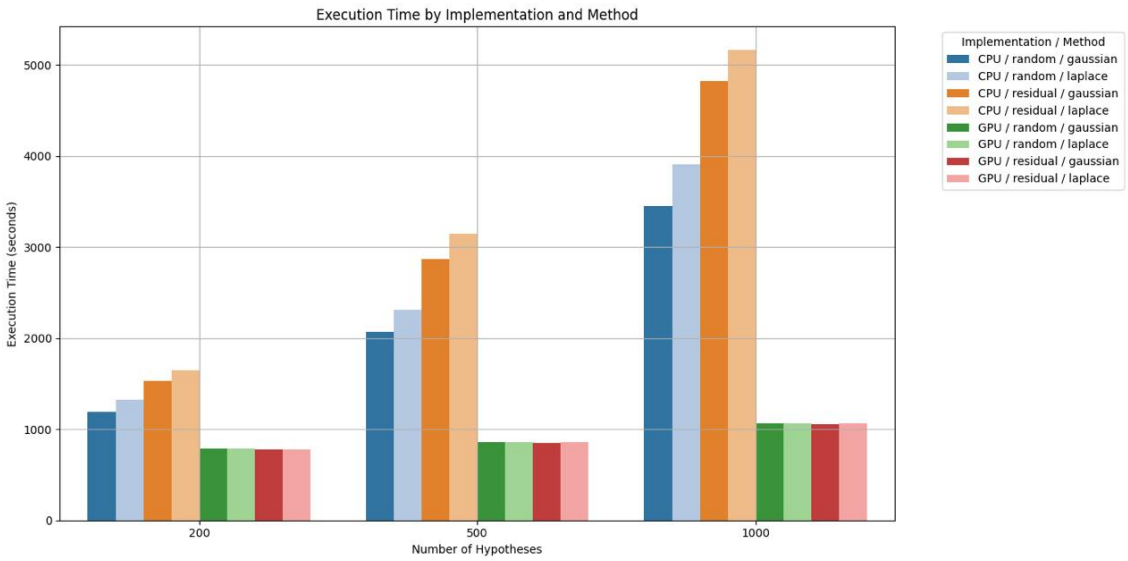


Figure 4.5: Execution time by Implementation and Method

Nevertheless, from the execution time perspective we can see in Figure 4.5 that GPU code is much more effective than CPU, as it was expected, and it scales better with higher number of hypotheses, which can be explained by JIT implementation of the code. As one can see, residual resampling and Laplacian noise take slightly more time (which is more visible at CPU part) comparing to random and Gaussian. The reason for that is quite trivial - Laplacian noise takes slightly less time to generate comparing to Gaussian. Albeit residual resampling should be faster than random, in our case it takes more time, probably due to imperfection of the code.

4.3 Summary

We evaluated the SOPPIR algorithm to predict the movement of particles sedov-50 and turb-50 scenarios with significant success. For the Sedov blast, the method demonstrated low prediction errors, with NRMSE values consistently below 5% comparing to 29% of null-noise model. SOPPIR

performed even better for subsonic turbulence, consistently demonstrating NRMSE below 3% comparing to 24% NRMSE of null-noise model. The implementation revealed that, while the overall algorithm performs well, the accuracy is not sensitive to noise and resampling techniques - at least in our case. From the time perspective, the GPU code for 200 hypotheses demonstrated the highest speed of predictions with lowest value being 755 seconds for prediction of a 1000 time steps.

These finding suggest that particle-based methods in combination with sequential online prediction techniques can be effectively incorporated in astrophysical simulations and related problems. It is clear, that SOPPIR can be used to fill in the gaps in data and make time steps more frequent which with relatively small errors which can be used by the research community.

Resulting comparison tables can be found in the Appendix.

Chapter 5

Conclusion

In this thesis we explored the applicability of time series analysis for astrophysical simulations, in particular applicability of Bayesian filters methods in combination with sequential online prediction. It focused specifically on SPH-EXA simulation for 2 scenarios - Sedov blast and subsonic turbulence. SOPPIR as a combination of particle filter and sequential online prediction methods demonstrated effectiveness of said methods in predicting the movement of particles in highly dynamic and chaotic systems. Our algorithm showed significant improvement over null-noise models and highlighted the possibility to fill in the possible gaps in the data and making time steps more frequent.

Sedov blast scenario posed a significant challenge, as it is missing around 80% of observations, however the resulting NRMSE of our model, which does not exceed 5%, shows a potential which can be expanded in this direction. These results show a suitability of SOPPIR method for predicting the movement of particles in linear-like scenarios with high volumes of missing data.

On the other hand, for subsonic turbulence scenario, which is much more chaotic and dynamic process, resulting NRMSE of our model does not exceed 3% even for the fastest and least complex versions of our algorithm.

We believe that these results show suitability of SOPPIR method for dynamic and chaotic systems as well. While in subsonic turbulence scenario we were missing as much as 20% of observations, it is possible to apply SOPPIR to more harsh conditions, as we did for sedov blast with 80% of observations missing, and, obviously, fill in the missing data and make time steps more frequent.

5.1 Future work

It is clear that there is a room of improvement of our method which would address the current limitations of our method:

1. **Other types of Noise** - in our study we assumed the usage of Laplacian and Gaussian noise which were quite effective, albeit showed almost no significant difference. Future work could explore other types of noise which would be more suitable for specific task.

2. **Observation Models** - in our study we used simplified observation model of astrophysical phenomena and it was one of the constraints of our method. However, future work may explore more complex observation models which could be closer to real model in their definition.

3. **Hybrid Prediction Techniques** - we were focusing mostly on particle filter as a basis for our technique, since it is the most suitable method for dynamic systems with non-Gaussian noise, however, in future work it is possible to combine our method with other filtering or prediction techniques.

4. **Code implementation** - while our code was written using Python, our goal was not to make it as effective as possible due to complexity of such task. In future work the code can be refactored to be more effective, or even rewritten on other programming languages which are faster than Python - for example, C++.

5. **Particle smoother** - while we suggested that our code can be used to fill in the missing observations and make time steps more frequent, future work could investigate into particle smoother algorithms, which can increase the precision of the data which will be used in generated time steps.

6. **Integration into SPH-EXA Framework** - finally, future work can incorporate our findings and algorithms in the SPH-EXA framework in order to increase the quality of the data and increase the resource allocation efficiency.

Bibliography

[1] M Sanjeev Arulampalam, Simon Maskell, Neil Gordon, and Tim Clapp. A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking. *IEEE Transactions on signal processing*, 50(2):174–188, 2002.

[2] Alan Bain and Dan Crisan. *Fundamentals of Stochastic Filtering*. Stochastic Modelling and Applied Probability. Springer New York, NY, 2009.

[3] Morgane Barbet-Massin, Quentin Rome, Claire Villemant, and Franck Courchamp. Can species distribution models really predict the expansion of invasive species? *PloS one*, 13(3):e0193085, 2018.

[4] Javier Blanco. Time series analysis: a gentle introduction. Blog post on quix.io, 2024.

[5] George EP Box, Gwilym M Jenkins, Gregory C Reinsel, and Greta M Ljung. *Time Series Analysis: Forecasting and Control*. John Wiley & Sons, 2015.

[6] Peter J Brockwell and Richard A Davis. *Introduction to time series and forecasting*. Springer, 2016.

[7] Ruben Cabezon et al. Sph-exa code. https://github.com/unibas-dmi-hpc/SPH-EXA, 2018.

[8] Herbert B Callen. *Thermodynamics and an Introduction to Thermostatistics*. John Wiley & Sons, 1985.

[9] Cheng Chang and R. Ansari. Kernel particle filter for visual tracking. *IEEE Signal Processing Letters*, 12(3):242–245, 2005.

[10] Zhe Chen. Bayesian filtering: From kalman filters to particle filters, and beyond. *Statistics*, 2003.

[11] PASC Consortium. Sph-exa2 project summary. *PASC Platform for Advanced Scientific Computing*, 2021.

[12] Arnaud Doucet and Adam M Johansen. A tutorial on particle filtering and smoothing: Fifteen years later. *Handbook of nonlinear filtering*, 12(656-704):3, 2009.

[13] Robert Ecke. The turbulence problem: An experimentalist's perspective. *Los Alamos Science*, 29:124–141, 2005.

[14] Víctor Elvira et al. Adapting the number of particles in sequential monte carlo methods through an online scheme for convergence assessment. *arXiv preprint arXiv:1509.04879*, 2017.

[15] Geir Evensen. The ensemble kalman filter: theoretical formulation and practical implementation. *Ocean Dynamics*, 53:343–367, 2003.

[16] Gene F Franklin, J David Powell, and Abbas Emami-Naeini. *Feedback control of dynamic systems*. Pearson, 2014.

[17] Belinda Gallardo, David C Aldridge, Pablo Gonz'alez-Moreno, Jan Pergl, Manuel Pizarro, Petr Pyšek, Wilfried Thuiller, Christopher Yesson, and Montserrat Vil'a. Protected areas offer refuge from invasive species spreading under climate change. *Global change biology*, 23(12):5331–5343, 2017.

[18] RA Gingold and JJ Monaghan. Smoothed particle hydrodynamics: theory and application to non-spherical stars. *Monthly notices of the royal astronomical society*, 181(3):375–389, 1977.

[19] Tilmann Gneiting and Adrian E Raftery. Strictly proper scoring rules, prediction, and estimation. *Journal of the American Statistical Association*, 102(477):359–378, 2007.

[20] Geoffrey Grimmett and David Stirzaker. *Probability and random processes*. Oxford University Press, 2001.

[21] Danilo Guerrera et al. Enhancing the scalability of sph codes via an exascale-ready sph mini-app. *arXiv preprint arXiv:1905.03344*, 2019.

[22] James D Hamilton. *Time series analysis*. Princeton university press, 2020.

[23] Elizabeth Hou et al. Penalized ensemble kalman filters for high dimensional non-linear systems. *PMC*, 2021.

[24] Simon J Julier and Jeffrey K Uhlmann. Unscented filtering and nonlinear estimation. *Proceedings of the IEEE*, 92(3):401–422, 2004.

[25] Nicholas Junge. What is a regular time series? Blog post on nicholasjunge.com, 2023.

[26] Rudolph E Kalman. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, 82(1):35–45, 1960.

[27] Matthias Katzfuss, Jonathan R. Stroud, and Christopher K. Wikle. Understanding the ensemble kalman filter. *The American Statistician*, 70(4):350–357, 2016.

[28] Sebastian Keller et al. The code for exaphoebos - hydrodynamics and gravity at exa-scale. PASC Presentation, 2022.

[29] Roger Labbe. Kalman and bayesian filters in python. https://github.com/rlabbe/Kalman-and-Bayesian-Filters-in-Python/, 2015.

[30] Bin Liu. Robust sequential online prediction with dynamic ensemble of multiple models: A review. *Neurocomputing*, 541:126345, 2023.

[31] Bin Liu, Zhiwei Ding, and Longxiang Lv. Sequential online prediction in the presence of outliers and change points: an instant temporal structure learning approach. *arXiv preprint arXiv:1907.06377*, 2019.

[32] Leon B Lucy. A numerical approach to the testing of the fission hypothesis. *The astronomical journal*, 82:1013–1024, 1977.

[33] Machine Learning TV. Kalman filter - part 1. YouTube, 2021.

[34] Joe J Monaghan. Smoothed particle hydrodynamics. *Annual review of astronomy and astrophysics*, 30(1):543–574, 1992.

[35] Benard Muma and Austin Karoki. Modeling GDP using autoregressive integrated moving average (ARIMA) model: A systematic review. *Open Access Library Journal*, 9:1–8, 2022.

[36] [Authors not provided]. Forecasting the main economic indicators for industry in the analytical system "horizon". *Munich Personal RePEc Archive*, (118887), 2023.

[37] Daniel J Price. Smoothed particle hydrodynamics and magnetohydrodynamics. *Journal of Computational Physics*, 231(3):759–794, 2012.

[38] Daniel J Price, James Wurster, Terrence S Tricco, Chris Nixon, St'even Toupin, Alex Pettitt, Conrad Chan, Daniel Mentiplay, Guillaume Laibe, Simon Glover, et al. Phantom: A smoothed particle hydrodynamics and magnetohydrodynamics code for astrophysics. *Publications of the Astronomical Society of Australia*, 35:e031, 2018.

[39] Christian P Robert and George Casella. *Monte Carlo Statistical Methods*. Springer Science & Business Media, 2013.

[40] Stephan Rosswog. Astrophysical smooth particle hydrodynamics. *Living reviews in computational astrophysics*, 1(1):1–109, 2015.

[41] Dan Simon. *Optimal state estimation: Kalman, H infinity, and nonlinear approaches*. John Wiley & Sons, 2006.

[42] Volker Springel. Smoothed particle hydrodynamics in astrophysics. *Annual Review of Astronomy and Astrophysics*, 48:391–430, 2010.

[43] Simo Särkkä. *Bayesian filtering and smoothing*. Cambridge University Press, 2013.

[44] Ruey S Tsay. Analysis of financial time series. *John Wiley & Sons*, 2005.

[45] Chao Wang et al. The role of turbulence in high-mass star formation. *Astronomy & Astrophysics*, 2024. Available online.

[46] Albrecht H. Weerts and Ghada Y. H. El Serafy. Particle filtering and ensemble kalman filtering for state updating with hydrological conceptual rainfall-runoff models. *Water Resources Research*, 42(9), 2006.

[47] Jie Xiong. *An Introduction to Stochastic Filtering Theory*. Oxford University Press, 2008.

[48] Linyang Zhu, Weiwei Zhang, Jiaqing Kou, and Yilang Liu. Machine learning methods for turbulence modeling in subsonic flows around airfoils. *Physics of Fluids*, 31(1):015105, 2019.

Appendix

Factorial Experiments for CPU

Number of hypotheses	Dataset	Resampling	Noise	Time (HH:MM:SS)	NRMSE
200	sedov	random	gaussian	19:55	4.64%
200	sedov	random	laplace	21:56	4.61%
200	sedov	residual	gaussian	25:37	4.51%
200	sedov	residual	laplace	27:17	4.48%
500	sedov	random	gaussian	34:33	4.50%
500	sedov	random	laplace	38:33	4.47%
500	sedov	residual	gaussian	47:55	4.41%
500	sedov	residual	laplace	52:26	4.39%
1000	sedov	random	gaussian	57:28	4.44%
1000	sedov	random	laplace	1:04:53	4.41%
1000	sedov	residual	gaussian	1:19:27	4.39%
1000	sedov	residual	laplace	1:25:45	4.35%
200	turb	random	gaussian	19:53	2.78%
200	turb	random	laplace	22:09	2.74%
200	turb	residual	gaussian	25:30	2.66%
200	turb	residual	laplace	27:29	2.61%
500	turb	random	gaussian	34:17	2.71%
500	turb	random	laplace	38:24	2.67%
500	turb	residual	gaussian	47:45	2.63%
500	turb	residual	laplace	52:25	2.57%
1000	turb	random	gaussian	57:22	2.68%
1000	turb	random	laplace	1:05:11	2.65%
1000	turb	residual	gaussian	1:21:14	2.61%
1000	turb	residual	laplace	1:26:16	2.56%

Factorial Experiments for GPU

Number of hypotheses	Dataset	Resampling	Noise	Time (HH:MM:SS)	NRMSE
200	sedov	random	gaussian	13:04	4.64%
200	sedov	random	laplace	13:03	4.61%
200	sedov	residual	gaussian	13:02	4.64%
200	sedov	residual	laplace	12:55	4.61%
500	sedov	random	gaussian	14:19	4.50%
500	sedov	random	laplace	14:18	4.47%
500	sedov	residual	gaussian	14:13	4.50%
500	sedov	residual	laplace	14:17	4.47%
1000	sedov	random	gaussian	17:46	4.44%
1000	sedov	random	laplace	17:35	4.41%
1000	sedov	residual	gaussian	17:38	4.44%
1000	sedov	residual	laplace	17:45	4.41%
200	turb	random	gaussian	13:06	2.78%
200	turb	random	laplace	13:03	2.74%
200	turb	residual	gaussian	12:58	2.78%
200	turb	residual	laplace	13:00	2.74%
500	turb	random	gaussian	14:09	2.71%
500	turb	random	laplace	14:20	2.67%
500	turb	residual	gaussian	14:09	2.71%
500	turb	residual	laplace	14:19	2.67%
1000	turb	random	gaussian	17:47	2.68%
1000	turb	random	laplace	17:52	2.65%
1000	turb	residual	gaussian	17:34	2.68%
1000	turb	residual	laplace	17:49	2.65%

University of Basel
Faculty of Science

Declaration on Scientific Integrity
(including a Declaration on Plagiarism and Fraud)
Translation from German original

Title of Thesis: Exploring the Applicability of Time Series Analysis to Astrophysical Simulations

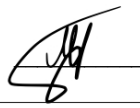
Name Assessor: Prof. Dr. Florina M. Ciorba

Name Student: Lev Maznichenko

Matriculation No.: 21-751-821

I attest with my signature that I have written this work independently and without outside help. I also attest that the information concerning the sources used in this work is true and complete in every respect. All sources that have been quoted or paraphrased have been marked accordingly.

Additionally, I affirm that any text passages written with the help of AI-supported technology are marked as such, including a reference to the AI-supported program used. This paper may be checked for plagiarism and use of AI-supported technology using the appropriate software. I understand that unethical conduct may lead to a grade of 1 or "fail" or expulsion from the study program.

Place, Date: 02.08.2024 Student: 

Will this work, or parts of it, be published?

☐ No

☑ Yes. With my signature I confirm that I agree to a publication of the work (print/digital) in the library, on the research database of the University of Basel and/or on the document server of the department. Likewise, I agree to the bibliographic reference in the catalog SLSP (Swiss Library Service Platform). (cross out as applicable)

Publication as of: _____

Place, Date: Basel, 02.08.2024 Student: 

Place, Date: _____ Assessor: _____

Please enclose a completed and signed copy of this declaration in your Bachelor's or Master's thesis.