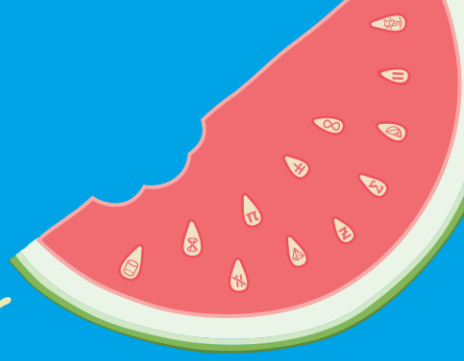
AMSI VACATION RESEARCH SCHOLARSHIPS 2021–22

Get a taste for Research this Summer



Data Assimilation for Korteweg-De Vries Equations

Michael D. Kaminski

Supervised by Dr. Andrew Zammit-Mangion

University of Wollongong

Vacation Research Scholarships are funded jointly by the Department of Education and the Australian Mathematical Sciences Institute.

AMSI AUSTRALIAN MATHEMATICAL SCIENCES INSTITUTE

Contents

Abstract

Data assimilation (DA) methods combine prior knowledge with observational data, in order to obtain more representative estimates of the true state of a system. This project focuses on two inferential sequential algorithms, through which DA is implemented: the Kalman filter (KF), and the ensemble Kalman filter (EnKF). These algorithms implement DA through a Bayesian update at each iteration, which incorporates data into the procured estimates. The main application of interest is DA for Korteweg-De Vries (KdV) equations, which model the behaviour of internal waves in shallow bodies of water. Necessary background theory underlying KdV equations and DA is discussed. A toy example is then constructed, in which both the KF and EnKF are implemented. The performance of the EnKF is benchmarked against the circumstantially optimal KF, from which it is determined that the EnKF is implemented correctly. Subsequently, the EnKF is implemented on simulated KdV data with known parameters, in order to estimate the underlying true process. The implemented EnKF is found to fit the simulated data well, providing reasonable estimates even when the measurements are sparse in space and time.

1 Introduction

In statistics, Bayesian inference is analogous to the scientific method: one holds a prior belief (or hypothesis), gathers observations, and alters the held belief in accordance with inferences made on the data. This system of inference, therefore, provides an ideal framework for combining scientific knowledge with data, which constitutes the modus operandi of data assimilation (DA). By combining information in this manner, DA methods make inferences on the underlying process responsible for generating the observed data (Wikle & Berliner 2007). This project investigates certain inferential sequential algorithms, falling under the umbrella of DA, which involve propagating an algorithm forward through time to estimate the underlying process. In particular, the two algorithms the project focuses on are the Kalman filter (KF), and the ensemble Kalman filter (EnKF). Both of these algorithms implement DA through a Bayesian update, which is performed at regular time intervals (Katzfuss et al. 2016).

Korteweg-De Vries (KdV) equations model the behaviour of internal waves in the ocean, under the key assumption that the body of water is relatively shallow. The KdV model changes form to account for varying physical processes and contexts, such as variable topography of the seabed. These changes are quantified by alterations to the model parameters, and the incorporation of additional terms into the equation (Holloway et al. 1997, Grimshaw et al. 2004). The main application of interest for this project involves estimating the true process of a simulated KdV, with known model parameters. Future work on this project will investigate parameter estimation, for KdV models with unknown parameters.

Section 2 introduces the necessary background theory for KdV equations, along with theoretical details pertaining to physical features of the ocean, and also provides an overview of how the KdV equations were discretised for implementation in Python. Section 3 provides the necessary background for data assimilation procedures, with thorough explanations given of the workings of both the KF and the EnKF. After having established the underlying theory, Section 4 introduces a toy example utilised for evaluating the efficacy of a Python-implemented EnKF. The toy example consists of a model for which the KF operates optimally; this enables the usage of the KF as a benchmark, against which the performance of the EnKF can be compared. After confirming that the EnKF is correctly implemented, it is modified in Section 5

for usage on a KdV model with known parameters, in order to estimate the true underlying KdV-governed process. By comparing the EnKF estimates with those given by a data-blind algorithm, where DA is not used to adjust the state trajectories, the importance of the Bayesian update in the EnKF is clearly illustrated.

1.1 Statement of Authorship

The project was formulated by Andrew Zammit-Mangion together with collaborators at the University of Western Australia (Matt Rayson and Nicole Jones). Code to run the KdV equations was provided by Matt Rayson. Andrew Zammit-Mangion, Nicole Jones, and Matt Rayson supervised Michael Kaminski. Code implementing the KF and EnKF on a toy example was written by Michael Kaminski (c.f. Appendix A). Moreover, EnKF implementation on the KdV simulations was carried out by Michael Kaminski (c.f. Appendix B), along with interpretation of the resulting output.

2 The Korteweg-De Vries Equation

The *Korteweg-De Vries equation*, or KdV equation for short, is well-known as an appropriate physical model for describing the behaviour of *internal waves* in relatively shallow bodies of water (Holloway et al. 1997). The defining characteristic of internal waves is that their largest vertical amplitudes occur in the interior of the fluid (Gerkema & Zimmerman 2008). The constant-coefficients KdV equation is given by

$$\partial_t \eta(x,t) + c\partial_x \eta(x,t) + \alpha\eta(x,t)\partial_x \eta(x,t) + \beta\partial_x^3 \eta(x,t) = 0 \tag{1}$$

where $\eta(x,t)$ is the vertical displacement of the internal wave, x is the horizontal coordinate, and t is time. Furthermore, α, β, and c are the nonlinearity, dispersion, and wave propagation speed (or phase speed) parameters, with units respectively given by s^{-1}, $\mathrm{m}^3\,\mathrm{s}^{-1}$, and $\mathrm{m}\,\mathrm{s}^{-1}$. A key underlying assumption of the KdV model is that the solutions are *long waves*, relative to depth; namely, that H/λ and a/H are small, where H is the local water depth, λ is a representative wavelength, and a is a representative wave amplitude (Holloway et al. 1997).

The constant-coefficients form (1) has limited real-world applicability, since it assumes a flat ocean bed. Variable topography of the ocean floor is incorporated by permitting the parameters α and β to vary horizontally as functions of x, and adding another term:

$$\partial_t \eta(x,t) + c\partial_x \eta(x,t) + \alpha(x)\eta(x,t)\partial_x \eta(x,t) + \beta(x)\partial_x^3 \eta(x,t) + \frac{c}{2Q(x)}\frac{\mathrm{d}Q(x)}{\mathrm{d}x}\eta(x,t) = 0. \tag{2}$$

The function Q represents the amplification factor due to a horizontally-variable ocean floor depth, oceanic density stratification, and current (Grimshaw et al. 2004). Various other versions of the KdV equation also exist, which account for additional physical processes (Rayson 2021).

2.1 Density Stratification

The ocean water density is derived from temperature, salinity, and pressure variables; for a particular water column, it can be measured at discrete depths from either a vertical mooring

or profiling instrument (Manderson et al. 2019). The density field may be regarded as the sum of a mean (background) density $\bar{\rho}(z)$ at a particular vertical coordinate z, together with the fluctuation $\rho'(x, z, t)$ about this mean (Phillips 1966). Using the time series of density readings $\widetilde{\rho}(z, t)$ obtained through the measurement apparatus installed at a particular water column, a filtering operation can be applied to extract the estimated background density from the raw data. It is the background density that is the dynamically significant quantity of interest for most analyses.

In low and midlatitudes (i.e. excluding polar regions), the vertical density structure typically consists of a surface mixed layer, a sharp gradient region referred to as the *pycnocline* where the density changes most rapidly, followed by a weaker gradient region beneath, where the density increases at an exponentially lower rate. An exception to this general picture are density stratifications consisting of double pycnoclines, most often found in subtropical regions. The double hyperbolic tangent (DHT) function has been implemented to reconstruct the vertical structure of the background density profile $\bar{\rho}(z)$ in the upper ocean from observational mooring data, and it is flexible enough to model density structures with one or two pycnoclines. The DHT function is therefore suitable in low and midlatitudes, but only for depths less than 500 metres. It has the form

$$\rho(z) = \beta_0 - \beta_1 \left[\tanh\left(\frac{z + \beta_2}{\beta_3} \right) + \tanh\left(\frac{z + \beta_4}{\beta_5} \right) \right] \tag{3}$$

where: β_0 ($\mathrm{kg\,m^{-3}}$) is the approximate average density across the profile; β_1 ($\mathrm{kg\,m^{-3}}$) is a scale for the density difference across the water column; β_2 and β_4 (m) are the respective middepths of the upper and lower pycnoclines; and β_3 and β_5 (m) are the respective widths of the upper and lower pycnoclines. (Manderson et al. 2019).

2.2 Simulation of the KdV

The KdV was simulated in Python, using code kindly provided by Matt Rayson from the Oceans Institute at the University of Western Australia. A discretisation of the KdV was implemented using an IMEX numerical discretisation scheme elucidated by Rayson (2021). The simulation was initialised using a background density profile, which was in turn created by specifying the parameter values for an implemented DHT function.

The units of space (distance and amplitude) were metres, while the units of time were seconds. The horizontal spatial domain consisting of coordinates $x \in [0, 100000)$ was discretised to consist of 2000 fixed points spaced by 50 metres. The temporal domain consisting of temporal points $t \in [0, 86400]$ was discretised, so that successive time points were separated by 15 seconds. The final time $T = 86400\,\mathrm{s}$ is equivalent to 24 hours. The vertical spatial domain, spanning from the ocean bed $z = -350\,\mathrm{m}$ to the surface $z = 0\,\mathrm{m}$, was discretised to consist of 50 fixed points spaced by $7\,\mathrm{m}$.

A sinusoidal disturbance was utilised as the boundary condition at the left-hand extremity of the spatial domain ($x = 0\,\mathrm{m}$), in order to model an internal tide coming in from the left of the domain. The sinusoid had the form $a_0 \sin(\omega t)$, with an angular frequency of $\omega = \frac{2\pi}{12\times 3600}\,\mathrm{rad\,s^{-1}}$ and an amplitude of $a_0 = 20\,\mathrm{m}$. The KdV implementation was capable of simulating both flat-bed and idealised variable-topography scenarios. However, only the flat-bed case corresponding to the constant-coefficients KdV (1) was considered, with a depth of $H = 350\,\mathrm{m}$.

3 Data Assimilation

Data assimilation (DA) involves combining observations with prior knowledge in order to obtain an estimate of the true state of a system and the associated uncertainty of that estimate (Katzfuss et al. 2016). The implementation of DA thus requires a statistical model for the observations (the data or measurement model), and an a priori statistical model for the state process (the state or process model). The paradigm of *Bayesian statistics* provides a coherent probabilistic approach for combining information, and it therefore provides an appropriate framework for data assimilation (Wikle & Berliner 2007).

Bayesian statistical inference consists of three steps. Firstly, one formulates a full probability model; the joint distribution of all observable and unobservable components of interest, including the data, process, and parameters. The next step involves obtaining the *posterior distribution*: the conditional distribution of the unobservable quantities of interest, given the observed data. Formally, this is accomplished by the application of Bayes' Theorem. Finally, one should evaluate the fit of the model, and its ability to adequately characterise the process of interest. In contrast to the posterior distribution, the *prior distribution* is the unconditional distribution of the unobservable quantities of interest. The prior quantifies a priori theoretical knowledge, while the posterior represents the update of the prior knowledge after taking the actual observations into account. Therefore, the Bayesian approach is analogous to the scientific method: one holds a prior belief, collects data, and then updates that belief given the new data (Wikle & Berliner 2007).

3.1 Sequential Inference

With subscripts denoting time, let $\mathbf{Y}_{1:t} := \{\mathbf{Y}_1, \ldots, \mathbf{Y}_t\}$ denote the observational data process consisting of temporally-indexed observation vectors, and let $\mathbf{X}_{0:t} := \{\mathbf{X}_0, \ldots, \mathbf{X}_t\}$ denote the state process consisting of temporally-indexed state vectors. Furthermore, let $\mathbf{y}_{1:t}$ and $\mathbf{x}_{0:t}$ denote the corresponding non-random realisations. Then, applying Bayes' Theorem, the posterior distribution of the states conditional on the observed data is given by

$$p(\mathbf{x}_{0:t}|\mathbf{y}_{1:t}) \propto p(\mathbf{y}_{1:t}|\mathbf{x}_{0:t})p(\mathbf{x}_{0:t}), \tag{4}$$

where $p(\mathbf{x}_{0:t})$ represents the prior knowledge of the state process, and $p(\mathbf{y}_{1:t}|\mathbf{x}_{0:t})$ represents the data or measurement distribution. Generally, the state process is assumed to be *Markovian*, so that the state at time t, when conditioned on all previous states, only depends on the state at time $t-1$. Mathematically, the Markov assumption for the prior can be expressed as

$$p(\mathbf{x}_{0:T}) = p(\mathbf{x}_0) \prod_{t=1}^{T} p(\mathbf{x}_t|\mathbf{x}_{t-1}), \tag{5}$$

where $p(\mathbf{x}_t|\mathbf{x}_{t-1})$ is the evolution distribution, $p(\mathbf{x}_0)$ is the distribution for the initial state, and T indicates the length of the analysis time period of interest.

An additional important assumption is that the observations are conditionally independent

of one another, given the true state, which is to say that

$$p(\mathbf{y}_{1:T}|\mathbf{x}_{0:T}) = \prod_{t=1}^{T} p(\mathbf{y}_t|\mathbf{x}_t). \tag{6}$$

The Markovian assumption (5) and the conditional independence assumption (6) jointly permit one to reformulate (4) as

$$p(\mathbf{x}_{0:T}|\mathbf{y}_{1:T}) \propto p(\mathbf{x}_0) \prod_{t=1}^{T} p(\mathbf{y}_t|\mathbf{x}_t)p(\mathbf{x}_t|\mathbf{x}_{t-1}).$$

This form illustrates the quintessence of sequential updating procedures: as new data become available, one can update the previous optimal estimate of the state process, without needing to start calculations from scratch (Wikle & Berliner 2007).

3.2 Filtering

Sequential procedures referred to as *filtering* procedures involve two steps at each time point t, assuming the availability of the *analysis distribution* (or filtering distribution) at time $t-1$, given by $p(\mathbf{x}_{t-1}|\mathbf{y}_{1:t-1})$. Firstly, the preceding analysis distribution at time $t-1$ is used to find the *forecast distribution* at time t, given by $p(\mathbf{x}_t|\mathbf{y}_{1:t-1})$. Then, a Bayesian update is performed to obtain the analysis distribution at time t, given by $p(\mathbf{x}_t|\mathbf{y}_{1:t})$. Leveraging the Markov assumption, the forecast distribution is given by

$$p(\mathbf{x}_t|\mathbf{y}_{1:t-1}) = \int p(\mathbf{x}_t|\mathbf{x}_{t-1})p(\mathbf{x}_{t-1}|\mathbf{y}_{1:t-1})\mathrm{d}\mathbf{x}_{t-1}$$

whence the analysis distribution is obtained using Bayes' rule:

$$p(\mathbf{x}_t|\mathbf{y}_{1:t}) = p(\mathbf{x}_t|\mathbf{y}_t, \mathbf{y}_{1:t-1}) \propto p(\mathbf{y}_t|\mathbf{x}_t, \mathbf{y}_{1:t-1})p(\mathbf{x}_t|\mathbf{y}_{1:t-1}) = p(\mathbf{y}_t|\mathbf{x}_t)p(\mathbf{x}_t|\mathbf{y}_{1:t-1}).$$

By iterating between these two steps as new data becomes available, the forecast and analysis distributions can be obtained for each time step (Wikle & Berliner 2007). In practice, the first step of each filtering iteration is termed the forecast step, while the second step is termed the update step.

The goal of filtering is to obtain the analysis distribution of the state at each time point, since the analysis distribution at time t is equivalent to the posterior distribution of $\mathbf{x}_t$ (the state) given $\mathbf{y}_{1:t}$ (the up-to-date data at time t). In particular, the forecast distributions quantify prior knowledge known about the state based on all previous observations, whereas the analysis distributions represent the updated knowledge obtained after taking the relevant up-to-date observational data into account (Katzfuss et al. 2016).

3.3 Kalman Filter

Assume a linear Gaussian state-space model (having linear model operators and Gaussian error distributions) of the form

$$\mathbf{y}_t = \mathbf{H}_t\mathbf{x}_t + \mathbf{v}_t, \qquad \mathbf{v}_t \sim N_{m_t}(\mathbf{0}, \mathbf{R}_t), \tag{7}$$

$$\mathbf{x}_t = \mathbf{M}_t\mathbf{x}_{t-1} + \mathbf{w}_t, \qquad \mathbf{w}_t \sim N_n(\mathbf{0}, \mathbf{Q}_t), \tag{8}$$

for $t = 1, \ldots, T$, where $\mathbf{y}_t$ is the observed m_t-dimensional observation vector at time t, $\mathbf{x}_t$ is the n-dimensional unobserved state vector of interest. Moreover, the observation error $\mathbf{v}_t$ and innovation/evolution error $\mathbf{w}_t$ are mutually and serially independent. Equations (7) and (8) are respectively the observation model and process (or evolution) model. The observation matrix $\mathbf{H}_t$ relates the states to the observations by mapping the state vectors to the observation space, and the evolution matrix $\mathbf{M}_t$ relates each state iteration to its predecessor, thus determining how the state evolves over time.

Filtering for the state-space model (7)–(8) can be undertaken with a *Kalman filter* (KF) (Katzfuss et al. 2016), which is in fact optimal for sequential updating in linear Gaussian models (Wikle & Berliner 2007). An iteration of the KF at time t initiates by assuming that the analysis distribution at the previous time $t-1$ is given by

$$\mathbf{x}_{t-1}|\mathbf{y}_{1:t-1} \sim N_n(\widehat{\boldsymbol{\mu}}_{t-1}, \widehat{\boldsymbol{\Sigma}}_{t-1}). \tag{9}$$

Then, utilising the process model (8), the forecast step computes the forecast distribution at time t as

$$\mathbf{x}_t|\mathbf{y}_{1:t-1} \sim N_n(\widetilde{\boldsymbol{\mu}}_t, \widetilde{\boldsymbol{\Sigma}}_t) \tag{10}$$

where $\widetilde{\boldsymbol{\mu}}_t = \mathbf{M}_t\widehat{\boldsymbol{\mu}}_{t-1}$ and $\widetilde{\boldsymbol{\Sigma}}_t = \mathbf{M}_t\widehat{\boldsymbol{\Sigma}}_{t-1}\mathbf{M}_t' + \mathbf{Q}_t$. The update step proceeds expectedly, modifying the forecast distribution using the new data $\mathbf{y}_t$. Exploiting Gaussianity, the update equations – with derivation omitted – are given by

$$\widehat{\boldsymbol{\mu}}_t = \widetilde{\boldsymbol{\mu}}_t + \mathbf{K}_t(\mathbf{y}_t - \mathbf{H}_t\widetilde{\boldsymbol{\mu}}_t), \tag{11}$$

$$\widehat{\boldsymbol{\Sigma}}_t = (\mathbf{I}_n - \mathbf{K}_t\mathbf{H}_t)\widetilde{\boldsymbol{\Sigma}}_t, \tag{12}$$

where $\mathbf{I}_n$ is the $n \times n$ identity matrix, and $\mathbf{K}_t = \widetilde{\boldsymbol{\Sigma}}\mathbf{H}_t'(\mathbf{H}_t\widetilde{\boldsymbol{\Sigma}}_t\mathbf{H}_t' + \mathbf{R}_t)^{-1}$ is referred to as the Kalman gain matrix, of dimension $n \times m_t$.

In summary, the KF provides the exact analysis distribution for linear Gaussian state-space models such as (7)–(8) at each time step, and can therefore be utilised as a benchmark for comparing with other sequential filtering algorithms on linear Gaussian models. However, if the observation or state vectors are large, computations of the covariance matrices and the matrix inversion in the Kalman gain formula become extremely expensive, necessitating approximations (Katzfuss et al. 2016). Furthermore, within the filtering framework, the forecast and analysis distributions cannot be obtained explicitly for non-Gaussian models and/or non-linear dynamic operators, thus limiting the scope of application for the Kalman filter (Wikle & Berliner 2007).

3.4 Ensemble Kalman Filter

The *ensemble Kalman filter* (EnKF) is essentially an approximate version of the KF, in which the state distribution is represented by an *ensemble*: a random sample which is propagated forward through time and updated when new data become available. The ensemble representation is a type of dimension reduction, greatly reducing computational demands for high-dimensional systems. An iteration of the EnKF at time t begins by assuming the availability of an ensemble $\widehat{\mathbf{x}}_{t-1}^{(1)}, \ldots, \widehat{\mathbf{x}}_{t-1}^{(N)}$ sampled randomly from the analysis distribution (9) at time $t-1$. In the case of a state-space model having the form (7)–(8), the EnKF forecast step obtains a sample from the forecast distribution (10) at time t by applying the evolution equation (8) to each ensemble member:

$$\widetilde{\mathbf{x}}_t^{(i)} = \mathbf{M}_t \widehat{\mathbf{x}}_{t-1}^{(i)} + \mathbf{w}_t^{(i)}, \qquad \mathbf{w}_t^{(i)} \sim N_n(\mathbf{0}, \mathbf{Q}_t) \tag{13}$$

for $i = 1, \ldots, N$ (Katzfuss et al. 2016). However, unlike the KF, the EnKF can also be applied for process models with nonlinear evolution operators, of the form

$$\mathbf{x}_t = \mathcal{M}(\mathbf{x}_{t-1}) + \mathbf{w}_t, \qquad \mathbf{w}_t \sim N_n(\mathbf{0}, \mathbf{Q}_t), \tag{14}$$

where $\mathcal{M}(\cdot)$ is the model evolution. In the case of a nonlinear process model (14), the matrix $\mathbf{M}_t$ in the forecast equation (13) would be substituted with the nonlinear operator $\mathcal{M}(\cdot)$ to obtain the forecast ensemble (Wikle & Berliner 2007).

Considering again a linear Gaussian state-space model (7)–(8), for the forecast ensemble (13), it holds that $\widetilde{\mathbf{x}}_t^{(i)} \sim N_n(\widetilde{\boldsymbol{\mu}}_t, \widetilde{\boldsymbol{\Sigma}}_t)$, and so each forecast ensemble member follows the KF forecast distribution. The process of updating the forecast ensemble based on new data can be carried out either stochastically or deterministically; this report will only investigate the stochastic variant of the EnKF, which employs the stochastic update. *Conditional simulation* from the state analysis distribution utilises the forecast ensemble (13), together with a set of simulated observations $\widetilde{\mathbf{y}}_t^{(1)}, \ldots, \widetilde{\mathbf{y}}_t^{(N)}$ from the observation forecast distribution. The simulated observations are computed as $\widetilde{\mathbf{y}}_t^{(i)} = \mathbf{H}_t \widetilde{\mathbf{x}}_t^{(i)} + \mathbf{v}_t^{(i)}$ where $\mathbf{v}_t^{(i)} \sim N_{m_t}(\mathbf{0}, \mathbf{R}_t)$. The forecast ensemble is shifted based on the difference between the simulated and actual observations:

$$\widehat{\mathbf{x}}_t^{(i)} = \widetilde{\mathbf{x}}_t^{(i)} + \mathbf{K}_t(\mathbf{y}_t - \widetilde{\mathbf{y}}_t^{(i)})$$

for $i = 1, \ldots, N$. It now holds that $\widehat{\mathbf{x}}_t^{(i)} \sim N_n(\widehat{\boldsymbol{\mu}}_t, \widehat{\boldsymbol{\Sigma}}_t)$, and so each analysis ensemble member follows the KF analysis distribution (Katzfuss et al. 2016). On the other hand, when the process model is nonlinear, the foregoing distributional results do not hold. In fact, for a nonlinear model evolution $\mathcal{M}(\cdot)$, the forecast distribution at time t cannot be Gaussian if the analysis distribution at time $t-1$ is Gaussian. Consequently, Gaussianity cannot hold for all time, causing the EnKF to yield biased samples and estimates (Wikle & Berliner 2007).

Conditional simulation therefore provides a means of updating the forecast ensemble to obtain an analysis ensemble, which is itself an exact sample from the analysis distribution when a linear Gaussian model is being considered. However, this requires computation of the Kalman gain, which in turn demands computation of the $n \times n$ forecast covariance matrix $\widetilde{\boldsymbol{\Sigma}}_t$. Since the calculation of this covariance matrix could potentially be very demanding, the update step of the EnKF approximates conditional simulation rather than directly implementing it, by substituting the Kalman gain with an estimate based on the forecast ensemble. Generally, the

estimated Kalman gain has the form

$$\widehat{\mathbf{K}}_t = \mathbf{C}_t\mathbf{H}_t'(\mathbf{H}_t\mathbf{C}_t\mathbf{H}_t' + \mathbf{R}_t)^{-1}$$

where $\mathbf{C}_t$ is an estimate of $\widetilde{\mathbf{\Sigma}}_t$, which can most simply be set equal to the sample covariance matrix of the forecast ensemble. In summary, the EnKF only requires storing and operating on N vectors (ensemble members) of length n, and the estimated Kalman gain can often be calculated rapidly. As $N \to \infty$, the EnKF converges to the KF for linear Gaussian models, but large values of N are usually infeasible in practice due to computational limitations (Katzfuss et al. 2016).

4 A Linear Gaussian Toy Example

As alluded to in Section 3.3, the standard Kalman filter is optimal for linear Gaussian state-space models, yielding exact analysis distributions at each iteration (Katzfuss et al. 2016). This allows one to use the KF as a benchmark for comparing performance with other filtering procedures, given a linear Gaussian model. In this Section, a linear Gaussian toy example will be used for the purpose of (i) implementing the EnKF on a simple example prior to its usage on a more complex KdV model, and (ii) comparing the performance of the EnKF with the KF benchmark to determine whether it has been implemented correctly.

The state-space model we consider has the form (7)–(8), with $m_t = n = 2$, and the observation matrix and covariance matrices set equal to the 2×2 identity matrix $\mathbf{I}_2$. Two formulations of the evolution matrix $\mathbf{M}_t = \mathbf{M}$ (constant in time) were tested, consisting of one diagonal matrix, and another non-diagonal matrix purposed to introduce correlations into the state process. Denoting the state vectors as $\mathbf{z}_t$, the resulting state-space model is given by

$$\mathbf{y}_t = \mathbf{z}_t + \mathbf{v}_t, \qquad \mathbf{v}_t \sim N_2(\mathbf{0}, \mathbf{I}_2), \tag{15}$$

$$\mathbf{z}_t = \mathbf{M}\mathbf{z}_{t-1} + \mathbf{w}_t, \qquad \mathbf{w}_t \sim N_2(\mathbf{0}, \mathbf{I}_2). \tag{16}$$

The uncorrelated and correlated renditions of $\mathbf{M}$ correspondingly had the forms

$$\begin{bmatrix} 0.5 & 0 \\ 0 & 0.2 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} 0.5 & -0.1 \\ 0.1 & 0.2 \end{bmatrix}$$

with eigenvalues of precisely 0.5 and 0.2 for the uncorrelated version, and 0.46 and 0.24 for the correlated version, to two decimal places. As a general rule for the linear evolution equation, if any of the eigenvalues have a modulus of at least unity, then the process model is unstable, and the state vectors will grow without bound as time increases (Wikle et al. 2019). Since the foregoing eigenvalues all have modulus less than one, both of the considered evolution matrices result in (16) being a stable process model.

4.1 Accuracy of Filtering Estimates

Let $\mathbf{z}_t$, $\mathbf{y}_t$, $\mathbf{z}_{\text{KF},t}$, and $\mathbf{z}_{\text{EnKF},t}$ respectively denote the values of the true process, observations, KF estimates, and EnKF estimates at time t. For the estimates and observations, the cumulative

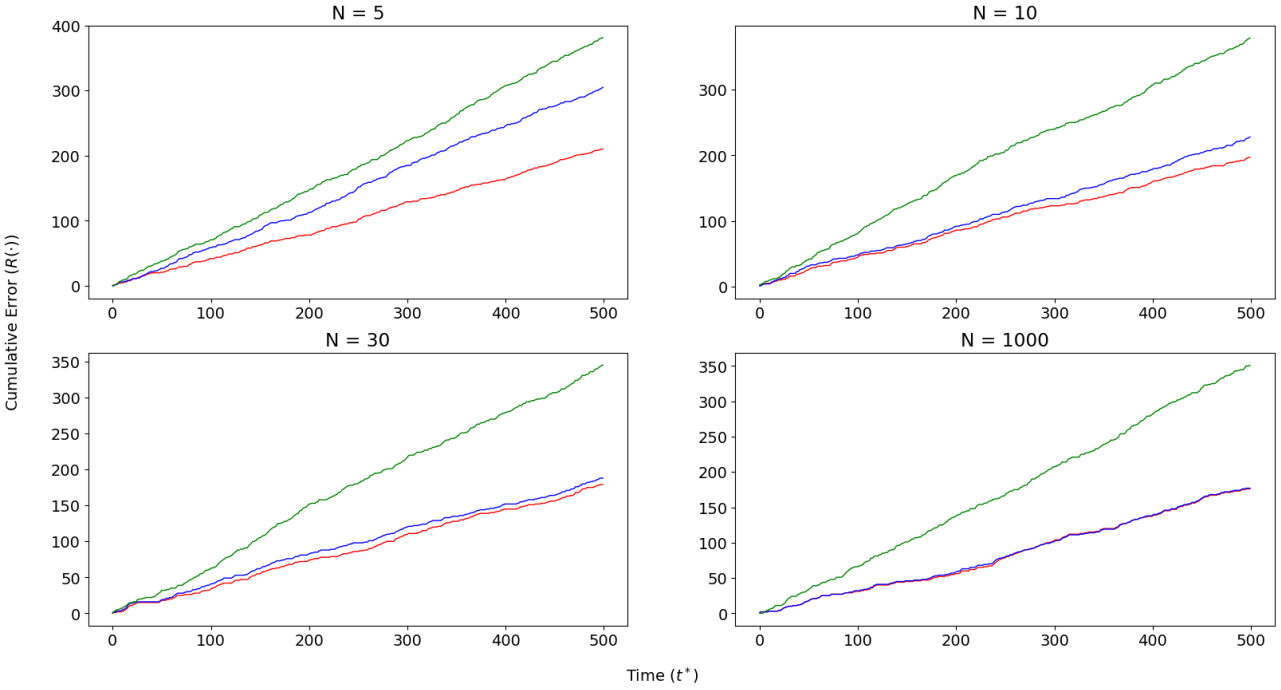


Figure 1: Plots of $R_1(\cdot)$ (red), $R_2(\cdot)$ (blue), and $R_3(\cdot)$ (green) against time

errors at time t^* are given by

$$R_1(t^*) = \sum_{t=1}^{t^*} \|\mathbf{z}_{\mathrm{KF},t} - \mathbf{z}_t\|_2, \qquad R_2(t^*) = \sum_{t=1}^{t^*} \|\mathbf{z}_{\mathrm{EnKF},t} - \mathbf{z}_t\|_2, \qquad R_3(t^*) = \sum_{t=1}^{t^*} \|\mathbf{y}_t - \mathbf{z}_t\|_2,$$

where $\|\cdot\|_2$ denotes the L^2-norm. The sums $R_1(\cdot)$, $R_2(\cdot)$, and $R_3(\cdot)$ correspondingly denote the cumulative errors for the KF, EnKF, and observations. These cumulative errors were plotted against time, with separate plots being generated for different EnKF ensemble sizes N. The goals of this plotting procedure were to: (i) confirm that the KF and EnKF (for reasonable choices of N) came closer to the true process than the noisy observations; (ii) assess the accuracy of the EnKF for various N relative to the KF benchmark; (iii) determine whether the plots were supportive of the EnKF converging towards the KF as N increased.

The panels in Figure 1 show the cumulative errors of the EnKF estimates for ensemble sizes $N = 5$, $N = 10$, $N = 30$, and $N = 1000$, together with those of the KF estimates and observations. The horizontal axes correspond to time, while the vertical axes correspond to the cumulative error. The process model corresponding to these plots utilised the correlated evolution matrix considered earlier; usage of the uncorrelated matrix generated similar output, with no discernable major differences. Firstly, the KF and EnKF both attain a lower cumulative error over time, compared with the observations. Moreover, the EnKF tends to attain intermediate accuracy between the KF estimate and the observations, with a lower error for larger ensembles. For the simulated ensemble with size $N = 1000$, the EnKF appears to attain a nearly identical cumulative error to the KF, which is indicative of the desired convergence quality. Based on these results, the EnKF appears to have been implemented correctly for the state-space model (15)–(16).

5 Inference for a KdV with Known Parameters

Retaining Gaussian error distributions, the EnKF employed for the linear Gaussian state-space model (15)–(16) was modified for application on the KdV. This was achieved by modifying the forecast step of the algorithm to account for a nonlinear evolution operator $\mathcal{M}(\cdot)$ (unavailable in closed form), which modeled the temporal evolution of the KdV. All points within the horizontal spatial domain of the discretised KdV (cf. Section 2.2) were utilised as the spatial points corresponding to the state process, resulting in state vectors of 2000 entries. In addition, 20 equally-spaced points nested within this domain were utilised as the spatial points corresponding to the observations, resulting in data vectors of 20 entries, and a 20×2000 constant observation matrix $\mathbf{H}$ mapping the state vectors to the observation space.

The resulting state-space model has the form

$$\mathbf{y}_t = \mathbf{H}\mathbf{x}_t + \mathbf{v}_t, \qquad \mathbf{v}_t \sim N_{20}(\mathbf{0}, \mathbf{R}), \tag{17}$$

$$\mathbf{x}_t = \mathcal{M}(\mathbf{x}_{t-1}) + \mathbf{w}_t, \qquad \mathbf{w}_t \sim N_{2000}(\mathbf{0}, \mathbf{Q}), \tag{18}$$

with observation and evolution covariance matrices $\mathbf{R} \in M_{20}(\mathbb{R})$ and $\mathbf{Q} \in M_{2000}(\mathbb{R})$, both positive definite and constant in time. The state vectors $\mathbf{x}_t$ represent the true amplitude of the internal wave at time t, across all 2000 spatial points, whereas the data vectors $\mathbf{y}_t$ represent the observed amplitude of the wave at time t, at the 20 horizontal distances corresponding to the locations of the data-gathering sensors. It should also be noted that whereas the KdV implementation had a temporal resolution of 15 seconds, the time steps of the state-space model (between t and $t+1$) were 10 minutes (600 seconds) long.

5.1 Incorporating Spatial Correlations

The observation covariance matrix $\mathbf{R} = \sigma_v^2 \mathbf{I}_{20}$ for appropriate $\sigma_v \in \mathbb{R}_+$ was formulated with the realistic assumption that the individual sensors gathering observational data operated in such a manner that the observational errors were mutually independent in space. The evolution covariance matrix $\mathbf{Q}$ was formulated to allow for spatial correlations, since any error in the state evolution equation (18) would likely be spatially correlated. This was achieved by constructing a *distance matrix*, and then applying a *Matérn covariance function* (Rasmussen & Williams 2006) to the entries of said matrix.

The 2000×2000 distance matrix $\mathbf{D}$ has the form

$$\mathbf{D} = \begin{bmatrix} 0 & 50 & \cdots & 99900 & 99950 \\ 50 & 0 & \ddots & \ddots & 99900 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 99900 & \ddots & \ddots & 0 & 50 \\ 99950 & 99900 & \cdots & 50 & 0 \end{bmatrix}$$

with entries representing spatial distances spaced by 50 metres, and zeros along the main diagonal. For an input distance d, non-negative integral parameter p (yielding positive $\nu =$

$p + 1/2$), and positive parameter l, the Matérn covariance function can be written as

$$k_{\nu=p+1/2}(d) = \exp\left(-\frac{\sqrt{2\nu}d}{l}\right)\frac{\Gamma(p+1)}{\Gamma(2p+1)}\sum_{i=0}^{p}\frac{(p+i)!}{i!(p-i)!}\left(\frac{\sqrt{8\nu}d}{l}\right)^{p-i}. \tag{19}$$

Specifying $p = 0$ yields the simplest form of (19):

$$k_{1/2}(d) = \exp\left(-\frac{d}{l}\right). \tag{20}$$

A value of l equal to one third of the spatial domain length has been used in practice, and parameter values of $p = 1$ and $p = 2$ have been suggested to yield forms of (19) which are of interest in many applications (Rasmussen & Williams 2006). Trial and error has however indicated that, for inference on the KdV, $p = 2$ and $p = 1$ both yield less desireable outcomes compared to $p = 0$, due to either generating less realistic waves, or causing algorithmic errors.

Thus, the Matérn covariance function in (20) with $l = 99950/3$ (one third of the spatial domain length) was applied to each entry of the distance matrix $\mathbf{D}$, with the resulting matrix denoted by $k_{1/2}(\mathbf{D})$. All entries of $k_{1/2}(\mathbf{D})$ were then multiplied by σ_w^2 for appropriate $\sigma_w \in \mathbb{R}_+$, in order to obtain the desired covariance matrix $\mathbf{Q} = \sigma_w^2 k_{1/2}(\mathbf{D})$ for the process model (18). Over the course of experimenting with various standard deviation values for the innovation and observation errors, the choices of $\sigma_v = 0.1$ and $\sigma_w = 0.125$ were found to yield desireable results, for the sake of demonstrating inference with the EnKF. Furthermore, the EnKF was initialised with an ensemble randomly sampled from $N_{2000}(\mathbf{0}, \mathbf{Q})$, i.e. the error distribution of the process model (18). This was found to yield an initial ensemble which: (i) had an appropriate spread; (ii) incorporated spatial correlations into the time-zero wave; and (iii) had zero mean to account for the form of the wave prior to the influence of the sinusoidal boundary condition.

5.2 Evaluation of EnKF Fit

A noisy KdV-governed process was generated, together with synthetic observations, in accordance with the state-space model (17)–(18), using the aforementioned parameter values. In addition, the covariance matrix $\mathbf{Q}$ was constructed utilising the procedure outlined above, in Section 5.1. The terminal time value, for both the simulation and EnKF implementation, was $T = 86400$ seconds (24 hours), thus illustrating the evolution of the internal wave over the course of a single day. Briefly reiterating details from Section 2.2 for convenience, the wave propagated rightwards from the sinusoidal boundary condition $a_0 \sin(\omega t)$ at the left extremity, over a horizontal distance of 100000 metres, within a 350-metre deep flat-bottomed ocean. As a result, the governing KdV equation had the constant-coefficients form (1).

The initialising background density profile was formulated with the DHT parametric model (3), having parameter values of $\beta_0 = 1023.9$, $\beta_1 = 0.91$, $\beta_2 = 30$, $\beta_3 = 40$, $\beta_4 = 117$, and $\beta_5 = 52$. These values were selected on the basis that they resembled parameter values estimated by Manderson et al. (2019), in the context of relatively shallow-water conditions (250-metre depth). The resulting KdV equation governing the internal wave dynamics had coefficient values of $\alpha = -0.009062\,\mathrm{s}^{-1}$ for the nonlinearity parameter, $\beta = 5965.233891\,\mathrm{m}^3\mathrm{s}^{-1}$ for the dispersion parameter, and $c = 1.115722\,\mathrm{ms}^{-1}$ for the phase speed (or wave propagation speed). The

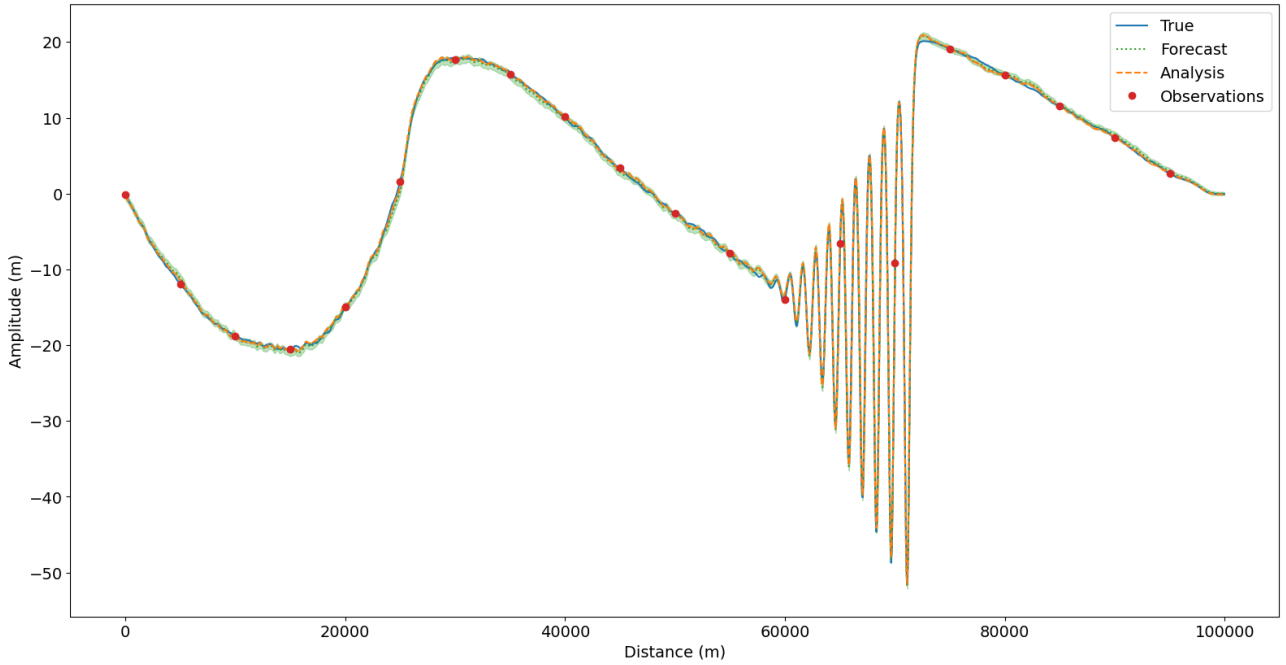


Figure 2: Final-time vertical displacements of the true process, EnKF estimates, and observations over distance

governing KdV model can be written as

$$\partial_t \eta + 1.115772 \partial_x \eta - 0.009062 \eta \partial_x \eta + 5965.233891 \partial_x^3 \eta = 0. \tag{21}$$

An EnKF with $N = 30$ ensemble members was implemented on the simulated data, with both forecast and analysis estimates at each time point being recorded. To elaborate, the forecast estimates were those given by the EnKF forecast step at each iteration, while the analysis estimates were those given by the EnKF analysis step at each iteration. The forecast estimates were therefore impacted by the Bayesian updates performed at all preceding temporal points, and so the accuracy of both the forecast and analysis estimates was regulated by the incorporation of observational data.

Figure 2 illustrates the process of interest, together with the estimates and observations, as functions of the horizontal distance x at the terminal time $t = T$. Lightly shaded regions – henceforth referred to as uncertainty bands – around the forecast/analysis estimates illustrate the uncertainty of the estimates, with the upper and lower bounds indicating a spread of ± 2 standard deviations. Uncertainty bands are displayed for the forecast and analysis estimates, shown in the same colours as the respective estimates. The plot is indicative of an overall fair fit for both the forecast and analysis estimates. At most distances, the analysis estimates do not display clearly superior accuracy to the forecast estimates, although improvements in accuracy are observable at spatial points near the synthetic observations. Furthermore, the analysis estimates were noticeably more precise than the forecasts, with thinner uncertainty bands at all distances.

The top pane of Figure 3 corresponds to the state evolution at the horizontal coordinate $x = 20000\,\text{m}$, where synthetic observational data was gathered; the bottom pane corresponds

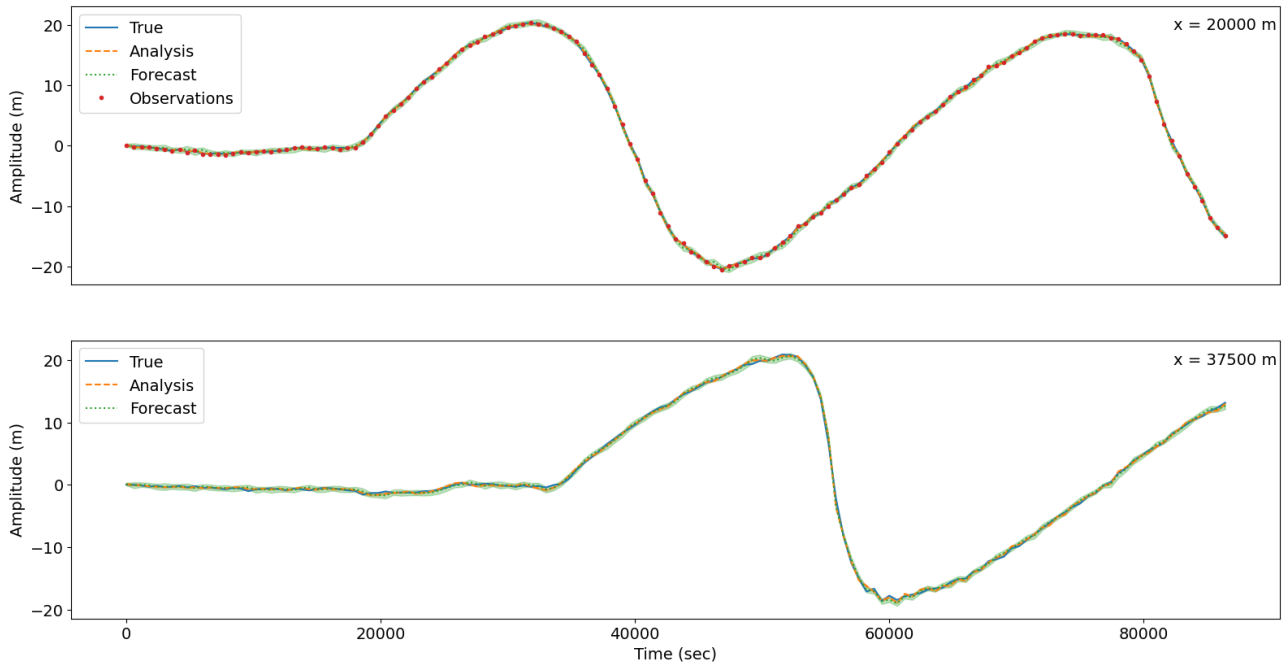


Figure 3: Vertical displacements of the true process, EnKF estimates, and observations over time

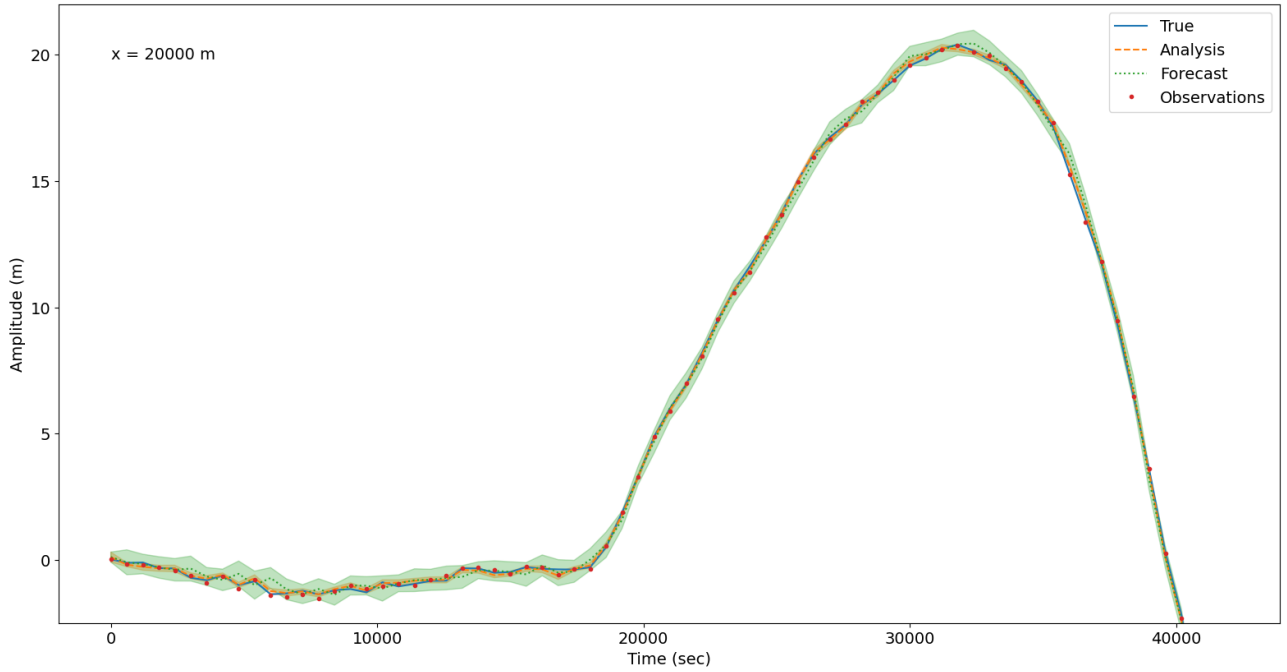


Figure 4: A close-up of the top pane in Figure 3 for the first 40000 seconds

to $x = 37500\,\text{m}$, where no observations were gathered. Despite the absence of observational data at the latter coordinate, the EnKF forecast and analysis estimates both held up very well; the analysis estimates were noticeably less uncertain, but only mildly more accurate relative to the forecasts. The analysis estimates in the top pane appear to adhere more closely with the true process, relative to those in the bottom pane. Since DA was only performed at the 20 spatial points where observations were present, this illustrates that: (i) the enhancing effects of DA extended well beyond those 20 spatial points, in fact impacting the entire spatio-temporal domain; and (ii) DA was most effective at locations where data was present. The close-up in Figure 4 focuses on the first 40000 seconds of the state evolution at $x = 20000\,\text{m}$, corresponding to the top pane of Figure 3. This close-up clarifies the enhancing effect of the updates on the precision and accuracy of the obtained state estimates, as can be seen by comparing the analysis and forecast estimates.

5.3 Importance of Data Assimilation

The foregoing plots incorporated forecast estimates, which estimated the true process only using observational data for all preceding time points. However, since the forecasts were cumulatively affected by the Bayesian updates iteratively performed in the EnKF, they do not represent estimates made without taking observations into account. By editing the update step out of the implemented EnKF in Python, and re-running the code, a data-blind algorithm was formed which only performed the EnKF forecast step at each iteration, without ever performing the update step. In other words, the data-blind algorithm never performed DA, only utilising the evolution model (18) to propagate the ensemble at each time step.

Figure 5 displays the data-blind estimates (labeled as the initial forecast), together with

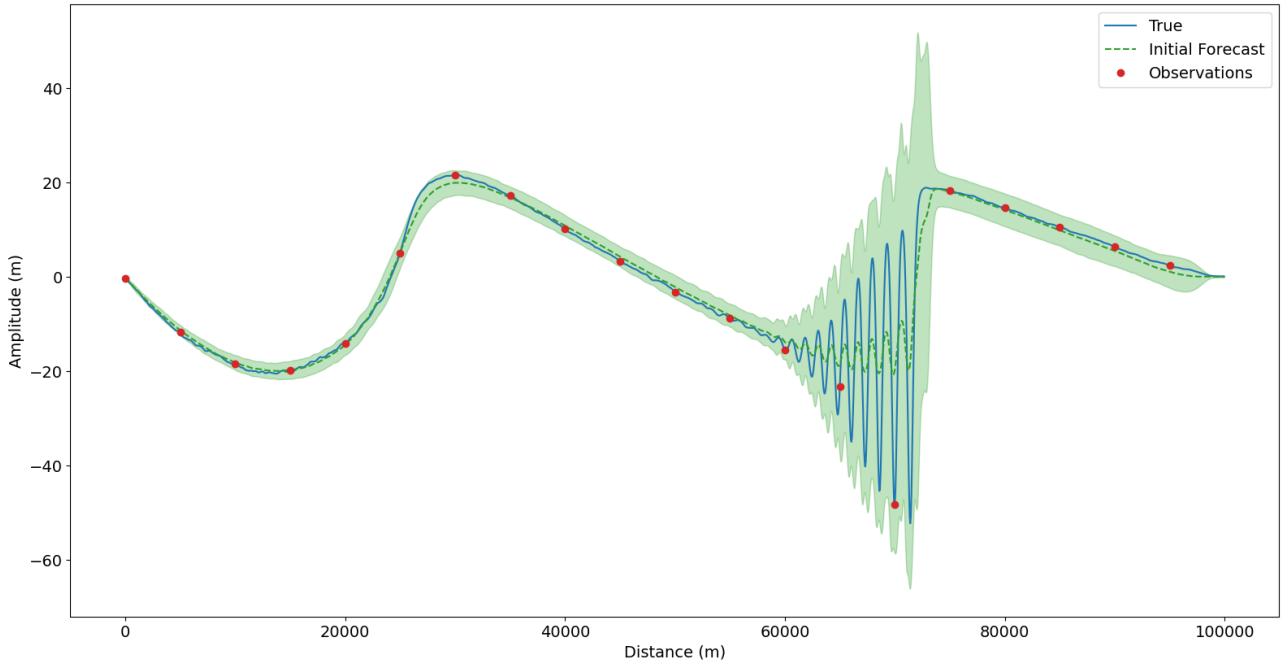


Figure 5: Final-time vertical displacements of the true process, data-blind estimates (initial forecast), and observations over distance

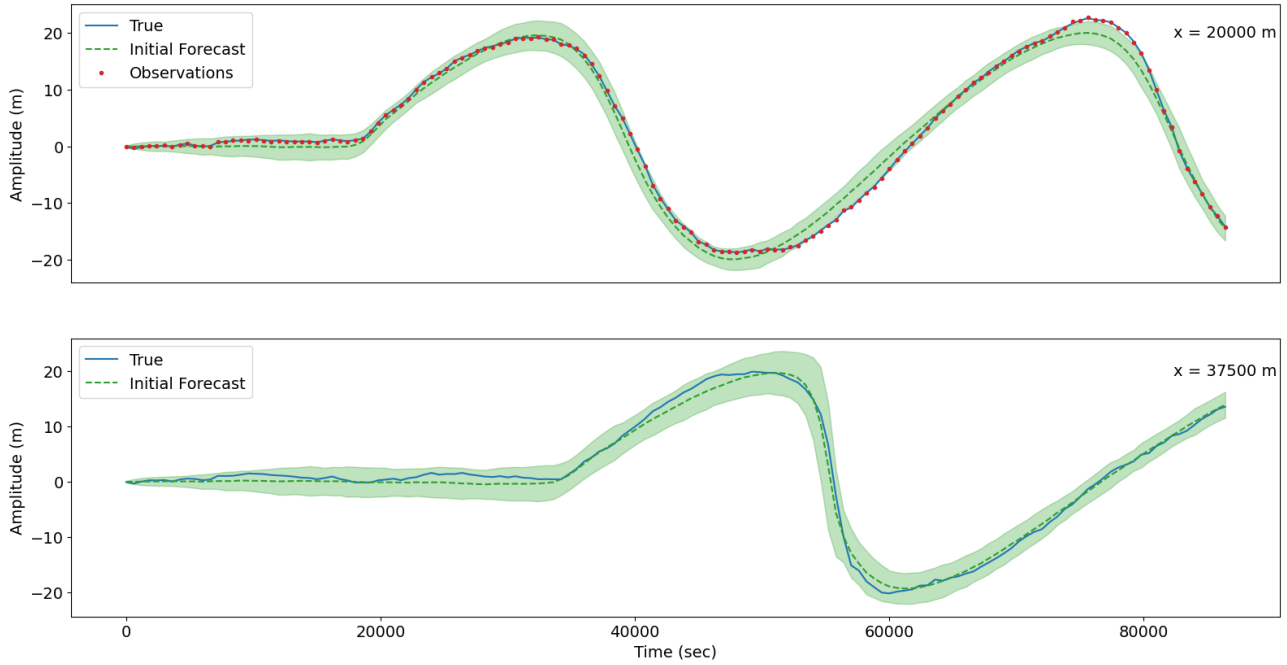


Figure 6: Vertical displacements of the true process, data-blind estimates (initial forecast), and observations over time

the observations and process of interest, as a function of x at the terminal time $t = T$. The true process appears nearly identical to that in Figure 2, due to the deterministic nature of the $\mathcal{M}(\cdot)$ evolution operator. The most visible difference between the data-blind estimates, and the earlier forecast/analysis estimates, is that the uncertainty bands have significantly greater spread. This indicates that the data-blind estimates had far higher uncertainty than the forecast/analysis estimates of the EnKF. Furthermore, in the horizontal region approximately between $x = 60000\,\text{m}$ and $x = 73000\,\text{m}$, the oscillating feature of the internal wave was captured very poorly by the data-blind estimates, relative to those of the EnKF. The uncertainty of the data-blind estimates also greatly intensified throughout the oscillating feature, unlike the EnKF estimates.

The time series plots in Figure 6 illustrate the state evolution together with the data-blind estimates (labeled as the initial forecast), with the panes corresponding to the same x-coordinates as the time series plots in Figure 3. While observations are also displayed in the top pane, they obviously had no impact on the data-blind estimates. Relative to the EnKF estimates, it appears that the data-blind estimates were substantially less accurate and more uncertain. While the data-blind estimates followed the same overall trend as the true process, as a consequence of utilising the same evolution model, it failed to capture the stochastic variations in the true process, which were generated by the innovation error added at each time step. The plots in Figure 6 also suggest that the uncertainty of the data-blind estimates increased with time.

In summary, the data-blind algorithm resulted in (i) extremely high-variance estimates, which (ii) failed to capture the stochastic variations of the true process, and (iii) poorly replicated the oscillations generated by the KdV implementation, despite utilising the deterministic KdV model evolution $\mathcal{M}(\cdot)$. The EnKF analysis estimates investigated earlier usually did not

bring significant improvements over the EnKF forecasts, indicating that – at each iteration – the updates had a minor enhancing impact on performance. However, by comparing the EnKF with the data-blind algorithm, it could be clearly seen that the Bayesian updates had a substantial cumulative impact on enhancing the state estimates obtained. Moreover, the improvements in precision appear to be more pronounced the further in time the algorithm is iterated, due to the observed effect of time-increasing uncertainty for the data-blind estimates.

6 Conclusion

This project primarily focused on performing data assimilation on a KdV-governed internal wave, utilising the ensemble Kalman filter. By formulating a data-blind algorithm which only performed the EnKF forecast step at each iteration, and comparing the data-blind estimates with those of the EnKF, the importance of the EnKF update step was clearly illustrated. Since the EnKF performs DA through the update step, the results therefore clearly demonstrate the utility of DA in providing better estimates. Moreover, the foregoing EnKF implementation utilised a constant-coefficients KdV model with known parameters. For cases in which the parameters are unknown, parameter estimation methods must be employed, utilising likelihood techniques. This will be investigated in future work on this project.

References

Gerkema, T. & Zimmerman, J. (2008), 'An Introduction to Internal Waves'.

Grimshaw, R., Pelinovsky, E., Talipova, T. & Kurkin, A. (2004), 'Simulation of the Transformation of Internal Solitary Waves on Oceanic Shelves', *Journal of Physical Oceanography* **34**(12), 2774–2791.

Holloway, P. E., Pelinovsky, E., Talipova, T. & Barnes, B. (1997), 'A Nonlinear Model of Internal Tide Transformation on the Australian North West Shelf', *Journal of Physical Oceanography* **27**(6), 871–896.

Katzfuss, M., Stroud, J. R. & Wikle, C. K. (2016), 'Understanding the Ensemble Kalman Filter', *The American Statistician* **70**(4), 350–357.

Manderson, A., Rayson, M. D., Cripps, E., Girolami, M. A., Gosling, J. P., Hodkiewicz, M. R., Ivey, G. N. & Jones, N. L. (2019), 'Uncertainty Quantification of Density and Stratification Estimates with Implications for Predicting Ocean Dynamics', *Journal of Atmospheric and Oceanic Technology* **36**(7), 1313–1330.

Phillips, O. M. (1966), *The Dynamics of the Upper Ocean*, Cambridge University Press, Bentley House, 200 Euston Road, London, p. 17.

Rasmussen, C. E. & Williams, C. K. I. (2006), *Gaussian Processes for Machine Learning*, Adaptive Computation and Machine Learning, The MIT Press, 55 Hayward Street, Cambridge, Massachusetts, pp. 85–86.

Rayson, M. D. (2021), Numerical Discretisation of the Variable-Coefficient Korteweg Equation. Preprint submitted to Ocean Modelling.

Wikle, C. K. & Berliner, L. M. (2007), 'A Bayesian Tutorial for Data Assimilation', *Physica D: Nonlinear Phenomena* **230**(1-2), 1–16.

Wikle, C. K., Zammit-Mangion, A. & Cressie, N. (2019), *Spatio-Temporal Statistics with R*, CRC Press, 6000 Broken Sound Parkway NW, Suite 300, Boca Raton, Florida, pp. 215–216.

A Python Code for the Toy Example

```
"""
Code for applying the KF and EnKF to a linear Gaussian toy problem
"""

import numpy as np
import scipy.linalg as la
import matplotlib.pyplot as plt
import numpy.random as rand

plt.rcParams.update({'font.size': 14})  # Adjust font size in plots

sigma_v = 1  # Observation error std dev
sigma_w = 1  # Innovation error std dev
sigma_1 = 1  # Initial state std dev; unsuitable choice may cause burn-in

H = np.eye(2)  # Observation matrix
Q = sigma_w**2 * np.eye(2)  # Evolution covariance
R = sigma_v**2 * np.eye(2)  # Observation covariance
P_b = sigma_1**2 * np.eye(2)  # Initial covariance (P_0|0)
M = np.asarray(np.mat("0.5 -0.1; 0.1 0.2"))  # Evolution matrix w/ correlation
# M = np.asarray(np.mat("0.5 0; 0 0.2"))  # Evolution matrix w/o correlation

# Check the eigenvalues of M (must be less than one)
eigvals = la.eigvals(M)

# Cholesky factors for sampling; need lower triangular factor
L_w = la.cho_factor(Q, lower=True)[0]
L_v = la.cho_factor(R, lower=True)[0]
L_1 = la.cho_factor(P_b, lower=True)[0]

T = 500  # Temporal horizon
lw = 1  # Line width in error plots
fig = plt.figure()

for ii in range(4):

    if ii == 0:
        Ne = 5
    elif ii == 1:
        Ne = 10
    elif ii == 2:
        Ne = 30
    else:
        Ne = 1000

    ########################
    # Generate true process

    x1 = L_1 @ rand.normal(size=2)  # Initial value
    x_true = []
    x_true.append(x1)

    for t in range(T-1):
```

```
    x_true.append(M @ x_true[t] + L_w @ rand.normal(size=2))  # Evolution model

########################
# Generate observations

y = []
for t in range(T):
    y.append(H @ x_true[t] + L_v @ rand.normal(size=2))  # Observation model

################
# Kalman filter

x_b = L_1 @ rand.normal(size=2)  # Background state (x_0|0)

kf_mean = []  # Analysis mean (x_t|t)
kf_mean.append(x_b)

kf_var = []  # Analysis variance (P_t|t)
kf_var.append(P_b)

for t in range(T-1):

    # Forecast step

    x_f = M @ kf_mean[t]  # Forecast mean (x_t|t-1)
    P_f = Q + M @ kf_var[t] @ M.T  # Forecast variance (P_t|t-1)

    # Update / Analysis step

    K1 = P_f @ H.T @ la.inv(H.T @ P_f @ H + R)  # Kalman gain matrix
    kf_mean.append(x_f + K1 @ (y[t+1] - H @ x_f))  # Analysis mean (x_t|t)
    kf_var.append((np.eye(2) - K1 @ H) @ P_f)  # Analysis variance (P_t|t)

#########################
# Ensemble Kalman filter

ens = []  # List for the ensemble

for n in range(Ne):
    ens.append(L_1 @ rand.normal(size=2))  # Generate initial ensemble

enkf_mean = []  # Analysis mean MC estimates (x_t|t)
enkf_mean.append(np.mean(ens, axis=0))  # Initial analysis mean (x_0|0)

# Compute sample covariance matrix of initial ensemble
init_cov = 0
for n in range(Ne):
    cov_factor = ens[n] - enkf_mean[0]
    init_cov += np.outer(cov_factor, cov_factor) / (Ne - 1)

enkf_var = []  # Analysis sample covariances (P_t|t)
enkf_var.append(init_cov)  # Initial analysis covariance (P_0|0)

for t in range(1, T):
```

```
    """
    Forecast step
    """

    for n in range(Ne):
        ens[n] = M @ ens[n] + L_w @ rand.normal(size=2)  # Forecast ensemble members
    x_f = np.mean(ens, axis=0)  # Forecast mean (x_t|t-1)

    # Compute forecast covariance (P_t|t-1)
    P_f = np.zeros((2, 2))
    for n in range(Ne):
        cov_factor = ens[n] - x_f
        P_f += np.outer(cov_factor, cov_factor) / (Ne - 1)

    """
    Update / Analysis step
    """

    K2 = P_f @ H.T @ la.inv(H @ P_f @ H.T + R)  # Kalman gain matrix
    for n in range(Ne):
        ens[n] = ens[n] + K2 @ (y[t] + L_v @ rand.normal(size=2) - H @ ens[n])  # Update forecasts
    x_a = np.mean(ens, axis=0)  # MC analysis mean (x_t|t)

    # Compute analysis covariance (P_t|t)
    P_a = np.zeros((2, 2))
    for n in range(Ne):
        cov_factor = ens[n] - x_a
        P_a += np.outer(cov_factor, cov_factor) / (Ne - 1)

    # Store the analysis mean and covariance estimates
    enkf_mean.append(x_a)
    enkf_var.append(P_a)

############################
# Compute cumulative errors

err = np.repeat(0, 3)  # Cumulative error vector at t=T [KF ENKF OBS]
err_cumul = np.zeros((3, T))  # Cumulative error matrix

for t in range(T):
    kf_err = la.norm(kf_mean[t] - x_true[t], ord=2)  # KF error
    enkf_err = la.norm(enkf_mean[t] - x_true[t], ord=2)  # EnKF error
    obs_err = la.norm(y[t] - x_true[t], ord=2)  # Observation error

    err[0] += kf_err  # Cumulative KF error
    err[1] += enkf_err  # Cumulative EnKF error
    err[2] += obs_err  # Cumulative observation error

    # Store cumulative errors
    err_cumul[0, t] = err[0]
    err_cumul[1, t] = err[1]
    err_cumul[2, t] = err[2]

#################
# Plot the output
```

```
    x = np.arange(1, T+1)
    plt.subplot(2, 2, ii+1)
    plt.title("N = " + str(Ne))
    plt.plot(x, err_cumul[0, :], 'r-', label="KF", linewidth=lw)  # First row (KF)
    plt.plot(x, err_cumul[1, :], 'b-', label="EnKF", linewidth=lw)  # Second row (EnKF)
    plt.plot(x, err_cumul[2, :], 'g-', label="Obs", linewidth=lw)  # Third row (observations)

fig.text(0.5, 0.04, 'Time ($t^*$)', ha='center')
fig.text(0.07, 0.5, 'Cumulative Error ($R(\cdot)$)', va='center', rotation='vertical')

plt.show()
```

B Python Code for KdV and EnKF Implementation

```
"""
Code for generating the KdV and running the EnKF for state inference
"""

import numpy as np
from iwaves.kdv.vkdv import vKdV  # KdV code provided by Matt Rayson
import matplotlib.pyplot as plt
import scipy.linalg as la
import numpy.random as rand
import scipy.special as spec

# IMEX options (IMEX was the discretisation method used)
imex = {
    'MCN_AX2': (1 / 8., 3 / 8.),
    'AM2_AX2': (1 / 2., 1 / 2.),
    'AI2_AB3': (3 / 2., 5 / 6.),
    'BDF2_BX2': (0., 0.),
    'BDF2_BX2s': (0., 1 / 2.),
    'BI2_BC3': (1 / 3., 2 / 3.),
}
imexscheme = 'AM2_AX2'
#imexscheme = 'MCN_AX2'
c_im = imex[imexscheme][0]
b_ex = imex[imexscheme][1]

# DHT function to create background density profile (from Manderson et al. 2019)
def rho_double_tanh(beta, z):
    return beta[0] - beta[1] * (np.tanh((z + beta[2]) / beta[3]) + np.tanh((z + beta[4]) / beta[5]))

# Sinusoidal boundary condition
def bcfunc(a0,t):
    T = 12*3600
    omega = 2*np.pi/T
    return a0*np.sin(omega*t)


##################################
# Inputs for the KdV Simulation
```

```
a0 = 20.  # Amplitude of initial sinusoidal disturbance

# Physical meaning of density parameters (from Manderson et al. 2019)
# beta0: Approximate mean density over the profile
# beta1: A scale for the density difference across the water column
# beta2: Middepth of the upper pycnocline
# beta3: Upper pycnocline width
# beta4: Middepth of the lower pycnocline
# beta5: Lower pycnocline width

avg_density = 1023.9
pyc_scale = 0.91
upper_pyc_depth = 30
upper_pyc_width = 40
lower_pyc_depth = 117
lower_pyc_width = 52

# NOTE: order of beta coefficients in vector input is 0,1,4,5,2,3
rho_params = np.array([avg_density, pyc_scale, lower_pyc_depth, lower_pyc_width, upper_pyc_depth,
                       upper_pyc_width])

H = 350  # Oceanic depth in metres
Nz = 50  # Number of discretised vertical points

# Time units are seconds
dt = 15  # 15 second time resolution
runtime = 24 * (6 * 600)  # Temporal horizon of 24 hrs / 1440 min
nsteps = int(runtime//dt)  # Number of 15 second steps until temporal horizon

# Spatial units are metres
Nx = 2000  # Number of discretised horizontal points (dimension of state vector)
dx = 50.  # 50 metre horizontal spatial resolution
L_d = Nx*dx  # Length of horizontal domain

mode = 0  # Wave mode
kdvargs = dict(
   N=Nx,
   dx=dx,
   dt=dt,
   spongedist=5e3,
   spongetime=60.,
   Nsubset=10,
   nonhydrostatic=1.,
   nonlinear=1.,
   c_im=c_im,
   b_ex=b_ex,
)

z = np.linspace(-H, 0, Nz)  # Vertical domain
x = np.arange(0, L_d, dx)  # Horizontal domain
rhoz = rho_double_tanh(rho_params,z)  # Background density profile, ordered from seabed to surface
h = np.repeat(H, repeats=Nx)  # Flat ocean floor
```

```
########################################################################
# Simulate the KdV to generate the true process and synthetic data

mykdv0 = vKdV(rhoz, z, h, x, mode, **kdvargs)  # Variable-coefficients KdV object
x_obs = np.arange(0, int(L_d * 1), L_d//20)  # Choose observation spatial points
Nx_obs = len(x_obs)  # Number of entries in observation vector

# Exception to enforce subset restriction
x_relcomp = np.setdiff1d(x_obs, x)
if len(x_relcomp) > 0:
    raise Exception("Error: Observational spatial points must be contained in horizontal domain.")

# Distance matrix (used to construct covariance matrix of state vectors)
dist_x = np.insert(arr=x, obj=0, values=np.flip(x[1:]))
dist_mx = np.zeros((Nx, Nx))
for k in range(Nx):
    dist_mx[:, k] = np.flip(dist_x[k:(Nx + k)])

sigma_v = 0.1  # Observation error std dev
sigma_w = 0.1  # Innovation error std dev
sigma_1 = sigma_w  # Initial state std dev

# Matern covariance function (p=0 only)
def matern_p0(d, l=x[-1]/3):
    return np.exp(-d / l)

# Matern covariance function (general form)
def matern_cov(d, p=0, l=x[-1]/3):
    nu = p + 0.5
    gamma_term = spec.gamma(p + 1) / spec.gamma(2*p + 1)
    sum_term = 0
    for i in range(p + 1):
        fact_term = spec.gamma(p + i + 1) / (spec.gamma(i + 1) * spec.gamma(p - i + 1))
        power_term = (np.sqrt(8 * nu) * d / l) ** (p - i)
        sum_term += fact_term * power_term
    return np.exp(-d * np.sqrt(2 * nu) / l) * gamma_term * sum_term

# Covariance matrices
R = sigma_v ** 2 * np.eye(Nx_obs)  # Observation covariance matrix
Q = sigma_w ** 2 * matern_p0(dist_mx)  # Evolution covariance matrix (incorporating spatial correlations)
P_b = Q  # Background covariance matrix (initial analysis covariance P_0|0)

# Cholesky factors (used for random sampling)
L_v = la.cho_factor(R, lower=True)[0]
L_w = la.cho_factor(Q, lower=True)[0]
L_1 = la.cho_factor(P_b, lower=True)[0]

# Observation matrix
H = np.zeros(shape=(Nx_obs, Nx))
x_index = np.intersect1d(x, x_obs, return_indices=True)[1]
H[:, x_index] = np.eye(Nx_obs)

timestep = 10*60  # Ten minute time steps for EnKF algorithm
n_dt = timestep // dt  # Number of 15 second steps in each EnKF time step
n_obs = nsteps // n_dt + 1  # Number of time steps with data present
```

```
# Initialise solution/data matrices; rows for time points and columns for spatial points
B_true = np.zeros(shape=(n_obs, Nx))  # True solution values
B_obs = np.zeros(shape=(n_obs, Nx_obs))  # Synthetic data values

# Compute KdV solution values
for ii in range(nsteps + 1):
    if ii % n_dt == 0:
        B_true[ii // n_dt, :] = mykdv0.B + L_w @ rand.normal(size=Nx)
        B_obs[ii // n_dt, :] = H @ B_true[ii // n_dt, :] + L_v @ rand.normal(size=Nx_obs)
        mykdv0.B = B_true[ii // n_dt, :]  # Store values in KdV object
    if mykdv0.solve_step(bc_left=bcfunc(a0, mykdv0.t)) != 0:  # Advance KdV forward in time
        print('Blowing up at step: %d' % ii)  # Safety check
        break


######################################################################
# Run an EnKF to infer the true process from the simulated data

Ne = 30  # Number of ensemble members
ens = []  # List for the ensemble

for n in range(Ne):
    ens.append(L_1 @ rand.normal(size=Nx))  # Generate initial ensemble (background state)

enkf_mean = [np.mean(ens, axis=0)]  # Analysis mean estimates (x_t|t)
enkf_prior = [np.mean(ens, axis=0)]  # Forecast mean estimates (x_t|t-1)
enkf_var = [P_b]  # Analysis sample covariances (P_t|t)
enkf_varF = [P_b]  # Forecast sample covariances (P_t|t)

rho_ens = np.tile(rhoz, reps=(Ne, 1)).T  # Columns are density profiles for each ensemble member
B_ens = np.zeros(shape=(Ne, n_obs, Nx))  # Ensemble array for all time steps; shape = # sets/rows/columns
for n in range(Ne):
    B_ens[n, 0, :] = ens[n]

mykdv = []
for n in range(Ne):
    mykdv.append(vKdV(rhoz, z, h, x, mode, **kdvargs))  # Distinct KdV objects initialised identically

# Function to step KdV forward by one time step (used in EnKF forecasts)
def kdv_stepper(k):
    for ii in range(n_dt):
        if mykdv[k].solve_step(bc_left=bcfunc(a0, mykdv[k].t)) != 0:
            print('Blowing up at step: %d' % ii)
            break
    density = np.flip(np.mean(mykdv[k].rhoZ, axis=1))
    mykdv[k].B = mykdv[k].B + L_w @ rand.normal(size=Nx)
    return mykdv[k].B, density

# Run the EnKF forward in time
for t in range(1, n_obs):

    """
    Forecast step
    """
```

```
    for n in range(Ne):
        ens_fwd_n = kdv_stepper(n)  # Forecast each ensemble member forward in time
        B_ens[n, t, :] = ens_fwd_n[0]  # Store ensemble forecasts
        rho_ens[:, n] = ens_fwd_n[1]  # Store density profiles
        ens[n] = B_ens[n, t, :]  # Update ensemble list with forecasts
        mykdv[n].B = ens[n]  # Store forecasts in associated KdV object

    x_f = np.mean(ens, axis=0)  # Forecast mean (x_t|t-1)

    # Compute forecast covariance (P_t|t-1)
    P_f = np.zeros((Nx, Nx))
    for n in range(Ne):
        cov_factor = ens[n] - x_f
        P_f += np.outer(cov_factor, cov_factor) / (Ne - 1)

    """
    Update / Analysis step
    """

    K2 = P_f @ H.T @ la.inv(H @ P_f @ H.T + R)  # Kalman gain matrix
    for n in range(Ne):
        ens[n] = ens[n] + K2 @ (B_obs[t, :] + L_v @ rand.normal(size=Nx_obs) - H @ ens[n])  # Update forecasts
        mykdv[n].B = ens[n]  # Store updates in associated KdV object
    x_a = np.mean(ens, axis=0)  # MC analysis mean (x_t|t)

    # Compute analysis covariance (P_t|t)
    P_a = np.zeros((Nx, Nx))
    for n in range(Ne):
        cov_factor = ens[n] - x_a
        P_a += np.outer(cov_factor, cov_factor) / (Ne - 1)

    # Store the analysis mean and covariance estimates
    enkf_mean.append(x_a)
    enkf_var.append(P_a)

    # Store the forecast mean and covariance estimates
    enkf_prior.append(x_f)
    enkf_varF.append(P_f)

B_enkf = np.asmatrix(enkf_mean)  # Array of EnKF analysis estimates
B_prior = np.asmatrix(enkf_prior)  # Array of EnKF forecast estimates
obs_index = np.intersect1d(x, x_obs, return_indices=True)[1]  # Observation indices within state vector

# Examine KdV parameters
print(mykdv[0].to_Dataset().data_vars)
mykdv[0].print_params()


##################
# Plot the output

plt.rcParams.update({'font.size': 14})  # Adjust font size in plots

# Function to plot KdV process and estimates as function of distance
```

```
def plot_kdv_space(prior=False, enkf_spread=False, prior_spread=False, only_prior=False, spread=False,
                   legend=False):

    if spread == True:
        prior_spread = True
        enkf_spread = True
    if only_prior == True:
        prior_spread = True
    if prior_spread == True:
        prior = True

    # Plot the true KdV solution against 1d space
    plt.figure()
    plt.plot(x, B_true[n_obs - 1], color="tab:blue", label="True")

    # Plot the EnKF prior estimates against 1d space
    if prior == True:
        if only_prior == False:
            plt.plot(x, enkf_prior[n_obs - 1], linestyle=":", color="tab:green", label="Forecast")
        else:
            plt.plot(x, enkf_prior[n_obs - 1], linestyle="--", color="tab:green", label="Initial Forecast")
        if prior_spread == True:
            stdev_f = []  # List containing standard deviations for the spatial point over time
            for d in range(Nx):
                stdev_f.append(np.sqrt(enkf_varF[nsteps // n_dt][d, d]))
            stdev_f = np.asarray(stdev_f)
            mean_f = np.asarray(B_prior[nsteps // n_dt, :]).T  # Require column 1d array for len() to work
            mean_f = mean_f.reshape(len(mean_f))
            plt.fill_between(x, mean_f - 2 * stdev_f, mean_f + 2 * stdev_f, alpha=0.3, color="tab:green")

    # Plot the EnKF posterior estimates against 1d space
    if only_prior == False:
        plt.plot(x, enkf_mean[n_obs - 1], linestyle="--", color="tab:orange", label="Analysis")
        if enkf_spread == True:
            stdev_a = []  # List containing standard deviations for the spatial point over time
            for d in range(Nx):
                stdev_a.append(np.sqrt(enkf_var[nsteps//n_dt][d, d]))
            stdev_a = np.asarray(stdev_a)
            mean_a = np.asarray(B_enkf[nsteps//n_dt, :]).T  # Require column 1d array for len() to work
            mean_a = mean_a.reshape(len(mean_a))
            plt.fill_between(x, mean_a - 2 * stdev_a, mean_a + 2 * stdev_a, alpha=0.3, color="tab:orange")

    # Plot the synthetic observations against 1d space
    plt.plot(x_obs, B_obs[n_obs - 1], linestyle="", marker="o", color="tab:red", label="Observations")

    plt.xlabel("Distance (m)")
    plt.ylabel("Amplitude (m)")
    if legend == True:
        plt.legend()

    plt.show()

# Function to plot KdV process and estimates as function of time
def plot_kdv_time(which_x=[0], prior=False, enkf_spread=False, prior_spread=False, only_prior=False,
                  spread=False, position="right", legend=False):
```

```
# Note: which_x input must be a list containing x value indices of interest

# Inputs
td = np.arange(0, runtime + 1, timestep)  # Time domain (10 minute intervals)
xstate_step = x[1]  # Spatial gap between state vector entries
xobs_step = x_obs[1]  # Spatial gap between observations
n_panes = len(which_x)  # Number of panes to plot (for each spatial point of interest)

if position not in ["left", "right"]:
    raise Exception("Invalid positional argument.")
if position == "left":
    legend_pos = "upper right"
else:
    legend_pos = "upper left"

if spread == True:
    prior_spread = True
    enkf_spread = True
if only_prior == True:
    prior_spread = True
if prior_spread == True:
    prior = True

# Plot KdV solution against time
plt.figure()
for p in range(n_panes):
    ax1 = plt.subplot(n_panes, 1, p + 1)
    xx = which_x[p]  # Indexing for list input
    plt.plot(td, B_true[:, xx], color="tab:blue", label="True")
    if only_prior == False:
        plt.plot(td, B_enkf[:, xx], linestyle="--", color="tab:orange", label="Analysis")

    # Standard deviation bands for analysis estimates
    if enkf_spread == True:
        stdev_a = []  # List containing standard deviations for the spatial point over time
        for k in range(n_obs):
            stdev_a.append(np.sqrt(enkf_var[k][xx, xx]))
        stdev_a = np.asarray(stdev_a)
        mean_a = np.asarray(B_enkf[:, xx])
        mean_a = mean_a.reshape(len(mean_a))
        plt.fill_between(td, mean_a - 2 * stdev_a, mean_a + 2 * stdev_a, alpha=0.3, color="tab:orange")

    if prior == True:
        if only_prior == False:
            plt.plot(td, B_prior[:, xx], linestyle=":", color="tab:green", label="Forecast")
        else:
            plt.plot(td, B_prior[:, xx], linestyle="--", color="tab:green", label="Initial Forecast")

        # Standard deviation bands for forecast estimates
        if prior_spread == True:
            stdev_f = []  # List containing standard deviations for the spatial point over time
            for k in range(n_obs):
                stdev_f.append(np.sqrt(enkf_varF[k][xx, xx]))
            stdev_f = np.asarray(stdev_f)
```

```
            mean_f = np.asarray(B_prior[:, xx])
            mean_f = mean_f.reshape(len(mean_f))
            plt.fill_between(td, mean_f - 2 * stdev_f, mean_f + 2 * stdev_f, alpha=0.3, color="tab:green")

        # Include observations when possible
        if len(np.intersect1d(obs_index, xx)) > 0:
            plt.plot(td, B_obs[:, int(xx * xstate_step // xobs_step)], marker=".", linestyle="",
                     color="tab:red", label="Observations")

        if p != n_panes - 1:
            ax1.axes.xaxis.set_ticklabels([])
            plt.tick_params(bottom=False)
            plt.ylabel("Amplitude (m)")

        if position == "right":
            plt.text(runtime, a0, str.format("x = {0} m", int(x[xx])), ha="center", va="center")
        else:
            plt.text(0, a0, str.format("x = {0} m", int(x[xx])), ha="left", va="center")

        if legend == True:
            plt.legend(loc=legend_pos)

    plt.xlabel("Time (sec)")
    plt.ylabel("Amplitude (m)")

    plt.show()

# Function to plot ensemble members as function of time
def plot_ens_time(xx=0):

    td = np.arange(0, runtime + 1, timestep)  # Time domain (10 minute intervals)

    fig = plt.figure()
    fig.suptitle(str.format("Ensemble Members at Distance x = {0} m with N = {1}", x[xx], Ne))

    # Plot each ensemble member
    plt.plot(td, B_ens[0][:, xx], linestyle="--", linewidth=1, label="Members")
    for ee in range(1, Ne):
        plt.plot(td, B_ens[ee][:, xx], linestyle="--", linewidth=1)

    # Plot ensemble mean
    plt.plot(td, B_prior[:, xx], linewidth=1.2, label="Mean", color="k")

    plt.ylabel("Amplitude (m)")
    plt.xlabel("Time (sec)")
    plt.grid(b=True)
    plt.legend()

    plt.show()

# Function to plot ensemble members as function of distance
def plot_ens_space():

    fig = plt.figure()
    fig.suptitle(str.format("Final-Time Ensemble Members with N = {0}, T = {1} min", Ne, runtime/60))
```

```
# Plot each ensemble member
plt.plot(x, B_ens[0][nsteps // n_dt, :], linestyle="--", linewidth=1, label="Members")
for ee in range(1, Ne):
    plt.plot(x, B_ens[ee][nsteps//n_dt, :], linestyle="--", linewidth=1)

# Plot ensemble mean
plt.plot(x, B_prior[nsteps//n_dt, :].T, linewidth=1.2, label="Mean", color="k")

plt.ylabel("Amplitude (m)")
plt.xlabel("Distance (m)")
plt.grid(b=True)
plt.legend()

plt.show()
```