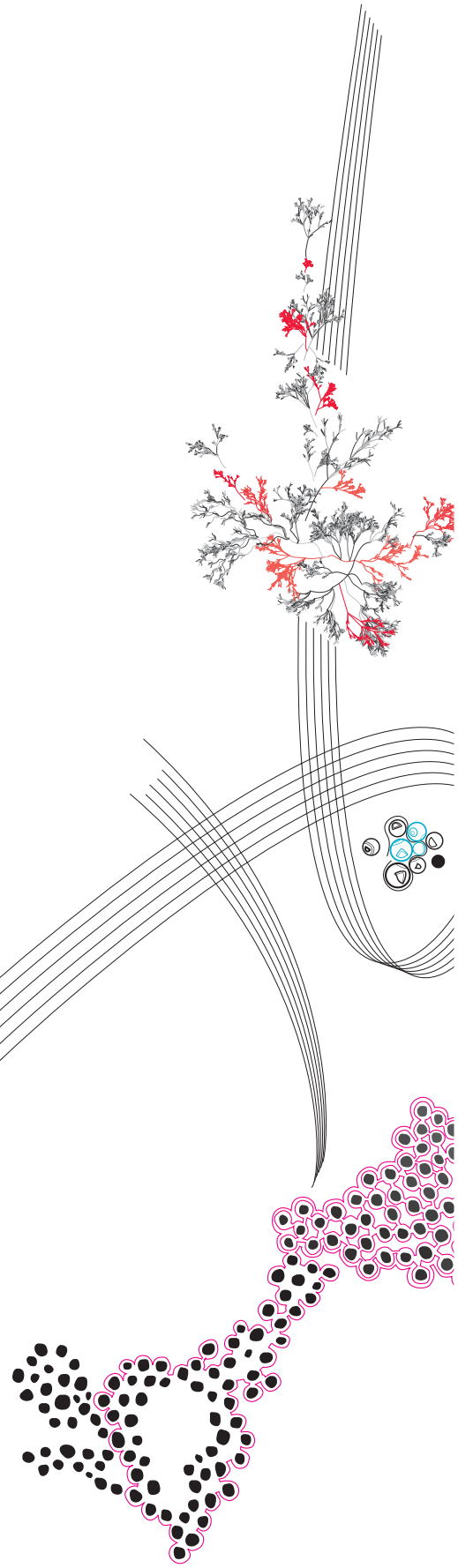


MSc Thesis Applied Mathematics

Investigation of 4DVarNet Algorithm for Image Reconstruction of Suspended Particulate Matter Dynamics Data

Fabio Mistrangelo

Supervisors: D. Ye (UT), F. Dols (Deltares), L. Meszaros (Deltares), C. Brune (UT)

November 18, 2024

Department of Applied Mathematics
Faculty of Electrical Engineering,
Mathematics and Computer Science

UNIVERSITY OF TWENTE.

Contents

1 Introduction **4**

2 Mathematical background **7**
2.1 Preliminaries . . . 7
2.2 Bayesian inference . . . 9
2.3 Recursive Bayesian formulation . . . 9
2.4 Problem formulation . . . 11
2.4.1 General smoother formulation . . . 11
2.4.2 Smoother formulation for perfect models . . . 12
2.4.3 Smoother formulation for parameters estimation . . . 12
2.4.4 Filter formulation . . . 13
2.4.5 Recursive smoother formulation . . . 13
2.5 Data assimilation: Gaussian assumption . . . 14
2.6 Data assimilation methods . . . 15
2.6.1 Optimal interpolation . . . 16
2.6.2 Kalman filter . . . 18
2.6.3 4DVar . . . 19
2.6.4 Particle filters . . . 23

3 Methods **24**
3.1 4DVarNet . . . 24
3.1.1 Problem statement of WC-4DVar . . . 24
3.1.2 Novelties . . . 25
3.1.3 End-To-End architecture . . . 27
3.1.4 Double-LSTM 4DVarNet . . . 28
3.2 Observing system experiment and observing system simulation experiments . . . 29
3.2.1 OSSE . . . 29
3.2.2 OSE . . . 30
3.3 Evaluation metrics . . . 30

4 Experiments and Results **31**
4.1 Data . . . 31
4.1.1 Satellite data . . . 31
4.1.2 Model data . . . 32
4.1.3 Data preprocessing . . . 32
4.2 Experiments . . . 33
4.3 Results . . . 35
4.3.1 Metrics . . . 35
4.3.2 Visual reconstruction . . . 36
4.3.3 Time series reconstruction . . . 38

5 Discussion **41**
5.1 Explanation of the results . . . 41
5.2 Limitations . . . 42
5.2.1 Data . . . 42
5.2.2 Second experiment . . . 44
5.2.3 Model complexity . . . 44
5.3 Related works and future researches . . . 44

6 Conclusion and Overlooks **46**

7 Acknowledgments **47**

8 Appendixes **48**
8.1 Appendix A . . . 48
8.2 Appendix B . . . 48

Bibliography **50**

Chapter 1

Introduction

Suspended particulate matter (SPM) consists of a mix of mineral or terrigenous sediments and particulate organic matter. This mixture is influenced by waves, tides, and marine currents, with contributions from both terrestrial and marine sources [6]. Apart from the natural forcing, which is the main cause of SPM, anthropogenic forces also commit to the formation of SPM. For example, activities such as fish trawling and maritime development including harbor sediment dredging and dumping, offshore wind farms, oil and gas pipelines also contribute [14].

The understanding of the SPM dynamics and the resulting turbidity contribute to the study of the growth of micro-organisms based on photosynthesis [30]. In particular, studies have shown how high concentrations of SPM harm marine ecosystems, especially those with rich organic sediments [55, 24, 48, 40, 68]. Those sediments tend to stick to coral tissue and seagrass leaves, which are species that have difficulties in removing this type of sediments compared to more inorganic-rich ones. Moreover, due to its lower density, organic SPM can be more easily resuspended and contribute to incrementing the turbidity of the water, even for long periods. It will result in the reduction of acidity and oxygen concentration in the water.

The study of SPM dynamics in some areas is more challenging. For example, the Dutch part of the Wadden Sea, see Fig. 1.1, is a shallow water area with a high concentration of SPM due to the sediments coming from the river estuary and the sediments from the islands, which often get partly flooded, spreading mud and sand in the nearby area.

In order to investigate the SPM concentration and dynamics, different types of data are typically collected and studied, including simulation data based on computational models, in situ data and satellite data using remote sensing. Simulation data provide complete and continuous spatio-temporal information over the study area, but due to the complex underlying processes, these data obtained from the computational models are subjected to simplifications and assumptions, resulting in discrepancy and uncertainty. In situ data contain very accurate data and are often even more reliable than satellite data. But they only present local information of where the sensors are placed and often the measurements are limited to a time window. For this reason, they are often used to validate the models or considered as reliable sourced of information to be integrated to the data from other sources. Finally, satellite data provide more realistic information, but they also involve limitations, such as instrumental errors and missing information due to cloud coverage on the study area. Therefore, the availability of SPM satellite and complete satellite data of the area is of great value for ecological and biogeochemical modellingmaking image reconstruction techniques essential for recovering missing portions of the satellite data.

A wide variety of methods have been developed for image reconstruction [17] and applied in oceanography. Numerical interpolation-based methods, i.e. optimal interpolation and kriging [31, 62, 61, 15], combine observed and simulated data to minimize the error of the interpolated field, using both spatial correlations of the data and model's error characteristics to create the best-guess estimate; a combination of Empirical Orthogonal Functions (EOF), also known as Principal Component Analysis [52], with data interpolation was proposed by Beckers et. al. [11], resulting in the so-called DInEOF. This is an iterative method that is able to identify dominant modes of variability and reconstruct missing values in a way that captures the main spatial and temporal structures of the given dataset. DInEOF, according to [17], is widely applied in real world scenarios

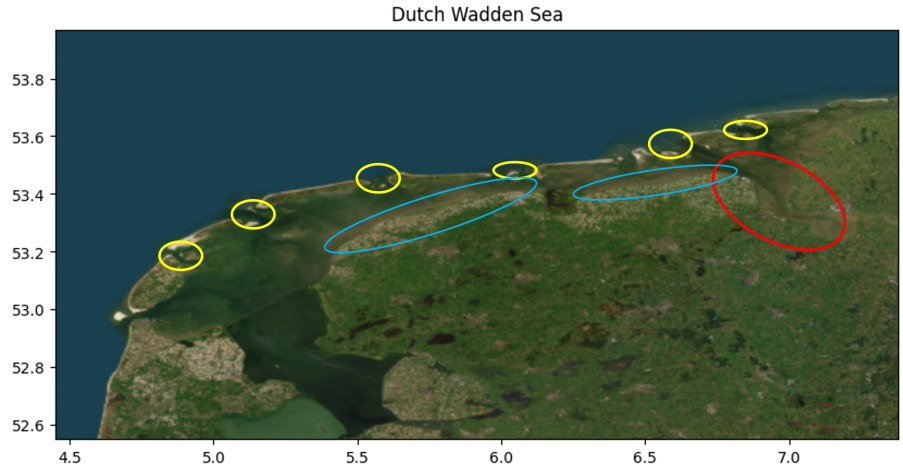


Figure 1.1: Satellite map of the Dutch Wadden Sea area. Tidal flats in between islands (yellow circles), flooding on the mainland (blue circles) and river estuary (red circle) all contribute to the SPM concentration in the area.

[50, 2, 69, 49, 75, 32] and researches [1, 60] of the past years. For example, Copernicus Marine Service provides some L4 products obtained with the use of DInEOF reconstructions. Other statistical methods have been used for data gap-filling, such as support vector regression, k-nearest neighbours and random forest/decision tree, but they gathered less attention compared to the two previous methods. However, all these methods present some issues, such as handling highly non-linear systems, being sensitive to the noise of the data or dealing with sparse data.

With the fast development of powerful computational resources (GPUs), machine learning methods have been significantly developed, showing better performances than classical numerical methods on various tasks. In the last decades, the application of machine learning techniques on image reconstruction has been widely explored [66, 56, 47, 59, 42, 41, 67]. Neural networks (NN) are getting more attention due to their flexibility and performance in handling complex, nonlinear relationships in data and their ability to generalize patterns without extensive feature engineering, making them especially valuable in domains with high-dimensional data, such as computer vision/imaging or natural language process. In particular, architectures as Convolutional Neural Network (CNN) and Autoencoders (AE) are the common techniques in computer vision tasks. CNNs' architecture leverages spatial hierarchies in images through a series of convolutional layers that can detect global and local complex features, such as edges, textures or shapes. It has been showcased its effectiveness in tasks such as object recognition, classification, and segmentation of images. On the other hand, AEs can efficiently learn low-dimensional representations (encodings) of data, which can capture essential features while reducing noise and dimensionality of the problem. In computer vision, this capability is valuable for scenarios such as image compression, denoising, and anomaly detection, where it is important to maintain image quality while reducing complexity. The advantage of these methods compared to traditional algorithms, other than computational efficiency, lies in the ability to automatically learn complex features and patterns, and it has driven their applications also in image gap-filling tasks [23, 16, 46, 7, 37, 58, 45, 22].

Amongst all these works, the method of Fablet et al. [28], based on the Weak-Constraint formulation of 4DVar (2.6.3) data assimilation method, demonstrated to be suitable for the reconstruction of SPM fields [73, 72]. The method they proposed is called 4DVarNet and it is an End-To-End algorithm for reconstruction and approximation of complex dynamical systems, which combines the classical data assimilation method framework with some neural network elements. The resulting architecture showed faster convergence and better accuracy than other data assimilation methods, making it appealing for satellite image reconstruction tasks.

The aim of this research is to apply and investigate 4DVarNet algorithm for image reconstruction of SPM data for the Dutch Wadden Sea area and compare it to two numerical methods, DInEOF and eDInEOF. More specifically, the 4DVarNet algorithm was implemented and trained on

an Observed System Simulated Experiment (OSSE) setup 3.2. OSSE is commonly used in oceanography because it efficiently combines information from simulated and satellite data. 4DVarNet was trained on simulated data from DELFT3D numerical model and tested on satellite data from CMEMS (see Sec. 4.1). Then, its performances were analysed and compared with two classical reconstruction methods, DInEOF and eDInEOF. A variation of 4DVarNet is also proposed, called Double LSTM, which exploits the Long Short Term Memory (LSTM) architecture of 4DVarNet to transfer information from one reconstructed batch to the next one.

This thesis work was carried out at Deltares, as part of the project EDITO Model Lab, aiming to develop a digital twin of the ocean "*to make ocean knowledge readily available to citizens, entrepreneurs, scientists and policy-makers by providing them with an innovative set of user-driven, interactive and visualisation tools*"[1].

The rest of the thesis is arranged as follows. In Chapter 2 the mathematical basis of data assimilation will be introduced and some of the most known methods will be briefly discussed. Chapter 3 introduces 4DVarNet, its novelties, the generic benchmarking framework for experiments and the metrics used. Chapter 4 is abouth the experiments setups and the numerical results. The discussion of the numerical results and related works is in Chapter 5. Finally, Chapter 6 talks about the conclusions and future outlooks.

[1]EDITO project: https://www.edito.eu.

Chapter 2

Mathematical background

Data assimilation is a statistical discipline, mainly developed for meteorology and earth sciences as a method for forecasting, combining theoretical knowledge (for example numerical models) and real-world observations. It heavily relies on mathematics, since it is assumed to have a numerical model of the dynamical system that we are studying. In the past decades, the application of data assimilation grew beyond forecasting and expanded to further tasks, such as finding the optimal state of a dynamical system, determining the initial conditions, estimating the model parameters or image reconstruction from sparse observations, thanks to its ability in combining data from different sources, which is a feature that could be applied to other fields.

The common mathematical approach to data assimilation is based on Bayes' theorem [27, 65] and then defining the problem setup. This chapter will briefly introduce the mathematical basis of data assimilation and common methods including Optimal Interpolation (OI) [31], Kalman filter ([44]), variational methods (3DVar and 4DVar) [51, 20], particle filters methods (sequential Monte Carlo [33]).

2.1 Preliminaries

This section aims to introduce the common elements of data assimilation to build the mathematical framework used for every method.

An assimilation time window is the temporal frame over which the data assimilation problem is solved. It should be a continuous time window, but since it is working with real-world applications, it will be considered a sequence of assimilation time windows in a discrete-time setup.

Each of the different data assimilation techniques uses in its own way the time window and they could be either regular intervals or irregular. For example, Kalman filters define the time window as the interval between each available measurement. They are not necessarily regularly sampled. Moreover, some methods assume that the time windows are independent (Optimal Interpolation, 3DVar) meanwhile others assume that they are sequentially dependent (4DVar, particle filters, Ensemble Kalman Filters [25], Extended Kalman Filter [43]).

Consider a model m representing the dynamics of a real-world system,

$$x_t = m(x_{t-1}, \theta, u_t, q_t) \tag{2.1}$$

where $x_t \in \mathbb{R}^n$ denotes the state of the system at time instance t, $\theta \in \mathbb{R}^p$ denotes the parameters of the system, $u_t \in \mathbb{R}^r$ is the sequence of model control which represents the time-dependent uncertain model forcing and $q_t \in \mathbb{R}^q$ denotes the sequence of model errors. It takes missing physics in the model equations and numerical discretization errors into consideration. The uncertainty over the assimilation time window is defined as follows,

$$x_0 = \hat{x}_0 + x'_0, \tag{2.2a}$$
$$\theta = \hat{\theta} + \theta', \tag{2.2b}$$
$$q_t^\top = \hat{q}_t^\top + q_t'^\top, \tag{2.2c}$$
$$u_t^\top = \hat{u}_t^\top + u_t'^\top, \tag{2.2d}$$

where $\hat{x}_0$, $\hat{\theta}$, $\hat{q}_t$ and $\hat{u}_t$ are the uncertain initial condition, prior values of parameters, uncertain model error and uncertain model controls respectively, while x'_0, θ', q'_t and u'_t are their corresponding uncertainties.

The sequence $x = \{x_0^\top, ..., x_K^\top\}$, where $x_i^\top \in \mathbb{R}^n$, is defined as the sequence of state model vectors over an assimilation time window. It describes the state of the dynamical system (or the model m we use to approximate it) at time t. The sequence $\bar{x}^\top = \{\bar{x}_0^\top, ..., \bar{x}_K^\top\} \in \mathbb{R}^n$ represents the sequence of true state vectors over an assimilation time window. Let's assume that the true state vectors evolve in time according to the equation:

$$\bar{x}_t = m(\bar{x}_{t-1}, \theta, u_t, q_t) + g(t-1), \tag{2.3}$$

where $g : \mathbb{Z} \longrightarrow \mathbb{R}^n$ is a function that it is supposed to be always unknown. This implies that the true state will be never known, so the goal of data assimilation is to find the best approximation of the true state, that is the model state.

Let's define the vector z as the state vector containing all the uncertainties to estimate in our data assimilation problem. The form of z has different variations depending on the problem, and generally can be summarized into two main formulations [27]:

1. $z^\top = (x^\top, \theta^\top, u^\top)$, which includes model state x (or model prediction) and the model error q is excluded here. This is called model-state formulation and it updates directly the model state x.

2. $z^\top = (x_0^\top, \theta^\top, u^\top, q^\top)$, which includes the model error as uncertain quantity that is desired to be estimated. This is known as forcing formulation because the estimated model errors q force the model. It can be noticed that given the form of z we can directly determine the model state x.

To provide an general version of the problem setting, multiple assimilation windows formulations are introduced here: given L assimilation windows, the model state is defined as the model-state trajectory $\mathcal{X}^\top = (x_1^\top, ..., x_L^\top)$, the model controls as the time sequence $\mathcal{U}^\top = (u_1^\top, ..., u_L^\top)$ and the model errors as the time sequence $\mathcal{Q}^\top = (q_1^\top, ..., q_L^\top)$. Therefore, the previous two formulations could be generalized over multiple assimilation windows in the notation $\mathcal{Z}^\top = (\mathcal{X}^\top, \theta^\top, \mathcal{U}^\top)$, or $\mathcal{Z}^\top = (\mathcal{X}_0^\top, \theta^\top, \mathcal{U}^\top, \mathcal{Q}^\top)$.

The measurements $\mathcal{D}$ of the model (predicted) state $\mathcal{X}$ the vector given by the equation

$$\mathcal{D} = \mathcal{H}(\mathcal{X}) + \mathcal{E}, \tag{2.4}$$

where $\mathcal{H} : \mathbb{R}^n \mapsto \mathbb{R}^m$ is the measurement operator, which maps the model state vector $\mathcal{X}$ into the measurement space, while the matrix $\mathcal{E}$ contains the measurement errors. A measurement error could consist of instrumental errors or a representation error, which is the error that includes the differences in the representation of the reality between measurements and models, the use of the wrong measurement operator or some errors from the measurement preprocessing.

Noticing that the state vector $\mathcal{Z}$ contains the model state $\mathcal{X}$, we can rewrite Eqn. (2.4) for each of the state vector formulation:

$$\mathcal{D} = \mathcal{G}(\mathcal{Z}) + \mathcal{E}, \tag{2.5}$$

for the model-state formulation, where $\mathcal{G}$ is a map from $\mathcal{Z}$ to the measurements, or

$$\mathcal{D} = \mathcal{H}(\mathcal{M}(\mathcal{Z})) + \mathcal{E}, \tag{2.6}$$

for the forcing formulation, where $\mathcal{M}$ is a map from $\mathcal{Z}$ to $\mathcal{X}$, called model operator, which corresponds to the model m in Eqn. (2.1) in the single assimilation window case.

With all the elements of data assimilation, a generic definition of the objective of data assimilation can be defined: given a sequence of observations $(y_t)_t \in \mathcal{D}$ of the true state $(\hat{x}_t)_t$ the objective of data assimilation is to determine the form of true state $\bar{x}$ (or a good approximation x, the model state). In particular, we want to make prediction on the state of the system: supposing we are at time t^* and the observations $(y_t)_{t \leq t^*}$ are known, our goal is to determine the observation forecasts $(y_t^f)_{t \geq t^*}$ of the predicted state with relative uncertainties.

2.2 Bayesian inference

The data assimilation problem can be formulated as a Bayesian inference problem. Let's denote with f a generic probability density function (PDF) and the function $f(\mathcal{Z})$ is known as the prior probability density function, of the quantity of interest $\mathcal{Z}$. In data assimilation, the observations (or measurements) are considered random variables, but in the moment we take them from the unknown true states of the system, they become determined. In the Bayesian framework, the likelihood $f(\mathcal{D}|\mathcal{Z})$ gives the probability of achieving the observation given the state vector Z. The observations, although they always contain errors, are not random variables. This means that the likelihood is not the PDF of the observation but rather a function of the states with the fixed observations. The randomness is given by the error $\mathcal{E}$ in Eqn. (2.4), so we can directly link the PDF of the measurement error with the likelihood:

$$f(\mathcal{D}|\mathcal{Z}) = f(\mathcal{D} - \mathcal{G}(\mathcal{Z})) = f(\mathcal{E}),$$

meaning that the function $f(\mathcal{E})$ has to be given to compute the likelihood. With the elements we have just introduced, a general form of the data assimilation problem can be formulated as a Bayesian problem: given the definition of conditioning:

$$f(\mathcal{Z}, \mathcal{D}) = f(\mathcal{Z}|\mathcal{D})f(\mathcal{D}) = f(\mathcal{D}|\mathcal{Z})f(\mathcal{Z}), \tag{2.7}$$

where $f(\mathcal{Z}, \mathcal{D})$ is the joint distribution of the random variables $\mathcal{Z}$ and $\mathcal{D}$, it leads to

$$f(\mathcal{Z}|\mathcal{D}) = \frac{f(\mathcal{D}|\mathcal{Z})f(\mathcal{Z})}{f(\mathcal{D})}, \tag{2.8}$$

known as the Bayes' Theorem. Therefore, the data assimilation problem is to find the posterior distribution of the state vector $\mathcal{Z}$ given the measurements $\mathcal{D}$. It is worth noting that Eqn. (2.8) is a point-wise equation. The denominator of the right side of the Eqn. (2.8) is the marginal distribution of the measurements and it acts as a normalization factor,

$$f(\mathcal{D}) = \int f(\mathcal{D}, \mathcal{Z}) d\mathcal{Z} = \int f(\mathcal{D}|\mathcal{Z}) f(\mathcal{Z}) d\mathcal{Z}, \tag{2.9}$$

where the normalization constant is called model evidence. The evidence is generally intractable, so methods such as variational inference and Monte Carlo Markov Chains are used to approximate it. However, [27] states that it is not necessary to always calculate its value, but only when it is needed to compare different numerical models using the same set of measurements. Note that Eqn. (2.8) refers to a forward problem and not an inverse problem, but it is known that data assimilation problems could be forward as well as inverse. Van Leeuwen et al. [70] introduce the inverse problem as a subset of the Bayesian inference.

2.3 Recursive Bayesian formulation

Since in most of the applications the observations become available sequentially, it is necessary to modify the previous formulation in more suitable way. Therefore, the Eqn. (2.8) can be adapted to two scenarios, Markov model and independent measurements.

A first-order Markov model represents a chain of stochastic events, in which the probability of each event transition depends only on the state reached by the previous event. So, there is no "memory" beyond the previous event. The chain of successive events is called a Markov process. Assume that the dynamical model is a first-order Markov process, given a sequence of state $f(z_t)_{t=0,...,l}$ over the l assimilation windows, the Markov property can be written as:

$$f(z_l|z_{l-1}, z_{l-2}, ..., z_0) = f(z_l|z_{l-1}). \tag{2.10}$$

Using this Eqn. (2.10), $f(\mathcal{Z})$ can be rewritten in a recursive way over the assimilation window $l \in (1, ..., L)$,

$$\begin{aligned} f(\mathcal{Z}) &= f(z_0)f(z_1|z_0)f(z_2|z_1)...f(z_L|z_{L-1}) \\ &= f(z_0)\prod_{l=1}^{L} f(z_l|z_{l-1}). \end{aligned} \tag{2.11}$$

The Markov assumption allows to recursively define model prior $f(\mathcal{Z})$, but it affects the evolution in time of the model-state.

Alternatively, let's assume that the measurements are independent between two different time assimilation windows. This means that the measurement errors have zero correlation coefficients (uncorrelated) in different time windows. With the assumption, the likelihood for the measurements vector $\mathcal{D}$ can be rewritten as a product of independent likelihoods, one for each assimilation window,

$$f(\mathcal{D}|\mathcal{Z}) = \prod_{l=1}^{L} f(d_l|z_l). \tag{2.12}$$

The measurements independence assumption is a strong assumption which is not commonly applied, however, it can be limited by assuming correlation only between the observations within the same assimilation window.

Combine Eqn. (2.11) and Eqn. (2.12) into the Bayes' formula Eqn. (2.8),

$$f(\mathcal{Z}|\mathcal{D}) \propto \prod_{l=1}^{L} f(d_l|z_l)f(z_l|z_{l-1})f(z_0). \tag{2.13}$$

By rearranging the order of multiplications, the recursive formulation can be obtained [26]:

$$f(z_1, z_0|d_1) = \frac{f(d_1|z_1)f(z_1|z_0)f(z_0)}{f(d_1)}, \tag{2.14a}$$

$$f(z_2, z_1, z_0|d_2) = \frac{f(d_2|z_2)f(z_2|z_1)f(z_1, z_0|d_1)}{f(d_2)}, \tag{2.14b}$$

$$\vdots$$

$$f(\mathcal{Z}|\mathcal{D}) = \frac{f(d_L|z_L)f(z_L|z_{L-1})f(z_{L-1}, ..., z_0|d_{L-1}, ..., d_1)}{f(d_L)}. \tag{2.14c}$$

A further assumption to simplify the recursive formula in Eqn. (2.14b) can be made by using the Markovian property of the model. By integrating out the model states of the previous assimilation windows, a new formulation of Eqn. (2.14b) is shown as follows,

$$f(z_1|d_1) = \frac{f(d_1|z_1)\int f(z_1|z_0)f(z_0)dz_0}{f(d_1)} = \frac{f(d_1|z_1)f(z_1)}{f(d_1)}, \tag{2.15a}$$

$$f(z_2|d_1, d_2) = \frac{f(d_2|z_2)\int f(z_2|z_1)f(z_1|d_1)dz_1}{f(d_2)} = \frac{f(d_2|z_2)f(z_2|d_1)}{f(d_2)}, \tag{2.15b}$$

$$\vdots$$

$$f(z_L|\mathcal{D}) = \frac{f(d_L|z_L)\int f(z_L|z_{L-1})f(z_{L-1}|d_{L-1},...,d_1)dz_{L-1}}{f(d_L)}$$
$$= \frac{f(d_L|z_L)f(z_L|d_{L-1})}{f(d_L)}. \tag{2.15c}$$

Since this is not the exact solution of the Bayes theorem Eqn. (2.8), but an approximation, it leads to the filtering assumption: the full smoother solution of Bayes Theorem is approximated with a sequential data-assimilation solution. The solution is updated only in the current assimilation window, so the previous one is not updated using the new measurements' information.

For example, at time l, all the information from the previous observations will be contained in z_l, including the one for the current lth assimilation window, making it a good starting point for the following predictions of $z_{l+1}, ..., z_L$.

Another appealing feature of this approximation is the similarity between each step. The rough idea is that, at each step, the state vector is available for the current time window with the information from the past observations it is combined with the new observations at the same assimilation window.

By denoting the model prediction as z_l and the observations d_l at time window l, the updating formula can be rewritten in a more general way:

$$f(z_l|d_l) = \frac{f(d_l|z_l)f(z_l)}{f(d_l)}, \tag{2.16}$$

or without assimilation window:

$$f(z|d) = \frac{f(d|z)f(z)}{f(d)}, \tag{2.17}$$

which resembles the Bayes Theorem Eqn. (2.8) but it refers only to a subset of the state vector and measurements, i.e. the current assimilation window.

2.4 Problem formulation

Until now, the state vector z had a generic definition, but depending on the type of the problem, there could be a different definition for the state vector z. In this section different formulations will be presented: the general smoother formulation 2.4.1 computes the solution over an assimilation window including the model error; the smoother formulation for perfect models 2.4.2 assume that the model is perfect, so there is no model error; the filter formulation 2.4.4 updates the model state at the end of each assimilation window, using Bayesian principles to predict observations based on the state, which leads to a sequential data assimilation process; the recursive smoother formulation 2.4.5 is similar to the filter formulation, but it updates the entire state vector at the end of the assimilation window.

2.4.1 General smoother formulation

Consider the state vector:

$$z = x = m(x_0, q), \tag{2.18}$$

as the model solution over the whole assimilation window with distributed measurements. Note that in this definition, model error q is included. The updating equation for the model state x, which in this case gives directly the observation predictions y^f, can be written as:

$$y^f = g(z) = h(x) = h(m(x_0, q)), \tag{2.19}$$

where, if considered the equality Eqn. (2.18), it can be noticed that the function g is just the measurement operator h that maps the model predictions to the observation predictions. Finally, the observation predictions are compared with the real observations.

Fig. 2.1 demonstrates how the update formula works in an ensemble setting. It starts with computing the prior ensemble integration over the assimilation window (blue lines). By combining the prior ensemble with the likelihood given by the observations (black dots), the updated ensemble estimate (green lines) is obtained updating at once over time and space. It is worth noting that the updated ensemble estimates are closer to the observations and have reduced uncertainties.

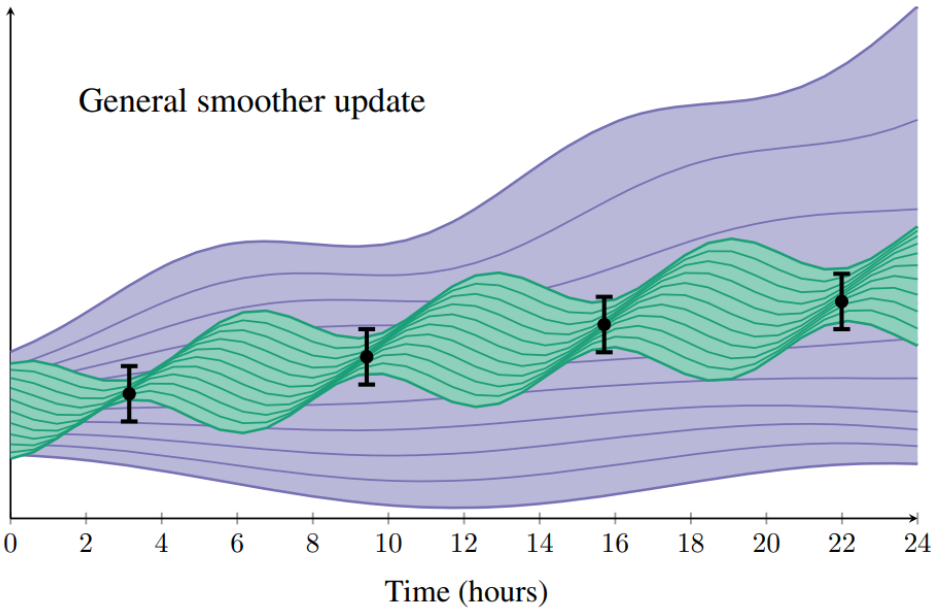


Figure 2.1: This is the visualization of a general ensemble smoother. The black dots are the observations with the relative error given with a standard deviation of 1. The blue lines are the full ensemble integration over the assimilation window. The green lines are the updated ensemble.

2.4.2 Smoother formulation for perfect models

Consider the state vector as

$$z = x_0. \tag{2.20}$$

It indicates that there are no errors and no uncertain parameters and controls are considered. This is a common setting in data assimilation problems. After the estimation of the initial model state for the assimilation window, the state is updated as follows:

$$y^f = g(z) = h(m(x_0)). \tag{2.21}$$

As shown in Fig. 2.2, it starts by integrating the model $x = m(x_0)$ to obtain the prediction. Successively, the measurement functional h is applied to the prediction to get the observation prediction and it is compared with the actual observation through the likelihood. Finally, this distance between the predicted and real observations is propagated backwards at the beginning of the assimilation window to update the model. This formulation is the basis for the *strong-constraint 4DVar* methods, which will be introduced later in Sec. 2.6.3, and in the iterative ensemble smoother.

2.4.3 Smoother formulation for parameters estimation

Similar to the smoother formulation with perfect models, let's assume here to have a model with no uncertain controls and model errors. Consider the state vector:

$$z = \theta, \tag{2.22}$$

namely, it contains only the uncertain model parameters. Using the same update as in 2.4.2, let's write:

$$y^f = g(z) = h(m(\theta)), \tag{2.23}$$

where the model m is supposed to be a forward one in order to assess the connection between the parameters and the observations.

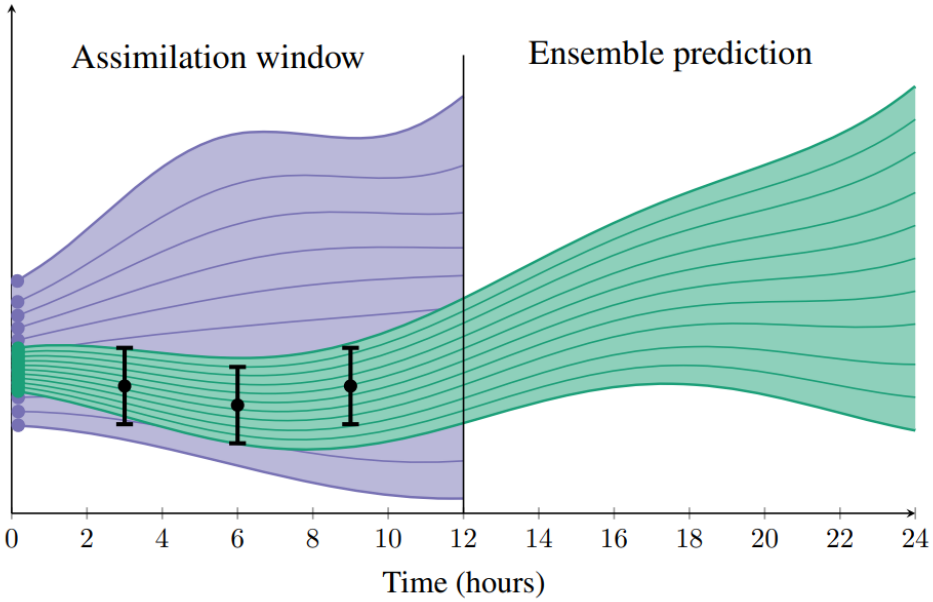


Figure 2.2: This is the visual representation of a recursive smoother formulation assuming a perfect model for ensemble models. After we define the assimilation window, we update the initial condition at the beginning of it and then we integrate the state (green lines).

2.4.4 Filter formulation

Consider the state vector,

$$z = x_K, \tag{2.24}$$

as the model solution at the end of the assimilation window, where it is assumed to have the measurements. What is called *filter solution* (usually solved by particle filters or Kalman filters), is the observation prediction y^f that is related to the state x^K as follows:

$$y^f = g(z) = h(x_K). \tag{2.25}$$

In this case, the operator g corresponds to the operator h that maps the model solution at the end of the assimilation window x_K to the observation predictions y^f. Fig.2.3 demonstrates how the filter works. The solution (observation predictions) is updated at the end of the assimilation window through Bayes' formula before integrating for the next time window. This means that at the end of the assimilation window, there is the marginal distribution of the model state where the information of the previous time steps has been integrated out.

The filter formulation has certain advantages compared to the general smoother one (2.4.1). Firstly, it divides the assimilation window with its observations into multiple shorter time intervals, each of them ending with an observation. It could be more accurate than the smoother formulation in the later assimilation windows. Secondly, in highly nonlinear problems, it can avoid getting "strong" non-Gaussian prior distributions, which are more likely to develop in the smoother formulation. Furthermore, the filter formulation leads to a sequential data assimilation problem with more frequent updates and keeps the model predictions closer to the real observations. Finally, the prior distribution at each observation time has a smaller standard deviation and it is more similar to a Gaussian distribution (compared to the smoother formulation), therefore the updating sequence results are more accurate.

2.4.5 Recursive smoother formulation

Consider the state vector:

$$z^\top = (x_0^\top, ..., x_{l-1}^\top, x_l^\top), \tag{2.26}$$

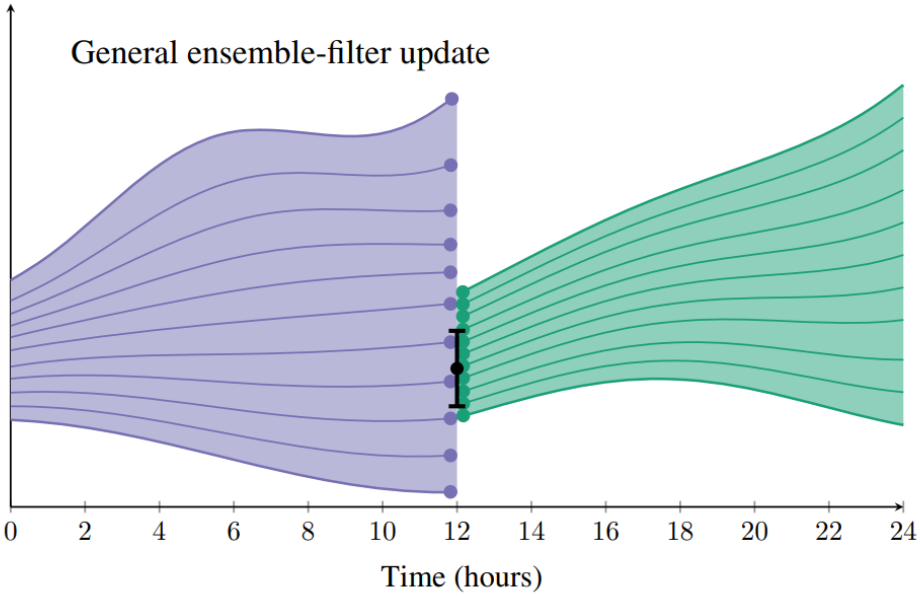


Figure 2.3: The figure visualizes a general ensemble-filter update of an ensemble prediction. The filter updates the ensemble prediction at the end of the previous assimilation window (blue lines) before continuing into the new one (green lines)

at assimilation window l, which contains all the previous model states until (and including) the one at time l. Given the observation at the end of the assimilation window (like in the filter formulation), then we would find the solution y^f given by:

$$y = g(z) = h(x_K), \tag{2.27}$$

but, differently from the filter approach, the entire vector z is updated. This formulation was introduced by Evensen and Van Leeuwen [26]. The idea is to solve the recursive formula of Eqn.2.14b instead of solving the marginal in Eqn. (2.15b). As shown in the filter formulation, this approach introduces the new measurements sequentially in the new assimilation windows and then it propagates them backwards to the past time windows. Consequently, this method inherits all the properties of the filter formulation mentioned before in 2.4.4.

Looking at Fig. 2.4, it can be noticed that the final prediction is the same as the filter formulation, the difference is only that the previous predictions are updated as well, making this approach suitable for hindcast problems [27].

2.5 Data assimilation: Gaussian assumption

In data assimilation problems, the setting is often simplified assuming that the prior is Gaussian. More accurately, it is assumed that both the state vector's elements z and observation errors ϵ are Gaussian.

Let's assume that the prior distribution of the state vector's components z and the observation errors ϵ are both Gaussian, that is:

$$f(z) = \mathcal{N}(z^b, C_{zz}), \tag{2.28}$$
$$f(y|g(z)) = f(\epsilon) = \mathcal{N}(0, C_{yy}), \tag{2.29}$$

where z^b is the background (or first guess) of the state vector and C_{zz} is its error covariance,

[h]

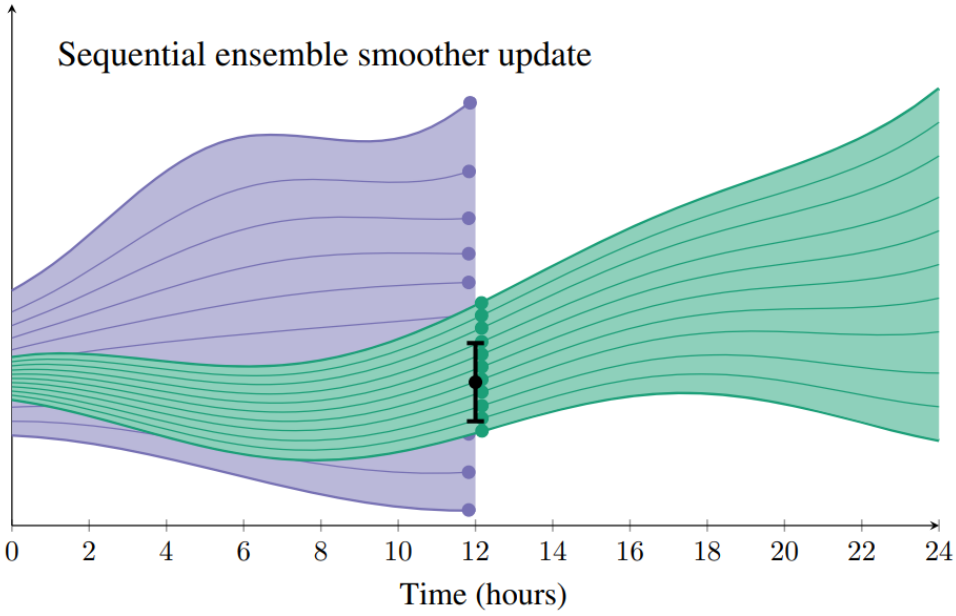


Figure 2.4: This is the illustration of a recursive ensemble smoother. Like in the filter approach, we update the ensemble (blue lines) at the end of the assimilation window, but we also propagate the new information backwards at the previous times (green lines).

containing all the covariances of the variables of the state vector z:

$$\mathbf{C}_{zz} = \begin{pmatrix} C_{x_0x_0} & C_{x_0\theta} & C_{x_0u} & C_{x_0q} \\ C_{\theta x_0} & C_{\theta\theta} & C_{\theta u} & C_{\theta q} \\ C_{ux_0} & C_{u\theta} & C_{uu} & C_{uq} \\ C_{qx_0} & C_{q\theta} & C_{qu} & C_{qq} \end{pmatrix}. \tag{2.30}$$

In this case, the vector z contains the model state at time 0 x_0 and the errors at the other times, known as forcing formulation. If it is reformulated to make z containing the model solution, then there won't be any model error in z. It is worth noting that for simplicity, often only the elements on the diagonal (the variances) are computed, assuming that the elements of vector z are independent.

Given the Gaussian prior in Eqn. 2.28, we obtain the posterior distribution as follows:

$$f(z|y) \propto exp\{-\mathcal{J}(z)\}, \tag{2.31}$$

where $\mathcal{J}(z)$ is called cost function and its explicit formula is:

$$\mathcal{J}(z) = \frac{1}{2}(z - z^b)^T C_{zz}^{-1}(z - z^b) + \frac{1}{2}(g(z) - y)^T C_{yy}^{-1}(g(z) - y), \tag{2.32}$$

where it is reminded that the function g is the (generally) nonlinear map from the state vector z to the measurements predictions y^f. Minimizing the cost function $\mathcal{J}(z)$ is equivalent to estimating the *Maximum A Posteriori (MAP)* of the posterior pdf in the Eqn. (2.31). To find the MAP solution, the gradient of the cost function can be computed and forced to be zero, which leads to:

$$\nabla_z \mathcal{J}(z) = C_{zz}^{-1}(z - z^b) + \nabla_z g(z) C_{yy}^{-1}(g(z) - y) = 0. \tag{2.33}$$

This is the *implicit closed form* solution of the minimization problem for the cost function $\mathcal{J}(z)$.

2.6 Data assimilation methods

In this section, it will be introduced and discussed the setting of some of the well-known data assimilation algorithms including particle filter, (ensemble) Kalman filter, 3DVar and 4DVar. Those

methods can be classified according to their common features: 1) recursive filtering methods, e.g. Kalman or particle filters; 2) methods that only approximate the mean of the PDF using a pre-fixed covariance, e.g. optimal interpolation; 3) methods that try to estimate the mean of the PDF through the minimization of a cost function by exploiting optimization algorithms, e.g.3DVar or 4DVar. More recently, those methods have been extended to an ensemble [25, 35, 57], which combine different data assimilation methods to obtain a better approximation, or more general methods to handle nonlinear problems.

2.6.1 Optimal interpolation

Optimal interpolation is one of the methods that solve the so-called gridding problem [10]. It consists of determining the field $\phi(r)$ on a regular grid of position r from random observations. r is a vector that is two-dimensional (horizontal plane), three-dimensional (longitude, latitude and depth) or four-dimensional (3D spatial + time).

In real-world applications, for example in oceanography, the observations are often sparse and also their distributions in both space and time are usually not uniform. The optimal interpolation methods aim to solve the gridding problem utilizing "optimally" these few pieces of information. Let's assume that we have a set of observations $\{d_1, ..., d_T\}$. Given this setting, objective analysis techniques, which are all those methods that use only empirical data for the solution, use mathematical formulations to infer the field $\phi(r)$ at the unobserved location based on the observations $\{d_1, ..., d_T\}$ [10]. In other words, objective analysis wants to solve the gridding problem by expressing the field in a mathematical way, usually defining it as a linear combination of the observations:

$$\phi(r) = \phi_b(r) + \sum_{j=1}^{N_d} w_j d_j, \tag{2.34}$$

where w_j are the weights and $\phi_b(r)$ is the background field (or first guess), which is fixed a priori. It can be noticed that this is a gridding problem since the field $\phi(r)$ can be evaluated in any position.

Problem formulation: Assume the state vector $z = x$ contains the information about the field values on the points of the grid, the observations vector y. In Sec. 2.1, the operator $\mathcal{H}$ has been introduced as the measurement operator. Here, let's define:

$$\mathcal{H}(z) = \mathcal{H}(x) = Hx, \tag{2.35}$$

where H is a matrix that, applied to the state vector z and returns the field interpolated at the locations of the observations Hx. Assume that the true state $\hat{x}$ are available, the goal is to achieve an analysis x^a the closest as possible to the true state, given an initial state (background) x^b and the observation y,

$$x^b = x^t + \eta^b, \tag{2.36}$$
$$y = Hx^t + \epsilon, \tag{2.37}$$

where η^b is the error associated with the first guess and ϵ is the observation error.

Error covariance: Assume that the first guess and the observations are unbiased, meaning that the expected values of the errors are zero,

$$E\left[\eta^b\right] = 0, \tag{2.38}$$
$$E\left[\epsilon\right] = 0, \tag{2.39}$$

meanwhile the error covariance matrices are defined as,

$$E\left[\eta^b(\eta^b)^\top\right] = P^b, \tag{2.40}$$
$$E\left[\epsilon\epsilon^\top\right] = R, \tag{2.41}$$
$$E\left[\eta^b\epsilon^\top\right] = 0, \tag{2.42}$$

where Eqn. (2.42) means that the two errors are assumed to be independent.

Analysis: Optimal Interpolation methods can be defined as finding the best linear unbiased estimator (BLUE) x^a, also called analysis, of the true state x^t, which follows the properties:

1. The estimator x^a is linear in x^b and y
2. The estimator is not biased:
$$E\left[x^a\right] = x^t, \tag{2.43}$$
3. The estimator has the minimal total variance, so no other estimators have a lower error variance than the BLUE estimator.

According to these properties, the only possible unbiased linear combination of x^b and y is expressed as:

$$x^a = x^b + K(y - Hx^b), \tag{2.44}$$

where the K is a matrix called Kalman gain. This matrix is the gridding operator since, when applied to the residual between the observations and the first guess, gives the gridded field. If the estimator error is computed using Eqn. (2.36) and (2.37), it will be obtained,

$$\begin{aligned}
\eta^a &= x^a - x^t \\
&= \cancel{x^t} + \eta^b - \cancel{x^t} + K(\cancel{Hx^t} + \epsilon - \cancel{Hx^t} - H\eta^b) \\
&= \eta^b + K(\epsilon - H\eta^b) \\
&= (I - KH)\eta^b + K\epsilon.
\end{aligned} \tag{2.45}$$

The covariance of the estimator error can be subsequently defined as:

$$P^a(K) = E\left[\eta^a(\eta^a)^\top\right] \tag{2.46}$$

$$= (I - KH)P^b(I - KH)^\top + KRK^\top. \tag{2.47}$$

The total error variance of the analysis is the trace of the matrix P^a:

$$tr(P^a(K)) = tr(P^b) + tr(KHP^bH^TK^\top) - 2tr(P^bH^TK^\top) + tr(KRK^\top). \tag{2.48}$$

Because it is wanted to minimize the total variance, for any small increment of the gain δK, it won't change the total variance in the first order of δK. Thus, it is obtained the following equivalences:

$$\begin{aligned}
0 &= tr(P^a(K + \delta K)) - tr(P^a(K)) \\
&= 2tr(KHP^bH^\top\delta K^\top) - 2tr(P^bH^\top\delta K^\top) + 2tr(KR\delta K^\top) \\
&= 2tr(\left[K(HP^bH^\top + R) - P^bH^\top\right]\delta K^\top).
\end{aligned} \tag{2.49}$$

Because the gain increment δK is arbitrary, the quantity in the square brackets in the last line of Eqn. (2.49):

$$K(HP^bH^\top + R) - P^bH^\top = 0, \tag{2.50}$$

and adjusting this equation, it can obtained an explicit expression of the Kalman gain K:

$$K = P^bH^\top(HP^bH^\top + R)^{-1}, \tag{2.51}$$

Therefore, if Eqn. (2.51) is plugged in Eqn. (2.46), it can be obtained the explicit formula for the error covariance of the BLUE estimator:

$$P^a = P^b - KHP^b \tag{2.52}$$

$$= P^b - P^bH^\top(HP^bH^\top + R)^{-1}HP^b. \tag{2.53}$$

For simplicity, often only the error variance (and not the covariance) of the analysis is calculated, which is given by the elements on the diagonal of P^a.

2.6.2 Kalman filter

Like in Optimal Interpolation, let's suppose that the measurement operator H is linear, i.e. $h(z) = Hz$, so the cost function Eqn. (2.32) becomes:

$$\mathcal{J}(z) = \frac{1}{2}(z - z^b)C_{zz}^{-1}(z - z^b) + \frac{1}{2}(Hz - y)^\top C_{yy}^{-1}(Hz - y). \tag{2.54}$$

By setting the gradient of the cost function equal to zero (that is solving the close form eqn. (2.33)):

$$C_{zz}^{-1}(z^a - z^b) + H^\top C_{yy}^{-1}(Hz^a - y) = 0, \tag{2.55}$$

it is obtained the so-called Kalman filter update equation:

$$z^a = z^b + (C_{zz}^{-1} + H^\top C_{yy}^{-1}H)^{-1}H^\top C_{yy}^{-1}(d - Hz^b). \tag{2.56}$$

This formulation solves the analysis z^a in the state space. Let's consider the following equivalence:

Woodbury Corollaries:

$$\left(\mathbf{C}^{-1} + \mathbf{G}^\top\mathbf{D}^{-1}\mathbf{G}\right)^{-1} = \mathbf{C} - \mathbf{C}\mathbf{G}^\top\left(\mathbf{G}\mathbf{C}\mathbf{G}^\top + \mathbf{D}\right)^{-1}\mathbf{G}\mathbf{C}, \tag{2.57}$$

$$\left(\mathbf{G}^\top\mathbf{D}^{-1}\mathbf{G} + \mathbf{C}^{-1}\right)^{-1}\mathbf{G}^\top\mathbf{D}^{-1} = \mathbf{C}\mathbf{G}^\top\left(\mathbf{G}\mathbf{C}\mathbf{G}^\top + \mathbf{D}\right)^{-1}, \tag{2.58}$$

if Eqn. (2.58) is used in the Kalman filter update, the standard form of it can be obtained:

$$\boxed{z^a = z^b + C_{zz}H^\top(HC_{zz}H^\top + C_{dd})^{-1}(d - Hz^b)}, \tag{2.59}$$

which this time has the solution in the observation space because the matrix inverted is defined in the observation space.

Now, the next step is to find an updated rule for the covariance matrix of the posterior. The Hessian of the cost function (Eqn. (2.54)) is:

$$\nabla_z\nabla_z\mathcal{J}(z) = C_{zz}^{-1} + HC^{-1}H^\top. \tag{2.60}$$

Now, knowing that the posterior is Gaussian, the cost function can be written as:

$$\mathcal{J}(z) = \frac{1}{2}(z - z^a)^\top(C_{zz}^a)(z - z^a) + c, \tag{2.61}$$

where c is a constant term that only depends on the measurements y. Then, the Hessian (second derivative) of this cost function is:

$$\nabla_z\nabla_z\mathcal{J}(z^a) = (C_{zz}^a)^{-1}. \tag{2.62}$$

Setting Eqn.(2.60) equals to Eqn.(2.62), because the two Hessians need to be equal:

$$(C_{zz}^a)^{-1} = C_{zz}^{-1} + HC^{-1}H^\top. \tag{2.63}$$

Using the matrix identity 2.57, it is obtained the update equation for the Kalman filter error-covariance:

$$\boxed{C_{zz}^a = C_{zz} - C_{zz}H^\top(HC_{zz}H^\top + C_{yy})^{-1}HC_{zz}}. \tag{2.64}$$

Now, the standard form of the Kalman filter can be defined in a recursive way:

$$\begin{aligned} z_k &= z^a, \\ C_{zz,k} &= C_{zz^a}, \\ z_{k+1} &= Mz_k, \qquad (2.65) \\ C_{zz,k+1} &= MC_{zz,k}M^\top + C_{qq}, \qquad (2.66) \end{aligned}$$

where starting at time t_k, the model is integrated until the next time t_{k+1}. Then, the model is updated using the new observations, setting first $z^b = z_{k+1}$ and $C_{zz} = C_{zz,k+1}$, and then substituting them in the update equations 2.59 and 2.64. This setting is similar to what is described in 2.3.

If it is assumed linear measurements operator H and Gaussian prior, the iterative solutions given by Eqn. (2.59), (2.64) solve the minimization problem given by (2.31), where $\mathcal{J}(z)$ has the form of (2.54). It can be noticed that, given the linear and Gaussian assumption, equations (2.59), (2.64) gives the posterior pdf $f(z|y)$.

For a better notation, the definition of Kalman Gain introduced in Eqn. (2.51) can be used, so Eqn.(2.59), (2.64) become:

$$z^a = z^b + \mathbf{K}(d - Hz^b) \tag{2.67}$$

$$C_{zz^a} = (I - \mathbf{K}H)C_{zz} \tag{2.68}$$

Notice that Optimal Interpolation is a simplification of the Kalman Filter if time-invariant error statistics are assumed.

It is also worth to mention that there exists another version of Kalman Filter, called extended Kalman Filter, which, indeed, extends the Kalman Filter formulation for nonlinear models and observation operator, but for brevity, it is not presented it here.

2.6.3 4DVar

Let's divide the 4DVar methods in two: strong-constraint 4DVar (SC-4DVar), where a dynamical system with a perfect model is assumed, and weak-constraint 4DVar (WC-4DVar), where a model error in the dynamical system is considered.

Strong-Constraint 4DVar

This formulation was introduced by Sasaki [64].
Let's assume a dynamical system with no model errors and uncertainties:

$$x_0 = x_0^b + x_0^{'}, \tag{2.69}$$

$$\theta = \theta^b + \theta^{'}, \tag{2.70}$$

$$x_{k+1} = m(x_k, \theta), \tag{2.71}$$

however, it is allowed for uncertain models initial conditions and parameters. Moreover, let's consider observation errors:

$$y = h(x) + \epsilon. \tag{2.72}$$

The goal is to estimate the uncertain initial conditions and parameters to obtain a model prediction close to the measurements, still maintaining them close to the first guesses according to their uncertainties. Therefore, the vector state is:

$$\mathbf{z} = \begin{pmatrix} x_0 \\ \theta \end{pmatrix}. \tag{2.73}$$

Now, if it is considered the cost function (2.32), using Eqn. (2.71), $g(z) = h(m(z)) = h(x)$, the cost function can be reformulated to obtain the SC-4DVar cost function:

$$\mathcal{J}(z) = \frac{1}{2}(z - z^b)C_{zz}^{-1}(z - z^b) + \frac{1}{2}(h(x) - y)^\top C_{yy}^{-1}(h(x) - y), \tag{2.74}$$

which is subject to the "perfect model" constraint (this is the definition of strong constraint) from Eqn. (2.71).

In the SC-4DVAr the covariance matrix of the state vector:

$$\mathbf{C}_{zz} = \begin{pmatrix} C_{x_0x_0} & 0 \\ 0 & C_{\theta\theta} \end{pmatrix}, \tag{2.75}$$

is better known as the background error covariance matrix, which characterizes the error covariances of the initial conditions and parameters. It can be pointed out that the 0 entrances indicate that no correlation is assumed between model's initial conditions and parameters.

Lagrangian formulation

The minimization problem of the cost function (2.74) with constrain Eqn. (2.71) can be expressed with a Lagrangian formulation as following:

$$\begin{aligned}\mathcal{L}(x_0,\dots,x_{K+1},\theta,\lambda_1,\dots,\lambda_{K+1}) = &\frac{1}{2}(x_0-x_0^b)^\top \mathbf{C}_{x_0x_0}^{-1}(x_0-x_0^b)\\ &+\frac{1}{2}(\theta-\theta^b)^\top \mathbf{C}_{\theta\theta}^{-1}(\theta-\theta^b)\\ &+\frac{1}{2}(h(x)-y)^\top \mathbf{C}_{yy}^{-1}(h(x)-y)\\ &+\sum_{k=0}^{K}\lambda_{k+1}^\top(x_{k+1}-\mathbf{m}(x_k,\theta)),\end{aligned} \tag{2.76}$$

where the last addend introduces the Lagrangian multipliers for the perfect-model constraint.

This formulation introduces more unknown variables in the equation, but it actually allows better solution method.

Now, let's introduce the notation for the gradients of the observation operator h and the model m:

$$\nabla\mathbf{h} = \nabla_x h(x)\big|_x, \tag{2.77}$$

$$\mathbf{M}_{x,k} = \nabla_{x_k} m(x,\theta)\big|_{x_k,\theta}, \tag{2.78}$$

$$\mathbf{M}_{\theta,k} = \nabla_{\theta} m(x,\theta)\big|_{x_k,\theta}, \tag{2.79}$$

where the notation is simplified from $(x_0,...,x_{K+1},\theta,\lambda_1,...,\lambda_{K+1})$ to (x,θ,λ) and the index k indicates that compute the gradient at different times t_k.

Lets focus for a moment on the matrix $\nabla\mathbf{h}$: it can be defined it as a composition of matrices $\nabla h_k \in \mathbb{R}^{n\times m}$ at time t_k, where m is the number of measurements and n is the size of the state vector

$$\nabla h = (\nabla h_0,...,\nabla h_k,...,\nabla h_{K+1}). \tag{2.80}$$

So, at every time t_k, given a set of observations, it is known how they are related to the model state through the block matrix ∇h_k.

With this construction, the matrices ∇h_k result sparse, where there are null rows when there are no observations. This implies that if the observations set is empty at time t_k, then the block matrix is null, i.e. $\nabla h_k = 0_{m\times n}$.

Euler-Lagrange Equations

Lets compute the gradient of the Lagrangian (2.76) w.r.t. x_k:

$$\nabla_{x_k}\mathcal{L}(x,\theta,\lambda) = \nabla h_k^\top C_{yy}^{-1}(h(x)-y) + \lambda_k - M_{x,k}^\top\lambda_{k+1}. \tag{2.81}$$

The matrix $M_{x,k}^T$ is the adjoint of the linear model: the model $M_{x,k}$ maps a vector from time t_k to t_{k+1}, so its adjoint maps backwards in time a vector from time t_{k+1} to t_k. For this reason, the Eqn. (2.81) is a reverse integration.

The gradient of the Lagrangian w.r.t. x_{K+1} (final time) is:

$$\nabla_{x_{K+1}}\mathcal{L}(x,\theta,\lambda) = \lambda_{K+1}, \tag{2.82}$$

meanwhile w.r.t. x_0 (initial time) is:

$$\nabla_{x_0}\mathcal{L}(x,\theta,\lambda) = C_{zz}^{-1}(x_0-x_0^b) - M_{x,0}^\top\lambda_1 \tag{2.83}$$

$$= C_{zz}^{-1}(x_0-x_0^b) - \lambda_0, \tag{2.84}$$

where λ_0 is just a "pseudo-variable" introduced for an easier notation.

The gradient w.r.t. the parameters θ is:

$$\nabla_\theta \mathcal{L}(x, \theta, \lambda) = C_{\theta\theta}^{-1}(\theta - \theta^b) - \sum_{k=0}^{K} M_{\theta,k}^\top \lambda_{k+1}. \tag{2.85}$$

The gradient w.r.t. λ_k is:

$$\nabla_{\lambda_k} \mathcal{L}(x, \theta, \lambda) = x_{k+1} - m(x_k, \theta). \tag{2.86}$$

Because the aim is to minimize the Lagrangian function (2.76), all the gradients that we just have computed are set equal to zero, obtaining a coupled system of Euler-Lagrange equations:

$$\begin{cases} x_0 & = x_0^b + C_{x_0 x_0}\lambda_0, \\ \theta & = \theta^b + C_{\theta\theta}\sum_{k=0}^{K} M_{\theta,k}^\top \lambda_{k+1}, \\ x_{k+1} & = m(x_k, \theta), \\ \lambda_{K+1} & = 0, \\ \lambda_k & = M_{x,k}^\top \lambda_{k+1} - H_k^\top C_{yy}^{-1}(h(x) - y), \end{cases} \tag{2.87}$$

where the first three equations are a forward model and the last two are a backward model. Solving this system is equivalent to solving a coupled two-point boundary-value problem in time [27].

Algorithmic formulation

Algorithm 1 shows the standard algorithmic form of SC-4DVar, where the gradients from Eqn. (2.83) and (2.85) are used in a Gauss-Newton method to iteratively update the initial conditions of the forward model.

Then, starting with the first guess solution of the model with $\lambda = 0$, it is obtained a solution for x using the forward model given by the first three equations of (2.87). Using the estimated x, solve the backward model for λ. Once λ is computed, the gradients is computed again from Eqn. (2.83) and (2.85) and the steps are repeated.

Algorithm 1 Standard SC-4DVar algorithm with parameter estimation

Require: $x^b \in \mathbb{R}^n$, $y \in \mathbb{R}^m$ ▷ Prior initial conditions and observations
$x_0 = x_0^b$ ▷ Initialization of x_0
$\theta = \theta^b$ ▷ Initialization of θ
repeat ▷ Iteration loop
  for $k = 0 : K$ **do** ▷ Integrate forward model
    $x_{k+1} = m(x_k, \theta)$
  end for
  $\lambda_{K+1} = 0$
  for $k = K : 0$ **do** ▷ Integrate backward adjoint model
    $\lambda_k = M_k^\top \lambda_{k+1} - \nabla h_k^\top \mathbf{C}_{yy}^{-1}(h(x) - d)$
  end for
  $x_0 \leftarrow x_0 - \gamma \mathbf{B}_{x_0} \nabla_{x_0} \mathcal{L}(x, \theta, \lambda)$ ▷ Update x_0 using Eq. (4.15)
  $\theta \leftarrow \theta - \gamma \mathbf{B}_\theta \nabla_\theta \mathcal{L}(x, \theta, \lambda)$ ▷ Update θ using Eq. (4.16)
until convergence

Weak-Constraint 4DVar

The 'weak-constraint" model concept was introduced by Sasaki [63] as an opposing approach to the "strong-constraint" model, and it was first used in data assimilation by Bennet and McIntosh [12]. In the literature, two different formulations have been proposed: the first approach, called forcing formulation, considers the model error as an additional model forcing that we need to estimate, meanwhile the second approach, called representer formulation, considers the model state over the assimilation window as the unknown variable. The second formulation was introduced by Bennett [13] and it results to be easier to solve than the first, but here only the forcing formulation is introduced since, it is the one used for the 4DVarNet algorithm 3.1.

Forcing formulation

Let's assume the model of the form:

$$x_k = m(x_{k-1}, q_k), \tag{2.88}$$

and the state vector of the form:

$$\mathbf{z} = \begin{pmatrix} x_0 \\ q_1 \\ \vdots \\ q_K \end{pmatrix}, \tag{2.89}$$

which contains initial conditions x_0 and the time-dependent model errors $(q_k)_k$.
The cost function is the same as Eqn. 2.32:

$$\mathcal{J}(z) = \frac{1}{2}(z - z^b)^\top C_{zz}^{-1}(z - z^b) + \frac{1}{2}(g(z) - y)^\top C_{yy}^{-1}(g(z) - y), \tag{2.90}$$

but subject to the model constraint 2.88.
The covariance matrix C_{zz} has the form:

$$\mathbf{C}_{zz} = \begin{pmatrix} \mathbf{C}_{x_0x_0} & 0 & \cdots & 0 \\ 0 & \mathbf{C}_{q_1q_1} & \cdots & \mathbf{C}_{q_1q_K} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \mathbf{C}_{q_Kq_1} & \cdots & \mathbf{C}_{q_Kq_K} \end{pmatrix}, \tag{2.91}$$

where it is assumed no correlation between the model errors and the first guess errors.
The operator $g(z)$ is the composed function following the constraint Eqn. (2.88).

State-space formulation

An alternative method can be also given to the forcing formulation just modifying the Eqn. (2.88), introducing an additional error:

$$x_k = m(x_{k-1}) + q_k. \tag{2.92}$$

Now, with this new formulation, let's explicitly substitute q_k in Eqn. (2.90):

$$\mathcal{J}(x) = \frac{1}{2}(x_0 - x_0^b)^\top C_{x_0x_0}^{-1}(x_0 - x_0^b) \tag{2.93}$$

$$+ \frac{1}{2}(h(x) - y)^\top C_{yy}^{-1}(h(x) - y) \tag{2.94}$$

$$+ \frac{1}{2}\sum_{r=1}^{K}\sum_{s=1}^{K}(x_r - m(x_{r-1}))^\top C_{qq}(r,s)(x_s - m(x_{s-1})), \tag{2.95}$$

where the double sum in the last term takes into account the correlation of the model errors in time.

In this formulation, the state vector is defined as:

$$z = x = \begin{pmatrix} x_0 \\ \vdots \\ x_K \end{pmatrix}, \tag{2.96}$$

therefore the model-error covariance matrix becomes:

$$C_{qq} = \begin{pmatrix} C_{q_1q_1} & \cdots & C_{q_1q_K} \\ \vdots & \ddots & \vdots \\ C_{q_Kq_1} & \cdots & C_{q_Kq_K} \end{pmatrix}, \tag{2.97}$$

which,consistently with the double sum in Eqn. (2.93), includes the correlated errors in time.

The cost function (2.93) can be seen as a weighted measure of:

- the background error (first term), that is the distance between the background and state vector. It can be also interpreted as a Tikhonov regularization term,
- the observation error (second term), that is the distance between the observation operator and the true observation.
- the prior error or prior cost (third term), that is the distance between the model assumption and the actual state.

2.6.4 Particle filters

Another important filtering method is the Particle Filter (or Sequential Monte Carlo), which indicates all the Monte Carlo algorithms used to approximate the solution of the filtering problem in the nonlinear systems. In particular, finding the approximated solution means to find the main features (mean, variance) of the posterior distribution. Since this method requires a proper mathematical introduction that is out of the scope of this work, here it will be just provided a general idea of how these approaches work. Now, let's introduce the approximation that allows all these methods. It is possible to approximate a probability density function, called target distribution, by a finite ensemble (or particles) of N model states as:

$$f(x) \approx \sum_{j=1}^{N} \frac{1}{N} \delta(x - x_j), \tag{2.98}$$

where $\delta(\cdot)$ is the Dirac-delta function.
There exists multiple sampling algorithms to sample the posterior pdf, known as Monte Carlo Markov Chains (MCMC), which sequentially generate the new samples only using the previous one by exploiting the Markovian property.

To obtain the samples from the target distribution $f(x)$, the strategy is to sample from intermediate distributions $f_0(x), f_1(x), ..., f_n(x) = f(x)$, starting from a simpler distribution f_0 and choosing the sequence of distributions in a way such that two consecutive distributions are "close" between each other.

Chapter 3

Methods

This Chapter will talk more specifically about the 4DVarNet algorithm 3.1, with its framework and novelties, the benchmarking setups 3.2 and the evaluation metrics used 3.3.

3.1 4DVarNet

In recent years, machine learning methods, especially deep learning techniques have shown excellent performance for the approximation of complex dynamical systems, both in computation time and accuracy. This is significant in data assimilation since in data assimilation, we need to deal with dynamical models, from real-world applications as mentioned in Chapter 1. In this section, the aim is to present and investigate 4DVarNet, an end-to-end Deep Learning framework proposed by Fablet et al. [28] for image reconstruction of SPM dynamics data. 4DVarNet is based on the WC-4DVar formulation 2.6.3. The novelties consist of a neural approximation of the dynamical model and a neural iterative solver used for the optimization problem. It will be firstly introduced the mathematical framework of 4DVarNet 3.1.1, then the novelties of 4DVarNet 3.1.2 and the resulting End-to-End framework 3.1.3 will be discussed. Finally, a new variation of 4DVarNet, called double LSTM 3.1.4, will be introduced and compared with the original version.

3.1.1 Problem statement of WC-4DVar

Let's consider the following continuous state-space formulation:

$$\begin{cases} \frac{\partial x(t)}{\partial t} = m(x(t)) + \eta(t) \\ y(t,p) = x(t,p) + \epsilon(t) \quad \forall t \in \{t_0, t_0 + \Delta t, ..., t_0 + N\Delta t\}, \forall p \in \Omega_{t_i} \end{cases}, \tag{3.1}$$

where x is the time-dependent state in the state space $\mathcal{X}$, m is the dynamical model and $\{t_0, t_0 + \Delta t, ..., t_0 + N\Delta t\}$ are the assimilation windows. Ω_{t_i} refers to a spatial domain of spatio-temporal dynamics or a list of indices for multivariate time processes. η and ϵ represent model errors and observation errors respectively, which are assumed to be Gaussian, e.g. $\eta, \epsilon \sim \mathcal{N}(0, 1)$.

As mentioned in Sec. 2.6.3, in the weak-constraint setup, the aim is to minimize the cost function Eqn. (2.93) in the state-space formulation, but they are made two assumptions to simplify the problem statement: 1) the background error was omitted for simplicity, 2) the covariance matrices C_{yy}^{-1} and C_{qq} are assumed being spherical. Remember that the model is assumed to be imperfect. Therefore a new cost function can be obtained, namely variational cost [28]:

$$U_\Phi(x, y, \Omega) = \lambda_1 \sum_i \|x(t_i) - y(t_i)\|^2_{\Omega_{t_i}} + \lambda_2 \sum_i \|x(t_i) - \Phi(x)(t_i)\|^2, \tag{3.2}$$

where $\Omega = \cup_i \Omega_{t_i}$ and $\|\cdot\|^2_\Omega$ is the evaluation of the squared norm with respect to a subdomain Ω. It is supposed that the series of observation $\{y(t_i)\}$ is available. Φ is the flow operator with the form:

$$\Phi(x)(t) = x(t - \Delta t) + \int_{t-\Delta t}^{t} m(x(u))du. \tag{3.3}$$

Eqn. (3.2) can be rewritten in a more compact form by dropping the time variable and assume that the norms are evaluated over some predefined time windows, typically from t_0 to $N\Delta t + t_0$,

$$U_\Phi(x, y, \Omega) = \lambda_1 \|x - y\|_\Omega^2 + \lambda_2 \|x - \Phi(x)\|^2. \tag{3.4}$$

In a continuous-time formulation, x and y refer to continuous-time processes, while in a discrete-time case, they refer to a concatenation of the states x_{t_i} and y_{t_i} from $N\Delta t + t_0$. Similarly, $\Omega = \{\Omega_{t_0}, ..., \Omega_{t_0+N\Delta t}\}$ accounts for the observation patterns with possible gaps over the considered time window.

A common method for the minimization problem of the variational cost is exploiting an iterative gradient-based scheme given an initial estimate $x^{(0)}$. The most straightforward scheme is the fixed-step gradient descent,

$$x^{(k+1)} = x^{(k)} - \alpha \nabla_x U_\Phi(x^{(k)}, y, \Omega) \tag{3.5}$$

where α is the gradient step coefficient. It could depend on time, i.e. α_t and we presume that the parametrization of Φ is differentiable.

3.1.2 Novelties

4DVarNet proposed the application of neural networks to achieve an end-to-end learning scheme. The neural network is applied two-folded: firstly, a neural (differentiable) formulation of the dynamical model Φ is constructed to compute the variational cost (3.4) and secondly, this parametrization is combined with a gradient-based neural solver that minimizes a given loss function. The important part of this framework is that it can jointly learn the solver for a predefined variational setting and the parametrization of the operator Φ. Note that the neural parametrization of Φ does not involve any physical assumption and it is purely data-driven, giving more generality to the model. Nevertheless, the neural formulation Φ in Eqn. (3.4) can represent the physics information in the 4DVarNet. Finally, the computation of the gradient in Eqn. (3.5) exploits automatic differentiation tools, also referred as backpropagation, to compute the gradient.

Neural network formulation for dynamics operator

Given the prior cost $\|x - \Phi(x)\|^2$ in Eqn. (3.4), it can be seen as a projection error, meaning that it can be approximated through neural network parametrizations. Excluding parametrizations of the identity operator $\Phi(x) = x$, there have been considered CNN architectures of the form of:

$$\Phi(x) = \phi(\psi(x)), \tag{3.6}$$

where operator ψ is a constrained convolutional layer [8], meaning that the element at the central location of the kernels is set to zero. Operator ϕ is CNN consisting of a concatenation of convolution and activation layers. In particular, the convolution kernels all have a size of 1 along time and/or space dimensions to avoid the parametrization of the identity. The operator Φ can be interpreted as a discretized numerical scheme of an ODE/PDE, since $\Phi(x)(t)$ depends only on some time neighborhoods $x(t - \delta), ..., x(t + \delta)$, where δ depends on the kernel width. Therefore, for all t,p, $\Phi(x)(t, p)$ involves only a local neighborhood in time and space.

Alternatively, the CNN can be constructed to represent the operator Φ in order to catch patterns of which at different scales. In this case, a two-scale representation can be shown as follows:

$$\Phi(x) = \mathcal{U}(\Phi_1(\mathcal{D}(x))) + \Phi_2(x) \tag{3.7}$$

where Φ_1, Φ_2 are parametrization Eqn. (3.6) but with different parameters, $\mathcal{U}$ is an upsampling operator, consisting in an average pool layer, and $\mathcal{D}$ is a downsampling operator, consisting in a convolutional transpose layer. This kind of architecture is known as U-Net [19]. Even though both $\Phi_1$andΦ_2 use the same convolution kernel size, they access different scales, since Φ_1 accesses downsampled data with lower frequencies while Φ_2 accesses higher frequencies. This will not avoid scale overlapping, but because they are applied at different scales, it is expected that they work complementary. As mentioned in 3.1.2, the CNN architecture is purely data-driven and does not have any prior knowledge about the physical model involved during learning.

Fablet et al. [29] presented a Residual Neural Network architecture for the reconstruction and forecasting of nonlinear dynamical systems, called Bilinear (Residual) Neural Network. This architecture is based on the 4th-order Runge-Kutta methods for approximating dynamical systems. Consider a dynamical system. the state of which X varies according to the following equation:

$$\frac{\mathrm{d}X(t)}{\mathrm{d}t} = M(X(t), \theta), \tag{3.8}$$

The 4th-order Runge Kutta method consists of a sequence of updates for a predefined integration step $\mathrm{d}t$:

$$X_{t_0+(n+1)\cdot dt} = X_{t_0+n\cdot dt} + \sum_{i=1}^{4} \alpha_i k_i \tag{3.9}$$

where k_i are define as $k_i = M(X_{t_0+\beta_i k_{i-1} dt}, \theta)$, with $k_0 = 0, \alpha_1 = \alpha_4 = \frac{1}{6}, \alpha_2 = \alpha_3 = \frac{2}{6}, \beta_1 = \beta_4 = 1, \beta_2 = \beta_3 = \frac{1}{2}$. Fablet et al. [29] proposed that the Runge-Kutta integration scheme (3.9) can be considered as a recurrent network with four residual neural network blocks, each of them sharing the same operator F, as shown in Fig. 3.1). In this architecture, the coefficients $\{\alpha_i\}_{i=1,\cdots,4}$ and $\{\beta_i\}_{i=1,\cdots,4}$ refer respectively to the relative weights given to the outputs of the four blocks F and to the relative weights given to the output from the $i-1$th block when added to input x_t to feed the ith block. So, the representation of the dynamical system M in (3.8) can be identified as the learning of the parametrization of the block F, meanwhile the other parameters $\{\alpha_i\}_i$ and $\{\beta_i\}_i$ can be either learned from the data or given the values from the 4th-order Runge-Kutta (Eqn. (3.9)).

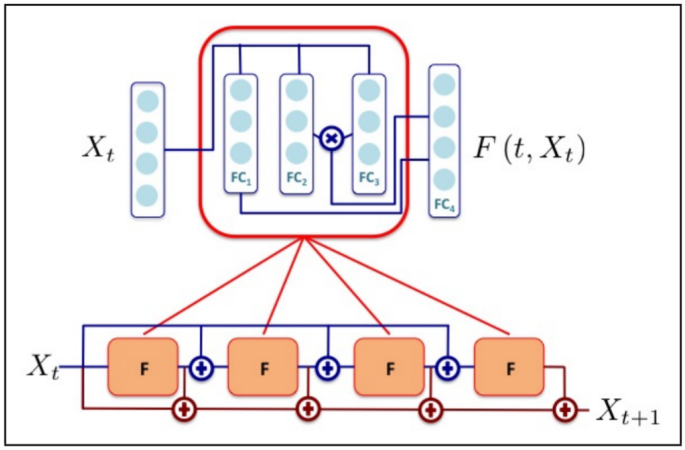


Figure 3.1: Bilinear neural network block ([29]) architecture to approximate a dynamical system as shown in (3.8). Based on the 4th-order Runge-Kutta scheme, it consists of four blocks of F and each of F is a four-layer residual neural network.

At this point, it is of fundamental importance to provide a proper parametrization for the block F. Considering that those dynamical systems involve not only non-linearities, as shown for example in Lorenz-63 [53] and Lorenz-96 dynamics [54], but also bilinear behavior, [29] introduce a bilinear neural network architecture that able to better capture the overall dynamics of the system compared to a vanilla neural network.

This architecture, which is shown in the upper part of Fig.3.1, combines linearly a fully connected layer (FC_1) with bilinear layers consisting in an Hadamard product of two fully connected layers FC_2 and FC_3. The motivation for using this parametrization lies on the fact that often physical dynamical systems present bilinear behavior, for example, multiplicative interactions between two physical variables. Of course, this parametrization can be extended for representing non-linearities, e.g. higher-order polynomial representations. Since [29] demonstrated that the bilinear neural network architecture outperforms the classical 4th-order Runge-Kutta on Lorenz-63 and Lorenz-96 dynamical systems. In this work it will be used in all the experiments without further demonstrations or comparisons.

Neural solver

Given a neural implementation of the operator Φ, the gradient of the variational cost can be computed as shown by Eqn. (3.4). Inspired by the work of Andrychowicz et al. [3] on meta-learning schemes, a recurrent neural network (RNN) structure has been proposed as iterative gradient-based solvers, namely LSTM-based solver, which updates the state at the k^{th} iteration as:

$$\begin{cases} g^{(k+1)} = \text{LSTM}\left[\alpha \cdot \nabla_x U_\Phi(x^{(k)}, y, \Omega), h(k), c(k)\right] \\ x^{(k+1)} = x^{(k)} - \mathcal{L}(g^{(k+1)}) \end{cases} \tag{3.10}$$

where α is a scalar parameter, similar to the step size in a classic gradient descent-based method; $\{h(k), c(k)\}$ are the internal states of the LSTM model and $\mathcal{L}$ is a linear layer to map the LSTM output to the space spanned by state x. The LSTM-based update is a typical parametrization of meta-learning schemes [3]. The updates are used as residual blocks of the residual network (ResNet) [36] with a fixed number of iterations. In Fig. 3.2, the ResNet structure is demonstrated with the light blue rectangle.

3.1.3 End-To-End architecture

Fig. 3.2 is a schematic summary of the learning scheme. The goal of the architecture is to reconstruct the hidden state x, using as inputs an initial state $x^{(0)}$, an observation series y and the associated domain Ω for the missing values. The End-To-End architecture combines the neural approximation of the dynamical model (Sec. 3.1.2) and the neural iterative solver (Sec. 3.1.2), denoted with Γ. Finally, the resulting model is denoted as $\Psi_{\Phi,\Gamma}(x^{(0)}, y, \Omega)$.

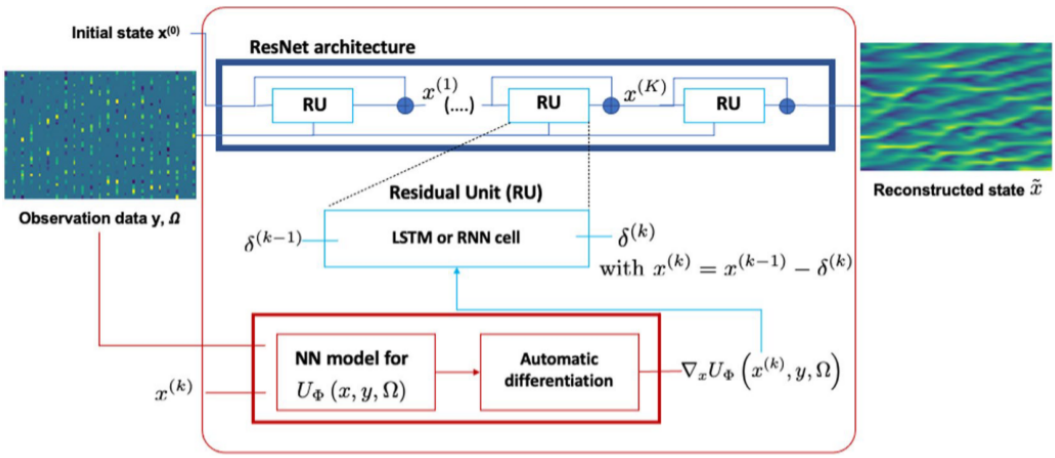


Figure 3.2: End-To-End architecture of 4DVarNet

Learning details

For the learning part of the end-to-end architecture, two setups have been introduced in [28]:

- **Unsupervised Learning** – Assume that a form of Φ is given (or previously learned) and some observation datasets are available, comprising a number of observation series $\{y_1, ..., y_n\}$ with associated missing data mask $\{\Omega_1, ..., \Omega_N\}$. The learning of the NN solver can be viewed as the minimization of the variational cost. The (simplified) formulation of the loss function can be written as:

$$\mathcal{L} = \sum_n U_\Phi(\Psi_{\Phi,\Gamma}(x_n^{(0)}, y_n, \Omega_n), y_n, \Omega_n) \tag{3.11}$$

  where parameters λ_1, λ_2 from Eqn. (3.4) are predefined and the index n indicates the nth sample of the dataset.

This learning function is the standard one used in variational data assimilation settings and it can be considered as unsupervised learning since no ground-truth data have been used for the reconstruction of the state x.

- **Supervised Learning** – Given a dataset composed of observation series $\{y_1, ..., y_N\}$, associated missing data masks $\{\Omega_1, ..., \Omega_N\}$ and true states $\{x_1, ..., x_N\}$, the loss function has the form:

$$\mathcal{L} = \sum_n \|x_n - \Psi_{\Phi,\Gamma}(x_n^{(0)}, y_n, \Omega_n)\|^2, \tag{3.12}$$

Note that the gradient used as input in the iterative solver Eqn. (3.10) is not the gradient of the loss function Eqn. (3.12) but it is the variational cost's one Eqn. (3.4).

3.1.4 Double-LSTM 4DVarNet

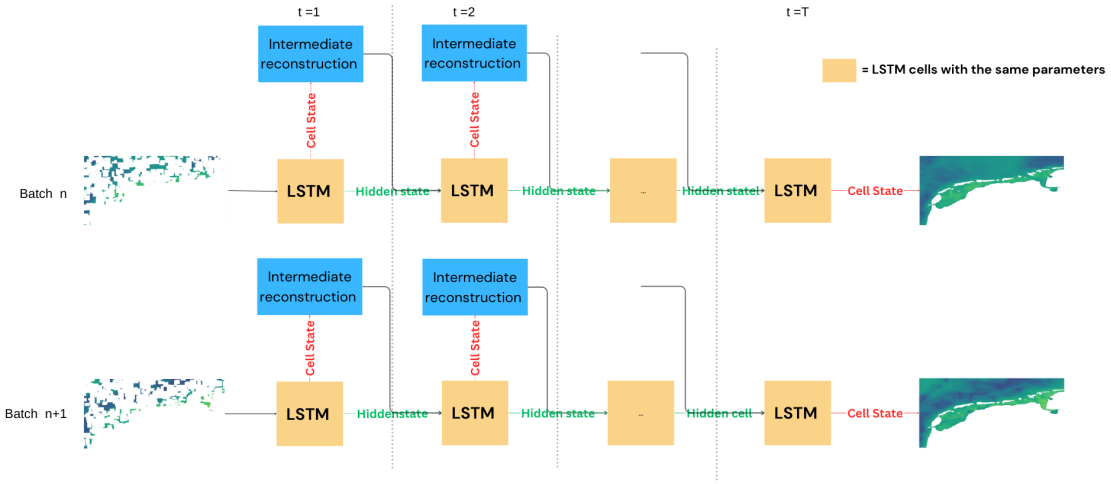


Figure 3.3: This illustration shows the workflow of 4DVarNet. Each batch of observations passes through an LSTM cell for a fixed number of iterations and the final reconstruction is given by the state cell of the last iteration's LSTM cell. Then we continue with the reconstruction of a new batch. We can see that all the LSTM cells share the same set of trainable parameters.

The LSTM iterative solver presented in 3.1.2 shares the parameters over all the iterations, resulting in a smaller number of parameters to train. One problem of the algorithm implementation (will be further discussed in Sec. 5) is its bias towards lower (zeros) values, which substitute the missing values at the beginning of the reconstruction. To overcome this issue, we proposed a new version of 4DVarNet, namely Double-LSTM 4DVarNet, which modifies the original structure from two perspectives: first, we assume that the LSTM cell in the first reconstruction iteration has different parameters from the LSTM cells used in the next iterations. It results in two sets of parameters to optimize instead of one. Second, this new LSTM cell takes as input hidden state the hidden state obtained at the end of the previous batch reconstruction.

Fig. 3.4 shows the workflow of this new algorithm. The core idea is to use the information stored in the cell state of the LSTM cell at the end of the reconstruction iterations and transfer them in the first LSTM cell of the new batch of reconstruction. To ensure that we use correctly this new information, we assume that the set of trainable parameters of the first LSTM cell of each iteration is learned separately from the parameters' set of the other LSTM cells. Therefore, the

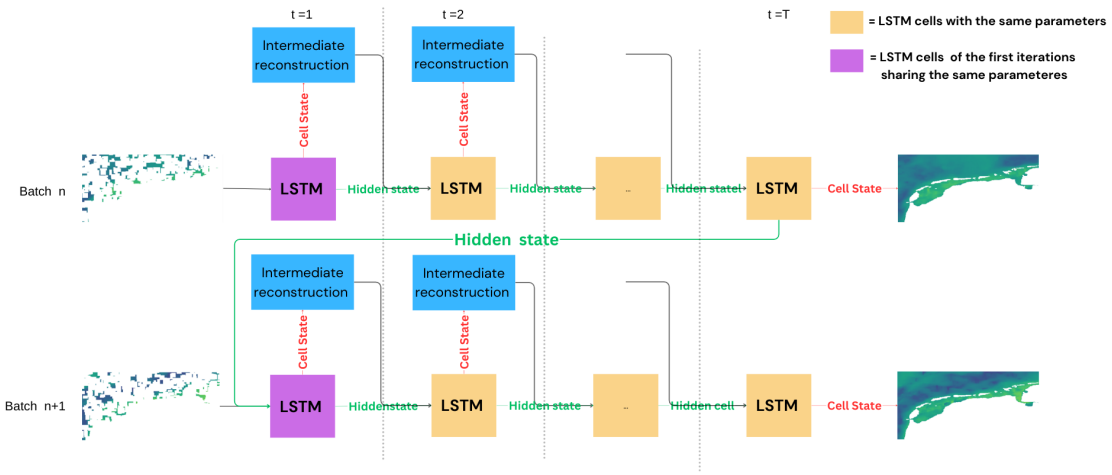


Figure 3.4: The workflow of Double-LSTM 4DVarNet. We can notice that the LSTM cells in the first iteration of each batch reconstruction has different trainable parameters than the next LSTM cells. They take as input the initial observation batch and the hidden state vector from the last iteration's LSTM cell (the green arrow).

resulting architecture has more trainable parameters and will affect the computation time during the training time. However it does not increase the overall complexity of the model, because the reconstruction steps are exactly the same as the original 4DVarNet and the structure of the LSTM cells remains the same.

3.2 Observing system experiment and observing system simulation experiments

Observing system experiment (OSE) and observing system simulation experiments (OSSE) [5] setup have been introduced to assess and evaluate the impact of, respectively, observation and simulated data on numerical models, especially for forecasting and data assimilation methods in weather forecast and oceanography. In this section their concept is introduced and how it is explained how they have been applied in the project, taking inspiration from the previous work of Vient et al. [73, 72].

3.2.1 OSSE

As introduced in Sec. 1, satellite images often suffer from missing data due to cloud coverage, making the data interpolation more challenging for the reconstruction of sediment dynamics. If the rate of missing data is low or moderate (less than 50%), satellite images can still be used for benchmarking different methods. But in scenarios with more than 80% of data missing, OSSE frameworks are more suitable for this task.

In particular, the metrics used for the evaluation are designed to ensure independence from the sampling method used for the observation data, which provides a more general strategy for evaluating the ability of the interpolation methods of spatial and temporal scales reconstruction. The OSSE setting needs a gap-free dataset as the ground-truth state for evaluation. Usually this dataset consists of simulation results generated from numerical models. In order to work with "close-to-reality" data, [73] generated missing data by applying a real cloud mask taken from real satellite images to the gap-free dataset. Another way to generate missing data is to cover the

images with random patches of some shapes. They have been used artificial masks composed by rectangles of different sizes, the same as we do in OSE. Finally, a Gaussian noise is added to simulate the satellite instrumental error.

3.2.2 OSE

In this setting, only satellite images are used (or are available) and because of the missing data, they can not be used as "gap-free" data in the same way as in the OSSE setting. Therefore a different strategy has been introduced. A binary mask is generated by random sampling and applied to the real data. In this way, a dataset with 'artificial' gaps is created and the original satellite data is used as reference data to evaluate the reconstructed states through the metrics. We need to point out that this dataset is not a ground-truth dataset since satellite images are noisy due to instrumental errors. Two different techniques have been proposed. The first consists of a random sampling for 50% of the available pixels, known as "pixel-wise" strategy. The second strategy is called "patch-wise" as the images are masked by patches of size $H \times W$ (H is the height and W is the width), where W and H are randomly sampled from a uniform distribution on a fixed interval. The number of patches is pre-defined as well. According to [72], the "patch-wise" strategy reproduces more realistically the independence between the training and test datasets.

3.3 Evaluation metrics

To evaluate the reconstruction quality of the models in terms of accuracy and visual similarity, three different metrics have been used: root mean square error, absolute error and structural similarity index measure. While the first two measure the errors between the values, the last one is an index measuring the visual similarity between two images.

- Mean square error (MSE) and root mean square error (RMSE): this metric describes quantitatively the discrepancy between the predicted and reference values. Although it is a good metric for evaluating and comparing the error of different models, it lacks interpretability when the values are on a different scale since errors for higher values will be automatically weighted more than errors of smaller values, and without a context, the error value on itself doesn't provide any information whether the model predictions are 'good' or 'bad'. The formula is the following:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}, \tag{3.13}$$

  where n is the number of predictions, y_i are the observed values and $\hat{y}_i$ are the predictions.

- Relative error: this metric is more suitable for comparing the errors of different units or values at different scales.

  The formula is the following:

$$\text{RE} = \frac{|y_i - \hat{y}_i|}{|y_i|} \tag{3.14}$$

  where y_i are the observed values and $\hat{y}_i$ are the predictions. The final error is in absolute error.

- Structural similarity index measure (SSIM): this metric measures the similarity between two images by evaluating the structural information of an image, such as luminance, contrast, and structure. All of these contribute to human perception of image similarity [74].

  The formula is the following:

$$SSIM(x, y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)} \tag{3.15}$$

  where x and y refer to the images, μ_x, μ_y are the pixel sample mean of the images, σ_x^2, σ_y^2 are the variance of the images, σ_{xy} is the covariance of the two images and c_1, c_2 are two variables to stabilize the division.

Chapter 4

Experiments and Results

This chapter is divided as follows: section 4.1 introduces the two types of data used and the preprocessing applied; section 4.2 describes the experiments setups and parameters used; finally, section 4.3 presents the numerical and visual results obtained and analyzes them.

4.1 Data

In this section, satellite data (CMEMS) and model data (DELFT3D) are introduced and described, together with their preprocessing methods. Both datasets present SPM concentration measured in milligrams per liter of water (mg/L).

4.1.1 Satellite data

Figure 4.1: The studying area (black box) for the Dutch Wadden Sea.

The satellite data that have been used in the experiments are the Level 3 (L3) biogeochemical product taken from the Copernicus Marine Environment Monitoring Service (CMEMS) for the North Sea [1]. The observations are obtained by a combination of data from different satellites including SeaWiFS, MODIS, MERIS, VIIRS-SNPP and JPSS1, and OLCI-S3A and S3B. The product provides biogeochemical information such as clorophyll-a concentration, phytoplankton functional types and SPM. In particular, this study focuses on the area (shown in Fig. 4.1) of the Dutch Wadden Sea, of coordinates 4.452W, 7,381E, 53.97N, 52,55S, with an original resolution of 1km, meaning that the dimensions of each image is 132×282 pixels.

[1] More detailed information at https://data.marine.copernicus.eu/product/OCEANCOLOUR_ATL_BGC_L3_MY_009_113/description

The dataset consists of daily observations of SPM from 2000 until now, since it is updated constantly. The dataset has on average more than 75% missing data due to cloud coverage, which makes it a high missing rate dataset and is used for comparing the reconstruction performances of 4DVarNet with DInEOF and eDInEOF. In Fig. 4.2, it is possible to see the rate of data available for the year 2018, with a concentration of more complete images during the spring-summer and more missing data in the winter period, which is due to the fact that there is more cloud coverage during winter than in summer. This seasonality trend appears in the other years as well.

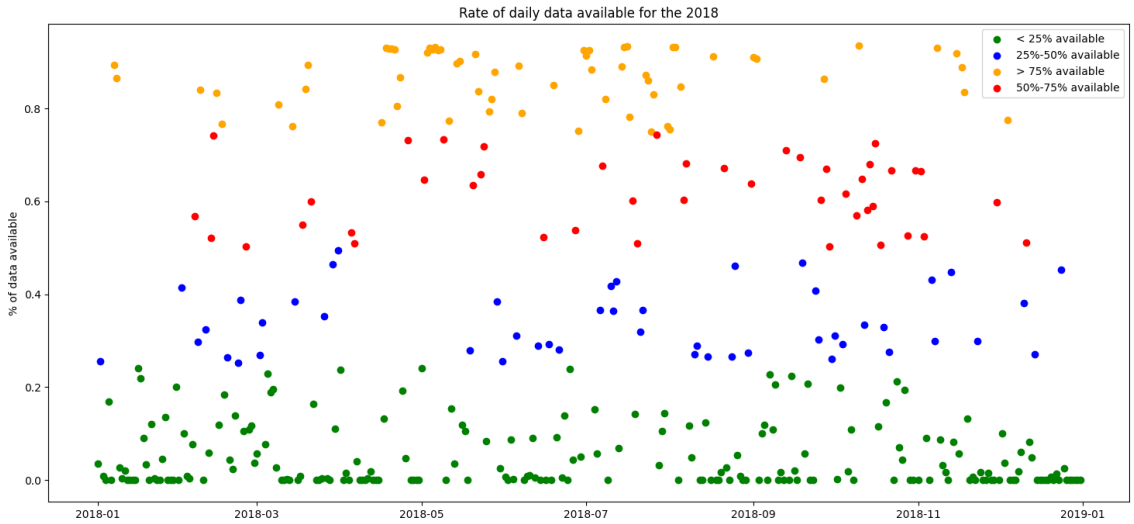


Figure 4.2: Rate of data (pixels) available on the study area for the year 2018.

A second dataset from CMEMS, with a resolution of 100m, was also used to explore the limits of the model and observation data. This higher-resolution dataset provides data on daily observations of SPM from 2020 until now, but it has a very high rate of missing data (around 95% on average).

4.1.2 Model data

Model data are generated by a Dutch Wadden Sea, based on DELFT3D FM[2], numerical solvers developed by Deltares for simulating hydrodynamics and sediment transportation. The dataset, which is the output of a model for mud dynamics, consists of 2 years (2016-2017) of daily SPM values and contains multiple hydrodynamics and sediment concentration variables. The ones that will be used are two different mud fractions: IM1 fraction represents macro flocs meanwhile IM2 represents micro flocs.

The data cover the area between coordinates 53.971N, 52.554S, 4.004W, 7.381E (Fig. 4.4). The horizontal data points are irregularly structured in curvilinear polygons, which are indicated by the faces and nodes (see Fig. 4.3). Each of the horizontal data points has 10 vertical layers, from the surface values to the bottom values, but the layers have irregular distances that vary depending on the bathymetry.

4.1.3 Data preprocessing

For both satellite and model data, preprocessing was applied to make them more suitable as input of 4DVarNet, especially for the training phase.

DELFT3D data were the ones required more preprocessing steps. As already mentioned in 4.1.2, the SPM values were obtained by combining the IM1 and IM2 variables, in particular summing them up. Moreover, the complete data from the DELFT3D-Flexible-Mesh model have 10 layers of values, giving a sort of "depth" information of the location, and the values are organized in flexible meshes, not in regular grids. Therefore, only the top layer was extracted and the data

[2]More details at https://www.deltares.nl/expertise/publicaties/dwsm-a-sixth-generation-3d-model-of-the-dutch-wadden-sea-2022-release-2

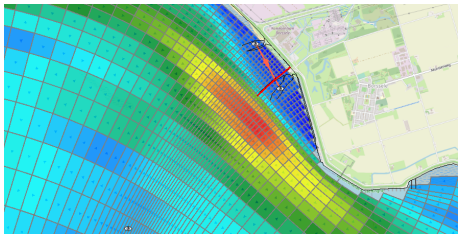


Figure 4.3: Example of how the flexible meshes look like in the DELFT3D model.

was processed from flexible meshes to regular grids using a python library developed by Deltares, called **dfm_tools**[3]. In particular, the function used is "*dfm_tools.get_nc.rasterize_ugrid*", which interpolates the values on a regular grid. The final resolution is approximately 800 meters. However, the model produced non-physical values due to the interpolation, resulting in concentrations above 10,000 and negative values. These are unrealistic as, firstly, one liter of water cannot hold such high SPM levels, and. secondly, SPM concentrations cannot be negative. After an analysis, it was found out that only a few data points over the entire dataset (less than one hundred data points for values over 10000 and one thousand four hundred data points with negative values), so two thresholds have been set to handle them, with the supposition that they wouldn't affect the outcomes. For the high values, the threshold has been set at 10000, meanwhile for negative value threshold was set at 0, since the values are all lower than the 10^{-8} order, meaning that they should have been zeros, but they became negative due to the smoothening process of the interpolation.

Since the original resolution of CMEMS data is of 1km, it was adjusted to the same resolution of model data using a built-in Python function [4] which exploits interpolation for data augmentation.

After these first steps, a logarithm with base 10 (log10) has been applied to both dataset: since log10 is a positive monotone function, the order of the values is maintained, but their distribution will be better distributed in a smaller range of values. In this way, 4DVarNet will learn better the dynamics and it won't be influenced by very high variations of SPM values. But before directly applying the log10, all the values lower than 0.1 have been cut because they would be converted to very high negative number after the log10, introducing more variance in the model. This assumption is possible because the values range from 0 to 100 for the satellite data and more than 1000 for the model data. Therefore, the threshold of 0.1 is low enough to consider it as 0.

Eventually, the final dataset to be used for the training and testing has been obtained, containing data points with matching coordinates and a similar value range. The main difference between the datasets is the values spatial distribution, this will be discussed more in Sec. 5.

The same pre-processing was applied on the 100-meter resolution dataset from CMEMS, which was downsampled to 200-meter resolution.

4.2 Experiments

Two main experiments were conducted using the OSSE setup. The training and validation phases were performed on simulated data (DELFT3D FM data), while the testing phase used model data (CMEMS data), preprocessed as described in Sec. 3.2. In particular, for all the training, validating and testing phases, the daily images have been artificially masked up to 75% to simulate the cloud coverage.

In the first experiment, we trained single LSTM and double LSTM on the one-year dataset (2016) and then tested on 1 year dataset from CMEMS (2018). Table 4.1 shows the hyperparameters and parameters of the two models: the time window was set at 10 days after exploratory experiments (it will be discussed in chapter. 5), the hidden layers dimension for both the autoencoder (AE) and the LSTM are set respectively at 64 and 96, same as the downsampling rate, at 2. The only difference is in the batch size, 4 for the single LSTM and 1 for the double LSTM: the reason for setting at 1 for double LSTM is because in this way the time difference between one

[3]The documentation can be found at https://github.com/Deltares/dfm_tools

[4]The function is xarray.DataArray.interp and its documentation can be found at https://docs.xarray.dev/en/stable/generated/xarray.DataArray.interp.html

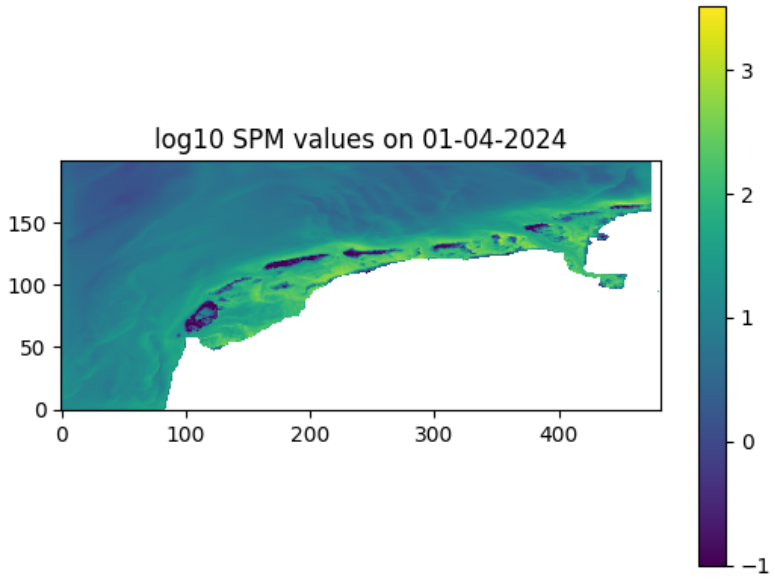


Figure 4.4: Example of a map of log10 SPM values on the from the DELFT3D model.

	Single LSTM	Double LSTM
Number of Channels	1	1
Time Window	10	10
Hidden layers dimension (AE)	64	64
downsampling (AE)	2	2
Batch size	4	1
Trainable parameters (AE)	666177	666177
Training Epochs	50	50
Solver iterations	15	15
Hidden layers dimension (LSTM)	96	96
Trainable parameters (Full)	1033860	1401543
Training time	879m 19s	1194m 15s

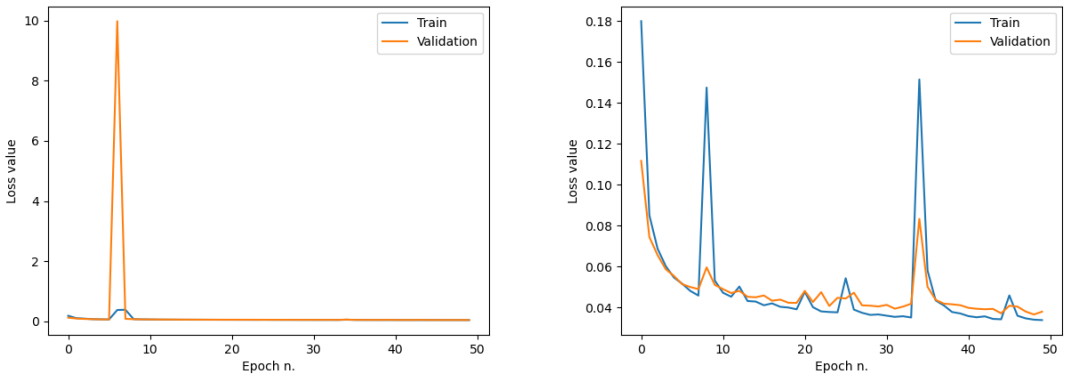
Table 4.1: Comparison of the model parameters and hyperparameters used in the first experiment. Single LSTM refers to the original 4DVarNet architecture.

batch and the following is of 1 day, allowing to use better the information from one batch to the following. This reflects in the total number of trainable parameters of the entire model, which is the number of parameters of the autoencoder plus 2 times the number of parameters of the LSTM structure.

After, the training is done over 50 epochs setting the number of iterations of the optimization solver at 15. The resulting training time is of about 15 hours for the single LSTM and about 20 hours for the double LSTM. As shown in Fig. 4.5, it can be observed that the loss function values of the two models during the training phase: even though they seem to have an unstable convergence due to the peaks, the algorithms adjust their parameters towards the correct values thanks to the meta-learning schemes inside the neural solver. In fact, at the end of each epoch, the value of the loss function is computed and if it is smaller than the previous epoch's one, the new model parameters are saved and used in the next epoch, otherwise they will be kept the old one.

Because of the peaks, especially in Fig 4.5a, the convergence of the loss function is not very clear from the images. For a more complete understanding, the numerical values are available in Appendix A 8.1

The second experiment used the same OSSE setup but on the 200m resolution datasets. The



(a) Loss function of single LSTM during the training phase

(b) Loss function of double LSTM during the training phase

Figure 4.5: The loss function values for the two models training phases. The peaks indicate that during the training, the parameters are updated wrongly, but then the solver adjusts itself with the correct values.

information about the model's parameters and training loss function can be found in Appendix B 8.2. In the discussion section 5, it will be explained why the results are not presented.

All the experiments have been run on a single NVIDIA H100 GPU node, available at Deltares cluster.

4.3 Results

In this section, the numerical results and visualization of image reconstruction are presented. The results refer to the reconstruction of one year (2018) of the CMEMS dataset, comparing the reconstruction abilities of 4DVarNet (single LSTM and double LSTM), DInEOF and eDInEOF. The assessment and comparison have been done on the overall dataset 4.3.1, a single day 4.3.2 and on some sites in the Wadden Sea 4.3.3.

For the one-year dataset, single LSTM takes 1m 18s to reconstruct, while double LSTM takes 1m and 33s, DInEOF takes 9m 36s and eDInEOF takes 23m 35s. While 4DVarNet models have comparable computational time for reconstructing the entire dataset, DInEOF takes more than 4 times and eDInEOF takes even 20 times more.

The results will be presented through metrics, as presented in Sec. 3.3, through visual reconstructions/spatial maps and through time series plots at individual locations.

4.3.1 Metrics

Table 4.2 shows the results of two 4DVarNet experiments, compared to benchmark techniques DInEOF and eDInEOF in three metrics. These are computed for the whole area, only for the Dutch Wadden Sea and only for the remaining area.

In terms of RMSE, the double LSTM shows the best overall performance with the lowest error (2.0472), suggesting it is the most accurate model. In contrast, the single LSTM has a slightly higher RMSE at 2.8089, with notable variation across different regions. Specifically, in the Wadden Sea, the single LSTM has a high RMSE of 4.9876, indicating poorer performance in this area, while its performance is significantly better in other regions, where it achieves an RMSE of 1.9010. DInEOF and eDInEOF, though somewhat comparable, exhibit higher RMSEs (2.282 and 2.814, respectively), with eDInEOF struggling especially in the remaining regions, where it shows an RMSE of 2.4071. Overall, double LSTM is the most accurate across the board, while eDInEOF demonstrates the poorest performance.

Metrics		single LSTM	double LSTM	DInEOF	eDInEOF
RMSE	All	2.8089	2.0472	2.282	2.814
	Wadden Sea	4.9876	3.1805	3.5019	4.0466
	remaining	1.9010	1.6431	1.8530	2.4071
RE	All	27.9055	42.1077	53.8607	79.9936
	Wadden Sea	28.3099	30.4148	29.8751	38.0782
	remaining	27.8036	45.0510	59.8984	90.5448
SSIM		0.9378	0.9167	0.5107	0.5102

Table 4.2: Metrics Comparison on the whole dataset, on only the Dutch Wadden Sea and only on the remaining area (offshore).

When looking at RE, the single LSTM model again shows strong performance with the lowest overall RE (27.9055). It is especially accurate in the offshore regions, where it achieves a low RE of 27.8036. However, double LSTM, despite excelling in RMSE, shows a relatively higher RE (42.1077), with a notable increase in the remaining regions (45.0510). The DInEOF model performs better than double LSTM in the Wadden Sea (with a RE of 29.8751), but its performance drops dramatically in the remaining regions (RE of 59.8984). Finally, eDInEOF records the highest RE across all regions (79.9936 overall, 90.5448 in the remaining regions), indicating it is the least accurate model in terms of relative error.

In terms of SSIM, which measures structural similarity, the single LSTM model performs best with a score of 0.9378, closely followed by the double LSTM at 0.9167. This indicates that both LSTM models maintain a high level of structural integrity in their predictions. In contrast, DInEOF and eDInEOF show significantly lower SSIM values, around 0.510, indicating that these models struggle to preserve structural similarity, which is a crucial factor in many applications.

In summary, the LSTM models, particularly double LSTM, show better accuracy in terms of RMSE, while single LSTM provides a more balanced performance across all metrics, including RE and SSIM. The EOF-based models, especially eDInEOF, generally perform worse across all metrics, with higher errors and lower structural similarity.

4.3.2 Visual reconstruction

Fig. 4.6 visualizes the reconstructed images for the day "22-05-2018" using the different models. On the top left, there is the original image, on the bottom left there is the image with 75% of data removed. Even though the visual maps are not indicative of the reconstruction abilities, it is possible to notice how the DInEOF and eDIneof reconstructions are more "pixelated", compared to the smoother images of single and double LSTM.

Table. 4.3 shows the metrics computed on a single day. The numerical results are similar to the ones shown in Tab. 4.2, where double LSTM has better RMSE than the other models, single LSTM has better RE and both of them have similar SSIM, with single LSTM being slightly better. However, in this case, single LSTM has better RMSE than DInEOF and eDInEOF, but DInEOF and eDInEOF have a similar RE to double LSTM, still performing worse than single LSTM.

Metrics	DInEOF	eDInEOF	single LSTM	double LSTM
RMSE	1.3949	1.3021	1.0939	0.9973
RE	20.2471	19.0484	11.4468	19.8685
SSIM	0.8252	0.8451	0.9564	0.9249

Table 4.3: This table shows evaluation metrics computed on the masked pixels for the reconstruction of the day 22-05-2018

To extend the analysis of the reconstructions, Fig. 4.7 shows the density functions of the values on the observed pixels (left plot) and masked pixels (right plot). For computing the density function, it was used the kernel density estimation method with a kernel width of 0.5, to ensure that the differences in the plotted functions are more visible. It is interesting to evaluate the density

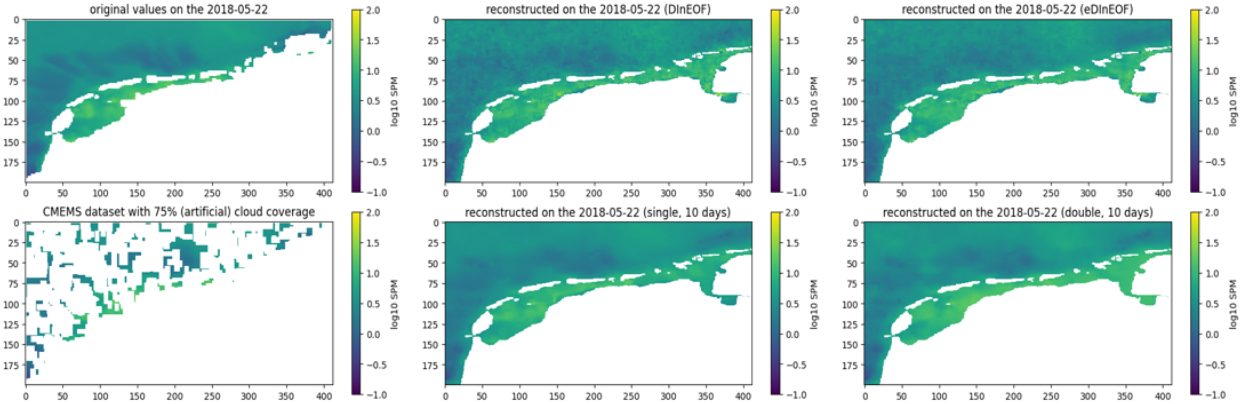


Figure 4.6: Example of image reconstruction for the day 22/05/2018. The original satellite image has few missing data (top left), so it has been masked to get 75% of missing data (bottom left). Then on the top center and right we can see the DInEOF and eDInEOF reconstruction, meanwhile on the bottom center and right the reconstruction for the single and double LSTM.

functions on both because all the models overwrite the values at each iteration of the algorithm, so even the values on the observed pixels can vary.

On the density plots of the observed pixels, all the models have more or less the same distribution as the original values, except for eDInEOF, which has a higher concentration of SPM values between 0.25 and 0.5 compared to the original data, but lower concentration of values in between 0.5 and 0.7 (approximately).

On the density plot of the masked pixels, all the models have lower values concentration between 0.5 and 0.7 compared to the original data, even though single LSTM outperforms the other models, meanwhile, they generally have higher concentration of values between 0.25 and 0.5, and again, single LSTM reproduces these concentrations more accurately, compared to the other models.

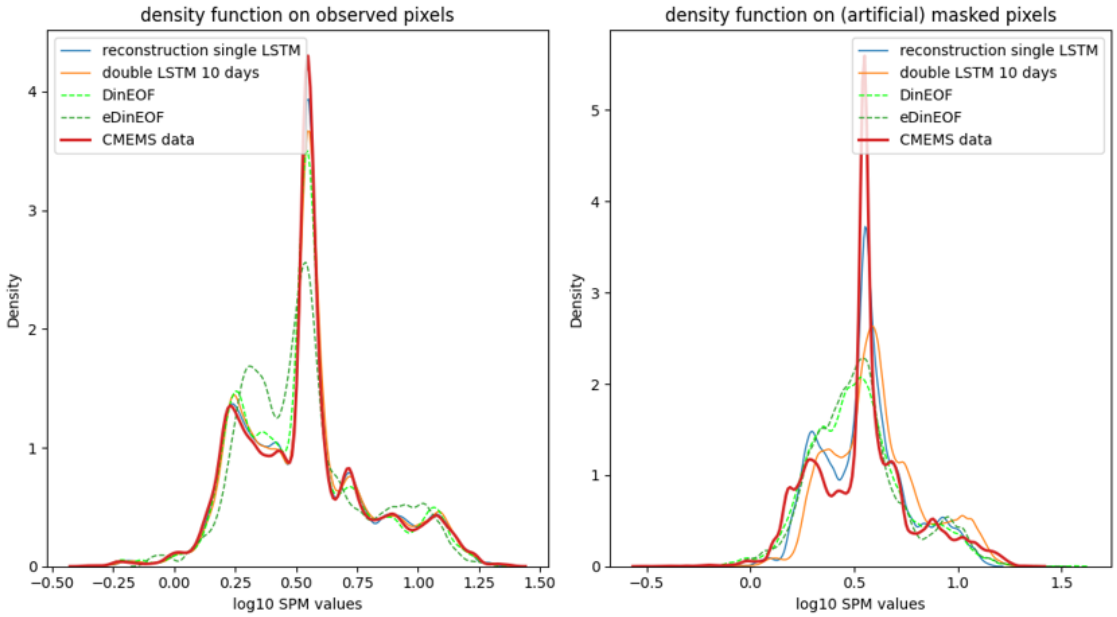


Figure 4.7: Plots of the density functions of the observed pixels and masked pixels. On the left there are the density functions of the observed pixels, while on the right there are the density functions of the masked pixels. The kernel width was set at 0.5.

	DInEOF	eDInEOF	single LSTM	double LSTM
KL divergence on observed pixels	0.002233	0.012773	0.000619	0.000895
KL divergence on masked pixel	0.037256	0.032883	0.019318	0.0179436

Table 4.4: The table shows the Kullback-Leibler divergence between the original values and the reconstructed ones from the different models.

Table 4.4 shows numerically how much different the densities of the reconstructed values of the models are compared to the original ones, using the Kullback-Leibler divergence. KL divergence measures how closely each model's predicted probability distribution matches the true distribution, with lower values indicating better performance.

On observed pixels, the single LSTM model performs the best, achieving the lowest KL divergence of 0.000619. The double LSTM model follows closely with a KL divergence of 0.000895, still showing strong accuracy but slightly less precise than the single LSTM. In comparison, DInEOF and eDInEOF have higher KL divergences on observed pixels, with values of 0.002233 and 0.012773, respectively. This indicates that these models are less accurate in capturing the distribution of the observed data. Meanwhile, when considering masked pixels, the double LSTM model achieves the lowest KL divergence of 0.0179436s. Single LSTM also performs well on masked pixels, with a KL divergence of 0.019318, though not quite as well as the double LSTM. Again, both DInEOF and eDInEOF show significantly higher KL divergence values of 0.037256 and 0.032883, respectively, indicating a reduced ability to generalize to missing data compared to the LSTM-based models. It can be concluded that single LSTM has a more similar density function to the density function of the original data, suggesting that it has better reconstructions on the masked pixels, meanwhile double LSTM has a more similar density function to the original one on the masked pixels. In both scenarios, the EOF-based models perform worse than 4DVarNet.

4.3.3 Time series reconstruction

In Fig. 4.8, two time series plots of two sites in the Dutch Wadden Sea are shown. The purple dots are the values observed and the green dots are the masked values, used to evaluate the reconstruction ability of the models.

Starting with the BOCHTVWTND station (see Fig. 4.8), among the models, the double LSTM (red line) closely follows the observed CMEMS data, demonstrating its ability to capture both peak and trough values throughout the year. The single LSTM (green line) also performs well, though it deviates slightly during certain peaks, underestimating some higher concentrations and predicting low values, especially in periods where there are no observed values. On the other hand, the EOF-based models, particularly DInEOF (blue dashed line), exhibit much greater variability. DInEOF tends to overestimate peaks, adding more oscillations to the data, especially during the high-concentration periods of early and late 2018. eDIneof has a more stable behaviour, but it still shows more oscillations than 4DVarNet models.

For the ZOUTKPLG station, the time series plots follow a similar behaviour to BOCHTVWTND. Once again, the Double LSTM offers the best match with the observed data, capturing key trends and fluctuations, particularly in mid-2018, where it tracks closely to the observed peaks. The Single LSTM also performs reasonably well but tends to smooth out some of the sharper spikes in concentration, such as those observed during March and April 2018. In contrast, the DInEOF and eDInEOF models, especially DInEOF, show higher variability and overestimate peak concentrations throughout the year. These models consistently introduce sharp fluctuations that do not align with the observed CMEMS data, particularly in the case of DInEOF, which overshoots the values during critical periods.

Tab. 4.5 shows the numerical results, which agree with what the time series plots show. For the BOCHTVWTND station, the Double LSTM model shows the lowest RMSE (4.1873), indicating that it is the most accurate in predicting SPM concentrations, followed by single LSTM with 7.7006. In terms of RE, Double LSTM also performs well, with a relative error of 31.8184, which is lower than that of the other models, though the Single LSTM (with an RE of 39.1200) also performed closely. On the other hand, both EOF-based models, DInEOF and eDInEOF, exhibit

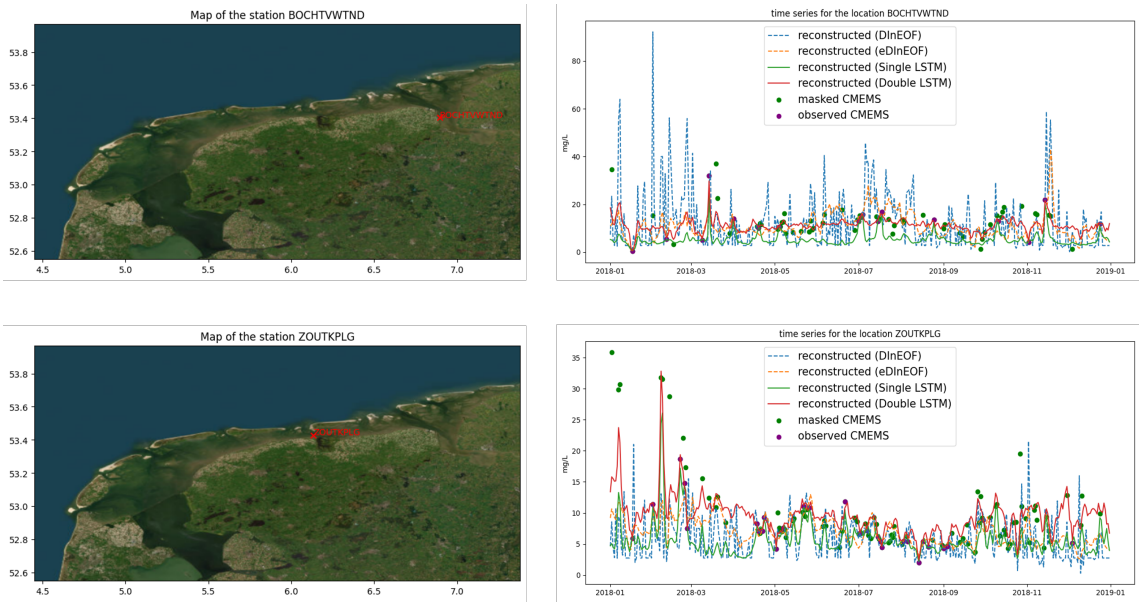


Figure 4.8: The figure shows the time series (right images) for two stations in the Wadden Sea, BOCHTVWTND and ZOUTKPLG, and their respective locations on the map (left images). DInEOF shows a high oscillating behaviour, meanwhile the other methods seem to be more stable. In particular, single and double LSTM interpolate the unobserved pixels (green dots) better. Double LSTM is able to reconstruct more consistently higher SPM values and it doesn't drop to values close to zero during intervals of missing values, which is the issue of single LSTM.

much higher RMSE values (12.6895 and 7.7720, respectively), confirming the visual observation that these models overestimate peaks and introduce significant variability. Their RE values are also notably higher, with DInEOF showing a large RE of 63.6614 and eDInEOF of 51.4314.

For the ZOUTKPLG station, the Double LSTM once again has the best performance, with an RMSE of 3.2551 and an RE of 17.9148. The Single LSTM also performs well, with an RMSE of 5.1861 and an RE of 18.1066, closely matching Double LSTM, although it slightly underestimates the variability seen in the observed data, as noted in the time series analysis. The EOF-based models again show weaker performance: DInEOF has an RMSE of 5.9302 and eDInEOF has an RMSE of 5.5775, both significantly worse than the LSTM-based models. Their RE values (22.5151 for DInEOF and 23.8033 for eDInEOF) are also higher, reflecting their difficulty in accurately reconstructing data at this station.

Metrics		**single LSTM**	**double LSTM**	**DInEOF**	**eDInEOF**
BOCHTVWTND	RMSE	7.7006	4.1873	12.6895	7.7720
	RE	39.1200	31.8184	63.6614	51.4314
ZOUTKPLG	RMSE	5.1861	3.2551	5.9302	5.5775
	RE	18.1066	17.9148	22.5151	23.8033

Table 4.5: The table reports the metrics computed for the time series of location BOCHTVWTND and ZOUTKPLG. In both location, double LSTM has better numerical results in both the metrics, but it is for the RMSE where it has the biggest improvement compared to the other methods.

Overall, the Double LSTM emerges as the most reliable model for both stations, according to both time series plots and numerical results, demonstrating better alignment with observed CMEMS data and accurately capturing the trends in SPM concentration. It consistently follows the peaks and troughs with fewer deviations compared to the other models. Single LSTM also performs well but occasionally it cannot reconstruct correctly higher values, especially in periods

with a high rate of missing values. In contrast, the EOF-based models (DInEOF and eDInEOF) exhibit more significant limitations, particularly in overestimating peaks and introducing higher variability.

Chapter 5

Discussion

In this chapter, the results will be discussed in depth and there will be given conclusions on the outcome of the experiments. Moreover, the limitations and issues encountered during the project will be introduced, and how they affected the experiment results will be explained. Finally, the last part will talk about related works and future research.

5.1 Explanation of the results

The results presented in Sec. 4 show the numerical and visual results of the different models applied to the CMEMS dataset. These results are further discussed to better contextualize them in relation to the research goals.

In Sec. 4.3.1, Tab. 4.2 shows that double LSTM has an overall RSME lower than the other models, but if we focus on the Wadden Sea area, the difference between the double LSTM value and the other models is bigger than the difference between double LSTM RMSE value with the other models' values in the remaining area (offshore). Conversely, single LSTM has an overall lower RE compared to the other models and the difference is significantly high. Interestingly, this difference becomes way smaller if we focus on the Wadden Sea area, meanwhile it increases in the offshore area.

All these results, which have been already comprehensively discussed in the previous chapter, suggest two interpretations: first, double LSTM performs better in areas with higher values, namely with a higher SPM concentration, and secondly, single LSTM, even if it performs slightly worse with higher SPM concentrations than double LSTM, has more consistent performances in the overall areas, even with lower SPM concentration.

As explained in Sec. 3.3, each metric has its own advantages and disadvantages. In particular, RMSE lacks of interpretability with values at different scales, which is the case in the area of interest of our study: usually, there are higher concentration of SPM in the Wadden Sea than in the rest of the study area, leading to a non-uniform spatial distribution of the values. For this reason, the value of the overall RMSE is more sensible to the errors in the Wadden Sea area than in the other areas, leading the double LSTM to have a better RMSE than the other models, which perform worse on the Wadden Sea. This is further confirmed by Tab. 4.2, where it is visible that the RMSE values for the Wadden Sea and the other areas have different value scales.

On the other hand, RE is more suitable to capture errors at different scales. From Tab. 4.2, we already know that single LSTM has the best RE among all the models. What is interesting is the fact that its overall RE values don't differ too much compared to the RE value for the Wadden Sea and other areas, suggesting that the model is able to reconstruct images with a constant proportional error on the entire spatial domain. Instead, all the other models have bigger gaps between RE values in the Wadden Sea and the other regions, where the error is larger offshore than in the coastal area. In particular, double LSTM results are more accurate in the Wadden Sea and less accurate in the other regions, which are offshore areas with lower SPM concentration, where even if the reconstructed values don't have high absolute error, they could have a higher relative error due to the smaller scale.

Moreover, SSIM indexes give further validation of the results shown by RMSE and RE, since they suggest that single and double LSTM reconstructed images are the most similar to the original ones, meanwhile DInEOF and eDInEOF are more different from the original images.

The Sec. 4.3.2 gives a better view of the reconstructions of a single day, which was picked for being almost complete (there are some data missing in thecoastal area). It has been already noticed that the 4DVarNet models predict smoother results than EOF-methods, which tend to yield more "pixelated" outputs in offshore areas while remaining quite accurate in the Wadden Sea. Comparing single LSTM with double LSTM, we can see how they are visually similar, but double LSTM has slightly higher reconstructed SPM values: this is confirmed by the density function plots in Fig. 4.7 (right figure). The density function plots of the two models are similar, but the one of double LSTM (orange line) is shifted slightly on the right. Thus, in the Wadden sea and for higher SPM values, double LSTM is more accurate but it overshoots lower values, while single LSTM is more consistent. Again, the metrics in Tab. 4.3 confirm that double LSTM has the best RMSE, because of the better accuracy on the high SPM values in the Wadden Sea, but single LSTM has a better RE, which reflects its consistency in reconstructing the values. Also in terms of similarity (SSIM), the two 4DVarNet models are visually more similar to the original image than the other two.

The results in Sec. 4.3.3 are interesting because they show the temporal reconstruction of the SPM values in specific sites in the Wadden Sea, BOCHTVWTND and ZOUTKPLG, where the fluctuations are more visible through the year. Specifically, we already talked about the high variance of the EOF-methods, but if we focus on the 4DVarNet methods, single LSTM doesn't reconstruct correctly SPM values, especially in long periods with no or barely any observations available, such as between May and July 2018 for BOCHTVWTND station and in between March and May 2018 for ZOUTKPLG station, showing its bias towards lower values described in 3.1.4. Meanwhile, double LSTM is able to reconstruct values that are closer to the real one, especially bigger values, but sometimes it doesn't interpolate correctly smaller values. Anyway, double LSTM shows better behaviour which is able to recognize daily fluctuations, peaks and lows of SPM concentration. In this case, double LSTM has better RMSE and RE than the other models, and the reason is that we are looking at values at the same scale, therefore the RMSE shows better the reconstructions quality and it is similar to the RE. This results from Tab. 4.5 are in agreement with the ones in Tab. 4.2 in the row of the Wadden Sea area.

In summary, 4DVarNet models outperform DInEOF and eDInEOF for satellite image reconstruction in terms of different metrics, as well as in computation speed, showing clearly their advantages. Single LSTM shows overall more constant results on the whole area, but double LSTM has better results on the Wadden Sea area, suggesting that this model is able to reconstruct more accurately areas with higher concentration of SPM.

5.2 Limitations

During the data preparation, different problems have been encountered and, therefore, some assumptions and simplifications have been applied. Moreover, there have been choices on the structure of the models, most of them based on the computation resources available for running the experiments. In this section, these issues will be discussed and explained how they influenced the outcomes of the experiments.

5.2.1 Data

As mentioned in 4.1.3, after the preprocessing of the dataset, we obtained two similar datasets, but they were still differing from each other for their values spatial distribution: from Fig. 5.2, it is clear how different are the two datasets in terms of the spatial distribution of the values. The model data are generally distributed towards higher values, meanwhile a large part of satellite data is concentrated around 0. Even the dynamics in the area are different, which is visible from Fig. 5.1.

This difference between the datasets prevents achieving a possible better reconstruction quality, since during the training phase of 4DVarNet, the dynamics behind the model data have been

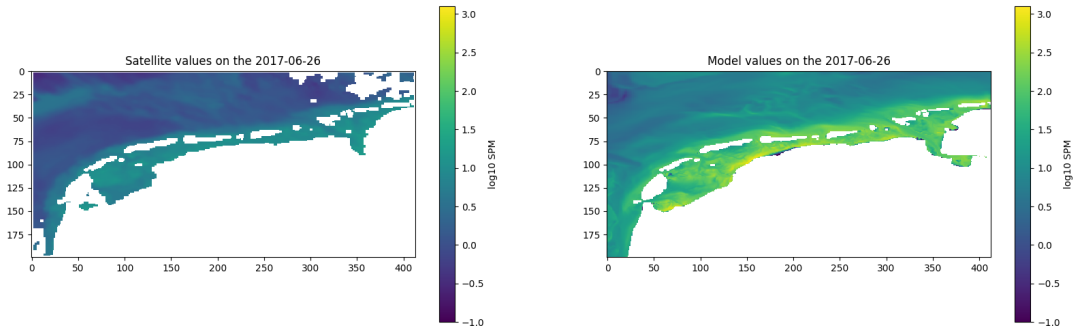


Figure 5.1: Here is the map of the satellite data (left) and model data (right) for the day 26-06-2017. It is possible to see how they differ in the values distribution, with model data having higher values overall, especially in the Wadden Sea area.

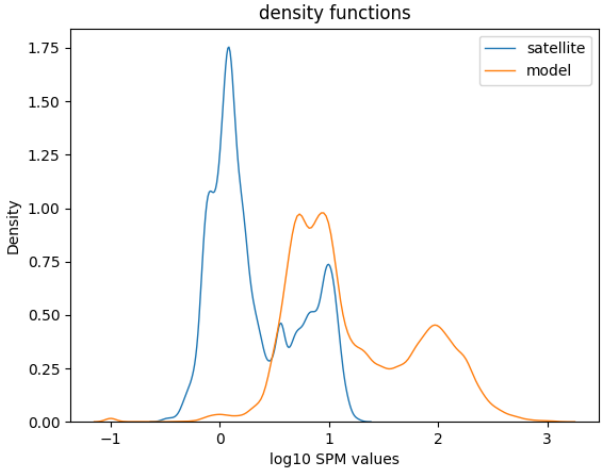


Figure 5.2: Plot of the densities functions of the SPM values for satellite data (orange line) and model data (blue line). For certain aspects, they have a similar distribution but satellite data have more values concentrated around 0 and in general less higher values.

learned, but when the pre-trained model is tested on the CMEMS data, which have different dynamics, therefore the reconstruction will be done on the base of the model data, not the satellite data, providing reconstructions that follow more the model data than the actual satellite data.

To overcome this issue, one solution was to fine-tune the pre-trained model on the satellite data, but in this case the results were not improving, if not getting worse. The reason could be that the CMEMS dataset isn't good enough to be used to train the model for its high rate of missing data, however further exploration in this way could be done, maybe changing some model hyperparameters or loss function. A second solution could be the availability of model data that agree better with satellite data, but this is more a limitation of the DELFT3D FM model and the way of generating data. A third solution would be finding more correlation between simulated data and satellite data with less missing data to align the datasets.

There are two further issues with the datasets: the first is that the DELFT3D dataset only provides two years worth of data only one year is used for training the models, which is a very limited period for the learning process. The second issue is with the CMEMS data, which have a high rate of missing data: this makes both the reconstruction and the performance analysis more challenging.

5.2.2 Second experiment

The reason why the results for the second experiments have been not presented in Sec. 4.3 is because the application of the 4DVarNet of the CMEMS dataset was not successful. The main explanation is that this dataset has a very high rate of missing data (more than 95% overall), so there are too few for the pre-trained model to be able to reconstruct in a good way the SPM values.

However, the aim of this second experiment is to show the limitations of the model in terms of number of parameters and computation time when we use higher-resolution data (and consequently, bigger datasets). Looking at the table 8.1, the batch size and the hidden layer dimension had to be downgraded in order to run the model on the GPU, and still, the training time resulted in more than 6 times than the single LSTM of the first experiment. When applied to the CMEMS dataset of 4 years, it gives the reconstructed output in 19 minutes, even if their quality is not good.

5.2.3 Model complexity

The choice of the models hyperparameters and their consequent complexity (see Tab. 4.1) was guided by two main factors: previous works in the literature and computation sources. For the first, the choice relied on the previous similar applications of 4DVarNet [73, 72] and a parallel work, that is still ongoing, at IMT Atlantique institute, team that firstly introduced 4DVarNet. Moreover, IMT Atlantique is one of Deltares' partners in the EDITO project, a collaboration in the scope of European Horizon projects. For example, in the parallel work with IMT Atlantique it was found that setting 10 days as time window is optimal, because increasing the number only leads to poorer performances of the algorithm.

For the computation power, the dimension of the hidden layers in the Autoencoder and in the LSTM, the downsampling rate and the batch size had to be fixed at the corresponding values because, during the training, the model reached the maximum capacity of the GPU of 80GB. Of course, there are no other ways to solve this limitation other than using higher capacity GPUs or parallelising the computation tasks, which was not possible on the Deltares cluster. However, compared to previous experiments, here it was used a higher number of parameters, especially because 3D convolutional layers have been used instead of two dimensional.

5.3 Related works and future researches

In the systematic literature review from the work of Catipovic et al. [16], it emerges that there exist several other studies in oceanography with the same aim of reconstructing satellite data from partial observation, some of them even more recent than 4DVarNet. Most of them exploit machine learning methods, in particular neural networks, highlighting how the research is shifting towards these new techniques that are demonstrated to be more efficient than numerical methods.

There has been also research on the side of neural network architectures for optimization convergences, related to the work of Andrychowicz et al. [3], which they were grouped together and defined as techniques for "learning to optimize" or "learning to learn", depending if the methods is based on meta-learning schemes or not. The aim of these techniques is to speed up the learning within tasks (improving the convergence) and to be able to transfer learning across tasks from the same distribution (generalization). The framework used in 4DVarNet is a simplified version of the one proposed in [3], but Chen et al. [18] showed how there exist other methods more complex and with faster convergences than this version. Moreover, the double LSTM variation introduced in this work has some similarities with the algorithm unrolling schemes [34]: the unrolling schemes are structured in the same way as the recurrent neural network, but the parameters are not shared between each iteration, meanwhile double LSTM has only the parameters of the first iteration that are not shared with the rest of the parameters. To summarize, double LSTM can be seen as a special case of unrolling algorithms, where only a subset of parameters is shared. Obviously, having more, if not all, parameters that are not shared across the iterations layers leads to a more complex model that would probably learn better the optimization task, but it will require more computation power as well.

For future research, the availability of more powerful computing sources will allow to expand the model parameters or explore more complex architectures, for example, tranformer-based ones

[71], which are demonstrated to be effective for the same problem by the work of Archambault et al. [4]. Transformers have already revolutionized other machine learning fields, i.e. natural language processes and, more specifically, large language models (LLMs), so it would be worth exploring them further to improve the performances. Another interesting technique is stable diffusion, which is part of what is nowadays called Generative AI. Stable diffusion models [38] are a type of probabilistic generative model and the main idea is to generate images simulating a diffusion process. These models could be used in different ways for the SPM reconstruction: the inpainting techniques [21] use stable diffusion models and they have been shown to be very powerful in filling incomplete images, which is a very similar task to the aim of this project. Another way of using stable diffusion models is for solving dynamical systems, for example, the work of Huang et al. [39] shows successfully the application of a diffusion model for forward and inverse problems. Lastly, a new LSTM-based architecture called extended LSTM (xLSTM) [9] was recently proposed, showing better performances than state-of-the-art models, such as the transformers, for LLMs and it is envisioned that it could be beneficial for time series predictions: in the context of this project, it could be interesting exploring whether the neural solver in the 4DVarNet can be enhanced using this new model. In conclusion, considering the amount of new research and developments in the deep learning field that have been just discussed, there are many different directions that could be taken in the future to improve 4DVarNet.

Chapter 6

Conclusion and Overlooks

This thesis project presents the application, analysis and comparison of 4DVarNet as a novel method for reconstructing SPM fields, focusing in particular on the Dutch Wadden Sea area. Based on the analysis and results discussed in Sec. 4 and 5, it is evident that the 4DVarNet algorithm provides significant improvements in the reconstruction of SPM data, particularly in areas with higher sediment concentration, such as the Dutch Wadden Sea. More specifically the single and double LSTM variations of 4DVarNet, have demonstrated their ability to outperform traditional methods like DInEOF and eDInEOF, both in terms of accuracy and computational efficiency.

The double LSTM model, in particular, showed superior performance in regions with high SPM concentrations, while the single LSTM exhibited more consistent performance across the entire study area, even in regions with lower SPM values. This suggests that double LSTM is better suited for environments characterized by complex sediment dynamics, whereas single LSTM may be more appropriate for broader, less variable regions.

There are multiple ways to improve the method and its performance, which require further experiments and digging more into the literature. With the advancements of research in the machine learning field.

In conclusion, the findings of this thesis not only confirm the effectiveness of 4DVarNet for satellite image reconstruction in marine environments, but also demonstrate its advantages compared to traditional numerical methods. Future work may focus on refining these models to further improve their accuracy and convergences, optimizing them for a better use of computing sources and extending the applicability of such methods to other variables. Moreover, it is possible to explore new alternative architectures that could eventually perform better. The contributions of this study, particularly in the context of the Dutch Wadden Sea, provide valuable insights for future research and practical applications in oceanography and marine ecosystems.

I would like to mention that in September 2024, I partecipated a hackathon in Grenoble for the EDITO Model lab and, in a team with other three researches, we deployed a walkthrough jupyter notebook for using a pre-trained model of 4DVarNet to be used on the EDITO Datalab platform [1]. For Deltares, this work will be converted in a product to be used by the modellers and researchers. Finally, in December 2024, I will present online this project at the American Geophysical Union (AGU) in Washington [2].

[1] You can find more information here https://datalab.dive.edito.eu/catalog/All.
[2] More info at https://www.agu.org/annual-meeting

Chapter 7

Acknowledgments

I would firstly thanks my supervisors from Deltares, Felix Dols and Lorinc Meszaros, who supported me during the entire duration of my master thesis' project, both in a professional and personal way, and facilitated me . I would also thanks my daily supervisor from University of Twente, Dongwei Ye, who gave me invaluable feedbacks and suggestions for delivering a proper academic work, and the committee chair, Prof. Christoph Brune, who also provided me with precious insights on how to improve my results.

Now, a special thanks goes to my family, and in particular to my parents and my sister, who supported me through the entire time of my master and believed in my ability to accomplish this important life's goal. Another big thank goes to my girlfriend, Roxana, who, despite the distance, always supported me and make me feel like we were closer the entire time.

The last thank goes to my all the friends I found here in Enschede, who since the beginning revealed to be some wonderful people and helped me to feel less alone in a completely new country, and with I spent some unforgettable moments.

Chapter 8

Appendixes

8.1 Appendix A

Values of training and validation loss functions of single LSTM:

- Training Loss values: [0.182994, 0.103153, 0.0882704, 0.0696425, 0.0641583, 0.059883, 0.375308, 0.383062, 0.0709362, 0.0629637, 0.0595105, 0.0568493, 0.0555639, 0.0540822, 0.0533253, 0.0523429, 0.0518311, 0.0511065, 0.0505824, 0.0499403, 0.0493538, 0.0488100, 0.0484268, 0.0480129, 0.0497744, 0.0474353, 0.0469372, 0.0466150, 0.0467049, 0.0456301, 0.0454793, 0.0451096, 0.0454020, 0.0444325, 0.0529801, 0.0461034, 0.0439058, 0.0434973, 0.0428512, 0.0428854, 0.0426534]

- Validation Loss values: [0.120422, 0.0924242, 0.0784136, 0.0696589, 0.0634635, 0.0610729, 9.98053, 0.0821068, 0.0688668, 0.0641610, 0.0621118, 0.0590476, 0.0577248, 0.0574321, 0.0552131, 0.0550912, 0.0544498, 0.0530637, 0.0524130, 0.0520975, 0.0521850, 0.0516933, 0.0511729, 0.0504027, 0.0500852, 0.0492786, 0.0492190, 0.0490129, 0.0478850, 0.0481029, 0.0476048, 0.0474192, 0.0475227, 0.0462603, 0.0564469, 0.0470933, 0.0464676, 0.0456783, 0.0453731, 0.0470752, 0.0448961]

Values of training and validation loss functions of double LSTM:

- Training Loss values: [0.179853, 0.0850784, 0.0687735, 0.0602103, 0.0546198, 0.0516198, 0.0480551, 0.0457882, 0.147422, 0.0531328, 0.0471642, 0.0452124, 0.0501932, 0.0431243, 0.0428593, 0.0410455, 0.041964, 0.040293, 0.0399238, 0.0390697, 0.047607, 0.0400847, 0.0380444, 0.0377549, 0.0375948, 0.0542451, 0.0389216, 0.0373682, 0.0363356, 0.036557, 0.0359506, 0.0353748, 0.03568, 0.0350718, 0.151393, 0.0580548, 0.0433809, 0.0409459, 0.0377469, 0.0370459, 0.0357325]

- Training Loss values: [0.111603, 0.0744371, 0.0655738, 0.0586927, 0.0553839, 0.0513756, 0.0499718, 0.0489341, 0.0595644, 0.051007, 0.0489389, 0.0470164, 0.04809, 0.0452134, 0.0449424, 0.0458248, 0.0432384, 0.0438112, 0.0422725, 0.0421993, 0.0480683, 0.0426299, 0.0474407, 0.0407002, 0.0446952, 0.0443446, 0.04718, 0.0409945, 0.0408469, 0.0404462, 0.041239, 0.0393663, 0.0403744, 0.0417896, 0.0832283, 0.0500941, 0.043683, 0.0418242, 0.0415099, 0.0410792, 0.0397813]

8.2 Appendix B

Second experiment model parameters and loss function.

	Single LSTM
Number of Channels	1
Time Window	10
Hidden layers dimension (AE)	64
downsampling (AE)	2
Batch size	1
Trainable parameters (AE)	666177
Training Epochs	50
Solver iterations	15
Hidden layers dimension (LSTM)	48
Trainable parameters (Full)	767076
Training time	5622m 57s

Table 8.1: Model parameters and hyperparameters used in the second experiment.

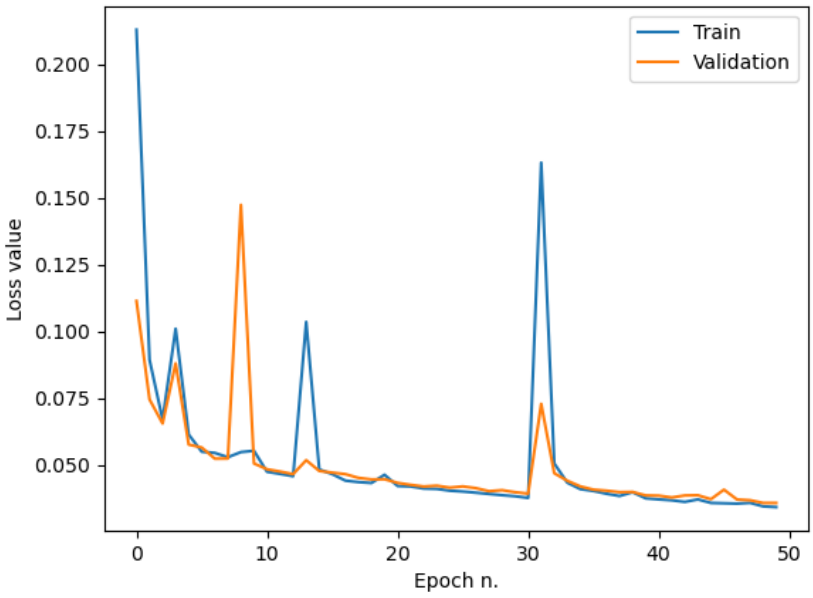


Figure 8.1: Loss function values for the training and validation phases.

Bibliography

[1] Aïda Alvera-Azcárate, Alexander Barth, Damien Sirjacobs, and J-M Beckers. Enhancing temporal correlations in eof expansions for the reconstruction of missing data using dineof. *Ocean Science*, 5(4):475–485, 2009.

[2] Aida Alvera-Azcárate, Quinten Vanhellemont, Kevin Ruddick, Alexander Barth, and Jean-Marie Beckers. Analysis of high frequency geostationary ocean colour data using dineof. *Estuarine, Coastal and Shelf Science*, 159:28–36, 2015.

[3] Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando De Freitas. Learning to learn by gradient descent by gradient descent. *Advances in neural information processing systems*, 29, 2016.

[4] Théo Archambault, Pierre Garcia, Anastase Alexandre Charantonis, and Dominique Béréziat. Deep sea surface height multivariate interpolation. In *EGU General Assembly Conference Abstracts*, page 17465, 2024.

[5] Charles P Arnold Jr and Clifford H Dey. Observing-systems simulation experiments: Past, present, and future. *Bulletin of the American Meteorological Society*, 67(6):687–695, 1986.

[6] Z Bainbridge, S Lewis, R Bartley, K Fabricius, C Collier, J Waterhouse, A Garzon-Garcia, B Robson, J Burton, A Wenger, et al. Fine sediment and particulate organic matter: A review and case study on ridge-to-reef transport, transformations, fates, and impacts on marine ecosystems. *Marine Pollution Bulletin*, 135:1205–1220, 2018.

[7] Alexander Barth, Aida Alvera-Azcárate, Matjaz Licer, and Jean-Marie Beckers. Dincae 1.0: A convolutional neural network with error estimates to reconstruct sea surface temperature satellite observations. *Geoscientific Model Development*, 13(3):1609–1622, 2020.

[8] Belhassen Bayar and Matthew C Stamm. Constrained convolutional neural networks: A new approach towards general purpose image manipulation detection. *IEEE Transactions on Information Forensics and Security*, 13(11):2691–2706, 2018.

[9] Maximilian Beck, Korbinian Pöppel, Markus Spanring, Andreas Auer, Oleksandra Prudnikova, Michael Kopp, Günter Klambauer, Johannes Brandstetter, and Sepp Hochreiter. xlstm: Extended long short-term memory. *arXiv preprint arXiv:2405.04517*, 2024.

[10] Charles Troupin Beckers. Introduction to optimal interpolation and variational analysis. 2008.

[11] Jean-Marie Beckers and Michel Rixen. Eof calculations and data filling from incomplete oceanographic datasets. *Journal of Atmospheric and oceanic technology*, 20(12):1839–1856, 2003.

[12] AF Bennett and PC McIntosh. Open ocean modeling as an inverse problem: Tidal theory. *Journal of physical oceanography*, 12(10):1004–1018, 1982.

[13] Andrew F Bennett. *Inverse methods in physical oceanography*. Cambridge university press, 1992.

[14] Ángel Borja, Mike Elliott, Jacob Carstensen, Anna-Stiina Heiskanen, and Wouter van de Bund. Marine management–towards an integrated implementation of the european marine strategy framework and the water framework directives. *Marine pollution bulletin*, 60(12):2175–2186, 2010.

[15] B Casey, R Arnone, and P Flynn. Simple and efficient technique for spatial/temporal composite imagery. published in the proceedings of spie. In *Conference on Coastal Ocean Remote Sensing, v6680, held in San Diego, CA on*, pages 26–30, 2007.

[16] L Ćatipović, H Kalinić, T Županović, S Sathyendranath, J Dingle, T Jackson, and F Matić. Implementation of gan-based satellite derived chlorophyll-a concentration gap reconstruction. 2023.

[17] Leon Ćatipović, Frano Matić, and Hrvoje Kalinić. Reconstruction methods in oceanographic satellite data observation—a survey. *Journal of marine science and engineering*, 11(2):340, 2023.

[18] Tianlong Chen, Xiaohan Chen, Wuyang Chen, Howard Heaton, Jialin Liu, Zhangyang Wang, and Wotao Yin. Learning to optimize: A primer and a benchmark. *Journal of Machine Learning Research*, 23(189):1–59, 2022.

[19] Özgün Çiçek, Ahmed Abdulkadir, Soeren S Lienkamp, Thomas Brox, and Olaf Ronneberger. 3d u-net: learning dense volumetric segmentation from sparse annotation. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2016: 19th International Conference, Athens, Greece, October 17-21, 2016, Proceedings, Part II 19*, pages 424–432. Springer, 2016.

[20] PHILIPPE Courtier, J-N Thépaut, and Anthony Hollingsworth. A strategy for operational implementation of 4d-var, using an incremental approach. *Quarterly Journal of the Royal Meteorological Society*, 120(519):1367–1387, 1994.

[21] Antonio Criminisi, Patrick Perez, and Kentaro Toyama. Object removal by exemplar-based inpainting. In *2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2003. Proceedings.*, volume 2, pages II–II. IEEE, 2003.

[22] Junyu Dong, Ruiying Yin, Xin Sun, Qiong Li, Yuting Yang, and Xukun Qin. Inpainting of remote sensing sst images with deep convolutional generative adversarial network. *IEEE geoscience and remote sensing letters*, 16(2):173–177, 2018.

[23] Matthew Ehrler and Neil Ernst. Vconstruct: Filling gaps in chl-a data using a variational autoencoder. *arXiv preprint arXiv:2101.10260*, 2021.

[24] D Eisma. Flocculation and de-flocculation of suspended matter in estuaries. *Netherlands Journal of Sea Research*, 20(2-3):183–199, 1986.

[25] Geir Evensen. Sequential data assimilation with a nonlinear quasi-geostrophic model using monte carlo methods to forecast error statistics. *Journal of Geophysical Research: Oceans*, 99(C5):10143–10162, 1994.

[26] Geir Evensen and Peter Jan Van Leeuwen. An ensemble kalman smoother for nonlinear dynamics. *Monthly Weather Review*, 128(6):1852–1867, 2000.

[27] Geir Evensen, Femke C Vossepoel, and Peter Jan Van Leeuwen. *Data assimilation fundamentals: A unified formulation of the state and parameter estimation problem*. Springer Nature, 2022.

[28] Ronan Fablet, Bertrand Chapron, Lucas Drumetz, Etienne Mémin, Olivier Pannekoucke, and François Rousseau. Learning variational data assimilation models and solvers. *Journal of Advances in Modeling Earth Systems*, 13(10):e2021MS002572, 2021.

[29] Ronan Fablet, Said Ouala, and Cedric Herzet. Bilinear residual neural network for the identification and forecasting of geophysical dynamics. In *2018 26th European signal processing conference (EUSIPCO)*, pages 1477–1481. IEEE, 2018.

[30] Jianxin Fan, Jiaxin Yang, Fulong Cheng, and Shikuo Zhang. The source, distribution, and environmental effects of suspended particulate matter in the yangtze river system. *Water*, 15(19):3429, 2023.

[31] Lev Semenovich Gandin. Objective analysis of meteorological fields. *Israel program for scientific translations*, 242, 1963.

[32] Unai Ganzedo, Aida Alvera-Azcarate, Ganix Esnaola, Agustin Ezcurra, and Jon Saenz. Reconstruction of sea surface temperature by means of dineof: a case study during the fishing season in the bay of biscay. *International journal of remote sensing*, 32(4):933–950, 2011.

[33] Neil J Gordon, David J Salmond, and Adrian FM Smith. Novel approach to nonlinear/non-gaussian bayesian state estimation. In *IEE proceedings F (radar and signal processing)*, volume 140, pages 107–113. IET, 1993.

[34] Karol Gregor and Yann LeCun. Learning fast approximations of sparse coding. In *Proceedings of the 27th international conference on international conference on machine learning*, pages 399–406, 2010.

[35] Thomas M Hamill and Chris Snyder. A hybrid ensemble kalman filter–3d variational analysis scheme. *Monthly Weather Review*, 128(8):2905–2919, 2000.

[36] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[37] Nobuyuki Hirahara, Motoharu Sonogashira, Hidekazu Kasahara, and Masaaki Iiyama. Denoising and inpainting of sea surface temperature image with adversarial physical model loss. In *Pattern Recognition: 5th Asian Conference, ACPR 2019, Auckland, New Zealand, November 26–29, 2019, Revised Selected Papers, Part I 5*, pages 339–352. Springer, 2020.

[38] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020.

[39] Jiahe Huang, Guandao Yang, Zichen Wang, and Jeong Joon Park. Diffusionpde: Generative pde-solving under partial observation. *arXiv preprint arXiv:2406.17763*, 2024.

[40] CF Jago, GM Kennaway, G Novarino, and SE Jones. Size and settling velocity of suspended flocs during a phaeocystis bloom in the tidally stirred irish sea, nw european shelf. *Marine Ecology Progress Series*, 345:51–62, 2007.

[41] Sihun Jung, Cheolhee Yoo, and Jungho Im. High-resolution seamless daily sea surface temperature based on satellite data fusion and machine learning over kuroshio extension. *Remote Sensing*, 14(3):575, 2022.

[42] Hrvoje Kalinić, Zvonimir Bilokapić, and Frano Matić. Can local geographically restricted measurements be used to recover missing geo-spatial data? *Sensors*, 21(10):3507, 2021.

[43] Rudolph E Kalman and Richard S Bucy. New results in linear filtering and prediction theory. 1961.

[44] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. 1960.

[45] Song-Hee Kang, Youngjin Choi, and Jae Young Choi. Restoration of missing patterns on satellite infrared sea surface temperature images due to cloud coverage using deep generative inpainting network. *Journal of Marine Science and Engineering*, 9(3):310, 2021.

[46] Vladimir Krasnopolsky, Sudhir Nadiga, Avichal Mehra, Eric Bayler, and David Behringer. Neural networks technique for filling gaps in satellite measurements: Application to ocean color observations. *Computational intelligence and neuroscience*, 2016(1):6156513, 2016.

[47] Ewa J Kwiatkowska and Giulietta S Fargion. Application of machine-learning techniques toward the creation of a consistent and calibrated global chlorophyll concentration baseline dataset using remotely sensed ocean color data. *IEEE Transactions on Geoscience and Remote Sensing*, 41(12):2844–2860, 2003.

[48] Steven N Liss, Ian G Droppo, Gary G Leppard, and Timothy G Milligan. *Flocculation in natural and engineered environmental systems*. CRC press, 2004.

[49] Xiaoming Liu and Menghua Wang. Analysis of ocean diurnal variations from the korean geostationary ocean color imager measurements using the dineof method. *Estuarine, Coastal and Shelf Science*, 180:230–241, 2016.

[50] Xiaoming Liu and Menghua Wang. Gap filling of missing data for viirs global ocean color products using the dineof method. *IEEE Transactions on Geoscience and Remote Sensing*, 56(8):4464–4476, 2018.

[51] Andrew C Lorenc. Analysis methods for numerical weather prediction. *Quarterly Journal of the Royal Meteorological Society*, 112(474):1177–1194, 1986.

[52] Edward N Lorenz. *Empirical orthogonal functions and statistical weather prediction*, volume 1. Massachusetts Institute of Technology, Department of Meteorology Cambridge, 1956.

[53] Edward N Lorenz. Deterministic nonperiodic flow. *Journal of atmospheric sciences*, 20(2):130–141, 1963.

[54] Edward N Lorenz. Predictability: A problem partly solved. In *Proc. Seminar on predictability*, volume 1. Reading, 1996.

[55] IN McCave. Introduction to the physics of cohesive sediment in the marine environment. *Geological Magazine*, 143(1):137, 2006.

[56] Hamid Mohebzadeh, Esmaiil Mokari, Prasad Daggupati, and Asim Biswas. A machine learning approach for spatiotemporal imputation of modis chlorophyll-a. *International Journal of Remote Sensing*, 42(19):7381–7404, 2021.

[57] Peter R Oke, Gary B Brassington, David A Griffin, and Andreas Schiller. The bluelink ocean data assimilation system (bodas). *Ocean modelling*, 21(1-2):46–70, 2008.

[58] Said Ouala, Ronan Fablet, Cédric Herzet, Bertrand Chapron, Ananda Pascual, Fabrice Collard, and Lucile Gaultier. Neural network based kalman filters for the spatio-temporal interpolation of satellite-derived sea surface temperature. *Remote Sensing*, 10(12):1864, 2018.

[59] Jinku Park, Jeong-Hoon Kim, Hyun-cheol Kim, Bong-Kuk Kim, Dukwon Bae, Young-Heon Jo, Naeun Jo, and Sang Heon Lee. Reconstruction of ocean color data using machine learning techniques in polar regions: Focusing on off cape hallett, ross sea. *Remote Sensing*, 11(11):1366, 2019.

[60] Bo Ping, Fenzhen Su, and Yunshan Meng. An improved dineof algorithm for filling missing values in spatio-temporal sea surface temperature data. *PLoS One*, 11(5):e0155928, 2016.

[61] Claire Pottier, Véronique Garçon, Gilles Larnicol, Joël Sudre, Philippe Schaeffer, and P-Y Le Traon. Merging seawifs and modis/aqua ocean color data in north and equatorial atlantic using weighted averaging and objective analysis. *IEEE transactions on geoscience and remote sensing*, 44(11):3436–3451, 2006.

[62] LD Pukhtyar, SV Stanichny, and IE Timchenko. Optimal interpolation of the data of remote sensing of the sea surface. *Physical Oceanography*, 19(4):225–239, 2009.

[63] Yoshikazu Sasaki. Numerical variational analysis with weak constraint and application to surface analysis of severe storm gust. *Monthly Weather Review*, 98(12):899–910, 1970.

[64] Yoshikazu Sasaki. Some basic formalisms in numerical variational analysis. *Monthly Weather Review*, 98(12):875–883, 1970.

[65] Andrew M Stuart. Inverse problems: a bayesian perspective. *Acta numerica*, 19:451–559, 2010.

[66] Swathy Sunder, RAAJ Ramsankaran, and Balaji Ramakrishnan. Machine learning techniques for regional scale estimation of high-resolution cloud-free daily sea surface temperatures from modis data. *ISPRS Journal of Photogrammetry and Remote Sensing*, 166:228–240, 2020.

[67] Swathy Sunder, RAAJ Ramsankaran, and Balaji Ramakrishnan. Machine learning techniques for regional scale estimation of high-resolution cloud-free daily sea surface temperatures from modis data. *ISPRS Journal of Photogrammetry and Remote Sensing*, 166:228–240, 2020.

[68] Xiao-Ling Tan, Guo-Ping Zhang, YIN Hang, Yoko FURUKAWA, et al. Characterization of particle size and settling velocity of cohesive sediments affected by a neutral exopolymer. *International Journal of Sediment Research*, 27(4):473–485, 2012.

[69] Marc H Taylor, Martin Losch, Manfred Wenzel, and Jens Schröter. On the sensitivity of field reconstruction and prediction using empirical orthogonal functions derived from gappy data. *Journal of Climate*, 26(22):9194–9205, 2013.

[70] Peter Jan Van Leeuwen. Representation errors and retrievals in linear and nonlinear data assimilation. *Quarterly Journal of the Royal Meteorological Society*, 141(690):1612–1623, 2015.

[71] A Vaswani. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.

[72] Jean-Marie Vient, Ronan Fablet, Frédéric Jourdin, and Christophe Delacourt. End-to-end neural interpolation of satellite-derived sea surface suspended sediment concentrations. *Remote Sensing*, 14(16):4024, 2022.

[73] Jean-Marie Vient, Frederic Jourdin, Ronan Fablet, Baptiste Mengual, Ludivine Lafosse, and Christophe Delacourt. Data-driven interpolation of sea surface suspended concentrations derived from ocean colour remote sensing data. *Remote Sensing*, 13(17):3537, 2021.

[74] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004.

[75] Mengmeng Yang, Faisal Ahmed Khan, Hongzhen Tian, and Qinping Liu. Analysis of the monthly and spring-neap tidal variability of satellite chlorophyll-a and total suspended matter in a turbid coastal ocean using the dineof method. *Remote Sensing*, 13(4):632, 2021.