RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

Université de Toulouse

Doctorat de l'Université de Toulouse

préparé à Toulouse INP

Assimilation de données en espace latent par des techniques de deep learning

---

Thèse présentée et soutenue, le 8 octobre 2024 par

Mathis PEYRON

École doctorale
EDMITT - Ecole Doctorale Mathématiques, Informatique et Télécommunications de Toulouse

Spécialité
Informatique et Télécommunications

Unité de recherche
CERFACS

Thèse dirigée par
Selime GÜROL et Serge GRATTON

Composition du jury
M. Ronan FABLET, Président, IMT Atlantique
M. Arthur VIDARD, Rapporteur, INRIA Grenoble
Mme Tijana JANJIC, Rapporteuse, Catholic University of Eichstätt-Ingolstadt
M. Alban FARCHI, Examinateur, École Nationale des Ponts et Chaussées
Mme Selime GÜROL, Directrice de thèse, Toulouse INP
M. Serge GRATTON, Co-directeur de thèse, Toulouse INP

Membres invités
M. Léo Nicoletti, Eviden
M. Mikaël Jacquemont, Eviden

À ma famille

Résumé

Mots clés : Assimilation de données, Apprentissage profond, Espace latent, Dynamique de substitution, Filtre de Kalman d'ensemble

Cette thèse, située à l'intersection de l'assimilation de données (AD) et de l'apprentissage profond (AP), introduit un concept nouveau : l'assimilation de données en espace latent. Elle permet une réduction considérable des coûts de calcul et des besoins mémoire, tout en offrant le potentiel d'améliorer la précision des résultats.

Il existe de nombreuses façons d'intégrer l'apprentissage profond dans les algorithmes d'assimilation de données, chacune visant des objectifs différents (Loh *et al.*, 2018; Tang *et al.*, 2020; Laloyaux *et al.*, 2020; Bonavita and Laloyaux, 2020; Brajard *et al.*, 2020; Farchi *et al.*, 2021b; Pawar and San, 2021; Leutbecher, 2019; Ruckstuhl *et al.*, 2021; Lin *et al.*, 2019; Deng *et al.*, 2018; Cheng *et al.*, 2024).

Nous étendons davantage l'intégration de l'apprentissage profond, en repensant le processus même d'assimilation. Notre approche s'inscrit dans la suite des méthodes à espace réduit (Evensen, 1994, 2009a; Bishop *et al.*, 2001; Hunt *et al.*, 2007; Courtier, 2007; Gratton and Tshimanga, 2009; Gratton *et al.*, 2013; Farrell and Ioannou, 2001; Lawless *et al.*, 2008; Cao *et al.*, 2007), qui résolvent le problème d'assimilation en effectuant des calculs dans un espace de faible dimension. Ces méthodes à espace réduit ont été principalement développées pour une utilisation opérationnelle, car la plupart des algorithmes d'assimilation de données s'avèrent être excessivement coûteux, lorsqu'ils sont implémentés dans leur forme théorique originelle.

Notre méthodologie repose sur l'entraînement conjoint d'un autœncodeur et d'un réseau de neurone surrogate. L'autœncodeur apprend de manière itérative à représenter avec précision la dynamique physique considérée dans un espace de faible dimension, appelé espace latent. Le réseau surrogate est entraîné si-

multanément à apprendre la propagation temporelle des variables latentes. Une stratégie basée sur une fonction de coût chaînée est également proposée pour garantir la stabilité du réseau surrogate. Cette stabilité peut également être obtenue en implémentant des réseaux surrogate Lipschitz.

L'assimilation de données à espace réduit est fondée sur la théorie de la stabilité de Lyapunov qui démontre mathématiquement que, sous certaines hypothèses, les matrices de covariance d'erreur de prévision et a posteriori se conforment asymptotiquement à l'espace instable-neutre (Carrassi *et al.*, 2022), qui est de dimension beaucoup plus petite que l'espace d'état. Alors que l'assimilation de données en espace physique consiste en des combinaisons linéaires sur un système dynamique non linéaire, de grande dimension et potentiellement multi-échelle, l'assimilation de données latente, qui opère sur les dynamiques internes sous-jacentes, potentiellement simplifiées, est davantage susceptible de produire des corrections significatives.

La méthodologie proposée est éprouvée sur deux cas tests : une dynamique à 400 variables - construite à partir d'un système de Lorenz chaotique de dimension 40 -, ainsi que sur le modèle quasi-géostrophique de la librairie OOPS (Object-Oriented Prediction System). Les résultats obtenus sont prometteurs.

Abstract

Key words: Data Assimilation, Deep Learning, Latent space, Surrogate dynamics, Ensemble Kalman Filter

This thesis, which sits at the crossroads of data assimilation (DA) and deep learning (DL), introduces latent space data assimilation, a novel data-driven framework that significantly reduces computational costs and memory requirements, while also offering the potential for more accurate data assimilation results.

There are numerous ways to integrate deep learning into data assimilation algorithms, each targeting different objectives (Loh *et al.*, 2018; Tang *et al.*, 2020; Laloyaux *et al.*, 2020; Bonavita and Laloyaux, 2020; Brajard *et al.*, 2020; Farchi *et al.*, 2021b; Pawar and San, 2021; Leutbecher, 2019; Ruckstuhl *et al.*, 2021; Lin *et al.*, 2019; Deng *et al.*, 2018; Cheng *et al.*, 2024).

We extend the integration of deep learning further by rethinking the assimilation process itself. Our approach aligns with reduced-space methods (Evensen, 1994; Bishop *et al.*, 2001; Hunt *et al.*, 2007; Courtier, 2007; Gratton and Tshimanga, 2009; Gratton *et al.*, 2013; Lawless *et al.*, 2008; Cao *et al.*, 2007), which solve the assimilation problem by performing computations within a lower-dimensional space. These reduced-space methods have been developed primarily for operational use, as most data assimilation algorithms are prohibitively costly, when implemented in their full theoretically form.

Our methodology is based on the joint training of an autoencoder and a surrogate neural network. The autoencoder iteratively learns how to accurately represent the physical dynamics of interest within a low-dimensional space, termed latent space. The surrogate is simultaneously trained to learn the time propagation of the latent variables. A chained loss function strategy is also proposed to ensure the stability of the surrogate network. Stability can also be achieved by implementing Lipschitz surrogate networks.

Reduced-space data assimilation is underpinned by Lyapunov stability theory, which mathematically demonstrates that, under specific hypotheses, the forecast and posterior error covariance matrices asymptotically conform to the unstable-neutral subspace (Carrassi *et al.*, 2022), which is of much smaller dimension than the full state space. While full-space data assimilation involves linear combinations within a high-dimensional, nonlinear, and possibly multi-scale dynamic environment, latent data assimilation, which operates on the core, potentially disentangled and simplified dynamics, is more likely to result in impactful corrections.

We tested our methodology on a 400-dimensional dynamics - built upon a chaotic Lorenz96 system of dimension 40 -, and on the quasi-geostrophic model of the Object-Oriented Prediction System (OOPS) framework. We obtained promising results.

Remerciements

Quelle aventure peu ordinaire, trépidante mais aussi parfois déroutante, que celle du doctorat... Terre de recherche à défricher où se côtoient amicalement chercheurs aguerris et experts en devenir !

L'espace de ces quelques pages ne saurait accueillir tous les mots nécessaires au compte-rendu fidèle des rencontres, des moments de partage, des encouragements, des écoutes patientes et attentives, ainsi que des aides bienveillantes qu'il m'a été offert de recevoir. Je vais néanmoins m'essayer à l'exercice, car nombreuses sont les personnes que je me dois de rétribuer par ces quelques mots.

S'il est un point de départ grâce auquel tout ceci a été rendu possible, il réside, sans aucun doute, entre les mains de Serge Gratton, professeur à l'ENSEEIHT. Au-delà du stage de fin d'études qu'il m'a proposé et qui augurait déjà de la thèse à venir, Serge a surtout œuvré avec beaucoup d'efforts et de détermination afin de créer, avec le concours de Stéphane Pralet, un montage CIFRE me permettant de travailler dans les meilleures conditions. Serge a également été un excellent encadrant, tant par ses qualités humaines que par ses précieux apports scientifiques.

Au cœur et au premier plan de cette réussite doctorale se trouve ma directrice de thèse, Selime Gürol. Les mots sont bien faibles pour rendre compte de l'infinie bienveillance, gentillesse, amitié, écoute et sympathie témoignées par Selime, à mon égard ainsi qu'à l'ensemble des membres de l'équipe. Il est rare de croiser sur sa route des personnes dont les valeurs humaines et les compétences professionnelles et scientifiques s'égalent au plus haut niveau. Selime est l'une d'entre elles, et je lui dois énormément, si ce n'est tout, dans la réussite qui est la mienne aujourd'hui. Je lui adresse toute ma gratitude ainsi que tous mes vœux de réussite dans ses projets professionnels et personnels, qu'ils soient présents ou à venir.

Je suis également reconnaissant envers le groupe Atos / Eviden de m'avoir fait confiance pour mener à bien une activité de recherche sur des thématiques innovantes, en m'intégrant au sein de l'équipe AI4Sim. Aussi, je transmets mes chaleureux et amicaux remerciements à Gaël Goret, Léo Nicoletti, Rémi Druilhe et Mikaël Jacquemont pour leur suivi régulier et leur accompagnement sans faille tout au long de mon travail. Toute ma sympathie et mon amitié vont également à l'ensemble des personnes que j'ai eu la chance de côtoyer dans le groupe : je pense en particulier à Christophe Bovalo, Luca Marradi, Virginie Megy, Benoit Pelletier, Bohémond Couka, Lionel Vincent, Alexis Giorkallos, Victor Trappler, Hugues de Laroussilhe et Rami Salem. Je remercie aussi Exaucé Luweh Adjim Ngarti, doctorant chez AI4Sim et ami, pour sa grandeur d'âme ainsi que pour l'ensemble des discussions philosophiques et spirituelles que nous avons eues.

Il en va naturellement de même pour Maksym Shpakovych, ami ukrainien au grand cœur qui s'inscrit dans cette même lignée de personnes savantes, sages et réfléchies : j'ai eu le plaisir de partager avec Maksym de nombreuses conversations, plus intéressantes et enrichissantes les unes que les autres !

À mon sens, ce sont véritablement l'encadrement et l'environnement de travail qui sont déterminants dans la réussite de la thèse, et ce, davantage encore que le sujet de recherche lui-même. Aussi, que ce soit au Cerfacs (où j'ai consacré la majeure partie de mon temps), mais aussi à l'IRIT ou l'ANITI, j'ai pu compter sur la bonne humeur, la bienveillance et la gentillesse partagées de tous et toutes : je tiens à remercier pour cela Théo Beuzeville, Valentin Mercier, Jérémy Briant, Alexandre Dupaquis, Sadok Jerad, Olivier Goux, Hadrien Godé, El Mehdi Ettaouchi, Oussama Mouhtal, Yanfei Xiang, Romain Espœys, Rachid El Montassir, Aakash, Arun Govind Neelan, Fernando Gonzalez, Anthony Weaver, Paul Mycek, Carola Cruse, Luc Giraud, Mayeul Destouches, Luciano Drozda, Brigitte Yzel, Lydia Otero et Fabrice Fleury.

Au cours de l'année écoulée, j'ai eu le plaisir de partager mon bureau avec Théo Briquet, ami lillois au cœur sur la main et aux nombreux talents. J'ai bien volontiers troqué ma casquette de thésard pour celle de critique culinaire, afin de déguster les multiples préparations de Théo (soupes, jus de fruits, purées, sorbets, cannelés, gâteaux, etc.), faisant peut-être du bureau G12 une antichambre de *Top Chef*, qui sait ? Au-delà de ses partages à travers le dessin, la musique, la poterie, la cuisine et la pâtiserie, Théo m'a toujours témoigné une très grande sympathie et amitié. Merci, grand chef, pour cela ainsi que pour toutes les belles discussions partagées, des plus ordinaires aux plus philosophiques et spirituelles.

Je remercie aussi Marie Lacombe, colocataire et amie, pour sa gentillesse, sa bonne humeur, son entrain, et sa sympathie constante à mon égard. Je suis heureux des échanges et des moments partagés ensemble, et lui suis reconnaissant de m'avoir toujours convié aux activités et évènements auxquels elle participait ou qu'elle organisait.

J'ai également une pensée amicale pour Romain Couillet, Paul Novello, Maxime Dalery, Sayan Saha et Miranda Boutillier.

Enfin, ces remerciements ne sauraient se passer de l'expression de ma profonde et infinie gratitude envers ma famille, toute ma famille, aimante et toujours à l'écoute. Je remercie tout particulièrement mes parents, Pascal et Véronique, ainsi que mon frère Timothé pour leur précieuse et inestimable présence à mes côtés.

Il est également celles et ceux qui m'ont vu commencer le doctorat, et qui se sont envolés vers une autre vie avant que je ne puisse écrire ces mots. Je porte en moi les semences de leur existence d'ici-bas, et souhaite les voir un jour éclore à l'aube d'un temps nouveau.

Contents

List of Tables

List of Figures

List of Algorithms

CHAPTER 1

Introduction and motivation

This thesis, which sits at the crossroads of data assimilation (DA) and deep learning (DL), introduces latent space data assimilation, a novel data-driven framework that significantly reduces computational costs and memory requirements, while also offering the potential for more accurate data assimilation results.

Operational data assimilation deals with high-dimensional, nonlinear, multi-scale and potentially unstable and chaotic dynamics, making computations exceedingly demanding and often limiting the accuracy of forecasts. Additionally, various components of the data assimilation process are typically only partially known, introducing potential sources of error. For example, covariance matrices may be imprecise, small-scale physical processes within the dynamics of interest might be unaccounted for (necessitating *ad-hoc* parameterizations), and physical models are very likely to contain inaccuracies.

Meanwhile, the rapid advancements in sensor technology and data acquisition methods - such as Earth observations (Kuenzer *et al.*, 2014; McNally *et al.*, 2014) - have positioned deep learning as an effective strategy across a wide range of applications, including data assimilation (Reichstein *et al.*, 2019; Geer, 2021; Cheng *et al.*, 2023). The core strength of neural networks lies in their ability to: (i) extract meaningful information from large raw datasets, (ii) derive inferential rules, and (iii) deliver fast, real-time predictions that markedly outpace traditional computational methods.

As a result, hybridizing data assimilation with deep learning has emerged as a promising approach to achieving faster and more accurate assimilation processes, as highlighted in recent literature (Cheng *et al.*, 2023).

Abarbanel *et al.* (2018) and Geer (2021) review equivalences and similarities between deep learning and data assimilation, advocating for their coupling and hybridization. For example, both DL and DA solve inverse problems and rely on gradient descent techniques; the cost function in variational DA is analogous to the loss function in training neural networks; and the adjoint method is mathematically identical to backpropagation (Hsieh and Tang, 1998).

There are numerous ways to integrate deep learning into data assimilation algorithms, each targeting different objectives. For instance, to achieve real-time forecasts, the numerical solvers of physical equations can be replaced with data-driven surrogate models, as demonstrated by Loh *et al.* (2018); Tang *et al.* (2020, 2021b,a). Additionally, integrating deep learning into data assimilation algorithms has proven effective for model error correction (Laloyaux *et al.*, 2020; Bonavita and Laloyaux, 2020; Brajard *et al.*, 2020; Wikner *et al.*, 2021; Farchi *et al.*, 2021b), parameter estimation (Pawar and San, 2021; Legler and Janjić, 2022), background covariance matrix estimation (Leutbecher, 2019; Chattopadhyay *et al.*, 2023), enforcing physical consistency in the analysis (Ruckstuhl *et al.*, 2021), and improving observational knowledge (Lin *et al.*, 2019; Deng *et al.*, 2018; Cheng *et al.*, 2024), among other applications.

While these significant advances only target specific components of the data assimilation process, we extend the approach further by rethinking the assimilation process itself through a data-driven framework. Our approach aligns with reduced-space methods, which solve the assimilation problem by performing computations within a lower-dimensional space relative. These reduced-space methods have been developed primarily for operational use, as most data assimilation algorithms, when implemented in their full theoretically form, are prohibitively computationally costly. Reduced-space methods include ensemble data assimilation algorithms (Evensen, 1994; Burgers *et al.*, 1998; Evensen, 2009a; Bishop *et al.*, 2001; Hunt *et al.*, 2007) for sequential approaches; and methods like the Physical Statistical Analysis System (PSAS) (Courtier, 2007), Restricted Preconditioned Conjugate Gradient (RPCG) (Gratton and Tshimanga, 2009; Gratton *et al.*, 2013), and balanced truncation (Farrell and Ioannou, 2001; Lawless *et al.*, 2008) in variational data assimilation. PCA-based methods have also been employed, as shown by Robert *et al.* (2005); Cao *et al.* (2007). Despite being operationally feasible, these approaches remain computationally intensive, and their accuracy may be limited, leaving room for further improvements.

Reduced-space data assimilation is underpinned by Lyapunov stability theory, which mathematically demonstrates that, under specific hypotheses, the forecast and posterior error covariance matrices asymptotically conform to the unstable-neutral subspace (Carrassi *et al.*, 2022), which is of much smaller dimension than the full state space. Even when the assumptions required for these mathematical proofs are relaxed, numerical evidence tends to confirm that the theoretical results still hold. This suggests that only a portion of the full state space information is necessary to accurately correct the prior estimate. Moreover, Principal Component Analysis (PCA) and, more recently, autoencoders have shown that large physical systems can be accurately represented within a lower-dimensional space. Since autoencoders are trained to construct a manifold that the state trajectory statistically adheres to, corrections made in latent data assimilation are likely to target the most error-sensitive directions. While full-space data assimilation involves linear combinations within a high-dimensional, nonlinear, and possibly multi-scale dynamic environment, latent data assimilation, which operates on the core, potentially disentangled and simplified dynamics, is more likely to result in impactful corrections. Furthermore, classical data assimilation faces an inherent mathematical limitation in that it relies on linear computations. Latent data assimilation, however, offers a way to overcome this limitation by performing the assimilation directly within the meaningful underlying structures of the data, obtained through nonlinear transformations.

The outline of the thesis is as follows. Chapter 2 introduces the fundamental concepts of data assimilation and provides the necessary background to understand the principles and objectives of our latent ETKF-Q algorithm. We begin with foundational methods in data assimilation, namely BLUE and 3D-Var algorithms, and then extend these to incorporate the time dimension, covering both variational and sequential approaches. We place particular emphasis on ensemble methods and their limitations, and specifically discuss the ETKF-Q algorithm developed by Fillion *et al.* (2020), which we selected and implemented for our latent space data assimilation approach.

Chapter 3 offers a literature review that focuses on the growing intersection and hybridization of deep learning and data assimilation. We begin by outlining the historical development of artificial intelligence and provide an in-depth overview of deep learning and common neural network architectures. This foundational background is crucial for understanding the deep learning concepts discussed later in the literature review, as well as in our work presented in chapter 4 and chapter 5. This chapter, therefore, sets the stage for the broader scientific context in which our research is situated.

In chapter 4, we give a thorough presentation of our latent data assimilation methodology. We provide criteria to select the latent dimension and discuss the desired properties of the latent space with respect to PCA and Lyapunov stability considerations (which both theoretically support our approach). The chapter also outlines the training strategies developed to ensure the stability of the surrogate neural network, as well as the modifications made to adapt the standard ETKF-Q algorithm to its latent version. Additionally, we introduce an extension of the algorithm that allows model error correction when the number of ensemble members exceeds the dimensionality of the assimilation state space, a scenario not addressed by the standard latent ETKF-Q method.

In chapter 5, we test our latent data assimilation approach on a tailored 40-dimensional chaotic Lorenz96 system and on the quasi-geostrophic model of the OOPS framework(1), which is collaboratively developed by ECMWF (European Centre for Medium-Range Weather Forecasts) and Météo-France. The chapter includes numerical results and graphs that confirm the validity of our approach and highlight the potential for further improvements.

In Chapter 6, we summarize the key insights and findings presented throughout this thesis. We draw conclusions about the achievements and advancements made, but also about the limitations and challenges that remain. We therefore explore potential future directions to further enhance, develop, and understand the potential of latent data assimilation.

Contributions

The main contribution of this thesis is to propose a latent space data assimilation methodology, in which the observations remain in their original space. Importantly, this approach can be adapted to other data assimilation algorithms (Melinc and Zaplotnik, 2024). The key aspects of this framework are:

- to define a joint training strategy for the surrogate and the autoencoder, which yields better results compared to sequential training.
- to propose two different ways of enforcing the stability of the surrogate network: an iterative training strategy through a chained loss function, and the utilization of Lipschitz networks as surrogates networks.
- to provide an extension of the approach when the number of ensembles is larger than the state space dimension.

(1)https://www.ecmwf.int/en/elibrary/77561-oops-common-framework-research-and-operations

- to demonstrate significant reduction of the computational cost and memory needs, along with the potential for more accurate analyses.

Some results from this work are published in Peyron *et al.* (2021).

Context of the PhD and collaborations

This PhD research was conducted within a collaborative framework involving CERFACS (Centre Européen de Recherche et de Formation Avancée en Calcul Scientifique), Eviden (formerly Atos), ANITI (Artificial and Natural Intelligence Toulouse Institute), and IRIT (Institut de Recherche en Informatique de Toulouse). This project was made possible through the support of Eviden, which funded the PhD via the CIFRE (Convention Industrielle de Formation par la Recherche) program, governed by the French state. The thesis is the result of close collaboration with the AI4Sim team at Eviden and the Algo-Coop team at CERFACS. As part of the ANITI initiative, we also benefited from valuable insights and ideas from Pierre Boudier, a collaborator from NVIDIA.

Part of the work developed throughout this thesis has been presented at the Congrès des Jeunes Chercheuses et Chercheurs en Mathématiques Appliquées(2) at École Polytechnique in 2021, and at the ECMWF–ESA Workshop on Machine Learning for Earth Observation and Prediction(3) in Reading in 2022. I also had the opportunity to participate in the 2023 edition of the CEMRACS(4) six-week summer school program. Along with two other PhD students, I explored the topic *1-Lipschitz neural networks for error control in function approximation*(5)(6), which later contributed to the development of stable surrogate networks. This work was presented at the ALGORITMY conference(7) of 2024 in Slovakia.

On a technical note, all the codes implemented during this research are written in Python, with the neural networks and deep learning components specifically relying on the PyTorch framework.

(2)https://cjc-ma2021.github.io/

(3)https://events.ecmwf.int/event/304/

(4)Centre d'Été Mathématique de Recherche Avancée en Calcul Scientifique, Marseille, France

(5)http://smai.emath.fr/cemracs/cemracs23/projects.html

(6)https://www.math.sk/alg2024/minisymposia-abstracts/

(7)https://www.math.sk/alg2024/minisymposia-abstracts/#machine_learning

CHAPTER 2

Data assimilation methods

In this thesis, situated at the intersection of **data assimilation** (DA) and **deep learning** (DL), a thorough introduction to DA key concepts is required for a deeper understanding of the interactions between inverse problems and neural networks.

Inverse problems seek the causes of phenomena from their observed effects. Unlike forward problems that model direct causal relationships between variables, parameters, and - physical - conditions, inverse problem theory aims to mathematically trace a process back from its result to its origin. These problems can be ill-defined, making them challenging to solve without additional knowledge. Hence, prior information on the causes or on the initial state is often introduced to regularize the problem.

Data assimilation, which will be developed throughout the following sections, belongs to the wide class of inverse problems. It involves using both an initial guess of a physical system's state and observation data, each with their uncertainties, to find an optimal estimate. This estimate is designed to be more accurate and reliable than either the initial guess or the observations alone. The chapter will later introduce a precise mathematical definition for the optimality criterion that this estimate must meet.

Within the framework of our research, we are mostly interested in weather forecast and geophysical DA applications. Advances in various scientific fields, including mathematics, physics, and more recently computer science, have allowed us to develop increasingly precise models of physical systems like atmospheric motions, oceanographic dynamics, or reservoir modeling.

To obtain precise estimates and reliable forecasts of the state of a physical system, three complementary conditions must be satisfied in practice.

First, accurately representing and forecasting a natural phenomenon lies primarily in the thorough understanding of the underlying physical laws. Thus, the integration of increasingly complex mathematical theories, concepts, and tools, alongside more precise observational instruments, has enabled us to better comprehend and model the world we live in.

A second critical requirement is a comprehensive knowledge of the system's initial state. While for simple systems, such as a gravity pendulum or free-falling objects, determining the initial state can be straightforward, it becomes significantly more challenging for complex, real-world physical processes, especially large-scale ones as encountered in weather forecasting, oceanography, or geosciences.

Lastly, once the first two conditions are met, either exactly or approximately, computational resources become necessary to carry out the calculations. For centuries, these were done manually or with the help of tools like slide rules, but the advent of computers has unlocked unprecedented computational power, opening up new and formidable opportunities for scientific exploration and modeling.

In real-world large-scale applications, not all these three conditions are precisely met, leading to errors. Also, many physical systems are chaotic, meaning even very tiny errors in our knowledge of the system's state will lead to significantly different trajectories after a finite number of time steps. Weather forecasting is a prime example of such chaotic systems, illustrated by Edward Lorenz's famous metaphorical **butterfly effect**:

Does the flap of a butterfly's wings in Brazil set off a tornado in Texas?

In this context, data assimilation has emerged as a key solution to complex challenges like weather forecasting, by merging knowledge-based and observational information together. The substantial growth in observational data and advancements in algorithms over recent decades have significantly enhanced the reliability and accuracy of forecasts.

The chapter is organized as follows:

- section 2.1 introduces the **BLUE** and **3D-Var** methods, which represent an accessible entry point into data assimilation, while underpinning the foundational principles of data assimilation.

- section 2.2 expands prior algorithms to include a temporal dimension, by using variational and sequential approaches.

While our research contributions are not tied to a specific data assimilation algorithm, we focus particularly on ensemble data assimilation, reflecting our main area of research interest.

Notably, the following subsections are significantly grounded on the remarkable and lucid lecture notes by Bocquet and Farchi (2014) and, to a lesser extent, to the thorough, well-articulated book “Data Assimilation” by Evensen (2009a). These works were chosen for their clarity, depth, and broad coverage of data assimilation topics, providing a solid foundation for the discussions in the subsequent sections.

2.1 Analysis state estimation approaches in a fixed time framework

Let us consider a physical phenomenon within a continuous spatial domain Ω, where the variable $\boldsymbol{x}^t$ denotes the **ground truth state** of the system across Ω. That is, for every point $\boldsymbol{u}$ within Ω, $\boldsymbol{x}^t(\boldsymbol{u})$ is the system's actual value at $\boldsymbol{u}$. In practice, numerical computations require to introduce a grid or a mesh, leading to considering the projection of $\boldsymbol{x}^t$ onto the discretized domain. In real-world applications, direct access to the ground truth state $\boldsymbol{x}^t$ is typically not possible. However, one often has access to a **prior information**, $\boldsymbol{x}^b \in \mathbb{R}^n$, referred to as the **background** and often provided by an expert or a numerical model, and to a vector of **observations**, $\boldsymbol{y} \in \mathbb{R}^p$, along with their associated uncertainties, $\boldsymbol{\eta}$ and $\boldsymbol{\varepsilon}$, respectively. Given these notations, we can mathematically define the two following equations:

$$\begin{cases} \boldsymbol{x}^b = \boldsymbol{x}^t + \boldsymbol{\eta}, & \text{(2.1a)} \\ \boldsymbol{y} \;= \boldsymbol{\mathcal{H}}(\boldsymbol{x}^t) + \boldsymbol{\varepsilon}, & \text{(2.1b)} \end{cases}$$

where $\boldsymbol{\eta} \in \mathbb{R}^n$, $\boldsymbol{\varepsilon} \in \mathbb{R}^p$ are random variables named **background error** and **observation error**, respectively. $\boldsymbol{\mathcal{H}}\colon \mathbb{R}^n \mapsto \mathbb{R}^p$ denotes the observation operator, and may include interpolations and/or unit transformations.

The background error $\boldsymbol{\eta}$ represents the mismatch between the prior guess $\boldsymbol{x}^b$ and the ground truth value $\boldsymbol{x}^t$. Likewise, $\boldsymbol{\varepsilon}$ accounts for all the errors and uncertainties related to both the observations $\boldsymbol{y}$ and the observation operator $\boldsymbol{\mathcal{H}}$. These errors can be divided into two categories:

- **measurements errors**: errors and uncertainties related to the data acquisition process. Instrumental devices have a limited and finite accuracy so that the confidence and the reliability of observations vary from one measurement to another.
- **representativeness errors**: when mapping from state $\boldsymbol{x}^t$ to the associated vector of observations $\boldsymbol{y}$, the operator $\boldsymbol{\mathcal{H}}$ inherently holds errors and uncertainties. Indeed, state variables and observations locations are unlikely to coincide exactly, which requires interpolations, therefore leading to approximations and uncertainties. Besides, we do not necessarily have direct measurements of the physical quantities of interest, *i.e.*, $\boldsymbol{x}$ variables, but sometimes the observed quantities have a different physical nature. That is why, operator $\boldsymbol{\mathcal{H}}$ also often hides complex physical and mathematical transformations that can be inexact, or error prone.

Within the data assimilation framework, it is conventionally assumed that both background and observation errors are additive. This assumption, while arbitrary, is primarily guided by mathematical convenience. Alternatives, such as multiplicative errors, could also be considered if they were deemed relevant.

Operational weather forecasting systems typically involve state variables with dimensions n that can reach up to 10^6 to 10^9, while p is generally about a hundred times smaller.

Given the formulation in equations (2.1a) and (2.1b), the aim of data assimilation is to derive an "optimal" estimate $\boldsymbol{x}^a$, from the background $\boldsymbol{x}^b$ and the observations $\boldsymbol{y}$, with the optimality criterion to be defined in subsequent discussions. The estimate $\boldsymbol{x}^a$ is often referred to as the **analysis state** (hence the upper-script "a").

To more precisely quantify uncertainties, we introduce the covariance matrices associated with $\boldsymbol{\eta}$ and $\boldsymbol{\varepsilon}$: the **background error covariance matrix** denoted by $\boldsymbol{B} \in \mathbb{R}^{n\times n}$, and the **observation error covariance matrix** represented by $\boldsymbol{R} \in \mathbb{R}^{p\times p}$ as:

- $[\boldsymbol{B}]_{ij} = \mathbb{E}\left[([\boldsymbol{\eta}]_i - \mathbb{E}[\boldsymbol{\eta}])([\boldsymbol{\eta}]_j - \mathbb{E}[\boldsymbol{\eta}])\right], \; \forall\, i,j \in [\![1,n]\!]$.
- $[\boldsymbol{R}]_{ij} = \mathbb{E}\left[([\boldsymbol{\varepsilon}]_i - \mathbb{E}[\boldsymbol{\varepsilon}])([\boldsymbol{\varepsilon}]_j - \mathbb{E}[\boldsymbol{\varepsilon}])\right], \; \forall\, i,j \in [\![1,p]\!]$.

where $\mathbb{E}$ denotes the **expectation operator**(1), notation $[\cdot]_{ij}$ indicates the element located in the i^{th} row and j^{th} column of a matrix, and $[\cdot]_i$ specifies the i^{th} element of a vector.

By definition, $\boldsymbol{B}$ and $\boldsymbol{R}$ are symmetric matrices. In addition, they are assumed to be **positive definite**, *i.e.*, they satisfy:

$$\forall\, \widetilde{\boldsymbol{x}} \in \mathbb{R}^n \backslash \{0\},\; \widetilde{\boldsymbol{x}}^T \boldsymbol{B} \widetilde{\boldsymbol{x}} > 0. \tag{2.3}$$

$$\forall\, \widetilde{\boldsymbol{y}} \in \mathbb{R}^p \backslash \{0\},\; \widetilde{\boldsymbol{y}}^T \boldsymbol{B} \widetilde{\boldsymbol{y}} > 0. \tag{2.4}$$

Within a data assimilation framework, we also conventionally make the following assumptions:

- background error $\boldsymbol{\eta}$ is unbiased, *i.e.*, $\mathbb{E}[\boldsymbol{\eta}] = 0$.
  Therefore, $[\boldsymbol{B}]_{ij} = \mathbb{E}\left[[\boldsymbol{\eta}]_i [\boldsymbol{\eta}]_j\right], \; \forall\, i,j \in [\![1,n]\!]$.
- observation error $\boldsymbol{\varepsilon}$ is unbiased, *i.e.*, $\mathbb{E}[\boldsymbol{\varepsilon}] = 0$.
  Therefore, $[\boldsymbol{R}]_{ij} = \mathbb{E}\left[[\boldsymbol{\varepsilon}]_i [\boldsymbol{\varepsilon}]_j\right], \; \forall\, i,j \in [\![1,p]\!]$.
- $\boldsymbol{\eta}$ and $\boldsymbol{\varepsilon}$ are uncorrelated, *i.e.*, $\mathbb{E}\left[\boldsymbol{\eta}\boldsymbol{\varepsilon}^T\right] = 0_{\mathbb{R}^{n\times p}}$.

In practice, not satisfying $\mathbb{E}[\boldsymbol{\eta}] = 0$ or $\mathbb{E}[\boldsymbol{\varepsilon}] = 0$ is not an insurmountable limitation, as it can be seamlessly circumvented when the bias is known or quantifiable. If $\boldsymbol{\eta}$ or $\boldsymbol{\varepsilon}$ is indeed biased, we can always subtract this bias and thereby define a new unbiased variable that could be used instead.

(1)
Definition 2.1 ***Expectation operator*** $\mathbb{E}$*: Let us consider a random variable Φ defined over the continuous domain Ω, and following the probability density function (or distribution) g. The expectation operator, also known as expectancy or expected value, is defined as:*

$$\mathbb{E}[\Phi] = \int_{\phi\in\Omega} \phi g(\phi) d\phi. \tag{2.2}$$

*Random variable Φ is said to be **unbiased** when $\mathbb{E}[\Phi] = 0$.*

2.1.1 Best Linear Unbiased Estimator (BLUE)

In the context of the so-called **Best Linear Unbiased Estimator**, we additionally assume the observation operator $\boldsymbol{\mathcal{H}}$ to be linear and therefore denote it by $\boldsymbol{H}$. Equations (2.1a) and (2.1b) can therefore be reformulated as:

$$\begin{cases} \boldsymbol{x}^b = \boldsymbol{x}^t + \boldsymbol{\eta}, & (2.5a) \\ \boldsymbol{y} \ = \boldsymbol{H}\boldsymbol{x}^t + \boldsymbol{\varepsilon}, & (2.5b) \end{cases}$$

As suggested by the terminology *Best Linear Unbiased Estimator*, we search for an estimate $\boldsymbol{x}^a$ in the form of a linear combination of the background $\boldsymbol{x}^b$ and the observations $\boldsymbol{y}$. Therefore, let us introduce the unknown matrices $\boldsymbol{L} \in \mathbb{R}^{n\times n}$ and $\boldsymbol{K} \in \mathbb{R}^{p\times n}$ and define $\boldsymbol{x}^a$ as follows:

$$\boxed{\boldsymbol{x}^a = \boldsymbol{L}\boldsymbol{x}^b + \boldsymbol{K}\boldsymbol{y}.} \tag{2.6}$$

We want the analysis error, denoted by $\boldsymbol{e}^a$, to be unbiased. Let us first express $\boldsymbol{e}^a$ as a function of the variables that are known to be unbiased:

$$\begin{aligned} \boldsymbol{e}^a &= \boldsymbol{x}^a - \boldsymbol{x}^t \\ &= \boldsymbol{L}\boldsymbol{x}^b + \boldsymbol{K}\boldsymbol{y} - \boldsymbol{x}^t \\ &= \boldsymbol{L}\big(\boldsymbol{x}^b - \boldsymbol{x}^t + \boldsymbol{x}^t\big) + \boldsymbol{K}\big(\boldsymbol{H}\boldsymbol{x}^t + \boldsymbol{\varepsilon}\big) - \boldsymbol{x}^t \\ &= \boldsymbol{L}\underbrace{\big(\boldsymbol{x}^b - \boldsymbol{x}^t\big)}_{=\boldsymbol{\eta}} + \boldsymbol{L}\boldsymbol{x}^t + \boldsymbol{K}\boldsymbol{H}\boldsymbol{x}^t + \boldsymbol{K}\boldsymbol{\varepsilon} - \boldsymbol{x}^t \\ &= \boldsymbol{L}\boldsymbol{\eta} + (\boldsymbol{L} + \boldsymbol{K}\boldsymbol{H} - \boldsymbol{I}_n)\boldsymbol{x}^t + \boldsymbol{K}\boldsymbol{\varepsilon}. \end{aligned} \tag{2.7}$$

Applying the expectation operator $\mathbb{E}[\cdot]$, and utilizing the assumption that $\boldsymbol{\eta}$ and $\boldsymbol{\varepsilon}$ are unbiased (*i.e.*, their expected values are zero), we obtain:

$$\begin{aligned} \mathbb{E}\left[\boldsymbol{e}^a\right] &= \boldsymbol{L}\underbrace{\mathbb{E}\left[\boldsymbol{\eta}\right]}_{=0_{\mathbb{R}^n}} + (\boldsymbol{L} + \boldsymbol{K}\boldsymbol{H} - \boldsymbol{I}_n)\mathbb{E}\left[\boldsymbol{x}^t\right] + \boldsymbol{K}\underbrace{\mathbb{E}\left[\boldsymbol{\varepsilon}\right]}_{=0_{\mathbb{R}^p}} \\ &= (\boldsymbol{L} + \boldsymbol{K}\boldsymbol{H} - \boldsymbol{I}_n)\boldsymbol{x}^t. \end{aligned} \tag{2.8}$$

To enforce $\boldsymbol{e}^a$ to be unbiased, a sufficient condition consists in choosing $\boldsymbol{L}$ and

$\boldsymbol{K}$ such that:

$$\boldsymbol{L} + \boldsymbol{K}\boldsymbol{H} - \boldsymbol{I}_n = 0_{\mathbb{R}^n},$$
$$\boxed{\boldsymbol{L} = \boldsymbol{I}_n - \boldsymbol{K}\boldsymbol{H}.} \tag{2.9}$$

Thus, by ensuring the linearity and unbiasedness criteria, our analysis estimate of equation (2.6) becomes:

$$\boldsymbol{x}^a = (\boldsymbol{I}_n - \boldsymbol{K}\boldsymbol{H})\boldsymbol{x}^b + \boldsymbol{K}\boldsymbol{y},$$
$$\boxed{\boldsymbol{x}^a = \boldsymbol{x}^b + \boldsymbol{K}\left(\boldsymbol{y} - \boldsymbol{H}\boldsymbol{x}^b\right),} \tag{2.10}$$

where $\left(\boldsymbol{y} - \boldsymbol{H}\boldsymbol{x}^b\right)$ is called the **innovation** vector and denotes the misfit between the observations and the background. Matrix $\boldsymbol{K} \in \mathbb{R}^{n\times p}$, known as a **gain** matrix, filters which information from the innovation is passed to correct the background.

Let us denote by $\boldsymbol{P}^a$ the posterior error covariance matrix associated with the analysis $\boldsymbol{x}_a$. The optimality criterion leading to the BLUE analysis consists in minimizing the trace of $\boldsymbol{P}^a$ with respect to $\boldsymbol{K}$. The resulting optimal gain will be denoted by $\boldsymbol{K}^*$. First, we therefore consider the expression of $\boldsymbol{P}^a$:

$$\boxed{\boldsymbol{P}^a = \mathbb{E}\left[\boldsymbol{e}^a(\boldsymbol{e}^a)^T\right].} \tag{2.11}$$

We can now derive $\boldsymbol{e}^a$ from equation (2.7) by substituting $\boldsymbol{L}$ with its known expression (*i.e.*, $\boldsymbol{I}_n - \boldsymbol{K}\boldsymbol{H}$):

$$\begin{aligned}
\boldsymbol{e}^a &= \boldsymbol{L}\boldsymbol{\eta} + (\boldsymbol{L} + \boldsymbol{K}\boldsymbol{H} - \boldsymbol{I}_n)\boldsymbol{x}^t + \boldsymbol{K}\boldsymbol{\varepsilon} \\
&= (\boldsymbol{I}_n - \boldsymbol{K}\boldsymbol{H})\boldsymbol{\eta} + \underbrace{(\boldsymbol{I}_n - \boldsymbol{K}\boldsymbol{H} + \boldsymbol{K}\boldsymbol{H} - \boldsymbol{I}_n)}_{=0_{\mathbb{R}^{n\times n}}}\boldsymbol{x}^t + \boldsymbol{K}\boldsymbol{\varepsilon} \\
&= \boldsymbol{\eta} - \boldsymbol{K}\boldsymbol{H}\boldsymbol{\eta} + \boldsymbol{K}\boldsymbol{\varepsilon} \\
&= \boldsymbol{\eta} + \boldsymbol{K}(\boldsymbol{\varepsilon} - \boldsymbol{H}\boldsymbol{\eta}).
\end{aligned} \tag{2.12}$$

Replacing $\boldsymbol{e}^a$ in equation (2.11) with its new formulation given by equation (2.12), we obtain:

$$\begin{aligned}
\boldsymbol{P}^a &= \mathbb{E}\left[(\boldsymbol{\eta} + \boldsymbol{K}(\boldsymbol{\varepsilon} - \boldsymbol{H}\boldsymbol{\eta}))(\boldsymbol{\eta} + \boldsymbol{K}(\boldsymbol{\varepsilon} - \boldsymbol{H}\boldsymbol{\eta}))^T\right] \\
&= \mathbb{E}\left[(\underbrace{(\boldsymbol{I}_n - \boldsymbol{K}\boldsymbol{H})}_{=\boldsymbol{L}}\boldsymbol{\eta} + \boldsymbol{K}\boldsymbol{\varepsilon})(\underbrace{(\boldsymbol{I}_n - \boldsymbol{K}\boldsymbol{H})}_{=\boldsymbol{L}}\boldsymbol{\eta} + \boldsymbol{K}\boldsymbol{\varepsilon})^T\right] \\
&= \mathbb{E}\left[(\boldsymbol{L}\boldsymbol{\eta} + \boldsymbol{K}\boldsymbol{\varepsilon})(\boldsymbol{L}\boldsymbol{\eta} + \boldsymbol{K}\boldsymbol{\varepsilon})^T\right] \\
&= \mathbb{E}\left[(\boldsymbol{L}\boldsymbol{\eta} + \boldsymbol{K}\boldsymbol{\varepsilon})\left(\boldsymbol{\eta}^T\boldsymbol{L}^T + \boldsymbol{\varepsilon}^T\boldsymbol{K}^T\right)\right] \\
&= \boldsymbol{L}\underbrace{\mathbb{E}\left[\boldsymbol{\eta}\boldsymbol{\eta}^T\right]}_{=\boldsymbol{B}}\boldsymbol{L}^T + \boldsymbol{L}\underbrace{\mathbb{E}\left[\boldsymbol{\eta}\boldsymbol{\varepsilon}^T\right]}_{=0_{\mathbb{R}^{n\times p}}}\boldsymbol{K}^T + \boldsymbol{K}\underbrace{\mathbb{E}\left[\boldsymbol{\varepsilon}\boldsymbol{\eta}^T\right]}_{=0_{\mathbb{R}^{p\times n}}}\boldsymbol{L}^T + \boldsymbol{K}\underbrace{\mathbb{E}\left[\boldsymbol{\varepsilon}\boldsymbol{\varepsilon}^T\right]}_{=\boldsymbol{R}}\boldsymbol{K}^T \\
&= \boldsymbol{L}\boldsymbol{B}\boldsymbol{L}^T + \boldsymbol{K}\boldsymbol{R}\boldsymbol{K}^T \\
&\boxed{= (\boldsymbol{I}_n - \boldsymbol{K}\boldsymbol{H})\boldsymbol{B}(\boldsymbol{I}_n - \boldsymbol{K}\boldsymbol{H})^T + \boldsymbol{K}\boldsymbol{R}\boldsymbol{K}^T.} \qquad (2.13)
\end{aligned}$$

Given equation (2.13), we can minimize $Tr(\boldsymbol{P}^a)$ with respect to $\boldsymbol{K}$, where $Tr(\cdot)$ denotes the trace operator and represents the sum of the variances (diagonal elements) of the error between the analysis state $\boldsymbol{x}^a$ and the ground truth state $\boldsymbol{x}^t$. We can mathematically derive the following expression for the optimal gain $\boldsymbol{K}^*$:

$$\boxed{\boldsymbol{K}^* = \boldsymbol{B}\boldsymbol{H}^T\left(\boldsymbol{R} + \boldsymbol{H}\boldsymbol{B}\boldsymbol{H}^T\right)^{-1}.} \qquad (2.14)$$

A thorough mathematical derivation of this solution is provided at the end of the manuscript, in appendix A.1.

When expressing the gain $\boldsymbol{K}^*$ as in equation (2.14), we notice that a p-by-p matrix has to be inverted. Relying on the **Sherman-Morrison-Woodbury** formula, it is possible to define an equivalent formulation of $\boldsymbol{K}^*$ for which a n-by-n matrix needs to be inverted. We show this equivalence in appendix A.2 and thereby derive the following second expression for $\boldsymbol{K}^*$:

$$\boxed{\boldsymbol{K}^* = \left(\boldsymbol{B}^{-1} + \boldsymbol{H}^T\boldsymbol{R}^{-1}\boldsymbol{H}\right)^{-1}\boldsymbol{H}^T\boldsymbol{R}^{-1}.} \qquad (2.15)$$

We thus have two different formulations for the optimal gain, $\boldsymbol{K}^*$, each offering practical advantages depending on the specific context of application. Besides, as shown in equation (2.13), the posterior error covariance matrix $\boldsymbol{P}^a$ depends on $\boldsymbol{K}$. According to the considered formula of $\boldsymbol{K}^*$, we can therefore derive two distinct expressions for $\boldsymbol{P}^a$, as demonstrated in appendix A.3:

$$\boxed{\boldsymbol{P}^a = \boldsymbol{B} - \boldsymbol{B}\boldsymbol{H}^T\left(\boldsymbol{R} + \boldsymbol{H}\boldsymbol{B}\boldsymbol{H}^T\right)^{-1}\boldsymbol{H}\boldsymbol{B},} \tag{2.16}$$

and,

$$\boxed{\boldsymbol{P}^a = \left(\boldsymbol{B}^{-1} + \boldsymbol{H}^T\boldsymbol{R}^{-1}\boldsymbol{H}\right)^{-1}.} \tag{2.17}$$

In summary, given the system of equations (2.5a) and (2.5b), the derivation of the Best Linear Unbiased Estimator (BLUE) yields the following formulas (derived through equations (2.14) to (2.17)):

$$\boxed{\begin{aligned}
\boldsymbol{x}^a &= \boldsymbol{x}^b + \boldsymbol{B}\boldsymbol{H}^T\left(\boldsymbol{R} + \boldsymbol{H}\boldsymbol{B}\boldsymbol{H}^T\right)^{-1}(\boldsymbol{y} - \boldsymbol{H}\boldsymbol{x}^b), && (2.18a)\\
&= \boldsymbol{x}^b + \left(\boldsymbol{B}^{-1} + \boldsymbol{H}^T\boldsymbol{R}^{-1}\boldsymbol{H}\right)^{-1}\boldsymbol{H}^T\boldsymbol{R}^{-1}(\boldsymbol{y} - \boldsymbol{H}\boldsymbol{x}^b). && (2.18b)\\
\boldsymbol{P}^a &= \boldsymbol{B} - \boldsymbol{B}\boldsymbol{H}^T\left(\boldsymbol{R} + \boldsymbol{H}\boldsymbol{B}\boldsymbol{H}^T\right)^{-1}\boldsymbol{H}\boldsymbol{B}, && (2.18c)\\
&= \left(\boldsymbol{B}^{-1} + \boldsymbol{H}^T\boldsymbol{R}^{-1}\boldsymbol{H}\right)^{-1}. && (2.18d)
\end{aligned}}$$

Within the BLUE framework, these equations provide a systematic method to integrate observations with prior information to produce an updated state with minimized error variance. The gain $\boldsymbol{K}^*$, in particular, optimally balances the trust placed in the background versus the observed data, a principle that is foundational to data assimilation methods. As illustrated throughout this section, BLUE analysis therefore stands as an accessible entry point into the theory of data assimilation. However, its reliance on the linearity assumption marks a significant limitation to practical applications: $\boldsymbol{\mathcal{H}}$ is indeed seldom linear in practice. Exploring new data assimilation methods that accommodate nonlinear observation operators is thus essential.

2.1.2 3D-Var

Shifting from BLUE analysis to an optimization-based framework, allows us to revisit our data assimilation problem. When the observation operator is no longer linear, we can indeed search for an estimate $\boldsymbol{x}^a$ by solving the following optimization problem:

$$\boxed{\boldsymbol{x}^a = \underset{\boldsymbol{x}\in\mathbb{R}^n}{\arg\min} \begin{cases} J: & \mathbb{R}^n \to \mathbb{R}^+ \\ & \boldsymbol{x} \mapsto \frac{1}{2}\left\|\boldsymbol{x}-\boldsymbol{x}^b\right\|^2_{\boldsymbol{B}^{-1}} + \frac{1}{2}\|\boldsymbol{y}-\boldsymbol{\mathcal{H}}(\boldsymbol{x})\|^2_{\boldsymbol{R}^{-1}}, \end{cases}} \tag{2.19}$$

where norm operators $\|\cdot\|_{\boldsymbol{B}^{-1}}$ and $\|\cdot\|_{\boldsymbol{R}^{-1}}$ are defined as follows:

$\forall(\boldsymbol{x},\boldsymbol{y}) \in (\mathbb{R}^n \times \mathbb{R}^p)$:

$$\|\boldsymbol{x}\|_{\boldsymbol{B}^{-1}} = \sqrt{\boldsymbol{x}^T\boldsymbol{B}^{-1}\boldsymbol{x}}. \tag{2.20}$$

$$\|\boldsymbol{y}\|_{\boldsymbol{R}^{-1}} = \sqrt{\boldsymbol{y}^T\boldsymbol{R}^{-1}\boldsymbol{y}}. \tag{2.21}$$

Notably, $\|\cdot\|_{\boldsymbol{B}^{-1}}$ and $\|\cdot\|_{\boldsymbol{R}^{-1}}$ are well-defined since $\boldsymbol{B}^{-1}$ and $\boldsymbol{R}^{-1}$ are positive definite.

The minimization problem defined in equation (2.19) is also known as the **three-dimensional variational (3D-Var)** optimization problem (Courtier *et al.*, 1998; Rabier *et al.*, 1998a; Andersson *et al.*, 1998). The term 3D-Var signifies variational data assimilation within a three-dimensional spatial domain, excluding temporal dynamics, unlike the so-called **4D-Var** (see section 2.2.1) which incorporates time. The designation "3D" does not imply that the state vector $\mathbb{R}^n$ is three-dimensional, but rather refers to the spatial dimensionality of the physical variables being assimilated (e.g., wind speed, temperature, pressure, humidity) across a three-dimensional grid or mesh.

Solving the unconstrained optimization problem of equation (2.19) requires the utilization of iterative optimization methods (Nocedal and Wright, 2006). The fundamental strategy involves identifying a descent direction, denoted by $\boldsymbol{p}^{(j)}$, and determining an appropriate step size, $\alpha^{(j)} \in \mathbb{R}^+$, at each iteration j. This iterative process is mathematically expressed as:

$$\boxed{\boldsymbol{x}^{(j+1)} = \boldsymbol{x}^{(j)} + \alpha^{(j)}\boldsymbol{p}^{(j)}.} \tag{2.22}$$

For **nonlinear least squares problems** (NLSPs) as in equation (2.19), it is common to apply the Gauss-Newton method (Gratton *et al.*, 2007; Nocedal and Wright, 2006): Gauss-Newton algorithm is an iterative procedure, that solves a sequence of linear least squares problems (LLSPs). At each step of the outer loop, the observation operator $\boldsymbol{\mathcal{H}}$ is linearized at the current iterate. Minimizing the linear least squares problem leads to finding the solution of an equation of the form $\boldsymbol{Ax} = \boldsymbol{b}$: this can be solved through direct or iterative methods, depending on $\boldsymbol{A}$ (Golub and Van Loan, 2013; Saad, 2003). For large-scale positive-definite systems, conjugate gradient methods are an efficient common choice.

When the observation operator is linear (*i.e.*, when $\boldsymbol{\mathcal{H}} = \boldsymbol{H}$), 3D-Var is equivalent to BLUE analysis, and the optimization problem reads:

$$\boxed{\boldsymbol{x}^a = \underset{\boldsymbol{x}\in\mathbb{R}^n}{\arg\min} \begin{cases} J: & \mathbb{R}^n \to \mathbb{R}^+ \\ & \boldsymbol{x} \mapsto \frac{1}{2}\left\|\boldsymbol{x} - \boldsymbol{x}^b\right\|^2_{\boldsymbol{B}^{-1}} + \frac{1}{2}\left\|\boldsymbol{y} - \boldsymbol{Hx}\right\|^2_{\boldsymbol{R}^{-1}}, \end{cases}} \tag{2.23}$$

We can mathematically solve equation (2.23) to demonstrate the equivalence between BLUE and 3D-Var in the linear case.

The cost function J of equation (2.23) is a quadratic function of the variable $\boldsymbol{x}$. Besides, J is strictly convex as the error covariance matrices $\boldsymbol{B}$ and $\boldsymbol{R}$ are positive definite. Therefore, there exists a unique $\boldsymbol{x}^a \in \mathbb{R}^n$ that minimizes J. We can mathematically derive that:

$$\boxed{\nabla J(\boldsymbol{x}) = \boldsymbol{B}^{-1}\left(\boldsymbol{x} - \boldsymbol{x}^b\right) - \boldsymbol{H}^T\boldsymbol{R}^{-1}(\boldsymbol{y} - \boldsymbol{Hx}).} \tag{2.24}$$

$$\boxed{\mathrm{Hess}_{\mathrm{J}}(\boldsymbol{x}) = \boldsymbol{B}^{-1} + \boldsymbol{H}^{\mathrm{T}}\boldsymbol{R}^{-1}\boldsymbol{H}.} \tag{2.25}$$

The minimum of the quadratic J is denoted by $\boldsymbol{x}^a$ and satisfies $\nabla J(\boldsymbol{x}^a) = 0$, which reads:

$$\begin{aligned}
& \boldsymbol{B}^{-1}\left(\boldsymbol{x}^a - \boldsymbol{x}^b\right) - \boldsymbol{H}^T\boldsymbol{R}^{-1}(\boldsymbol{y} - \boldsymbol{Hx}^a) = 0 \\
\Leftrightarrow\; & \boldsymbol{B}^{-1}\boldsymbol{x}^a - \boldsymbol{B}^{-1}\boldsymbol{x}^b - \boldsymbol{H}^T\boldsymbol{R}^{-1}\boldsymbol{y} + \boldsymbol{H}^T\boldsymbol{R}^{-1}\boldsymbol{Hx}^a = 0 \\
\Leftrightarrow\; & \left(\boldsymbol{B}^{-1} + \boldsymbol{H}^T\boldsymbol{R}^{-1}\boldsymbol{H}\right)\boldsymbol{x}^a = \boldsymbol{B}^{-1}\boldsymbol{x}^b + \boldsymbol{H}^T\boldsymbol{R}^{-1}\boldsymbol{Hy} \\
\Leftrightarrow\; & \left(\boldsymbol{B}^{-1} + \boldsymbol{H}^T\boldsymbol{R}^{-1}\boldsymbol{H}\right)\boldsymbol{x}^a = \left(\boldsymbol{B}^{-1} + \boldsymbol{H}^T\boldsymbol{R}^{-1}\boldsymbol{H} - \boldsymbol{H}^T\boldsymbol{R}^{-1}\boldsymbol{H}\right)\boldsymbol{x}^b + \boldsymbol{H}^T\boldsymbol{R}^{-1}\boldsymbol{Hy} \\
\Leftrightarrow\; & \left(\boldsymbol{B}^{-1} + \boldsymbol{H}^T\boldsymbol{R}^{-1}\boldsymbol{H}\right)\boldsymbol{x}^a = \left(\boldsymbol{B}^{-1} + \boldsymbol{H}^T\boldsymbol{R}^{-1}\boldsymbol{H}\right)\boldsymbol{x}^b + \boldsymbol{H}^T\boldsymbol{R}^{-1}\boldsymbol{H}\left(\boldsymbol{y} - \boldsymbol{Hx}^b\right)
\end{aligned}$$

$$\Leftrightarrow \boxed{\boldsymbol{x}^a = \boldsymbol{x}^b + \left(\boldsymbol{B}^{-1} + \boldsymbol{H}^T\boldsymbol{R}^{-1}\boldsymbol{H}\right)^{-1}\boldsymbol{H}^T\boldsymbol{R}^{-1}\boldsymbol{H}\left(\boldsymbol{y} - \boldsymbol{H}\boldsymbol{x}^b\right).} \tag{2.26}$$

Therefore, solving the 3D-Var optimization problem given by equation (2.23) yields the same estimate as the BLUE under similar assumptions. Notably, in the case of 3D-Var, we also have that $\boldsymbol{P}^a = \mathrm{Hess_J(x)} = \boldsymbol{B}^{-1} + \boldsymbol{H}^{\mathrm{T}}\boldsymbol{R}^{-1}\boldsymbol{H}$, $\forall \boldsymbol{x} \in \mathbb{R}^{\mathrm{n}}$.

2.2 Including temporal dimension

BLUE analysis (section 2.1.1) and 3D-Var (section 2.1.2) are two data assimilation methodologies for refining the estimation of a physical system's state by leveraging prior knowledge $\boldsymbol{x}^b$, observational data $\boldsymbol{y}$, and the error covariance matrices associated with their respective random noises. However, BLUE and 3D-Var do not explicitly account for the (possible) temporal evolution of the analysis and its covariance matrix through time, since no dynamical model is included within the DA equations (2.1a) and (2.1b).

We now consider the following system of equations:

$$\begin{cases} \boldsymbol{x}_k^t = \boldsymbol{\mathcal{M}}_k\left(\boldsymbol{x}_{k-1}^t\right) + \boldsymbol{\eta}_k, & (2.27a) \\ \boldsymbol{y}_k = \boldsymbol{\mathcal{H}}_k(\boldsymbol{x}_k^t) + \boldsymbol{\varepsilon}_k, & (2.27b) \end{cases}$$

where $\boldsymbol{\mathcal{M}}_k$ - sometimes also denoted by $\boldsymbol{\mathcal{M}}_{(k\text{-}1)\to k}$ - is a knowledge-based (and as such imperfect) model, that propagates the system's state from time t_k to time t_{k+1}. The error between the model prediction and the true state value at time t_k is denoted by $\boldsymbol{\eta}_k$. All variables are temporally indexed by k, with k lying in $[\![0, T]\!]$.

To complement equations (2.27a) and (2.27b), we commonly make the following assumptions:

- model error $\boldsymbol{\eta}_k$ is unbiased, *i.e.*, $\mathbb{E}\left[\boldsymbol{\eta}_k\right] = 0_{\mathbb{R}^n}$, $\forall k \in [\![0, T]\!]$.
- model errors $(\boldsymbol{\eta}_k)_{k\in[\![0,T]\!]}$ are uncorrelated in time.

  Therefore, $\mathbb{E}\left[\boldsymbol{\eta}_k\boldsymbol{\eta}_l^T\right] = \begin{cases} \boldsymbol{Q}_k \text{ if } k = l \\ 0_{\mathbb{R}^{n\times n}} \text{ otherwise} \end{cases}$, $\forall (k, l) \in [\![0, T]\!] \times [\![0, T]\!]$.
- observation error $\boldsymbol{\varepsilon}_k$ is unbiased, *i.e.*, $\mathbb{E}\left[\boldsymbol{\varepsilon}_k\right] = 0_{\mathbb{R}^n}$, $\forall k \in [\![0, T]\!]$.

- observation errors $(\boldsymbol{\varepsilon}_k)_{k\in[\![0,T]\!]}$ are uncorrelated in time.
  Therefore, $\mathbb{E}\left[\boldsymbol{\varepsilon}_k\boldsymbol{\varepsilon}_l^T\right] = \begin{cases} \boldsymbol{R}_k \text{ if } k=l \\ 0_{\mathbb{R}^{p\times p}} \text{ otherwise} \end{cases}, \ \forall(k,l)\in[\![0,T]\!]\times[\![0,T]\!]$.
- $(\boldsymbol{\eta}_k)_{k\in[\![0,T]\!]}$ and $(\boldsymbol{\varepsilon}_k)_{k\in[\![0,T]\!]}$ are uncorrelated.
  Therefore, $\mathbb{E}\left[\boldsymbol{\eta}_k\boldsymbol{\varepsilon}_l^T\right] = \mathbb{E}\left[\boldsymbol{\eta}_k\right]\mathbb{E}\left[\boldsymbol{\varepsilon}_k^T\right] = 0_{\mathbb{R}^{n\times p}}, \ \forall(k,l)\in[\![0,T]\!]\times[\![0,T]\!]$.

Within the framework of equations (2.27a) and (2.27b) - and under the aforementioned assumptions -, we establish the expression of an optimal estimate along with its forward error statistics, in the subsequent sections. This is done from two different lens: **variational** and **sequential** data assimilation.

2.2.1 4D-Var

Given equations (2.27a) and (2.27b), the **four-dimensional variational (4D-Var)** assimilation algorithm (Le Dimet and Talagrand, 1986; Rabier *et al.*, 1998b; Mahfouf and Rabier, 2000; Klinker *et al.*, 2000; Lorenc, 1986) expands the 3D-Var method by adding a time dimension. It is defined as the following optimization problem:

$$\boxed{\boldsymbol{x}_0^a = \underset{\boldsymbol{x}_0\in\mathbb{R}^n}{\arg\min} \begin{cases} J\colon \ \mathbb{R}^n \to \mathbb{R}^+ \\ \quad \boldsymbol{x}_0 \mapsto \frac{1}{2}\left\|\boldsymbol{x}_0-\boldsymbol{x}_0^b\right\|^2_{\boldsymbol{B}^{-1}} + \frac{1}{2}\sum_{k=0}^{\mathcal{T}}\|\boldsymbol{y}_k-\boldsymbol{\mathcal{G}}_k(\boldsymbol{x}_0)\|^2_{\boldsymbol{R}_k^{-1}}, \end{cases}} \tag{2.28}$$

where $\boldsymbol{\mathcal{G}}_k$ is called the **generalized observation operator** and is defined as follows:

$$\boldsymbol{\mathcal{G}}_k(\boldsymbol{x}_0) = \begin{cases} \boldsymbol{\mathcal{H}}_k(\boldsymbol{x}_k) = \boldsymbol{\mathcal{H}}_k\circ\boldsymbol{\mathcal{M}}_k\circ\cdots\circ\boldsymbol{\mathcal{M}}_2\circ\boldsymbol{\mathcal{M}}_1(\boldsymbol{x}_0), \ \forall k\in[\![1,T]\!] \\ \boldsymbol{\mathcal{H}}_0(\boldsymbol{x}_0) \text{ if k=0} \end{cases} \tag{2.29}$$

4D-Var operates over time windows whose length is here denoted by $\mathcal{T}$. For each time period, 4D-Var iteratively optimizes for the best initial condition $\boldsymbol{x}_0$ of the considered window: all the available observations are assimilated at once, thanks to models $\boldsymbol{\mathcal{M}}_1,\ldots,\boldsymbol{\mathcal{M}}_{\mathcal{T}}$ and observation operators $\boldsymbol{\mathcal{H}}_0,\boldsymbol{\mathcal{H}}_1,\ldots,\boldsymbol{\mathcal{H}}_{\mathcal{T}}$. 4D-var is therefore known to have a smoothing effect over the state trajectory, compared to sequential approaches.

Figure 2.2.1 illustrates the principle of the 4D-Var algorithm. For the sake of simplicity, observations $\boldsymbol{y}_k$ and state variables $\boldsymbol{x}_0^b$,$\boldsymbol{x}_k^f$ and $\boldsymbol{x}_k^a$ are depicted in the same space. From a prior guess $\boldsymbol{x}_0^b$, 4D-Var searches for an optimal estimate $\boldsymbol{x}_0^a$ so that the resulting trajectory error over the entire time window is minimized. The double-headed green arrows represent the discrepancy between the observations $\boldsymbol{y}_k$ and the corrected trajectory states $\boldsymbol{x}_k^a$. Likewise, the double-headed blue arrows denote the difference between the forecasts computed from $\boldsymbol{x}_0^b$ and the analyses.

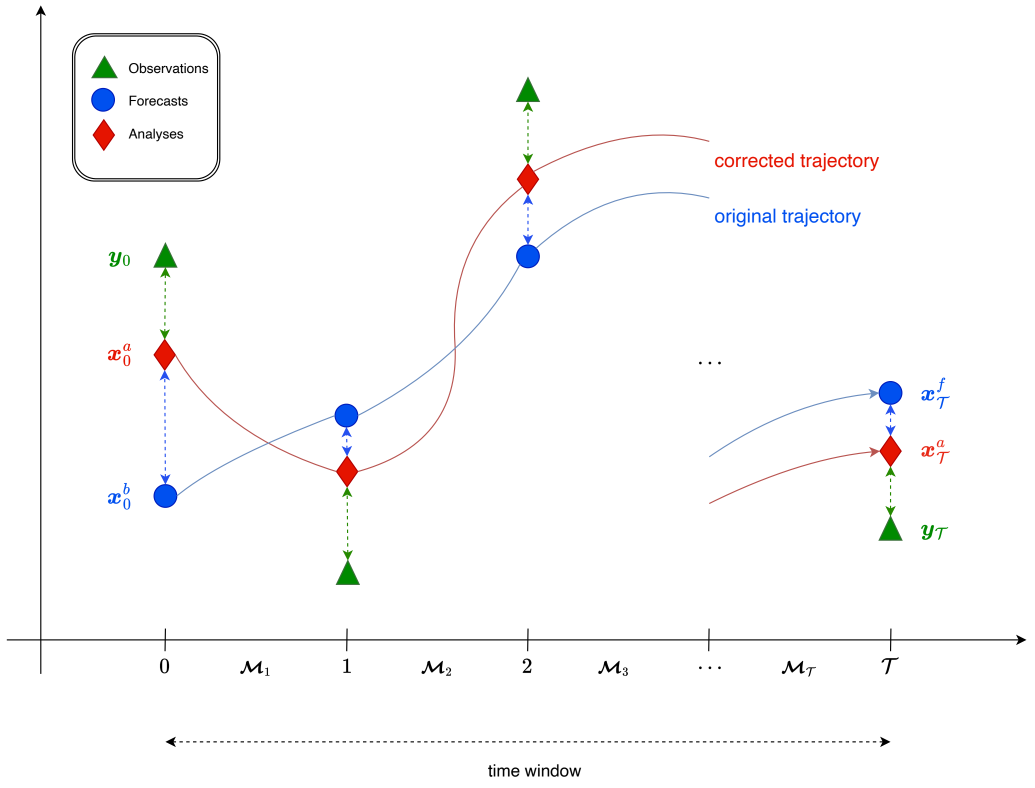


Figure 2.2.1: Illustration of 4D-Var algorithm over the i^{th} time window of the assimilation process.

In equation (2.28), we implicitly assume that models $\boldsymbol{\mathcal{M}}_1, \ldots, \boldsymbol{\mathcal{M}}_k$ are perfect: the optimization problem is therefore named **strong-constraint 4D-Var**. There exists extensions, the so-called **weak-constraint 4D-Var** (Trémolet, 2006, 2007).

One way to introduce model error is to optimize for both the initial state $\boldsymbol{x}_0$ and an estimation of a systematic bias $\boldsymbol{\eta}$, therefore defining the following optimization problem:

$$\boxed{\boldsymbol{x}_0^a, \boldsymbol{\eta}^* = \underset{\boldsymbol{x}_0, \boldsymbol{\eta} \in \mathbb{R}^n}{\arg\min} \begin{cases} J: \ \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^+ \\ \quad \boldsymbol{x}_0, \boldsymbol{\eta} \mapsto \frac{1}{2}\left\|\boldsymbol{x}_0 - \boldsymbol{x}_0^b\right\|^2_{\boldsymbol{B}^{-1}} + \frac{1}{2}\sum_{k=0}^{\mathcal{T}} \|\boldsymbol{y}_k - \boldsymbol{\mathcal{G}}_k(\boldsymbol{x}_0)\|^2_{\boldsymbol{R}_k^{-1}} + \frac{1}{2}\left\|\boldsymbol{\eta} - \boldsymbol{\eta}^b\right\|^2_{\boldsymbol{Q}^{-1}}, \end{cases}} \tag{2.30}$$

where $\boldsymbol{\eta}^b$ is a background prior of the bias, and $\boldsymbol{\mathcal{G}}_k$ is redefined as

$$\boldsymbol{\mathcal{G}}_k(\boldsymbol{x}_0) = \begin{cases} \boldsymbol{\mathcal{H}}_k(\boldsymbol{\mathcal{M}}_k(\cdots \boldsymbol{\mathcal{M}}_2(\boldsymbol{\mathcal{M}}_1(\boldsymbol{x}_0) + \boldsymbol{\eta}) + \boldsymbol{\eta}) + \cdots \boldsymbol{\eta}), \ \forall k \in [\![1, T]\!] \\ \boldsymbol{\mathcal{H}}_0(\boldsymbol{x}_0) \text{ if k=0} \end{cases} \tag{2.31}$$

This weak-constraint formulation is implemented operationally in the ECMWF's Integrated Forecasting System (IFS) (Bocquet and Farchi, 2014; Trémolet and Fisher, 2010) and accounts for a systematic bias that is assumed invariant through time.

In the following, we will be considering only the case depicted by equation (2.28). Similarly to 3D-Var, the optimization problem of equation (2.28) can be solved by using the Gauss-Newton approach (Gratton *et al.*, 2007; Nocedal and Wright, 2006). Instead of optimizing for $\boldsymbol{x}$, we rather iteratively search for an increment $\boldsymbol{\delta x} \in \mathbb{R}^n$, by solving a sequence of LLSPs. At iteration j of the outer loop, we solve the following LLSP (with $\boldsymbol{x}_0^{(0)} = \boldsymbol{x}^b$):

$$\boxed{\boldsymbol{\delta x}_0^{(j,*)} = \underset{\boldsymbol{\delta x}_0^{(j)} \in \mathbb{R}^n}{\arg\min} \begin{cases} J: \quad \mathbb{R}^n \to \mathbb{R}^+ \\ \quad \boldsymbol{\delta x}_0^{(j)} \mapsto \frac{1}{2}\left\|\boldsymbol{x}_0^{(j)} + \boldsymbol{\delta x}_0^{(j)} - \boldsymbol{x}_0^b\right\|^2_{\boldsymbol{B}^{-1}} + \frac{1}{2}\sum_{k=0}^{\mathcal{T}} \left\|\boldsymbol{y}_k - \boldsymbol{\mathcal{G}}_k\left(\boldsymbol{x}_0^{(j)}\right) - \boldsymbol{G}_k\left(\boldsymbol{x}_0^{(j)}\right)\boldsymbol{\delta x}_0^{(j)}\right\|^2_{\boldsymbol{R}_k^{-1}}, \end{cases}} \tag{2.32}$$

where $\boldsymbol{G}_k\left(\boldsymbol{x}_0^{(j)}\right) \in \mathbb{R}^{p \times n}$ is the linear approximation of operator $\boldsymbol{\mathcal{G}}_k$ at point $\boldsymbol{x}_0^{(j)}$.

Equation (2.32) is a quadratic functional with respect to $\boldsymbol{\delta x}_0^{(j)}$, its gradient is given by:

$$\begin{aligned}\nabla J\left(\boldsymbol{\delta x}_0^{(j)}\right) =& \boldsymbol{B}^{-1}\left(\boldsymbol{\delta x}_0^{(j)} + \boldsymbol{x}_0^{(j)} - \boldsymbol{x}_0^b\right) - \boldsymbol{H}_0\left(\boldsymbol{x}_0^{(j)}\right)^T \underbrace{\boldsymbol{R}_0^{-1}\left(\boldsymbol{y}_0 - \boldsymbol{\mathcal{H}}_0\left(\boldsymbol{x}_0^{(j)}\right) - \boldsymbol{H}_0\left(\boldsymbol{x}_0^{(j)}\right)\boldsymbol{\delta x}_0^{(j)}\right)}_{\Delta_0} \\ & - \boldsymbol{M}_1\left(\boldsymbol{x}_0^{(j)}\right)^T \boldsymbol{H}_1\left(\boldsymbol{x}_0^{(j)}\right)^T \underbrace{\boldsymbol{R}_1^{-1}\left(\boldsymbol{y}_1 - \boldsymbol{\mathcal{H}}_1\left(\boldsymbol{\mathcal{M}}_1\left(\boldsymbol{x}_0^{(j)}\right)\right) - \boldsymbol{H}_1\left(\boldsymbol{M}_1\left(\boldsymbol{x}_0^{(j)}\right)\right)\boldsymbol{\delta x}_0^{(j)}\right)}_{\Delta_1} \\ & - \boldsymbol{M}_1^T \boldsymbol{M}_2^T \boldsymbol{H}_2^T \underbrace{\boldsymbol{R}_2^{-1}\left(\boldsymbol{y}_2 - \boldsymbol{\mathcal{G}}_2\left(\boldsymbol{x}_0^{(j)}\right) - \boldsymbol{H}_2 \boldsymbol{M}_2 \boldsymbol{M}_1 \boldsymbol{\delta x}_0^{(j)}\right)}_{\Delta_2} \qquad (2.33) \\ & - \boldsymbol{M}_1^T \boldsymbol{M}_2^T \dots \boldsymbol{M}_{\mathcal{T}}^T \boldsymbol{H}_{\mathcal{T}}^T \underbrace{\boldsymbol{R}_{\mathcal{T}}^{-1}\left(\boldsymbol{y}_{\mathcal{T}} - \boldsymbol{\mathcal{G}}_{\mathcal{T}}\left(\boldsymbol{x}_0^{(j)}\right) - \boldsymbol{H}_{\mathcal{T}} \boldsymbol{M}_{\mathcal{T}} \dots \boldsymbol{M}_2 \boldsymbol{M}_1 \boldsymbol{\delta x}_0^{(j)}\right)}_{\Delta_{\mathcal{T}}}. \qquad (2.34)\end{aligned}$$

where $\boldsymbol{H}_k\left(\boldsymbol{x}_k^{(j)}\right)$, $\forall k \in [\![0, \mathcal{T}]\!]$ and $\boldsymbol{M}_k\left(\boldsymbol{x}_k^{(j)}\right)$, $\forall k \in [\![1, \mathcal{T}]\!]$ respectively denote the linearized observation and model operators at point $\boldsymbol{x}_k^{(j)}$ (with $\boldsymbol{x}_k^{(j)} = \boldsymbol{\mathcal{M}}\left(\boldsymbol{x}_{k-1}^{(j)}\right)$ for $k > 1$). From equation (2.33) - and in the following as well - we do not indicate the linearization point, so that they are simply denoted by $\boldsymbol{H}_k$ and $\boldsymbol{M}_k$.

By factorizing the upfront terms $\boldsymbol{H}_0^T$, $\boldsymbol{M}_1^T$, $\boldsymbol{M}_2^T$, ..., $\boldsymbol{M}_{\mathcal{T}-1}^T$, we can obtain a **Horner factorization** of $\nabla J\left(\boldsymbol{\delta x}_0^{(j)}\right)$:

$$\begin{aligned}\nabla J\left(\boldsymbol{\delta x}_0^{(j)}\right) =& \boldsymbol{B}^{-1}\left(\boldsymbol{\delta x}_0^{(j)} + \boldsymbol{x}_0^{(j)} - \boldsymbol{x}_0^b\right) \\ & - \left(\boldsymbol{H}_0^T \Delta_0 + \boldsymbol{M}_1^T\left[\boldsymbol{H}_1^T \Delta_1 + \boldsymbol{M}_2^T\left[\boldsymbol{H}_2^T \Delta_2 + \dots + \boldsymbol{M}_{\mathcal{T}}^T \boldsymbol{H}_{\mathcal{T}}^T \Delta_{\mathcal{T}}\right] \dots\right]\right). \qquad (2.35)\end{aligned}$$

The computation of the gradient based on this Horner factorization is detailed in Algorithm 2.1: it provides a numerical advantage by efficiently computing $\nabla J\left(\boldsymbol{\delta x}_0^{(j)}\right)$ and minimizing unnecessary matrix product computations. When considering Algorithm 2.1, we implicitly assume the error covariance matrices $\boldsymbol{P}_0^f$, $\boldsymbol{R}_0, \dots, \boldsymbol{R}_{\mathcal{T}}$, along with operators $\boldsymbol{M}_0, \dots, \boldsymbol{M}_{\mathcal{T}}$ and $\boldsymbol{H}_0, \dots, \boldsymbol{H}_{\mathcal{T}}$ to be available.

The direct implementation of the gradient derivation masks a significant challenge related to the mathematical representation of the adjoint (or transpose here) of the model matrices $\boldsymbol{M}_k$ and observation operators $\boldsymbol{H}_k$. In practical data assimilation applications, physical models are often implemented as operators in programming languages (*e.g.*, Fortran, C, or C++), rather than being straightforward

Algorithm 2.1: Computation of $\nabla J\left(\boldsymbol{\delta x}_0^{(j)}\right)$ (4D-Var)

```
   Inputs:
      Background $\boldsymbol{x}_0^b \in \mathbb{R}^n$ ;
      Current initial estimate $\boldsymbol{x}_0^{(j)} \in \mathbb{R}^n$ ;
      Observations $\{\boldsymbol{y}_0, \dots, \boldsymbol{y}_{\mathcal{T}}\} \in \mathbb{R}^{p\times(\mathcal{T}+1)}$ ;

   Forward propagation step:
 1 for $k = 0, 1, \dots, \mathcal{T}$ do
      // Compute forecast states
 2    $\boldsymbol{x}_{k+1}^{(j)} = \boldsymbol{\mathcal{M}}_{k+1}\left(\boldsymbol{x}_k^{(j)}\right)$
      // Propagate increments with the linearized model
 3    $\boldsymbol{\delta x}_{k+1}^{(j)} = \boldsymbol{M}_{k+1}\boldsymbol{\delta x}_k^{(j)}$
      // Compute the normalized innovations
 4    $\Delta_k = \boldsymbol{R}_k^{-1}\left(\boldsymbol{y}_k - \boldsymbol{\mathcal{H}}_k\left(\boldsymbol{x}_k^{(j)}\right) - \boldsymbol{H}_k\boldsymbol{\delta x}_k^{(j)}\right)$
 5 end

   Backward propagation step:
 6 $\widetilde{\boldsymbol{\delta x}}_{\mathcal{T}}^{(j)} = \boldsymbol{H}_{\mathcal{T}}^T\Delta_{\mathcal{T}}$
 7 for $k = \mathcal{T}, \dots, 1, 0$ do
      // Iteratively compute the gradient's terms
 8    $\widetilde{\boldsymbol{\delta x}}_k^{(j)} = \boldsymbol{H}_k^T\Delta_k + \boldsymbol{M}_{k+1}^T\widetilde{\boldsymbol{\delta x}}_k^{(j)}$
 9 end
10 return $\nabla J\left(\boldsymbol{\delta x}_0^{(j)}\right) = -\widetilde{\boldsymbol{\delta x}}_0$
```

matrix operations. Observation operators are also mappings that rarely take the form of matrices. Consequently, computing the adjoint of these operators - essentially finding the practical equivalent of $\boldsymbol{M}_k^T$ and $\boldsymbol{H}_k^T$ - is far from trivial and demands considerable effort to accurately reflect the theoretical operations in code. Additionally, since physical phenomena are mostly driven by nonlinear equations, and as state variables are not always the observed ones, deriving the gradient of the functional J very often involves linearizing these operators at the considered point $\boldsymbol{x}_k^{(j)}$. All these mathematical and computational considerations are practical impediments to a simple and straightforward implementation of 4D-Var.

In order to compute the minimum of the functional J, let us introduce the following notation:

$$\boldsymbol{M}_{k,0} = \boldsymbol{M}_k \boldsymbol{M}_{k-1}, \dots, \boldsymbol{M}_2 \boldsymbol{M}_1,\ \forall k \in [\![1, \mathcal{T}]\!]. \tag{2.36}$$

We can then reformulate equation (2.35) into:

$$\begin{aligned}
\nabla J\left(\boldsymbol{\delta x}_0^{(j)}\right) &= \boldsymbol{B}^{-1}\left(\boldsymbol{\delta x}_0^{(j)} + \boldsymbol{x}_0^{(j)} - \boldsymbol{x}_0^b\right) - \sum_{k=0}^{\mathcal{T}} \boldsymbol{M}_{k,0}^T \boldsymbol{H}_k^T \boldsymbol{R}_k^{-1}\left(\boldsymbol{y}_k - \boldsymbol{\mathcal{G}}_k\left(\boldsymbol{x}_0^{(j)}\right) - \boldsymbol{H}_k \boldsymbol{M}_{k,0} \boldsymbol{\delta x}_0^{(j)}\right) \\
&= \boldsymbol{B}^{-1}\left(\boldsymbol{\delta x}_0^{(j)} + \boldsymbol{x}_0^{(j)} - \boldsymbol{x}_0^b\right) - \left(\sum_{k=0}^{\mathcal{T}} \boldsymbol{M}_{k,0}^T \boldsymbol{H}_k^T \boldsymbol{R}_k^{-1}\left(\boldsymbol{y}_k - \boldsymbol{\mathcal{G}}_k\left(\boldsymbol{x}_0^{(j)}\right)\right)\right) - \left(\sum_{k=0}^{\mathcal{T}} \boldsymbol{M}_{k,0}^T \boldsymbol{H}_k^T \boldsymbol{R}_k^{-1} \boldsymbol{H}_k \boldsymbol{M}_{k,0}\right) \boldsymbol{\delta x}_0^{(j)} \\
&= \left(\boldsymbol{B}^{-1} + \sum_{k=0}^{\mathcal{T}} \boldsymbol{M}_{k,0}^T \boldsymbol{H}_k^T \boldsymbol{R}_k^{-1} \boldsymbol{H}_k \boldsymbol{M}_{k,0}\right) \boldsymbol{\delta x}_0^{(j)} - \sum_{k=0}^{\mathcal{T}} \boldsymbol{M}_{k,0}^T \boldsymbol{H}_k^T \boldsymbol{R}_k^{-1}\left(\boldsymbol{y}_k - \boldsymbol{\mathcal{G}}_k\left(\boldsymbol{x}_0^{(j)}\right)\right) - \boldsymbol{B}^{-1}\left(\boldsymbol{x}_0^{(j)} - \boldsymbol{x}_0^b\right).
\end{aligned} \tag{2.37}$$

When searching for $\boldsymbol{\delta x}_0^{(j)}$ such that $\nabla J\left(\boldsymbol{\delta x}_0^{(j)}\right) = 0$, we obtain:

$$\boldsymbol{\delta x}_0^{(j,*)} = \left(\boldsymbol{B}^{-1} + \sum_{k=0}^{\mathcal{T}} \boldsymbol{M}_{k,0}^T \boldsymbol{H}_k^T \boldsymbol{R}_k^{-1} \boldsymbol{H}_k \boldsymbol{M}_{k,0}\right)^{-1} \left(\boldsymbol{B}^{-1}\left(\boldsymbol{x}_0^{(j)} - \boldsymbol{x}_0^b\right) + \sum_{k=0}^{\mathcal{T}} \boldsymbol{M}_{k,0}^T \boldsymbol{H}_k^T \boldsymbol{R}_k^{-1}\left(\boldsymbol{y}_k - \boldsymbol{\mathcal{G}}_k\left(\boldsymbol{x}_0^{(j)}\right)\right)\right), \tag{2.38}$$

where the matrix to be inverted is a first-order approximation of the Hessian of J.

In real-world applications, the size of the matrices involved in the assimilation process is significantly large, so that a direct inversion of $\left(\boldsymbol{B}^{-1} + \sum_{k=0}^{\mathcal{T}} \boldsymbol{M}_{k,0}^T \boldsymbol{H}_k^T \boldsymbol{R}_k^{-1} \boldsymbol{H}_k \boldsymbol{M}_{k,0}\right)^{-1}$ is unaffordable. We instead use iterative methods to approximate $\boldsymbol{\delta x}_0^{(j,*)}$.

If we now assume that operators $\boldsymbol{H}_k$ and $\boldsymbol{M}_k$ are linear, and models $\boldsymbol{M}_k$ are supposed to be perfect, 4D-Var algorithm is characterized by a **transferability of optimality** property (Li and Navon, 2001; Bocquet and Farchi, 2014). This property implies that solving the 4D-Var problem over $[t_0, t_\mathcal{T}]$ yields the same initial state estimate $\boldsymbol{x}_0^a$ as sequentially optimizing over two sub-windows $[t_0, t_i]$ and $[t_i, t_\mathcal{T}]$. More precisely, let us start by considering the analysis state at time t_0, denoted $\boldsymbol{x}_0^{a,*}$ and obtained by optimizing over $[t_0, t_i]$ using 4D-Var. Given this state $\boldsymbol{x}_0^{a,*}$, the transferability of optimality property states that conducting a subsequent optimization for $\boldsymbol{x}_0$ considering the observations from $[t_i, t_\mathcal{T}]$ and using $\boldsymbol{x}_0^{a,*}$ as the background state, is equivalent to a direct 4D-Var application across the entire interval $[t_0, t_\mathcal{T}]$, and therefore yields $\boldsymbol{x}_0^{a,\dagger}$ which is equal to $\boldsymbol{x}_0^a$. We can mathematically derive this equivalence, but this would extend beyond the scope of this thesis.

2.2.2 Sequential approaches

In the following subsections, we present four sequential data assimilation approaches along with their specificities and differences: the **Kalman Filter** (KF), the **Extended Kalman Filter** (EKF), the **Ensemble Transform Kalman Filter** (ETKF) and the **Ensemble Transform Kalman Filter with model error** (ETKF-Q). In essence, these approaches are grounded on BLUE analysis and offer extensions of it in the context of equations (2.27a) and (2.27b).

2.2.2.1 Kalman Filter

The **Kalman filter** (KF) (Kalman, 1960; Welch *et al.*, 1995) extends BLUE methodology to cases involving dynamic models and time-sequential observations. On top of the BLUE analysis step yielding $\boldsymbol{x}_k^a$, the Kalman filter provides a forecast step which not only propagates $\boldsymbol{x}_k^a$ forward in time, but also calculates the associated error covariance matrix $\boldsymbol{P}_{k+1}^f$, both of these variables being required for the next analysis at time t_{k+1}.

Within the context of the Kalman filter, we assume that the models and the observation operators are linear, so that equations (2.27a) and (2.27b) become:

$$\begin{cases} \boldsymbol{x}_k^t = \boldsymbol{M}_k \boldsymbol{x}_{k-1}^t + \boldsymbol{\eta}_k, & (2.39a) \\ \boldsymbol{y}_k = \boldsymbol{H}_k \boldsymbol{x}_k^t + \boldsymbol{\varepsilon}_k, & (2.39b) \end{cases}$$

Figure 2.2.2 illustrates how the Kalman filter sequentially computes an analysis estimate from the background $\boldsymbol{x}_k^f$, the observations $\boldsymbol{y}_k$, and their associated uncertainties.

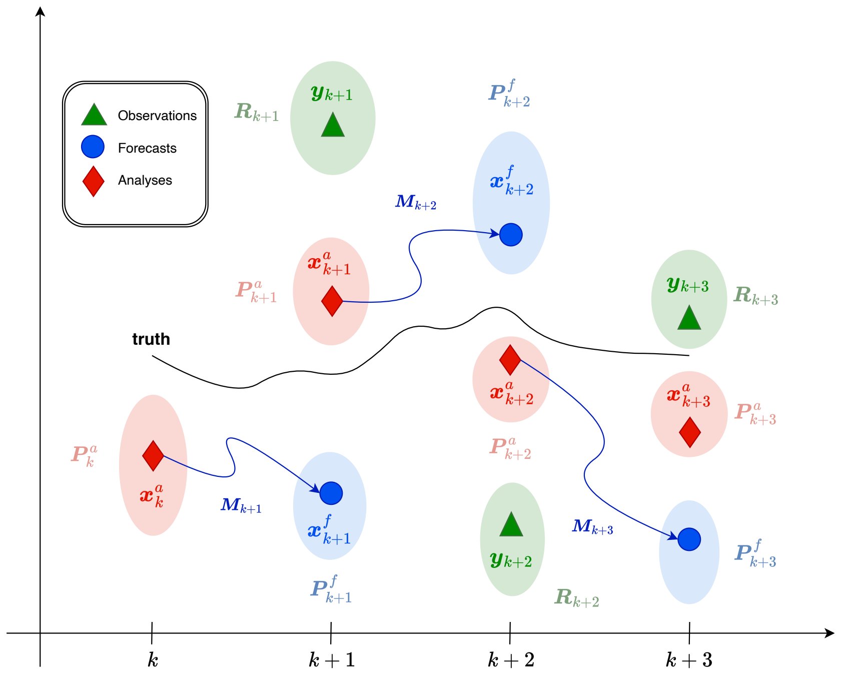


Figure 2.2.2: Illustration of the Kalman filter algorithm.

The analysis step of the Kalman filter is the same as the BLUE approach, albeit with updated notation to reflect the temporal dimension:

$$\boxed{\begin{aligned} \boldsymbol{x}_k^a &= \boldsymbol{x}_k^f + \boldsymbol{K}_k\Big(\boldsymbol{y}_k - \boldsymbol{H}_k\boldsymbol{x}_k^f\Big), && (2.40a)\\ \boldsymbol{P}_k^a &= (\boldsymbol{I}_n - \boldsymbol{K}_k\boldsymbol{H}_k)\boldsymbol{P}_k^f, && (2.40b)\end{aligned}}$$

where $\boldsymbol{x}_k^a$ is the analysis estimate at time t_k, $\boldsymbol{x}_k^f$ represents the forecast derived from the previous analysis through model $\boldsymbol{\mathcal{M}}_k$ (*i.e.*, $\boldsymbol{x}_k^f = \boldsymbol{\mathcal{M}}_k\big(\boldsymbol{x}_{k-1}^a\big)$, $\forall k > 1$), and $\boldsymbol{x}_0^f$ denotes the initial background state $\boldsymbol{x}_0^b$. The Kalman gain matrix $\boldsymbol{K}_k$, observation operator $\boldsymbol{H}_k$, and observations $\boldsymbol{y}_k$ are all specified for time t_k.

One crucial point when forecasting the analysis $\boldsymbol{x}_k^a$ to yield $\boldsymbol{x}_{k+1}^f$ is the evolution of its error covariance matrix $\boldsymbol{P}_k^a$ and its link with respect to $\boldsymbol{P}_{k+1}^f$. Therefore, let us derive the error $\boldsymbol{e}_{k+1}^f = \boldsymbol{x}_{k+1}^f - \boldsymbol{x}_{k+1}^t$ in order to estimate $\boldsymbol{P}_{k+1}^f = \mathbb{E}\left[\boldsymbol{e}_{k+1}^f\left(\boldsymbol{e}_{k+1}^f\right)^T\right]$:

$$\begin{aligned}
\boldsymbol{e}_{k+1}^f &= \boldsymbol{x}_{k+1}^f - \boldsymbol{x}_{k+1}^t \\
&= \boldsymbol{M}_{k+1}\boldsymbol{x}_k^a - \boldsymbol{x}_{k+1}^t \\
&= \boldsymbol{M}_{k+1}(\boldsymbol{x}_k^a - \boldsymbol{x}_k^t + \boldsymbol{x}_k^t) - \boldsymbol{x}_{k+1}^t \\
&= \boldsymbol{M}_{k+1}\underbrace{(\boldsymbol{x}_k^a - \boldsymbol{x}_k^t)}_{=\boldsymbol{e}_k^a} - \underbrace{(\boldsymbol{x}_{k+1}^t - \boldsymbol{M}_{k+1}\boldsymbol{x}_k^t)}_{=\boldsymbol{\eta}_{k+1}}.
\end{aligned} \tag{2.41}$$

From which we derive:

$$\begin{aligned}
\boldsymbol{P}_{k+1}^f &= \mathbb{E}\left[\boldsymbol{e}_{k+1}^f\left(\boldsymbol{e}_{k+1}^f\right)^T\right] \\
&= \mathbb{E}\left[\left(\boldsymbol{M}_{k+1}\boldsymbol{e}_k^a - \boldsymbol{\eta}_{k+1}\right)\left((\boldsymbol{e}_k^a)^T\boldsymbol{M}_{k+1}^T - \boldsymbol{\eta}_{k+1}^T\right)\right] \\
&= \mathbb{E}\left[\boldsymbol{M}_{k+1}\boldsymbol{e}_k^a(\boldsymbol{e}_k^a)^T\boldsymbol{M}_{k+1}^T - \boldsymbol{M}_{k+1}\boldsymbol{e}_k^a\boldsymbol{\eta}_{k+1}^T - \boldsymbol{\eta}_{k+1}(\boldsymbol{e}_k^a)^T\boldsymbol{M}_{k+1}^T + \boldsymbol{\eta}_{k+1}\boldsymbol{\eta}_{k+1}^T\right] \\
&= \boldsymbol{M}_{k+1}\underbrace{\mathbb{E}\left[\boldsymbol{e}_k^a(\boldsymbol{e}_k^a)^T\right]}_{=\boldsymbol{P}_k^a}\boldsymbol{M}_{k+1}^T - \boldsymbol{M}_{k+1}\underbrace{\mathbb{E}\left[\boldsymbol{e}_k^a\boldsymbol{\eta}_{k+1}^T\right]}_{=0_{\mathbb{R}^{n\times n}}} - \underbrace{\mathbb{E}\left[\boldsymbol{\eta}_{k+1}(\boldsymbol{e}_k^a)^T\right]}_{=0_{\mathbb{R}^{n\times n}}}\boldsymbol{M}_{k+1}^T + \underbrace{\mathbb{E}\left[\boldsymbol{\eta}_{k+1}\boldsymbol{\eta}_{k+1}^T\right]}_{=\boldsymbol{Q}_{k+1}} \\
&\boxed{= \boldsymbol{M}_{k+1}\boldsymbol{P}_k^a\boldsymbol{M}_{k+1}^T + \boldsymbol{Q}_{k+1}.}
\end{aligned} \tag{2.42}$$

The Kalman Filter method is summarized in Algorithm 2.2 (we remind that most notations along with algorithms structures, are inspired from the lecture notes of Bocquet and Farchi (2014)).

In real-world scenarios, the linearity assumption for the model and observation operators ($\boldsymbol{H}_k$ and $\boldsymbol{M}_k$) rarely holds, necessitating adaptations to the standard Kalman filter framework for nonlinear systems. This adaptation involves the linearization of these operators around the current estimate, allowing to keep using the Kalman filter with some slight modifications. Specifically, the nonlinear operators $\boldsymbol{\mathcal{H}}_k$ and $\boldsymbol{\mathcal{M}}_k$ are employed directly to map the background state into the observation space and for temporal propagation of the analysis state $\boldsymbol{x}^a$, respectively. Meanwhile, $\boldsymbol{H}_k$ and $\boldsymbol{M}_k$ are redefined to represent the Jacobian matrices of $\boldsymbol{\mathcal{H}}_k$ and $\boldsymbol{\mathcal{M}}_k$ at the current estimate, effectively serving as their linear approxi-

Algorithm 2.2: Kalman Filter

```
  Inputs:
      Background $\boldsymbol{x}_0^f \in \mathbb{R}^n$ ;
      Observations $\{\boldsymbol{y}_0, \dots, \boldsymbol{y}_T\} \in \mathbb{R}^{p\times(T+1)}$ ;
1 for $k = 0, 1, \dots, T$ do
                  Analysis step
      // Compute the Kalman gain
2     $\boldsymbol{K}_k = \boldsymbol{P}_k^f \boldsymbol{H}_k^T \left(\boldsymbol{H}_k \boldsymbol{P}_k^f \boldsymbol{H}_k^T + \boldsymbol{R}_k\right)^{-1}$
      // Derive the analysis
3     $\boldsymbol{x}_k^a = \boldsymbol{x}_k^f + \boldsymbol{K}_k \left(\boldsymbol{y}_k - \boldsymbol{H}_k \boldsymbol{x}_k^f\right)$
      // Compute the error covariance matrix of the analysis
4     $\boldsymbol{P}_k^a = (\boldsymbol{I}_n - \boldsymbol{K}_k \boldsymbol{H}_k)\boldsymbol{P}_k^f$
                  Propagation step
      // Forecast the analysis
5     $\boldsymbol{x}_{k+1}^f = \boldsymbol{M}_{k+1} \boldsymbol{x}_k^a$
      // Compute the error covariance matrix of the forecast
         analysis
6     $\boldsymbol{P}_{k+1}^f = \boldsymbol{M}_{k+1} \boldsymbol{P}_k^a \boldsymbol{M}_{k+1}^T + \boldsymbol{Q}_{k+1}$
7 end
```

mations. Also, $\boldsymbol{H}_k^T$ and $\boldsymbol{M}_k^T$ represent the adjoint of the observation operator and the transpose of the Jacobian matrix of the model, respectively. This approach, which relaxes the linearity assumption, is known as the **Extended Kalman Filter** (EKF), and maintains the iterative forecast-analysis cycle of the Kalman filter while accommodating nonlinear model and observation operators.

The extended Kalman filter is presented in Algorithm 2.3 and operates by:

- linearizing the observation operator $\boldsymbol{\mathcal{H}}_k$ to get the matrix $\boldsymbol{H}_k$, and computing $\boldsymbol{H}_k^T$, the adjoint of $\boldsymbol{\mathcal{H}}_k$. These three operators are then used to calculate the Kalman gain and update the state estimate in the analysis step. The nonlinear operator $\boldsymbol{\mathcal{H}}_k$ is directly used in the calculation of the innovation term.
- using $\boldsymbol{\mathcal{M}}_k$ to forecast the state forward in time, directly applying the nonlinear model. The derivation of the linear operator $\boldsymbol{M}_k$ and its transpose $\boldsymbol{M}_k^T$ allows for the derivation of $\boldsymbol{P}_k^a$ from $\boldsymbol{P}_k^f$ and $\boldsymbol{Q}_k$.

Therefore, the Kalman filter equation (2.40) and the propagation equation can be reformulated as follows:

$$\boxed{\begin{aligned} \boldsymbol{x}_k^a &= \boldsymbol{x}_k^f + \boldsymbol{K}_k\Big(\boldsymbol{y}_k - \boldsymbol{H}_k\Big(\boldsymbol{x}_k^f\Big)\Big), && (2.43a)\\ \boldsymbol{x}_k^f &= \boldsymbol{\mathcal{M}}_k\big(\boldsymbol{x}_{k-1}^a\big). && (2.43b) \end{aligned}}$$

Despite being grounded on strong theoretical foundations, the applicability of the KF and the EKF is challenged by the high dimensionality and inherent nonlinearity of many real-world systems.

A first significant limitation arises from the sheer high-dimensionality of the considered physical problem. In weather forecast, oceanography or geosciences, the state vector's dimension n can reach up to 10^6 to 10^9, meaning that error covariance matrices ($\boldsymbol{P}_k^f$ and $\boldsymbol{P}_k^a$) can contain 10^{12} to 10^{18} entries. The mere storage of these covariance matrices is very often not possible, simply because they do not fit in memory. Also, mathematical computations involving covariance matrices can be intractable in practice, the most significant limitation being the computation of $\boldsymbol{P}_{k+1}^f$ from $\boldsymbol{P}_k^a$: it indeed requires matrix-vector products with $\boldsymbol{M}_{k+1}$ and $\boldsymbol{M}_{k+1}^T$. Consequently, KF and EKF are naturally rather confined to low-dimensional problems.

Algorithm 2.3: Extended Kalman Filter

```
  Inputs:
      Background $\boldsymbol{x}_0^f \in \mathbb{R}^n$ ;
      Observations $\{\boldsymbol{y}_0, \dots, \boldsymbol{y}_T\} \in \mathbb{R}^{p \times (T+1)}$ ;
1 for $k = 0, 1, \dots, T$ do
                  Analysis step
      // Compute the Kalman gain
2     $\boldsymbol{K}_k = \boldsymbol{P}_k^f \boldsymbol{H}_k^T \left( \boldsymbol{H}_k \boldsymbol{P}_k^f \boldsymbol{H}_k^T + \boldsymbol{R}_k \right)^{-1}$
      // Derive the analysis (with nonlinear observation operator)
3     $\boldsymbol{x}_k^a = \boldsymbol{x}_k^f + \boldsymbol{K}_k \left( \boldsymbol{y}_k - \boldsymbol{\mathcal{H}}_k \left( \boldsymbol{x}_k^f \right) \right)$
      // Compute the error covariance matrix of the analysis
4     $\boldsymbol{P}_k^a = (\boldsymbol{I}_n - \boldsymbol{K}_k \boldsymbol{H}_k) \boldsymbol{P}_k^f$
                  Propagation step
      // Forecast the analysis (with nonlinear model)
5     $\boldsymbol{x}_{k+1}^f = \boldsymbol{\mathcal{M}}_{k+1}(\boldsymbol{x}_k^a)$
      // Compute the error covariance matrix of the forecast
         analysis
6     $\boldsymbol{P}_{k+1}^f = \boldsymbol{M}_{k+1} \boldsymbol{P}_k^a \boldsymbol{M}_{k+1}^T + \boldsymbol{Q}_{k+1}$
7 end
```

The second major hurdle occurs when the dynamics of interest is nonlinear. Indeed, the evolution of $\boldsymbol{P}_k^a$ is given by $\mathbb{E}\left[\left(\boldsymbol{x}_{k+1}^t - \boldsymbol{\mathcal{M}}_{k+1}(\boldsymbol{x}_k^a)\right)\left(\boldsymbol{x}_{k+1}^t - \boldsymbol{\mathcal{M}}_{k+1}(\boldsymbol{x}_k^a)\right)^T\right]$ and the derivation of $\left(\boldsymbol{x}_{k+1}^t - \boldsymbol{\mathcal{M}}_{k+1}(\boldsymbol{x}_k^a)\right)$ requires to linearize around the analysis state $\boldsymbol{x}_k^a$. Linearizing introduce approximations and therefore possible significant errors and instabilities, which may accumulate over time, leading to divergence from the true system state. Accounting for higher closure schemes when linearizing could resolve this issue but is unaffordable in practice because of their high-dimensionality.

2.2.2.2 Ensemble Transform Kalman Filter (ETKF)

In order to overcome the aforementioned limitations of the Kalman filter and the extended Kalman filter, ensemble approaches based on reduced rank approximations have been proposed. The **Ensemble Kalman Filter** (**EnKF**) is a stochastic algorithm firstly introduced in 1994 by Evensen, and further developed by Burgers *et al.* (1998); Houtekamer and Mitchell (1998); Evensen (2009a): instead of performing the analysis step on a single state $\boldsymbol{x}_k^f$ (as done in KF or EKF), the EnKF considers a set of states, and assimilates a perturbed observation vector for each one of them. Later, Bishop *et al.* (2001) and Hunt *et al.* (2007) published a deterministic ensemble algorithm, named **Ensemble Transform Kalman Filter** (**ETKF**): the ETKF algorithm is inspired from the reduced rank square root filter (RRSQRT) (Verlaan and Heemink, 1997; Evensen, 1994), and does not require to individually perturb the observation $\boldsymbol{y}_k$ for each state of the ensemble. The Singular Evolutive Extended Kalman Filter (SEEK) (Tuan Pham *et al.*, 1998) is another common ensemble method that is well-known in oceanography.

Ensemble algorithms are optimal under linear and Gaussian assumptions. These methods utilize a collection of state estimates, known as ensembles, to update the statistics of the data assimilation estimates. Typically, ensembles consist of tens to hundreds of state estimates, referred to as "members.". Integrating ensembles within the KF or EKF frameworks allows for a strategic and effective circumvention of the high-dimensionality and nonlinearity issues faced by these filters. A low-rank approximation of the error covariance matrix can be empirically derived from the ensemble, enabling partial recovery of the information contained in $\boldsymbol{P}_k^f$ from a relatively small number of members compared to the system's dimension. This reduces the need for large storage capacities and extensive linearization schemes. This frees the constraints for large storage capabilities and extensive closure linearization schemes. However, representing a high-dimensional covariance matrix with a relatively small number of members necessarily introduces large sampling errors. These errors can lead to filter divergence if not properly addressed. In the following subsection, we will discuss **localization** and **inflation**, two common techniques used to mitigate sampling errors in ensemble algorithms.

Assuming that the considered physical phenomenon follows Gaussian statistics, ensemble methods update the prior probability density function given the distribution of the observations. Figure 2.2.3 visually represents how ETKF algorithm is performed through time.

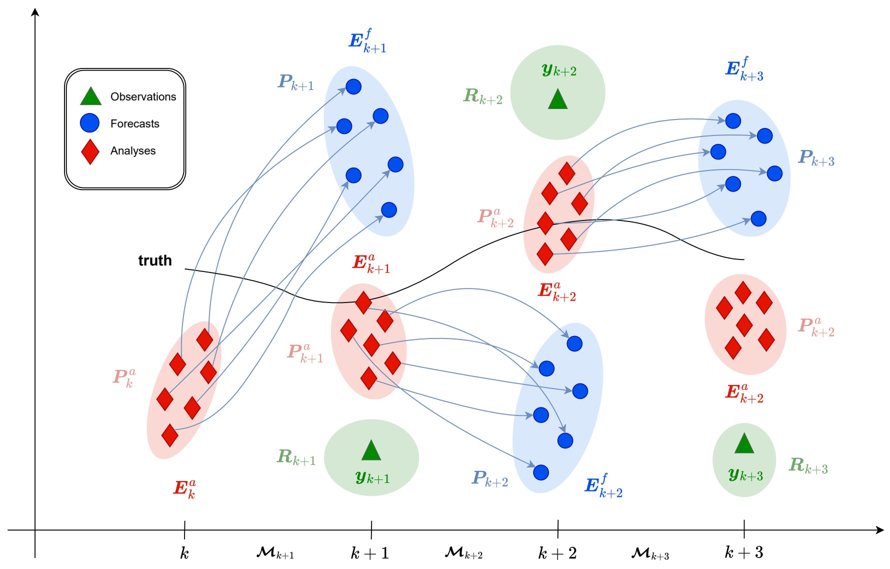


Figure 2.2.3: Illustration of the ETKF algorithm.

The EnKF and ETKF stand out for their simplicity, ease of implementation, and applicability to real-world scenarios. Contrary to methods like 3D-Var, EKF or 4D-Var for which deriving the tangent linear and adjoint operators is known to be possibly arduous, EnKF and ETKF do not require such heavy computations.

In the following of this subsection, we focus on the Ensemble Transform Kalman Filter (ETKF) as detailed in the seminal works of Bishop *et al.* (2001); Hunt *et al.* (2007). Although we present our methodology based on the ETKF, it is important to note that the approach outlined in this thesis is quite general and readily adaptable to various other types of data assimilation algorithms.

Likewise in the KF and the EKF, equations (2.27a) and (2.27b) describe our physical and observational dynamical system. We define a prior ensemble of m members (with $m \ll n$) at time k as follows:

$$\boldsymbol{E}_k^f = \left\{ \boldsymbol{x}_k^{f,1}, \boldsymbol{x}_k^{f,2}, \dots, \boldsymbol{x}_k^{f,m} \right\}, \tag{2.44}$$

where $\boldsymbol{E}_0^f$ can be generated by randomly perturbing an initial background estimate $\boldsymbol{x}_0^f$.

Given this ensemble of m members, we can empirically derive a low-rank approximation of the error covariance matrix $\boldsymbol{P}_k^f$ as follows:

$$\boxed{\boldsymbol{P}_k^f = \frac{1}{m-1} \sum_{i=1}^{m} \left(\boldsymbol{x}_k^{f,i} - \overline{\boldsymbol{x}}_k^f \right) \left(\boldsymbol{x}_k^{f,i} - \overline{\boldsymbol{x}}_k^f \right)^T = \boldsymbol{X}_k^f \left(\boldsymbol{X}_k^f \right)^T,} \tag{2.45}$$

where $\overline{\boldsymbol{x}}_k$ denotes the mean of $\boldsymbol{E}_k$. $\boldsymbol{X}_k^f \in \mathbb{R}^{n \times m}$ is termed **anomalies** or **ensemble anomalies** and is defined such that:

$$\boxed{\left[\boldsymbol{X}_k^f \right]_i = \frac{\boldsymbol{x}_k^{f,i} - \overline{\boldsymbol{x}}_k^f}{\sqrt{m-1}},} \tag{2.46}$$

with $[\cdot]_i$ denoting the i^{th} column of matrix $\boldsymbol{X}_k$.

Likewise the normalized anomalies, we introduce their observational counterparts to represent $\boldsymbol{H}_k \boldsymbol{X}_k^f$, namely the **observation anomalies** $\boldsymbol{Y}_k^f$:

$$\boxed{\left[\boldsymbol{Y}_k^f \right]_i = \left[\boldsymbol{H}_k \boldsymbol{X}_k^f \right]_i = \frac{\boldsymbol{\mathcal{H}}_k \left(\boldsymbol{x}_k^{f,i} \right) - \overline{\boldsymbol{y}}_k^f}{\sqrt{m-1}},} \tag{2.47}$$

where $\overline{\boldsymbol{y}}_k^f = \frac{1}{m} \sum_{i=1}^{m} \boldsymbol{\mathcal{H}}_k \left(\boldsymbol{x}_k^{f,i} \right)$. For the sake of simplicity, we use the notations $\boldsymbol{H}_k \boldsymbol{X}_k^f$ and $\left(\boldsymbol{X}_k^f \right)^T \boldsymbol{H}_k^T$ to represent $\frac{\boldsymbol{\mathcal{H}}_k (\boldsymbol{x}_k^{f,i}) - \overline{\boldsymbol{y}}_k^f}{\sqrt{m-1}}$ and $\left(\frac{\boldsymbol{\mathcal{H}}_k (\boldsymbol{x}_k^{f,i}) - \overline{\boldsymbol{y}}_k^f}{\sqrt{m-1}} \right)^T$, respectively.

Unlike the **stochastic ensemble Kalman filter** (Evensen, 1994; Burgers *et al.*, 1998; Evensen, 2009a) which assimilates perturbed observations for every member, in the ensemble transform Kalman filter the assimilation is performed on the mean of $\boldsymbol{E}_k^f$ solely:

$$\boxed{\overline{\boldsymbol{x}}_k^a = \overline{\boldsymbol{x}}_k^f + \boldsymbol{K}^*\left(\boldsymbol{y}_k - \boldsymbol{\mathcal{H}}_k\left(\overline{x}_k^f\right)\right).} \tag{2.48}$$

We then develop equation (2.48) and substitute the forecast error covariance matrix $\boldsymbol{P}_k^f$ by its low-rank approximation so that:

$$\begin{aligned}
\overline{\boldsymbol{x}}_k^a &= \overline{\boldsymbol{x}}_k^f + \boldsymbol{K}^*\left(\boldsymbol{y}_k - \boldsymbol{\mathcal{H}}_k\left(\overline{x}_k^f\right)\right) \\
&= \overline{\boldsymbol{x}}_k^f + \boldsymbol{P}_k^f\boldsymbol{H}_k^T\left(\boldsymbol{H}_k\boldsymbol{P}_k^f\boldsymbol{H}_k^T + \boldsymbol{R}_k\right)^{-1}\left(\boldsymbol{y}_k - \boldsymbol{\mathcal{H}}_k\left(\overline{x}_k^f\right)\right) \\
&\approx \overline{\boldsymbol{x}}_k^f + \boldsymbol{X}_k^f\underbrace{\left(\boldsymbol{X}_k^f\right)^T\boldsymbol{H}_k^T}_{=\left(\boldsymbol{Y}_k^f\right)^T}\left(\underbrace{\boldsymbol{H}_k\boldsymbol{X}_k^f}_{=\boldsymbol{Y}_k^f}\underbrace{\left(\boldsymbol{X}_k^f\right)^T\boldsymbol{H}_k^T}_{=\left(\boldsymbol{Y}_k^f\right)^T} + \boldsymbol{R}_k\right)^{-1}\left(\boldsymbol{y}_k - \boldsymbol{\mathcal{H}}_k\left(\overline{x}_k^f\right)\right) \\
&\approx \overline{\boldsymbol{x}}_k^f + \boldsymbol{X}_k^f\underbrace{\left(\boldsymbol{Y}_k^f\right)^T\left(\boldsymbol{Y}_k^f\left(\boldsymbol{Y}_k^f\right)^T + \boldsymbol{R}_k\right)^{-1}\left(\boldsymbol{y}_k - \boldsymbol{\mathcal{H}}_k\left(\overline{x}_k^f\right)\right)}_{\boldsymbol{w}^a\in\mathbb{R}^m} && (2.49) \\
&\boxed{\approx \overline{\boldsymbol{x}}_k^f + \boldsymbol{X}_k^f\boldsymbol{w}^a.} && (2.50)
\end{aligned}$$

Notably, the analysis estimate $\overline{\boldsymbol{x}}_k^a$ is not unique. Indeed, for any $\alpha \in \mathbb{R}$ and for any $\boldsymbol{w} \in \mathbb{R}^m$, we have:

$$\begin{aligned}
\boldsymbol{X}_k^f(\boldsymbol{w} + \alpha\boldsymbol{1}) &= \boldsymbol{X}_k^f\boldsymbol{w} + \alpha\boldsymbol{X}_k^f\boldsymbol{1} \\
&= \boldsymbol{X}_k^f\boldsymbol{w} + \alpha\frac{\sum_{i=1}^m\left(\boldsymbol{x}_k^{f,i} - \overline{\boldsymbol{x}}_k^f\right)}{\sqrt{m-1}} \\
&= \boldsymbol{X}_k^f\boldsymbol{w} + \alpha\frac{\left(\sum_{i=1}^m\boldsymbol{x}_k^{f,i}\right) - \left(\sum_{i=1}^m\overline{\boldsymbol{x}}_k^f\right)}{\sqrt{m-1}} \\
&= \boldsymbol{X}_k^f\boldsymbol{w} + \alpha\frac{m\overline{\boldsymbol{x}}_k^f - m\overline{\boldsymbol{x}}_k^f}{\sqrt{m-1}} \\
&= \boldsymbol{X}_k^f\boldsymbol{w},
\end{aligned} \tag{2.51}$$

where $\boldsymbol{1} = [1,\ldots,1]^T$. Solutions have been proposed in order to get the uniqueness of $\overline{\boldsymbol{x}}_k^a$, such as the gauge-fixing term of Bocquet and Sakov (2014), and the deviation matrices of Fillion *et al.* (2020).

Applying Sherman-Morrison-Woodbury, we get:

$$\boxed{\boldsymbol{w}^a = \left(\boldsymbol{I}_m + \left(\boldsymbol{Y}_k^f\right)^T \boldsymbol{R}_k^{-1}\boldsymbol{Y}_k^f\right)^{-1}\left(\boldsymbol{Y}_k^f\right)\boldsymbol{R}^{-1}\left(\boldsymbol{y}_k - \boldsymbol{\mathcal{H}}_k\left(\overline{x}_k^f\right)\right).} \tag{2.52}$$

The analysis can therefore be rewritten as follows:

$$\overline{\boldsymbol{x}}_k^a = \overline{\boldsymbol{x}}_k^f + \boldsymbol{X}_k^f\left(\boldsymbol{I}_m + \left(\boldsymbol{Y}_k^f\right)^T \boldsymbol{R}_k^{-1}\boldsymbol{Y}_k^f\right)^{-1}\left(\boldsymbol{Y}_k^f\right)\boldsymbol{R}^{-1}\left(\boldsymbol{y}_k - \boldsymbol{\mathcal{H}}_k\left(\overline{x}_k^f\right)\right). \tag{2.53}$$

In order to iterate this assimilation procedure, we need to generate a posterior ensemble $\boldsymbol{E}_k^a$. It requires to derive a square-root matrix $\boldsymbol{X}_k^a$ of the error covariance matrix $\boldsymbol{P}_k^a$, so that we could perturb $\overline{\boldsymbol{x}}_k^a$ according to $\boldsymbol{P}_k^a$ statistics. We would like to express $\boldsymbol{P}_k^a$ under the form $\boldsymbol{X}_k^a(\boldsymbol{X}_k^a)^T$. Therefore, we consider the expression of $\boldsymbol{P}_k^a$ as given in equation (A.10):

$$\begin{aligned}
\boldsymbol{P}_k^a &= (\boldsymbol{I}_n - \boldsymbol{K}_k^*\boldsymbol{H}_k)\boldsymbol{P}_k^f \\
&\approx \left(\boldsymbol{I}_n - \boldsymbol{X}_k^f\left(\boldsymbol{Y}_k^f\right)^T\left(\boldsymbol{Y}_k^f\left(\boldsymbol{Y}_k^f\right)^T + \boldsymbol{R}_k\right)^{-1}\boldsymbol{H}_k\right)\boldsymbol{X}_k^f\left(\boldsymbol{X}_k^f\right)^T \\
&\approx \left(\boldsymbol{X}_k^f - \boldsymbol{X}_k^f\left(\boldsymbol{Y}_k^f\right)^T\left(\boldsymbol{Y}_k^f\left(\boldsymbol{Y}_k^f\right)^T + \boldsymbol{R}_k\right)^{-1}\boldsymbol{H}_k\boldsymbol{X}_k^f\right)\left(\boldsymbol{X}_k^f\right)^T \\
&\boxed{\approx \boldsymbol{X}_k^f\left(\boldsymbol{I}_m - \left(\boldsymbol{Y}_k^f\right)^T\left(\boldsymbol{Y}_k^f\left(\boldsymbol{Y}_k^f\right)^T + \boldsymbol{R}_k\right)^{-1}\boldsymbol{Y}_k^f\right)\left(\boldsymbol{X}_k^f\right)^T.}
\end{aligned} \tag{2.54}$$

Consequently, we can set $\boldsymbol{X}_k^a = \boldsymbol{X}_k^f\left(\boldsymbol{I}_m - \left(\boldsymbol{Y}_k^f\right)^T\left(\boldsymbol{Y}_k^f\left(\boldsymbol{Y}_k^f\right)^T + \boldsymbol{R}_k\right)^{-1}\boldsymbol{Y}_k^f\right)^{-1/2}$. Fortunately, we notice that $\boldsymbol{X}_k^a$ comprises the same expression as in equation (2.49) that can be simplified with the Sherman-Morrison-Woodbury formula:

$$
\begin{aligned}
\boldsymbol{X}_k^a &= \boldsymbol{X}_k^f \left(\boldsymbol{I}_m - \left(\boldsymbol{I}_m + \left(\boldsymbol{Y}_k^f\right)^T \boldsymbol{R}_k^{-1} \boldsymbol{Y}_k^f \right)^{-1} + \left(\boldsymbol{Y}_k^f\right)^T \boldsymbol{R}_k^{-1} \boldsymbol{Y}_k^f \right)^{1/2} \\
&= \boldsymbol{X}_k^f \left[\left(\boldsymbol{I}_m + \left(\boldsymbol{Y}_k^f\right)^T \boldsymbol{R}_k^{-1} \boldsymbol{Y}_k^f \right)^{-1} \left(\left(\boldsymbol{I}_m + \left(\boldsymbol{Y}_k^f\right)^T \boldsymbol{R}_k^{-1} \boldsymbol{Y}_k^f \right) - \left(\boldsymbol{Y}_k^f\right)^T \boldsymbol{R}_k^{-1} \boldsymbol{Y}_k^f \right) \right]^{1/2} \\
&= \boldsymbol{X}_k^f \left[\left(\boldsymbol{I}_m + \left(\boldsymbol{Y}_k^f\right)^T \boldsymbol{R}_k^{-1} \boldsymbol{Y}_k^f \right)^{-1} \left(\boldsymbol{I}_m + \underbrace{\left(\boldsymbol{Y}_k^f\right)^T \boldsymbol{R}_k^{-1} \boldsymbol{Y}_k^f - \left(\boldsymbol{Y}_k^f\right)^T \boldsymbol{R}_k^{-1} \boldsymbol{Y}_k^f}_{=0_{\mathbb{R}^{m \times m}}} \right) \right]^{1/2} \\
&\boxed{= \boldsymbol{X}_k^f \underbrace{\left(\boldsymbol{I}_m + \left(\boldsymbol{Y}_k^f\right)^T \boldsymbol{R}_k^{-1} \boldsymbol{Y}_k^f \right)^{-1/2}}_{\boldsymbol{T}_k},} \qquad (2.55)
\end{aligned}
$$

where $\boldsymbol{T}_k$ is known as the ensemble transform matrix.

The members of the posterior ensemble can be derived accordingly:
$\forall i \in [\![1, m]\!]$:

$$
\boxed{\boldsymbol{x}_k^a = \overline{\boldsymbol{x}}_k^{a,i} + \sqrt{m-1}\boldsymbol{X}_k^f \left[\boldsymbol{T}_k\right]_i .} \qquad (2.56)
$$

This version of the ETKF algorithm we derived is summarized in Algorithm 2.4.

The smooth and seamless derivation of the ETKF algorithm might suggest it is a superior and flawless data assimilation method. However, and as the adage says, "*there is no free lunch*" and therefore the ETKF is not the odd one out. More specifically, a primary challenge arises when attempting to faithfully represent real-world error covariance matrices of $\mathbb{R}^{n \times n}$ (with n on the order of $\mathcal{O}(10^6 - 10^9)$) using only a few tens up to a few hundreds of ensemble members: in such cases, large sampling errors are ineluctable. Even if the so-called **unstable directions** - introduced in section 4.2.1, with further details provided in Strogatz (1994); Legras and Vautard (1996); Carrassi *et al.* (2022) - are generally much lower than the state space dimension, accurately capturing these directions remains a challenge for any ensemble method due to practical limits on the number of ensemble members. The computational burden to forecast thousands or millions of members at every cycle is unaffordable nearly all the time. Also, the Gaussian assumption made in the

Algorithm 2.4: Ensemble Transform Kalman Filter

```
Inputs:
    Background ensemble $\boldsymbol{E}_0^f = \left\{\boldsymbol{x}_0^1, \ldots, \boldsymbol{x}_0^m\right\} \in \mathbb{R}^{n\times m}$ ;
    Observations $\{\boldsymbol{y}_0, \ldots, \boldsymbol{y}_T\} \in \mathbb{R}^{p\times(T+1)}$ ;

1 for $k = 0, 1, \ldots, T$ do
                  Analysis step
      // Compute ensemble forecast mean
2     $\overline{\boldsymbol{x}}_k^f = \boldsymbol{E}_k^f \mathbf{1}/m$
      // Compute anomalies
3     $\boldsymbol{X}_k^f = \left(\boldsymbol{E}_k^f - \overline{\boldsymbol{x}}_k^f \mathbf{1}^T\right)/\sqrt{m-1}$
4     $\boldsymbol{Y}_k^f = \left(\boldsymbol{\mathcal{H}}_k\left(\boldsymbol{E}_k^f\right) - \boldsymbol{\mathcal{H}}_k\left(\overline{\boldsymbol{x}}_k^f\right)\mathbf{1}^T\right)/\sqrt{m-1}$
      // Compute the transform matrix $\boldsymbol{T}_k$
5     $\boldsymbol{T}_k = \left(\boldsymbol{I}_m + \left(\boldsymbol{Y}_k^f\right)^T \boldsymbol{R}_k^{-1} \boldsymbol{Y}_k^f\right)^{-1}$
      // Compute $\boldsymbol{w}^a$
6     $\boldsymbol{w}^a = \boldsymbol{T}_k \left(\boldsymbol{Y}_k^f\right)^T \boldsymbol{R}_k^{-1} \left(\boldsymbol{y}_k - \boldsymbol{\mathcal{H}}_k\left(\overline{\boldsymbol{x}}_k^f\right)\right)$
      // Generate posterior ensemble
7     $\boldsymbol{E}_k^a = \overline{\boldsymbol{x}}_k^f \mathbf{1}^T + \boldsymbol{X}_k^f \left(\boldsymbol{w}_k^a \mathbf{1}^T + \sqrt{m-1}\boldsymbol{T}_k^{1/2}\right)$
                  Propagation step
      // Forecast posterior ensemble
8     $\boldsymbol{E}_{k+1}^f = \boldsymbol{\mathcal{M}}_{k+1}(\boldsymbol{E}_k^a)$
9 end
```

context of ensemble data assimilation is unlikely to hold for most real-world applications. Therefore, if used as described in Algorithm 2.4, the ETKF algorithm could yield estimates that diverge from the ground truth.

To mitigate these sampling errors, two strategies known as **localization** and **inflation** offer complementary solutions:

Localization addresses the issue by two means:

- Spatial localization: it involves performing local analyses that assimilate only nearby observations for the considered spatial window.
- Covariance localization: this technique post-processes the error covariance matrix $\boldsymbol{P}_k^f$ by dampening spurious long-range correlations that low-rank approximations introduce. A Schur product between $\boldsymbol{P}^f$ and a well-defined sparse matrix effectively eliminates unwanted non-zero off-diagonal terms.

Inflation serves as an additional countermeasure to sampling errors which may still accumulate over successive assimilation cycles. By inflating the covariance matrix $\boldsymbol{P}_k^a$ or the posterior ensemble with a scalar factor, named **inflation**, this approach helps to avoid divergence of the filter. The inflation parameter has empirically proved effective when being slightly above 1. Inflation is case-dependent and as such can be fine-tuned.

When including an inflation parameter, denoted by λ, in Algorithm 2.4, we get Algorithm 2.5.

2.2.2.3 ETKF-Q: an ensemble Kalman Filter algorithm that accounts for model error

In the following, we detail a specific version of the ETKF algorithm that accounts for model errors $\boldsymbol{Q}_k$, and therefore named **ETKF-Q** (Fillion *et al.*, 2020). What we call ETKF-Q method precisely denotes the **IEnKS-Q** algorithm as detailed in (Fillion *et al.*, 2020, Algorithm 4.1) with parameters (L=0, K=0, S=1, G=0, one Gauss Newton loop, transform version).

In equation (2.51), we prove that the decomposition of $\boldsymbol{x}_k^a$ is not unique due to the rank deficient matrix $\boldsymbol{X}_k^f$. This yields an ill-defined change of variables in

Algorithm 2.5: Ensemble Transform Kalman Filter with inflation

```
Inputs:
    Background ensemble $\boldsymbol{E}_0^f = \left\{\boldsymbol{x}_0^1, \dots, \boldsymbol{x}_0^m\right\} \in \mathbb{R}^{n\times m}$ ;
    Observations $\{\boldsymbol{y}_0, \dots, \boldsymbol{y}_T\} \in \mathbb{R}^{p\times(T+1)}$ ;
    Inflation parameter $\lambda \in \mathbb{R}$.
1 for $k = 0, 1, \dots, T$ do
                    Analysis step
      // Compute ensemble forecast mean
2     $\overline{\boldsymbol{x}}_k^f = \boldsymbol{E}_k^f \mathbf{1}/m$
      // Compute anomalies
3     $\boldsymbol{X}_k^f = \left(\boldsymbol{E}_k^f - \overline{\boldsymbol{x}}_k^f \mathbf{1}^T\right)/\sqrt{m-1}$
4     $\boldsymbol{Y}_k^f = \left(\boldsymbol{\mathcal{H}}_k\left(\boldsymbol{E}_k^f\right) - \boldsymbol{\mathcal{H}}_k\left(\overline{\boldsymbol{x}}_k^f\right)\mathbf{1}^T\right)/\sqrt{m-1}$
      // Compute the transform matrix $\boldsymbol{T}_k$
5     $\boldsymbol{T}_k = \left(\boldsymbol{I}_m + \left(\boldsymbol{Y}_k^f\right)^T \boldsymbol{R}_k^{-1} \boldsymbol{Y}_k^f\right)^{-1}$
      // Compute $\boldsymbol{w}^a$
6     $\boldsymbol{w}^a = \boldsymbol{T}_k \left(\boldsymbol{Y}_k^f\right)^T \boldsymbol{R}_k^{-1}\left(\boldsymbol{y}_k - \boldsymbol{\mathcal{H}}_k\left(\overline{\boldsymbol{x}}_k^f\right)\right)$
      // Generate posterior ensemble
7     $\boldsymbol{E}_k^a = \overline{\boldsymbol{x}}_k^f \mathbf{1}^T + \boldsymbol{X}_k^f \boldsymbol{w}_k^a \mathbf{1}^T + \lambda \times \sqrt{m-1}\boldsymbol{X}_k^f \boldsymbol{T}_k^{1/2}$
                    Propagation step
      // Forecast posterior ensemble
8     $\boldsymbol{E}_{k+1}^f = \boldsymbol{\mathcal{M}}_{k+1}(\boldsymbol{E}_k^a)$
9 end
```

the ensemble space that has to be fixed with the so-called gauge-fixing term (Bocquet and Sakov, 2014). As an alternative Fillion *et al.* (2020) introduces deviation matrices to overcome this problem.

Definition 2.2 *Deviation matrix. A deviation matrix* $\boldsymbol{\Delta}$ *of a symmetric semi-definite-positive matrix* $\boldsymbol{\Sigma}$ *is an* ***injective*** *factor verifying:* $\boldsymbol{\Delta}\boldsymbol{\Delta}^T = \boldsymbol{\Sigma}$*. A deviation matrix of an ensemble is a deviation matrix of its sample covariance matrix.*

Therefore, we aim to find a deviation matrix $\boldsymbol{\Delta}_k$ of $\boldsymbol{P}_k^f$ such that the formulation in equation (2.53) yields a unique estimate $\boldsymbol{x}_k^a$. Hence, we apply (Fillion *et al.*, 2020, Proposition 3.2) to $\boldsymbol{x}_k^a$ in order to ensure such a requirement. Thereby, there exists a unique vector $\boldsymbol{w}_k^a \in \mathbb{R}^{m-1}$ such that:

$$\boxed{\boldsymbol{x}_k^a = \overline{\boldsymbol{x}}_k^f + \boldsymbol{\Delta}_k \boldsymbol{w}_k^a,} \tag{2.57}$$

and

$$\boxed{\begin{aligned} \mathbb{E}\left[\boldsymbol{w}_k^a\right] &= \boldsymbol{0}_{m-1}, & (2.58a)\\ \mathbb{C}\left[\boldsymbol{w}_k^a\right] &= \boldsymbol{I}_{m-1}, & (2.58b) \end{aligned}}$$

where notation $\mathbb{C}\left[\cdot\right]$ is used to represent the covariance operator.

The issue that remains is to calculate a deviation matrix $\boldsymbol{\Delta}_k \in \mathbb{R}^{n\times(m-1)}$ of $\boldsymbol{P}_k^f$. We therefore rely on (Fillion *et al.*, 2020, Proposition 3.3) which reads:

Proposition 2.1 *(Deviation matrix and ensemble construction): Let* $n, m, l \in \mathbb{N}$ *such that* $n \geq m, l = m-1$*. Let* $\boldsymbol{U}_m \in \mathbb{R}^{m\times l}$ *such that* $\left[\frac{\boldsymbol{1}_m}{\sqrt{m}} \;\; \boldsymbol{U}_m\right] \in \mathbb{R}^{m\times m}$ *be an orthonormal matrix. If* $\boldsymbol{E} \in \mathbb{R}^{n\times m}$ *is a full column rank ensemble, then the mean* $\boldsymbol{\mu} \in \mathbb{R}^n$ *and a deviation matrix* $\boldsymbol{\Delta} \in \mathbb{R}^{n\times l}$ *of* $\boldsymbol{E}$*:*

$$\left[\boldsymbol{\mu} \;\; \boldsymbol{\Delta}\right] = \boldsymbol{E} \times \left[\frac{\boldsymbol{1}_m}{m} \;\; \frac{\boldsymbol{U}_m}{\sqrt{l}}\right]. \tag{2.59}$$

Conversely, if $\boldsymbol{\mu} \in \mathbb{R}^n$ *and* $\boldsymbol{\Delta} \in \mathbb{R}^{n\times l}$ *then the ensemble* $\boldsymbol{E} \in \mathbb{R}^{n\times m}$ *defined by*

$$\boldsymbol{E} = \left[\boldsymbol{\mu} \;\; \boldsymbol{\Delta}\right] \times \left[\boldsymbol{1}_m \;\; \sqrt{l}\boldsymbol{U}_m\right]^T \tag{2.60}$$

has $\boldsymbol{\mu}$ *as sample mean and* $\boldsymbol{\Delta}\boldsymbol{\Delta}^T$ *as sample covariance matrix.*

With equation (2.59), we can compute $\boldsymbol{\Delta}_k$, a deviation matrix of $\boldsymbol{P}_k^f$ (we remind that $l = m-1$):

$$\boxed{\boldsymbol{\Delta}_k = \left[\boldsymbol{x}_k^{f,1}, \boldsymbol{x}_k^{f,2}, \dots, \boldsymbol{x}_k^{f,m}\right] \frac{\boldsymbol{U}_m}{\sqrt{m-1}}.} \tag{2.61}$$

The matrix $\boldsymbol{U}_m \in \mathbb{R}^{m \times (m-1)}$, is such that $\left[\frac{\mathbf{1}_m}{\sqrt{m}} \ \boldsymbol{U}_m\right]$ is orthonormal (where $\mathbf{1}_m$ denotes the m-length vector $[1, 1, \dots, 1]^T$). It is worth mentioning that $\boldsymbol{U}_m$ can be constructed thanks to Householder's rotations.

When propagating through time, we know that our model $\boldsymbol{\mathcal{M}}_k$ is not perfect and has an intrinsic error denoted by $\boldsymbol{\eta}_k$ (see 2.27a). However, we have not yet included this particular knowledge in our analysis, keeping the erroneous prediction as it is. Some approaches attempt to leverage this information in order to perform a model error correction and thus improve the quality of the predictions (Sakov *et al.*, 2018; Mitchell and Carrassi, 2015; Sakov and Bocquet, 2018; Mandel *et al.*, 2016; Amezcua *et al.*, 2017; Sommer and Janjić, 2018).

We now come to the core of the ETKF-Q algorithm, by taking model error into account in the expression of the covariance matrix of $\boldsymbol{x}_k^t$ (we remind that $\boldsymbol{x}_k^t$ denotes the ground truth physical state):

$$\boldsymbol{x}_k^t = \boldsymbol{\mathcal{M}}_k\left(\boldsymbol{x}_{k-1}^t\right) + \boldsymbol{\eta}_k, \tag{2.62}$$

$$\mathbb{C}\left[\boldsymbol{x}_k^t | \boldsymbol{y}_{0:k-1}\right] = \mathbb{C}\left[\boldsymbol{\mathcal{M}}_k\left(\boldsymbol{x}_{k-1}^t\right) + \boldsymbol{\eta}_k | \boldsymbol{y}_{0:k-1}\right], \tag{2.63}$$

where $\boldsymbol{y}_{0:j}$ denotes the sequence of all the observations from time 0 to time j.

We suppose that $\boldsymbol{\eta}_k \ \forall k \in [\![0, T-1]\!]$ and $\boldsymbol{x}_0^t$ are mutually independent. Then, as $\boldsymbol{\mathcal{M}}_k\left(\boldsymbol{x}_{k-1}^t\right)$ is a function of $\boldsymbol{x}_0^t$ and of $\boldsymbol{\eta}_0, \boldsymbol{\eta}_1, \dots, \boldsymbol{\eta}_{k-1}$, it emerges that $\boldsymbol{\mathcal{M}}_k\left(\boldsymbol{x}_{k-1}^t\right)$ and $\boldsymbol{\eta}_k$ are independent, which yields that

$$\mathbb{C}\left[\boldsymbol{x}_k^t | \boldsymbol{y}_{0:k-1}\right] = \mathbb{C}\left[\boldsymbol{\mathcal{M}}_k\left(\boldsymbol{x}_{k-1}^t\right) | \boldsymbol{y}_{0:k-1}\right] + \mathbb{C}\left[\boldsymbol{\eta}_k | \boldsymbol{y}_{0:k-1}\right]. \tag{2.64}$$

We also assume that propagation and observation errors $\boldsymbol{\eta}_k$ and $\boldsymbol{\varepsilon}_k$ are mutually independent $\forall k \in [\![0, T-1]\!]$. We then have $\mathbb{C}\left[\boldsymbol{\eta}_k | \boldsymbol{y}_{0:k-1}\right] = \mathbb{C}\left[\boldsymbol{\eta}_k\right] = \boldsymbol{Q}_k$. This yields:

$$\mathbb{C}\left[\boldsymbol{x}_k^t | \boldsymbol{y}_{0:k-1}\right] = \mathbb{C}\left[\boldsymbol{\mathcal{M}}_k(\boldsymbol{x}_{k-1}) | \boldsymbol{y}_{0:k-1}\right] + \boldsymbol{Q}_k. \tag{2.65}$$

However, $\mathbb{C}\left[\mathcal{M}_k(\boldsymbol{x}_{k-1}) | \boldsymbol{y}_{0:k-1}\right]$ has been empirically approximated by $\boldsymbol{P}_k^f =$

$\boldsymbol{\Delta}_k \boldsymbol{\Delta}_k^T$.

Hence we obtain that

$$\boxed{\mathbb{C}\left[\boldsymbol{x}_k^t | \boldsymbol{y}_{0:k-1}\right] \approx \boldsymbol{\Delta}_k \boldsymbol{\Delta}_k^T + \boldsymbol{Q}_k.} \tag{2.66}$$

Deviation matrices of $\boldsymbol{\Delta}_k \boldsymbol{\Delta}_k^T + \boldsymbol{Q}_k$ are supposed to lie in $\mathbb{R}^{n \times n}$, but since a $n \times l$ deviation matrix is required for the next cycle, a reduction must be performed.

As $\boldsymbol{\Delta}_k \boldsymbol{\Delta}_k^T + \boldsymbol{Q}_k$ is symmetric (as a sum of symmetric matrices), its eigendecomposition by using the first $\ell = m - 1$ dominant eigenvectors yields $\boldsymbol{V}_k \in \mathbb{R}^{n \times (m-1)}$ and $\boldsymbol{\Lambda}_k \in \mathbb{R}^{(m-1) \times (m-1)}$ such that:

$$\boxed{\left(\boldsymbol{\Delta}_k \boldsymbol{\Delta}_k^T + \boldsymbol{Q}\right) \boldsymbol{V}_k \approx \boldsymbol{V}_k \boldsymbol{\Lambda}_k.} \tag{2.67}$$

One notices that this approximation is the best one in the matrix Frobenius norm.

A square root approximation of $\boldsymbol{\Delta}_k \boldsymbol{\Delta}_k^T + \boldsymbol{Q}_k$ is thus given by $\boldsymbol{V}_k \boldsymbol{\Lambda}_k^{1/2}$. We hence update $\boldsymbol{\Delta}_k = \boldsymbol{V}_k \boldsymbol{\Lambda}_k^{1/2}$. Then, equation (2.60) enables to update ensemble $\boldsymbol{E}_k$ according to this new statistic:

$$\boxed{\boldsymbol{E}_k = \overline{\boldsymbol{x}}_k^f + \boldsymbol{\Delta}_k \sqrt{m-1} \boldsymbol{U}_m^T.} \tag{2.68}$$

Similarly, we apply equation (2.59) to the observation ensemble to produce $\boldsymbol{Y}_k^f$ which is analogous to the observation anomalies in the regular ETKF algorithm:

$$\boldsymbol{Y}_k^f = \left[\boldsymbol{\mathcal{H}}_k(\boldsymbol{x}_k^1), \boldsymbol{\mathcal{H}}_k(\boldsymbol{x}_k^2), \ldots, \boldsymbol{\mathcal{H}}_k(\boldsymbol{x}_k^m)\right] \frac{\boldsymbol{U}_m}{\sqrt{m-1}}. \tag{2.69}$$

For the remainder of the algorithm, we straightforwardly apply the standard ETKF. The ETKF-Q approach that we derived is summarized in Algorithm 2.6 (we remind that some notations are borrowed from Fillion *et al.* (2020)).

Algorithm 2.6: ETKF-Q algorithm

```
Inputs:
    Background ensemble $\boldsymbol{E}_0^f = \{\boldsymbol{x}_0^1, \dots, \boldsymbol{x}_0^m\} \in \mathbb{R}^{n\times m}$ ;
    Observations $\{\boldsymbol{y}_0, \dots, \boldsymbol{y}_T\} \in \mathbb{R}^{p\times(T+1)}$ ;
    Inflation parameter $\lambda \in \mathbb{R}$.
Initialization:
    Construct $\boldsymbol{U}_m$ matrix such that $\left[\frac{\mathbf{1}_m}{\sqrt{m}} \ \boldsymbol{U}_m\right]$ is orthonormal ;
    Define $\boldsymbol{\mathcal{U}} = \left[\frac{\mathbf{1}_m}{m} \ \frac{\boldsymbol{U}_m}{\sqrt{m-1}}\right]$ ;
1  for $k = 0, 1, \dots, T$ do
                        Analysis step
       // Mean and deviation matrix of forecast ensemble
2      $\left[\overline{\boldsymbol{x}}_k^f \ \ \boldsymbol{\Delta}_k\right] = \boldsymbol{E}_k^f \times \boldsymbol{\mathcal{U}}$
       // Calculate eigenpairs of the error covariance matrix
3      $\left(\boldsymbol{\Delta}_k\boldsymbol{\Delta}_k^T + \boldsymbol{Q}_k\right)\boldsymbol{V}_k \approx \boldsymbol{V}_k\boldsymbol{\Lambda}_k$
4      where $\boldsymbol{V}_k \in \mathbb{R}^{n\times(m-1)}$ and $\boldsymbol{\Lambda}_k \in \mathbb{R}^{(m-1)\times(m-1)}$
       // Update the error covariance matrix accordingly
5      $\boldsymbol{\Delta}_k = \boldsymbol{V}_k\boldsymbol{\Lambda}_k^{1/2}$
       // Update the ensemble with the new statistics
6      $\boldsymbol{E}_k^f = \left[\overline{\boldsymbol{x}}_k^f \ \ \boldsymbol{\Delta}_k\right] \times \boldsymbol{\mathcal{U}}^{-1}$
       // Mean and deviation matrix of observation ensemble
7      $\left[\overline{\boldsymbol{y}}_k \ \ \boldsymbol{Y}_k^f\right] = \boldsymbol{\mathcal{H}}_k\left(\boldsymbol{E}_k^f\right) \times \boldsymbol{\mathcal{U}}$
       // Compute transform matrix $\boldsymbol{T}_k \in \mathbb{R}^{(m-1)\times(m-1)}$
8      $\boldsymbol{T}_k\boldsymbol{T}_k^T = \left(\boldsymbol{I}_{m-1} + \left(\boldsymbol{Y}_k^f\right)^T \boldsymbol{R}^{-1}\boldsymbol{Y}_k^f\right)^{-1}$
       // Compute $\boldsymbol{w}^a$
9      $\boldsymbol{w}_k^a = \boldsymbol{T}_k\boldsymbol{T}_k^T\boldsymbol{Y}_k^f\boldsymbol{R}^{-1}(\boldsymbol{y}_k - \overline{\boldsymbol{y}}_k)$
       // Generate posterior ensemble
10     $\boldsymbol{E}_k^a = \overline{\boldsymbol{x}}_k\mathbf{1}_m^T + \lambda \times \boldsymbol{\Delta}_k\left(\boldsymbol{w}_k^a\mathbf{1}_m + \sqrt{m-1}\boldsymbol{T}_k\right)$
                        Propagation step
       // Forecast posterior ensemble
11     $\boldsymbol{E}_{k+1}^f = \boldsymbol{\mathcal{M}}_{k+1}(\boldsymbol{E}_k^a)$
12 end
```

CHAPTER 3

Literature review: enhance data assimilation by incorporating deep learning models

This chapter explores the emerging interactions and hybridizations between deep learning and data assimilation, structured as a literature review. In recent years, deep learning has distinguished itself as a formidable tool across various domains, including numerical weather prediction, natural language processing, and pattern recognition, among others. The core strength of neural networks is threefold and resides in their ability to: (i) extract meaningful information from large raw datasets, (ii) derive inferential rules, and (iii) deliver fast predictions in real-time, markedly outpacing traditional computational methods.

Data assimilation in real-world applications, such as numerical weather prediction and geosciences, often deals with high-dimensional, nonlinear and multi-scale physical systems, making computations exceedingly demanding. The state space dimension can reach $\mathcal{O}(10^7 - 10^9)$, with observational data typically one to two orders of magnitude smaller — but still computationally significant. Managing such high-dimensional data constrains the fineness of the spatial discretization and requires approximations when performing linear algebra operations or when solving optimization problems. Besides, some data assimilation components are only partially known, introducing potential errors: for example, covariance matrices may be imprecise, small-scale physical processes within the dynamics of interest might be unaccounted for (necessitating *ad-hoc* parameterizations), and physical models are very likely to contain inaccuracies.

As a result, hybridizing data assimilation with deep learning has emerged as a promising way to achieve a faster and more accurate assimilation process.

The organization of this chapter is as follows:

- section 3.1 outlines the historical background of artificial intelligence, emphasizing deep learning. Theoretical aspects of neural networks along with different types of neural architectures are detailed.
- section 3.2 is a literature review about the replacement of some data assimilation components by neural networks.
- section 3.3 highlights groundbreaking progress in numerical weather prediction (NWP), showcasing novel data-driven models that, for the first time, rival or even surpass the performances of operational data assimilation systems for short-term and medium range forecasts.
- Section section 3.4 presents research works that achieved to completely replace the traditional data assimilation process with deep learning solutions.

3.1 Essentials of deep learning

3.1.1 Historical introduction to artificial intelligence

John McCarthy, who coined the term **Artificial Intelligence** (AI), provides the following definition in McCarthy (2007):

> [Artificial Intelligence] is the science and engineering of making intelligent machines, especially intelligent computer programs. It is related to the similar task of using computers to understand human intelligence, but AI does not have to confine itself to methods that are biologically observable. Intelligence is the computational part of the ability to achieve goals in the world. Varying kinds and degrees of intelligence occur in people, many animals and some machines.

Before artificial intelligence led to the revolution of **neural networks** (NNs) and **deep learning** (DL), it firstly represented a conceptual, even philosophical, leap in the evolution of programming and computational problem-solving. Traditional programming paradigms were primarily algorithmic procedures, so that they were designed to carry out well-defined and repetitive tasks. This ranged from basic operations like simple mathematical and physical computations (*e.g.*, aerodynamic calculations, solving differential equations), and storage of business or government recordings (*e.g.*, payroll), to more advanced and complex processes including text editing, email writing, large dimensional numerical simulations, image and video processing, and the management of network communications, among others (Pottenger *et al.*, 2023; Crosley, 2022). The hallmark of these conventional programs is their limitation to operate strictly within a predefined framework, thus lacking the ability for creativity or adaptation beyond their initial programming boundaries.

Artificial intelligence revolutionized computer science by developing systems capable of handling tasks traditionally believed to be within the human domain. AI aims to endow computers with decision-making competences, problem-solving skills, and even the ability to learn from experience.

Whereas traditional programming relies on explicit instructions defined by developers to achieve specific outcomes, AI - particularly through **machine learning** (ML) and deep learning -, involves creating algorithms able to learn patterns, make inferences, or take actions based on data. This effectively allows the software to "learn" and improve over time without being explicitly programmed for each specific task.

This paradigm shift represents a transition from static, rule-based computing to dynamic, adaptive, and increasingly autonomous systems. AI challenges the very notion of programming by sometimes blurring the lines between human and machine capabilities, thereby redefining what programming could mean.

Nowadays, AI covers a wide range of applications such as object detection (Girshick *et al.*, 2014; Ren *et al.*, 2015; Redmon *et al.*, 2016; Redmon and Farhadi, 2018), image segmentation (Long *et al.*, 2015; Ronneberger *et al.*, 2015; Milletari *et al.*, 2016; Chen *et al.*, 2017), image classification (Krizhevsky *et al.*, 2012; Simonyan and Zisserman, 2014; He *et al.*, 2016), natural language processing (NLP) (Mikolov *et al.*, 2013; Vaswani *et al.*, 2017; Devlin *et al.*, 2018), or, physics surrogate modeling (Loh *et al.*, 2018; Raissi *et al.*, 2019; Tang *et al.*, 2020; Lucor *et al.*, 2021), among others.

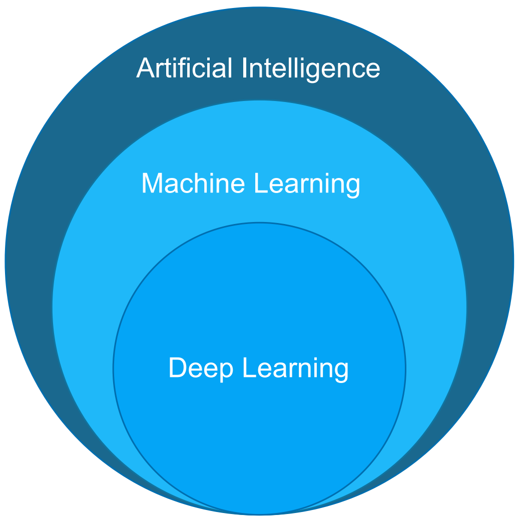


Figure 3.1.1: Nested links between Artificial Intelligence, Machine Learning and Deep Learning. Inspired from: `https://metaverseadmedia.com/`.

The commonly used terms **artificial intelligence**, **machine learning** and **deep learning** share a nested relationship, as illustrated in Figure 3.1.1. More precisely, machine learning - which includes deep learning techniques -, is a subfield of AI that extends beyond just neural networks (NNs) training. ML is a fundamental component of data science, representing the set of statistical methods that enable a computer to perform tasks without the need for explicit instructions (*e.g.*, regression, clustering, and classifications).

Regarding deep learning, it consists in iteratively updating the parameters (referred to as **weights** and **biases**) of a so-called **artificial neural network** (ANN) through a stochastic process. The optimization aims to minimize an objective function, known as the **loss function**, over successive iterations. This process, which is referred to as **learning** or **training**, allows the neural network to progressively improve its performance on a specified task, but requires a substantial amount of data. Since neural networks are inherently built upon data, they are often termed data-driven models. Neural networks' structure is inspired by the architecture of the human brain and, thus, aim to artificially mimic the learning process strategy of acquiring knowledge and extracting information given previously collected data.

Deep learning's origins date back to the 1940s when McCulloch and Pitts (1943) introduced the first mathematical model of a neural network, with binary inputs and outputs, but no optimization strategy. The first concrete advancements in machine learning emerged later in the 1950s with the development of a championship-level chess-playing software by Arthur Samuel, an IBM engineer, who is also credited with popularizing the term machine learning. Rather than exhaustively exploring the tree of possible moves, the chess program aims to maximize the chances of winning, using the so-called **alpha-beta pruning** algorithm (Knuth and Moore, 1975).

Deep learning methods gained incredible traction in data science alongside the remarkable increase in computational power over the last few decades. Notably, the development of **Graphical Processing Unit** (GPU) has substantially reduced training times due to its parallel computing capabilities. Figure 3.1.2 displays the first artificial neural network developed by Marvin Minsky and Dean Edmonds in 1951 (refer to Gladchuk (2020)), together with the Nvidia H100 GPU launched on March 21st and offered in three versions, the SXM, the PCIe, and the NVL (in ascending order of performance). At the time of writing this thesis, it stands as the most powerful GPU used in data centers for large-scale AI projects, boasting from 756 up to $1,979$ teraflops performance (with a TensorFloat-32 precision format), from 2TB/s to 7.8TB/s memory bandwidth, and from 600 GB/s to 900 GB/s interconnects, depending on the version[(1)].

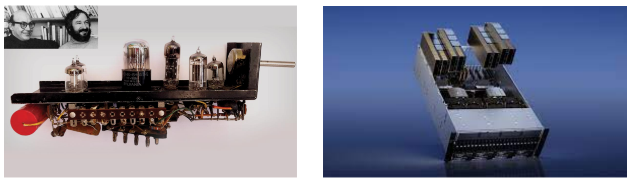


Figure 3.1.2: Left: the Stochastic Neural Analog Reinforcement Calculator (SNARC) machine (40 interconnected neurons), known as the first artificial neural network in history. Right: Nvidia H100 GPU, released in March 2023 and becoming to be widespread in AI projects.

In general, machine learning and deep learning algorithms can be assigned to one of the following categories: **supervised** learning, **unsupervised** learning and **reinforcement** learning. In supervised learning (LeCun *et al.*, 2015; Goodfellow *et al.*, 2016), the algorithm is trained on a labeled dataset, where each input is paired with the corresponding desired output. The goal is for the model to learn

(1) https://www.nvidia.com/en-us/data-center/h100/

the mapping from inputs to outputs, allowing it to infer the correct label of new, *i.e.*, unseen, data. For instance, in classification tasks, neural networks have proven to be effective in automatically assigning the correct digit to an input image from the MNIST dataset[(2)] (LeCun and Cortes, 2005): MNIST comprises images of handwritten digits with dimensions of 28×28.
Unsupervised learning (Hinton and Salakhutdinov, 2006; Van der Maaten and Hinton, 2008) involves training a model on an unlabeled dataset where the algorithm must discover patterns, structures, or relationships within the data without explicit guidance. The objective is often to uncover hidden representations or groupings in the absence of predefined output labels.
Reinforcement learning (Mnih *et al.*, 2015; Sutton and Barto, 2018) is a type of learning in which an agent learns to make decisions by interacting with an environment. The agent receives feedback in the form of rewards or penalties based on its actions. The goal is to learn a policy that maximizes the cumulative reward over time.
Given the context of our study, we will exclusively focus on the supervised and **autoassociative self-supervised** (Hinton and Salakhutdinov, 2006; Vincent *et al.*, 2008) learning cases. Autoassociative self-supervised learning is an intermediate training framework, in-between supervised and unsupervised learning: it does not require labels in the conventional sense, it indeed generates its own supervisory targets from the input data. **Autoencoders** (see section 3.1.5) are a prime example of neural network architectures that are trained in a self-supervised manner, and that will be developed and discussed throughout this manuscript.

3.1.2 Perceptron and Multi-Layer Perceptron

From a mathematical standpoint, and within the framework of this thesis, a neural network can be defined by:

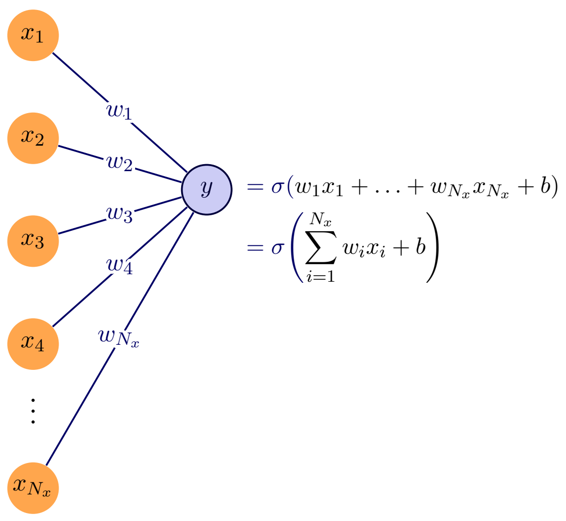
$$\boxed{\begin{aligned} f_\theta \colon \mathcal{X} &\to \mathcal{Y} \\ \boldsymbol{x} &\mapsto f_\theta\left(\boldsymbol{x}\right), \end{aligned}} \tag{3.1a}$$

where $\mathcal{X}$ and $\mathcal{Y}$ are typically vector spaces. In the following, they are denoted by $\mathbb{R}^{N_x}$ and $\mathbb{R}^{N_y}$, respectively. As for $\theta \in \mathbb{R}^d$, it represents the neural network's parameters, namely the weights and the biases.

(2) https://www.kaggle.com/datasets/hojjatk/mnist-dataset

To clearly distinguish the dimension of the data assimilation problem from the input dimension of a neural network, we use distinct notations for the variable $\boldsymbol{x}$ in chapter 2 and chapter 3. In the former, $\boldsymbol{x}$ lies in $\mathbb{R}^n$, while in the latter, $\boldsymbol{x} \in \mathbb{R}^{N_x}$. This distinction is intended to indicate that the variable $\boldsymbol{x}$ represents different quantities in each chapter. As chapter 4 and chapter 5 will later introduce our latent data assimilation method, which integrates deep learning within a data assimilation framework, the variable $\boldsymbol{x}$ will be common to both data assimilation and neural networks equations.

The most elementary neural network structure, known as the **perceptron** (Rosenblatt, 1958; Rosenblatt *et al.*, 1962; LeCun *et al.*, 2015; Goodfellow *et al.*, 2016), is depicted in Figure 3.1.3: $\boldsymbol{x} \in \mathbb{R}^{N_x}$ denotes the input vector, $\boldsymbol{w} \in \mathbb{R}^{N_x}$ represents the weights, $b \in \mathbb{R}$ is the bias, and σ denotes the so-called **activation function** (LeCun *et al.*, 2015; Goodfellow *et al.*, 2016). Activation functions are crucial in deep learning as they represent the nonlinear part of a neural network, all the other mathematical computations being vector-matrix and matrix-matrix operations as detailed later. Figure 3.1.4 presents the graphs of commonly utilized activation functions: *sigmoid* (Rumelhart *et al.*, 1986), *tanh*, *ReLU* (Rectified Linear Unit)(Nair and Hinton, 2010; Glorot *et al.*, 2011), *Leaky ReLU*, and *ELU* (Exponential Linear Unit) (Clevert *et al.*, 2015). Originally, activation functions were designed to simulate the brain's decision-making process, determining whether a neuron should be fired or not:. For example, as depicted in Figure 3.1.4, the *sigmoid* and *ReLU* functions exhibit no activation (*i.e.*, $\sigma(x_i) \approx 0$) for $x_i < 0$, and propagate the input information x_i forward through the network when $x_i > 0$. To mitigate the dying ReLU issue, variations such *Leaky ReLU*, *ELU* or SiLU (Sigmoid Linear Unit, also known as swish) (Ramachandran *et al.*, 2017) functions, which limit the occurrences of zero activation have been introduced. These variations feature a gentle negative slope instead of a plateau at zero, as illustrated in Figure 3.1.4. Additionally, fully linear neural networks exist, though they are less common. Given these notations, the perceptron, hereafter denoted by f_θ^p, can be mathematically defined as follows:



x_1
x_2
x_3
x_4
x_{N_x}
w_1
w_2
w_3
w_4
w_{N_x}
y
$= \sigma(w_1x_1 + \ldots + w_{N_x}x_{N_x} + b)$
$= \sigma\left(\sum_{i=1}^{N_x} w_i x_i + b\right)$

Figure 3.1.3: Architecture of the perceptron.

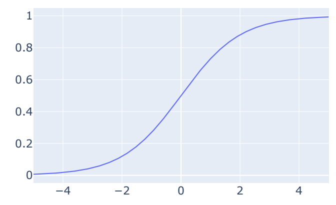
$$\boxed{\begin{aligned} f_\theta^p : \mathbb{R}^{N_x} &\to \mathbb{R} \\ \boldsymbol{x} &\mapsto \sigma(\boldsymbol{w}^T\boldsymbol{x} + b). \end{aligned}} \tag{3.2a}$$

The perceptron serves as the entry point to deep learning. Strictly speaking, the perceptron is considered as a **shallow** neural network (as opposed to **deep** ones): it only consists of one linear transformation - *i.e.*, $\boldsymbol{x} \mapsto \boldsymbol{w}^T\boldsymbol{x} + b$ - followed by one activation function σ. These two mathematical operations define the so-called **layer** of a neural network.

Let us modify the perceptron as defined in equation (3.2a), and let us introduce the weights matrix $\boldsymbol{W} \in \mathbb{R}^{N_x \times N_h}$ and the bias vector $\boldsymbol{b} \in \mathbb{R}^{N_h}$ so that the perceptron now maps from $\mathbb{R}^{N_x}$ to $\mathbb{R}^{N_h}$:

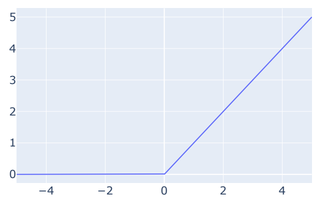
$$\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_{N_h} \end{pmatrix} = \sigma \left[\begin{pmatrix} w_{1,1} & \ldots & w_{1,N_x} \\ w_{2,1} & \ldots & w_{2,N_x} \\ \vdots & \ddots & \vdots \\ w_{N_h,1} & \ldots & w_{N_h,N_x} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_{N_x} \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_{N_h} \end{pmatrix} \right] \tag{3.3}$$

The perceptron's output is therefore a vector, $\boldsymbol{y} \in \mathbb{R}^{N_h}$, that can be fed on another similar perceptron. By stacking several perceptrons one after the other, we



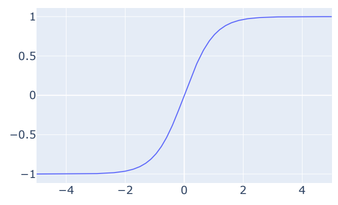
Sigmoid

$$x \mapsto \frac{1}{1+e^{-x}}$$



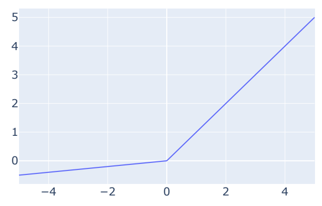
ReLU

$$x \mapsto max(0,x)$$



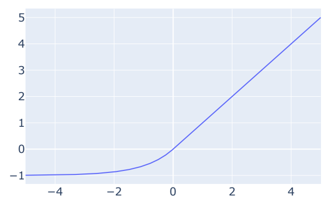
Tanh

$$x \mapsto \frac{e^x - e^{-x}}{e^x + e^{-x}}$$



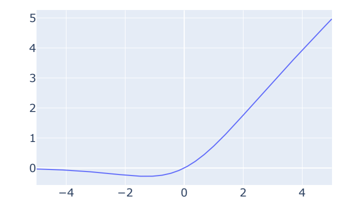
Leaky ReLU

$$x \mapsto \begin{cases} x & \text{if } \geq 0 \\ \alpha x & \text{if } x < 0 \end{cases}$$



ELU

$$x \mapsto \begin{cases} x & \text{if } x \geq 0 \\ \gamma(e^x - 1) & \text{if } x < 0 \end{cases}$$



SiLU (swish)

$$x \mapsto x \times log\left(\frac{1}{1+e^{-x}}\right)$$

Figure 3.1.4: Common activation functions. For the Leaky ReLU and ELU activation functions, α and γ are tunable hyper-parameters that control the steepness of the slope for negative inputs. Their default values in Pytorch are 0.01 and 1, respectively. While we retained $\gamma = 1$, we adjusted α to 0.1 for better visual clarity in the figure.

create the **Multi-Layer Perceptron** (MLP), also known as a **fully-connected feedforward neural network** (Ivakhnenko *et al.*, 1965; Ivakhnenko and Lapa, 1967; Amari, 1967; LeCun *et al.*, 2015; Goodfellow *et al.*, 2016). Figure 3.1.5 illustrates the architecture of a MLP. Notably, the first and the last layers are named **input layer** and **output layer**, respectively, while the remaining ones are called **hidden layers**. Mathematically, we can define the MLP as:

$$\boxed{\begin{aligned} f_\theta \colon \mathbb{R}^{N_x} &\to \mathbb{R}^{N_y} \\ \boldsymbol{x} &\mapsto f_\theta\left(\boldsymbol{x}\right). \end{aligned}} \tag{3.4a}$$

Given $\boldsymbol{x} \in \mathbb{R}^{N_x}$ and a MLP with L layers, f_θ can be expressed as follows:

$$f_\theta\left(\boldsymbol{x}\right) = \sigma^{(L)}\left(\boldsymbol{W}^{(L)}\boldsymbol{x}^{(L-1)} + \boldsymbol{b}^{(L)}\right), \tag{3.5}$$

$$\boldsymbol{x}^{(1)} = \sigma^{(1)}\left(\boldsymbol{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)}\right), \tag{3.6}$$

$$\boldsymbol{x}^{i} = \sigma^{(i)}\left(\boldsymbol{W}^{(i)}\boldsymbol{x}^{(i-1)} + \boldsymbol{b}^{(i)}\right), \tag{3.7}$$

where $\boldsymbol{W}^{(1)} \colon \mathbb{R}^{N_x} \to \mathbb{R}^{N_1}$, $\boldsymbol{W}^{(i)} \colon \mathbb{R}^{N_{(i-1)}} \to \mathbb{R}^{N_i}$, $\forall i \in [\![2, L-2]\!]$, $\boldsymbol{W}^{(L)} \colon \mathbb{R}^{N_{(L-1)}} \to \mathbb{R}^{N_y}$, and biases $\boldsymbol{b}^{(i)}$ are defined correspondingly to match the output dimension of each layer.

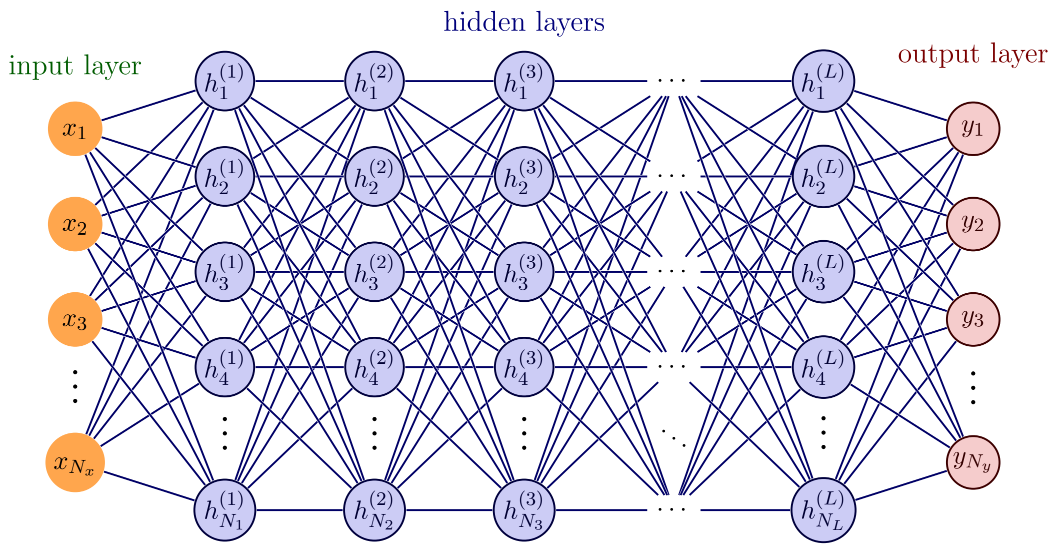


Figure 3.1.5: Illustration of a fully-connected feedforward neural network.

As shown by Cybenko (1989), MLPs are universal function approximators, *i.e.*, they can theoretically learn any nonlinear mapping to any desired accuracy, given a sufficiently large and representative dataset and a suitable architecture (sufficiently deep/wide network).

3.1.3 Convolutional Neural Networks (CNN)

When dealing with images, fully connected neural networks are no longer the most suitable architecture. Images are often high dimensional which poses practical issues in terms of memory and computations. For instance, a standard classification image with dimensions of 256×256 RGB values results in $196,608$ input dimensions. In fully connected networks, where hidden layers are typically at least as large as the input size, this can lead to an extensive number of weights, exceeding billions easily, and necessitating a more appropriate architecture.
The statistical relationship between nearby pixels in images is very informative and should be preserved when feeding a neural network. While fully connected networks treat every input equally without accounting for proximity, **convolutional layers** are designed to recognize local patterns. This enables them to capture spatial dependencies and exploit statistical relationships between neighboring pixels effectively.
The stability of image interpretation under geometric transformations is another

noteworthy aspect. For example, an image of a given object essentially holds the same information for a human being when shifted downwards, upwards, rightwards or leftwards by a few pixels. However, in a fully connected model, this shift alters every input, necessitating the model to redundantly learn pixel patterns for a specific object at different positions. Convolutional layers tackle this by independently processing local regions, ensuring the model generalizes efficiently across spatial variations. They also utilize a limited number of parameters, which allows these networks to capture spatial hierarchies and recognize patterns in a translationally invariant manner, making **Convolutional Neural Networks** (CNNs) (LeCun *et al.*, 1989, 1998; Krizhevsky *et al.*, 2012; Simonyan and Zisserman, 2014; Szegedy *et al.*, 2015; He *et al.*, 2016) well-suited for image-related tasks.

By taking advantage of images structures, CNNs illustrate the concept of **inductive bias** (Mitchell, 1980; LeCun *et al.*, 2015; Goodfellow *et al.*, 2016; Battaglia *et al.*, 2018). Inductive bias represents the set of assumptions, factors, or constraints that steer a model - specifically here, a neural network -, towards effectively generalizing from training data to new unseen observations. Choosing a CNN architecture for image processing over a MLP introduces such a bias, compelling the network to leverage existing knowledge about image structures. This prioritizes a particular way of computing or representing input data, which is shown to be more effective in practice, compared to MLP.

Moreover, the utility of CNNs extends beyond image analysis. They are equally capable when applied to any form of spatial data, such as physical fields where each input **channel** (see definition below) may represent a distinct physical quantity. In these scenarios, CNNs can exploit spatial correlations and the proximity of data points to process and analyze physical phenomena effectively.

Hereafter, we define some architectural features encountered in most convolutional neural networks:

- **kernel**: also known as a filter, it denotes a small matrix used for the convolution operation. It slides over the input data to detect specific patterns or features. Kernels are learnable parameters, analogous to the weights in a fully connected network, that capture relevant information from the input, helping the network recognize hierarchical features.
- **stride**: it refers to the step size with which the kernel moves across the

input data during the convolution operation. A larger stride results in a downsampled output, reducing the spatial dimensions. Stride influences the amount of overlap between the receptive fields of neighboring convolutional operations and affects the spatial resolution of the output.

- **padding**: padding involves adding extra border pixels to the input data before applying convolution operations. This is done to prevent the reduction of spatial dimensions, especially at the edges of the input. Padding ensures that the convolutional operations consider information near the borders, helping to preserve spatial relationships and prevent the loss of valuable features.

- **channel**: in image processing, it refers to the red, green and blue (RGB) color channels. Channels allow the network to process and learn features from distinct input colors. Notably, the information associated with a given channel will not be processed independently of the one contained in the other channels: they will rather be combined throughout the network, enhancing the model's ability to capture complex patterns. More broadly, channels are not confined to images and can, therefore, also represent physical quantities, for example.

- **max pooling**: it is a downsampling operation involving selecting the maximum value from a group of adjacent values in the input. Max pooling helps reduce the spatial dimensions of the data while retaining the most prominent features. It aids in extracting dominant patterns and making the network more computationally efficient.

Figure 3.1.6 shows a schematic representation of a convolutional neural network, which helps visualize how the information contained in an image is extracted and passed forward through the network. With this illustration, one also better picturize how the image representation evolves in terms of size and number of channels, when applying convolution and pooling layers.

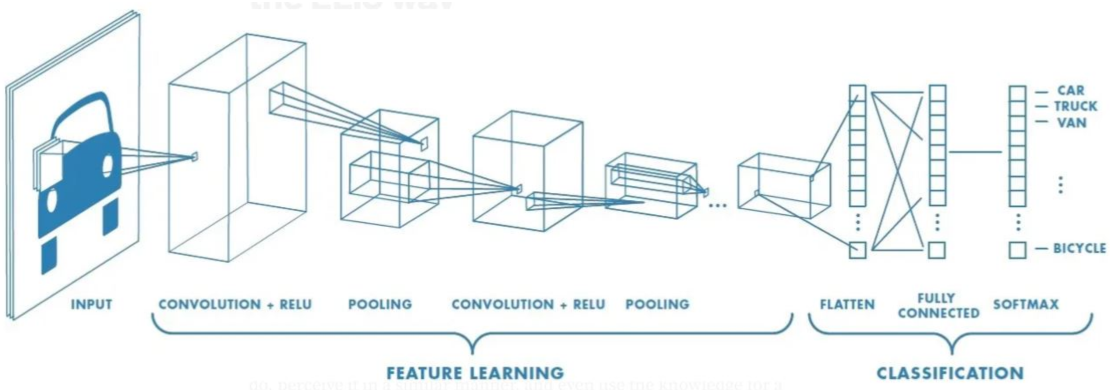


Figure 3.1.6: Illustration of a convolutional neural network architecture. Credit to Saha (2018).

3.1.4 Recurrent Neural Networks (RNN)

In the same way that CNNs are suitable for images, **recurrent neural networks** (RNN) are tailored to efficiently learn on sequential data (*e.g.*, time series). RNNs find applications in a broad range of fields, including language translation, speech recognition and time propagation of physical systems among others. The term "recurrent" was introduced by Rumelhart and McClelland (1987) even if the core idea can be traced back to at least the work of Minsky and Papert (1969). The key aspect of RNNs lies in their internal memory state whose values are modified as input data are processed through the network. This memory or hidden state is often denoted by h_t where the subscript t refers to the t^{th} time the network is called.

The most elementary recurrent neural network is known as the **Elman network** (Elman, 1990). Considering an input sequence $[\boldsymbol{x}_0, \boldsymbol{x}_1, \dots, \boldsymbol{x}_T]$, an initial hidden memory $\boldsymbol{h}_0$, and denoting $\boldsymbol{W}_h$, $\boldsymbol{W}_y$, $\boldsymbol{U}_h$ and $\boldsymbol{U}_y$ the weights matrices, $\boldsymbol{b}_h$ and $\boldsymbol{b}_y$ the bias, σ^h and σ^y the activation functions and $\boldsymbol{y}_t$ the output at iteration t, the Elman network reads:

$$\boldsymbol{h}_t = \sigma_h \left(\boldsymbol{W}_h \boldsymbol{x}_t + \boldsymbol{U}_h \boldsymbol{h}_{t-1} + \boldsymbol{b}_h \right). \tag{3.8}$$

$$\boldsymbol{y}_t = \sigma_y \left(\boldsymbol{W}_y \boldsymbol{h}_t + \boldsymbol{b}_y \right), \tag{3.9}$$

Since a recurrent neural network is fed ordered input pairs, computing the gradients of the weights requires unfolding the neural network in time, necessitating the so-called **backpropagation through time** (BPTT) (Robinson and Fallside, 1987; Werbos, 1988; Mozer, 1995).

Figure 3.1.7 illustrates the way RNNs process input data and update their internal state accordingly. The figure also represents the unrolling operation that is performed during BPTT.

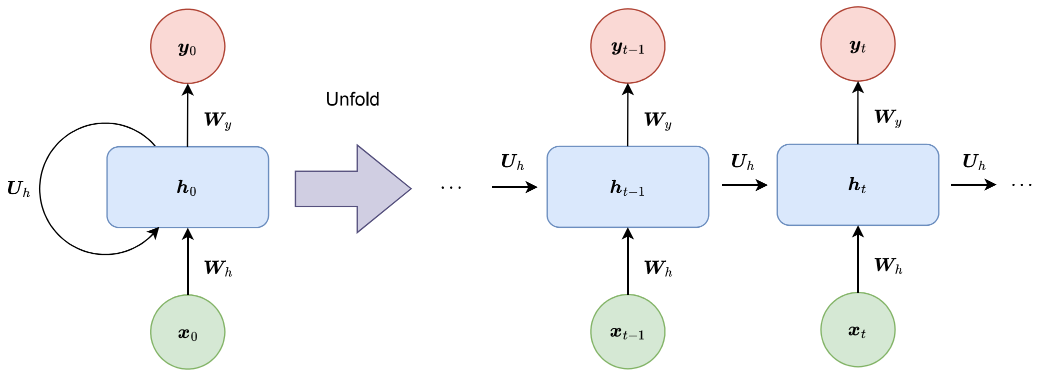


Figure 3.1.7: Illustration of a recurrent neural network (biases are not represented). Inspired from Kale and Altun (2023).

Despite their theoretical appeal for handling sequential data, training RNNs presents practical challenges. Specifically, RNNs are susceptible to vanishing or exploding gradients due to the nature of long data sequences. Vanishing gradients occur when information diminishes across temporal data sequences, hindering the network's learning, whereas exploding gradients result from cumulative gradient errors, making weight updates unmanageable.

Long Short-Term Memory (LSTM) networks are a type of recurrent neural networks architectures designed to overcome these challenges, encountered when learning long-range dependencies in sequential data. Introduced by Hochreiter and Schmidhuber (1997) and followed by Gers *et al.* (1999), LSTMs address the vanishing and exploding gradient problems that traditional RNNs often face: in addition to the hidden state $\boldsymbol{h}_t$, LSTM networks utilize a memory cell $\boldsymbol{c}_t$, equipped with gates that control the flow of information. Contrary to the recurrent networks presented earlier, the output of an LSTM is $\boldsymbol{h}_t$ itself. This dual functionality enables LSTMs to be chained together in sequences, making them powerful tools for modeling time-series data or sequences. There exists different variants of LSTM architectures, and Figure 3.1.8 depicts the most widespread one: it includes an input, an output, and a forget gate, along with a candidate memory cell. These gates allow the model to selectively retain or discard information over time, enabling the network to capture and remember relevant patterns across varying time scales. The forget gate has a direct effect on the amount of information that will be removed from $\boldsymbol{c}_{t-1}$ to $\boldsymbol{c}_t$, while the input gate controls how much data of the new memory candidate $\widetilde{\boldsymbol{c}}_t$ will be added to $\boldsymbol{c}_t$. The output gate represents the impact of the memory cell $\boldsymbol{c}_t$ to the output $\boldsymbol{h}_t$. We can mathematically summarize this procedure through the following equations:

- $\boldsymbol{f}_t = \sigma(\boldsymbol{W}_f \boldsymbol{x}_t + \boldsymbol{U}_f \boldsymbol{h}_{t-1} + \boldsymbol{b}_f)$,
- $\boldsymbol{i}_t = \sigma(\boldsymbol{W}_i \boldsymbol{x}_t + \boldsymbol{U}_i \boldsymbol{h}_{t-1} + \boldsymbol{b}_i)$,
- $\boldsymbol{o}_t = \sigma(\boldsymbol{W}_o \boldsymbol{x}_t + \boldsymbol{U}_o \boldsymbol{h}_{t-1} + \boldsymbol{b}_o)$,
- $\widetilde{\boldsymbol{c}}_t = tanh(\boldsymbol{W}_c \boldsymbol{x}_t + \boldsymbol{U}_c \boldsymbol{h}_{t-1} + \boldsymbol{b}_c)$,
- $\boldsymbol{c}_t = \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \odot \widetilde{\boldsymbol{c}}_t$,
- $\boldsymbol{h}_t = \boldsymbol{o}_t \odot tanh(\boldsymbol{c}_t)$.

This unique architecture makes LSTMs particularly effective in tasks involving sequential data, such as natural language processing, speech recognition, and time series analysis, where capturing dependencies over extended temporal contexts is crucial for accurate predictions. Figure 3.1.8 provides further architectural details about LSTMs.

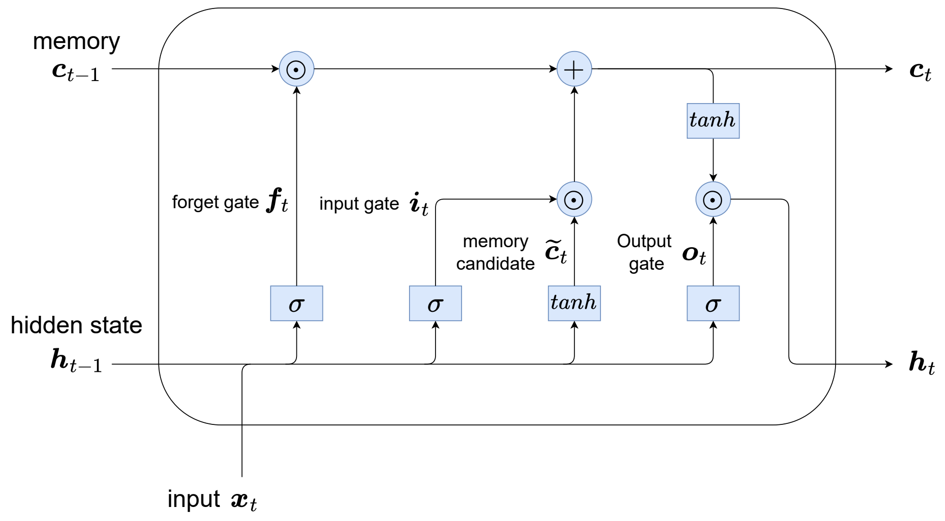


Figure 3.1.8: Architecture of a Long-Short Term Memory neural network. Inspired from Calzone (2022).

3.1.5 Autoencoders (AE)

Autoencoders (AEs) (Hinton and Zemel, 1993; Hinton and Salakhutdinov, 2006; Vincent *et al.*, 2008, 2010) are a particular type of neural network architecture trained in an autoassociative self-supervised (Hinton and Salakhutdinov, 2006; Vincent *et al.*, 2008) manner. Unlike supervised learning where an input/output pair $(\boldsymbol{x}, \boldsymbol{y}) \in \mathbb{R}^{N_x} \times \mathbb{R}^{N_y}$ is provided, autoencoders solely require $\boldsymbol{x}$ to serve both as the input and the target. Historically, the primary goal of autoencoders is to reconstruct the input data with the constraint of finding a low dimensional representation of it (Kramer, 1991). For example, given a dataset $\boldsymbol{X} = \boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_{\mathcal{N}} \in \mathbb{R}^{N_x \times \mathcal{N}}$ the AE iteratively seeks a reduced-space - specifically termed "**latent space**" - of a fixed dimension $\ell \ll N_x$. The absence of the data reduction constraint would essentially lead to learning the identity function, which is of no interest. From a mathematical point of view, the autoencoder updates its parameters, by iteratively minimizing a reconstruction cost function (see equation (3.10)) under the latent space representation constraint. Figure 3.1.9 illustrates the architecture of a classic autoencoder.

Autoencoders can also be used to map from the input space to a higher dimensional space. This can be particularly relevant in the context of the Koopman theory (Koopman, 1931), which posits that the dynamics of any finite-dimensional nonlinear system can be represented as a linear system in an infinite-dimensional function space.

Within the context of our research, we will only be referring to autoencoders that exhibit a low dimensional representation of the input data.

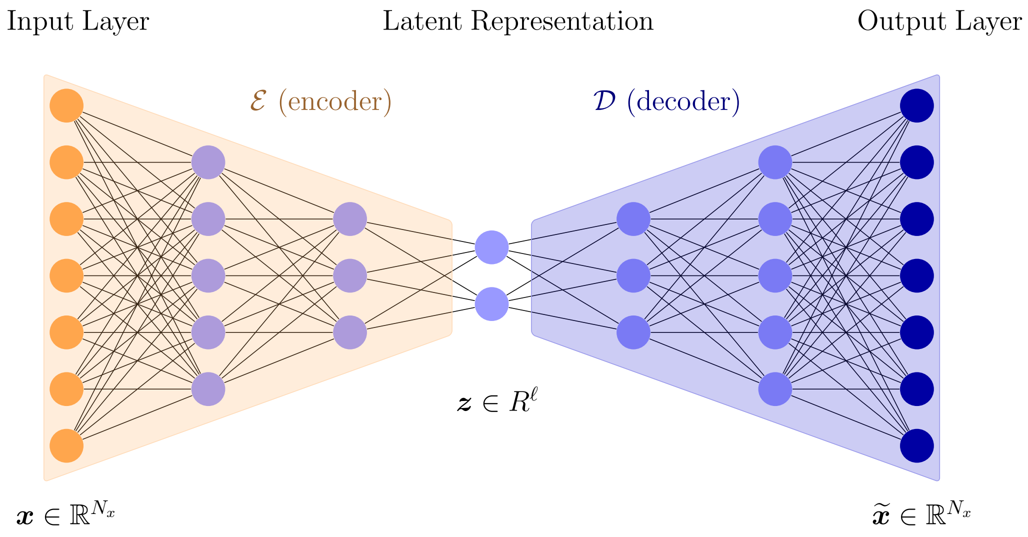

Input Layer
Latent Representation
Output Layer
$\mathcal{E}$ (encoder)
$\mathcal{D}$ (decoder)
$\boldsymbol{z} \in R^{\ell}$
$\boldsymbol{x} \in \mathbb{R}^{N_x}$
$\widetilde{\boldsymbol{x}} \in \mathbb{R}^{N_x}$

Figure 3.1.9: Architecture of an autoencoder

An autoencoder comprises three parts:

- the **encoder**, denoted by $\mathcal{E}$ and parameterized by $\theta_{\mathcal{E}}$, maps from the full space to the latent space through dimension-decreasing layers.
- the latent representation of the data also known as the bottleneck of the autoencoder.
- the **decoder**, denoted by $\mathcal{D}$ and parameterized by $\theta_{\mathcal{D}}$, that maps back the latent variable to the full space.

When training an autoencoder, the most straightforward error metric is a reconstruction cost function. Given $\boldsymbol{x}_i \in \boldsymbol{X}$, it is computed as follows:

$$\boxed{\mathcal{L}_{AE}(\boldsymbol{x}_i;\theta_{\mathcal{E}};\theta_{\mathcal{D}}) = \frac{1}{N_x}\|\boldsymbol{x}_i - \mathcal{D}(\mathcal{E}(\boldsymbol{x}_i))\|_2^2.} \tag{3.10}$$

Autoencoders have been very successful in a wide range of applications (*e.g.*, denoising tasks, reducing the computational burden when processing high-dimensional data, etc.).

Without any constraint, the latent space is unlikely to be regular, smooth, organized and interpretable. Let us consider one has trained an autoencoder over the MNIST dataset (LeCun and Cortes, 2005) and let us denote $\boldsymbol{z}_1$ and $\boldsymbol{z}_2$ the latent representations of digits 0 and 1, respectively. By decoding $\boldsymbol{z}_3 = (\boldsymbol{z}_1 + \boldsymbol{z}_2)/2$, one would expect to obtain a new digit close to a combination of a 0 and a 1. This is not what happens in practice with traditional autoencoders. If some regularity, smoothness or interpretability within the latent space are required for a given application, one can resort to the so-called **Variational Autoencoders** (VAEs).

VAEs (Kingma and Welling, 2013; Doersch, 2016; Higgins *et al.*, 2016) are capable of generating completely new data by enforcing the smoothness and the continuity of the latent space. They enable seamless interpolation between different points in the latent space, preventing the occurrence of gaps within it and thereby avoiding unrealistic outputs to be returned by the decoder. To do so, VAEs encode the input data in terms of probability density functions (PDF). In practice such a probability distribution is assumed to be Gaussian, therefore the encoder is trained to yield a mean and a covariance matrix associated with a given input. Figure 3.1.10 gives an illustration of a VAE.

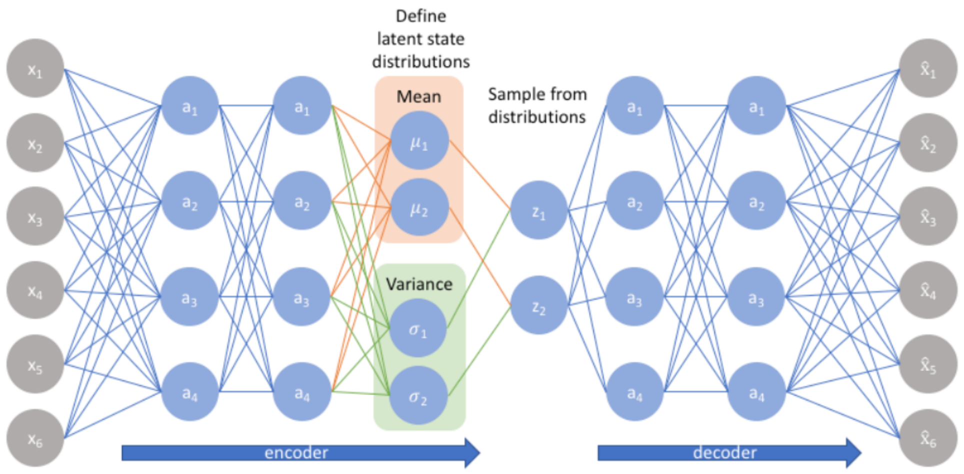


Figure 3.1.10: Architecture of a Variational Autoencoder (VAE).

Enforcing the probability distribution to be Gaussian can be done by adding a penalization term in the loss function, known as the **Kullback-Leibler divergence** or KL-divergence (Kullback and Leibler, 1951). It measures the discrepancy between two probability distributions. Given $\boldsymbol{x}_i \in \boldsymbol{X}$ and $\boldsymbol{z}_i$ its latent representation, the loss of a VAE to be minimized, also known as the **Evidence Lower BOund** (ELBO), reads:

$$\boxed{\mathcal{L}\left(\boldsymbol{x}_i;\theta_\mathcal{E};\theta_\mathcal{D}\right) = \underbrace{\mathbb{E}_{q_{\theta_\mathcal{E}}\left(\boldsymbol{z}_i|\boldsymbol{x}_i\right)}\left[ln\left(p_{\theta_\mathcal{D}}\left(\boldsymbol{x}_i|\boldsymbol{z}_i\right)\right)\right]}_{\text{reconstruction term}} - \underbrace{\mathrm{KL}\left[q_{\theta_\mathcal{E}}\left(\boldsymbol{z}_i|\boldsymbol{x}_i\right)||p\left(\boldsymbol{z}_i\right)\right]}_{\text{regularizing term}},} \tag{3.11}$$

where:

- $q_{\theta_\mathcal{E}}\left(\boldsymbol{z}_i|\boldsymbol{x}_i\right)$ is the posterior distribution of latent variable $\boldsymbol{z}_i$ given $\boldsymbol{x}_i$.
- $p_{\theta_\mathcal{D}}\left(\boldsymbol{x}_i|\boldsymbol{z}_i\right)$ represents the probability distribution of reconstructing $\boldsymbol{x}_i$ given the latent variable $\boldsymbol{z}_i$.

The first term is the expected negative log-likelihood of the input $\boldsymbol{x}_i$, which encourages the decoder to accurately reconstruct the inputs. It acts as the reconstruction loss. The second term, the Kullback-Leibler divergence between the encoder's distribution $q_{\theta_E}\left(\boldsymbol{z}_i|\boldsymbol{x}_i\right)$ and $p\left(\boldsymbol{z}_i\right)$, the prior distribution over the latent variable (often chosen to be the standard normal distribution $\mathcal{N}\left(0,\boldsymbol{I}\right)$, acts as a regularization term. It encourages the encoder to produce a representation $\boldsymbol{z}_i$ that is close to the prior, ensuring a well-structured latent space.

Figure 3.1.11 shows the shape of the latent spaces for three different trainings, when learning a 2D low-dimensional representation of MNIST data (LeCun and Cortes, 2005). We observe that the autoencoder produces an interpretable but non-smooth reduced-space (left picture), while the VAE yields a smooth, regular and interpretable latent space (right image). Besides, only minimizing the KL-divergence term results in a smooth latent space, but with no spatial correlation and clarity between encoded points.

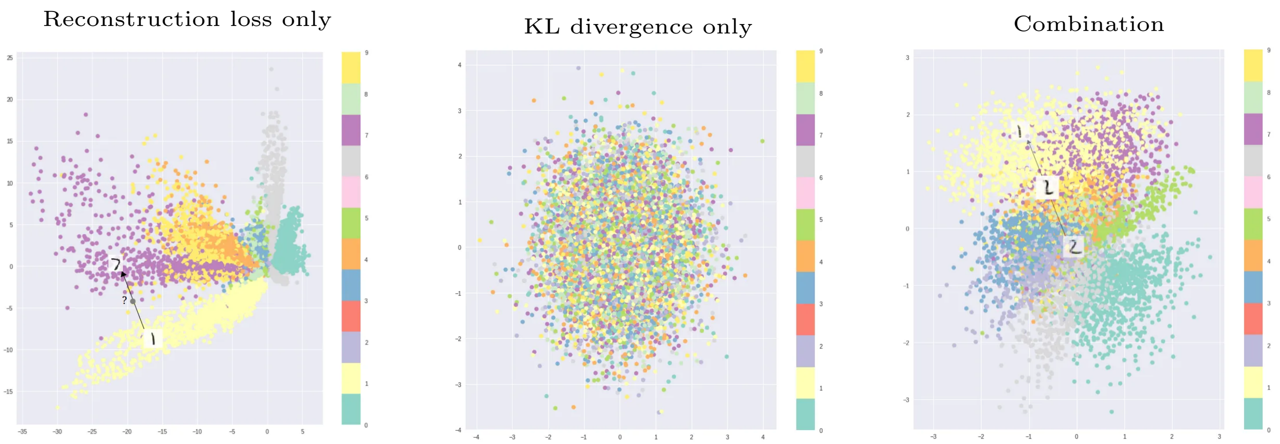


Figure 3.1.11: Left: 2D latent space of an autoencoder trained on MNIST dataset (LeCun and Cortes, 2005). Middle: 2D latent space of a variational autoencoder trained with the Kullback-Leibler divergence penalization only. Right: 2D latent space of a variational autoencoder trained with a two-terms loss function (reconstruction and penalization terms). Images from Shafkat (2018).

3.1.6 Training neural networks: an optimization problem

In the following, f_θ no longer specifically denotes a MLP or a CNN, but any type of neural network.

The success of deep learning is also related to the effectiveness of the optimization algorithms utilized to iteratively adjust neural networks' parameters. The optimization process aims to find the set of parameters that minimizes the discrepancy between the model's predictions and the true values with respect to a given metric $\mathcal{L}$, commonly referred to as the **loss** or **cost function** (LeCun *et al.*, 2015;

Goodfellow *et al.*, 2016).

When training a neural network model in a supervised context, one split their global dataset into **train**, **validation** and **test** sets (LeCun *et al.*, 2015; Goodfellow *et al.*, 2016): a common rule-of-thumb consists in randomly picking 80% of the data for the train stage, 10% for the validation step and the remaining 10% for the test evaluation. The train dataset is used to iteratively fit the model while the validation data are dedicated to assessing the performance of the model through the training and serves for **hyperparameters** tuning. As for the test dataset, it is used to evaluate trained models.

Let us denote $\mathcal{U}, \mathcal{V}$ and $\mathcal{T}$ the train, validation and test datasets, respectively. Each dataset is made up of $(\boldsymbol{x}, \boldsymbol{y})$ pairs, where $\boldsymbol{x}$ represents the input to the neural network and $\boldsymbol{y}$ the associated label.

Mathematically, the minimization problem reads:

$$\boxed{\min_{\theta \in \mathbb{R}^d} \sum_{(\boldsymbol{x}_i, \boldsymbol{y}_i) \in \mathcal{U}} \mathcal{L}(\boldsymbol{y}_i, f_\theta(\boldsymbol{x}_i)).} \tag{3.12}$$

Gradient Descent (GD) (Cauchy *et al.*, 1847; Ruder, 2017) stands as the cornerstone of neural network optimization. It involves iteratively adjusting the model parameters in the direction of the steepest decrease in the loss function. The update rule for each parameter is given by:

$$\boxed{\theta_i^{new} = \theta_i^{old} - \eta \frac{\partial \mathcal{L}}{\partial \theta_i}, \ \forall\, i \in \{1, d\},} \tag{3.13}$$

where η is the **learning rate**, controlling the step size during parameter updates. The computation of $\frac{\partial \mathcal{L}}{\partial \theta_i}$ is enabled thanks to the so-called **backpropagation** (Kelley, 1960; Rosenblatt *et al.*, 1962; Werbos, 1974; Rumelhart *et al.*, 1986; LeCun *et al.*, 1989), short for "backward propagation of the errors", which relies on the application of the Leibniz chain rule (von Leibniz *et al.*, 1920): specifically, backpropagation requires all the mathematical operations involved in the considered neural network to be fully differentiable. In practice, backpropagation is performed thanks to **automatic differentiation** (Griewank and Walther, 2008).

Several variants of gradient descent algorithms have been proposed (Ruder,

2017) to address challenges such as oscillations and slow convergence. Notable among these are:

- **Stochastic Gradient Descent** (SGD) (Robbins and Monro, 1951; Zhang, 2004; Bottou, 2010; Ruder, 2017): contrary to GD which processes all the data at once, SGD is an optimization variant that introduces stochasticity by randomly picking one sample in the training dataset, computing the loss function and updating the parameters accordingly. The remaining data are then iteratively processed one by one. Stochastic Gradient Descent is significantly less expensive than Gradient Descent. It also offers faster progress in the initial stages of the training.
- **Mini-batch Gradient Descent** (Dekel *et al.*, 2012; Li *et al.*, 2014; Ruder, 2017): an intermediate approach between GD and SGD, mini-batch gradient descent computes parameter updates based on a small, randomly selected subset (mini-batch) of the training data. Similarly as SGD, Mini-batch Gradient Descent is stochastic but has the advantage to process multiple samples at once.
- **Adam** (Adaptive Moment Estimation) (Kingma and Ba, 2017; Ruder, 2017; Reddi *et al.*, 2019; Loshchilov and Hutter, 2019): Adam smooths out weights' updates by adapting the learning rates for each parameter based on their past gradients. Accounting for previous gradients helps in faster convergence and in overcoming the challenges of local minima and saddle points, which are common in complex optimization landscapes like those found in deep learning. The learning rate is updated by the so-called first and second moments which involve the gradients and the square of the gradients of the weights, respectively.

When training a neural network, **regularization techniques** are commonly used in order to enhance its generalization capabilities and prevent overfitting to the training data. Here-after, we present three well-known regularization techniques: weight decay (or L2 regularization), dropout, and early stopping.

Weight decay (Krogh and Hertz, 1991; Goodfellow *et al.*, 2016), also known as L2 regularization, adds a penalty proportional to the sum of the squared weights of the model to the loss function. This technique discourages the model from assigning excessively large weights to any single feature, thereby promoting simpler and more generalizable models. The modified loss function with weight decay is given by:

$$\boxed{\mathcal{L}_{new} = \mathcal{L}_{original} + \rho \sum_i \theta_i^2,} \tag{3.14}$$

where ρ is the regularization parameter. We remind that θ_i represent the parameters (*i.e.*. weights and biases) of the model. The constant ρ controls the strength of the regularization, with higher values leading to stronger regularization effects.

Dropout (Srivastava *et al.*, 2014; Gal and Ghahramani, 2016) is a technique where, during training, randomly selected neurons are deactivated or "dropped out" at each iteration. This prevents the network from becoming too reliant on specific neurons and forces it to learn more robust and distributed representations. In each training iteration, each neuron is retained with a probability p and dropped with a probability $(1 - p)$. Typically, p is set to 0.5 for hidden layers and 1.0 (*i.e.*., no dropout) for output layers during training. During inference, all neurons are used, but their activations are scaled by p to maintain the same expected output.

Early stopping involves monitoring the model's performance on the validation set during training and halting the training process when the performance on this set stops improving. This helps prevent the model from overfitting to the training data. During training, the performance metric (*e.g.*., validation loss or accuracy) is tracked at the end of each epoch. A patience parameter is set, which specifies the number of epochs to wait for an improvement before stopping the training. For example, if the patience is set to 5, training will stop if there is no improvement in the validation performance for 5 consecutive epochs. The best-performing model on the validation set is saved during training. If early stopping is triggered, this best model is restored.

In addition to weight decay, dropout, and early stopping, several other regularization methods are commonly used: **batch normalization** (Ioffe and Szegedy, 2015), **data augmentation** (Shorten and Khoshgoftaar, 2019), **L1 regularization (Lasso)** (Tibshirani, 1996), **Elastic Net** (Zou and Hastie, 2005), **knowledge distillation** (Hinton *et al.*, 2015).

Section 3.1 provided the reader with all the essentials of deep learning necessary to fully understand the neural network architectures we will be discussing, along with the terminology and optimization procedures specific to this field. For a more detailed and comprehensive introduction to deep learning, accompanied by further explanations, the reader can refer to LeCun *et al.* (2015); Goodfellow *et al.* (2016); Arnulf Jentzen and von Wurstemberger (2023); Prince (2023).

3.2 Neural networks as key ingredients for substituting data assimilation components

Data assimilation is concerned with accurately predicting the state of a dynamical system using an imperfect physical predictive model coupled with possibly sparse, unevenly distributed and often noisy observations. The DA process also accounts for the associated uncertainties enhancing the reliability of the output state beyond that of each individual input information considered apart. Further information and details about data assimilation can be found in chapter 2. We remind the propagation and observation equations, as already defined in equations (2.27a) and (2.27b):

$$\begin{cases} \boldsymbol{x}_k^t = \boldsymbol{\mathcal{M}}_k\left(\boldsymbol{x}_{k-1}^t\right) + \boldsymbol{\eta}_k, \\ \boldsymbol{y}_k = \boldsymbol{\mathcal{H}}_k\left(\boldsymbol{x}_k^t\right) + \boldsymbol{\varepsilon}_k, \end{cases}$$

Operators $\boldsymbol{\mathcal{M}}$ and $\boldsymbol{\mathcal{H}}$ often stem from limited and incomplete physical knowledge, making them error-prone, especially in real-world scenarios.

The rapid growth in sensor technology and data acquisition methods (*e.g.*, Earth observations (Kuenzer *et al.*, 2014; McNally *et al.*, 2014)) has positioned deep learning as a highly effective strategy for extracting and processing pertinent information from raw data. Its extraordinary ability to infer or recover underlying physics, connections and patterns without relying on prior knowledge or specific research guidelines, is a significant breakthrough in science and in particular in data assimilation (Reichstein *et al.*, 2019; Geer, 2021; Cheng *et al.*, 2023).

Notably, the works of Abarbanel *et al.* (2018) and Geer (2021) establish the equivalence between machine learning and data assimilation, both of them could be unified under the Bayesian framework.

The most straightforward approaches to incorporating deep learning into data assimilation algorithms consist in substituting DA operators or components (*e.g.*, $\boldsymbol{\mathcal{M}}_k$, $\boldsymbol{\mathcal{H}}_k$, $\boldsymbol{P}_k^f$, $\boldsymbol{R}_k$, $\boldsymbol{Q}_k$) with neural networks.

Going one step further leads to the development of data-driven models whose forecasting capabilities rival or even surpass the ones of operational data assimilation systems for short-term and medium range forecasts.

The ultimate aim is to conduct data assimilation in a fully data-driven manner. To this end, researchers have developed standalone deep learning models that operate independently of any DA system, relying solely on prior analysis and current observations.

The following subsections examine a range of hybrid methods that incorporate neural networks into data assimilation algorithms, and provide examples from the literature. Although these methods are presented within a unified scope, each piece of research addresses distinct issues under a unique set of assumptions.

Following this, the subsequent two sections present approaches that further integrate data-driven models towards a more standalone online running position, smoothly relegating data assimilation to a background and supportive role.

3.2.1 Surrogate physical models

Data assimilation is applied in diverse fields such as weather forecasting, oceanography, climatology, geosciences, epidemiology, or wildfire prediction. As suggested by these real-world contexts, physical models used in DA can often demand substantial computational resources. For instance, accurately solving the motion equations of the Earth's atmosphere on fine grids represents a high-dimensional and computationally intensive challenge. Given this, deep learning emerges as a compelling alternative: by harnessing the power of GPUs and parallel computing, deep learning offers the potential for faster, and possibly even real-time forecasting.

In 2018, the study by Loh *et al.* (2018) effectively utilizes the large sensor database available in the gas well industry to train a data-driven surrogate model. While this task is traditionally performed by costly physical models, the authors demonstrate the feasibility of training a LSTM network to efficiently predict the temporal evolution of flow rates in mature gas wells. More precisely, the neural network learns on a specific gas well, labeled as "A", and characterized by certain dynamic behaviors. The trained model proves to be applicable to another gas well, "B", which exhibits a slightly different production evolution. This data-driven model is integrated within an EnKF framework that includes parameter estimation, as presented in Evensen (2003). Therefore, the data assimilation algorithm corrects not only the physical variables of interest, but also the bias of the last layer of the network. This is achieved by concatenating the bias vector to the state variables:

the method is known as state augmentation. Notably, the augmented system approach allows for the dynamic updating of the weight bias in the last layer during the assimilation process. This method has the advantage of being applicable to real natural gas well production in a real-time context.

Geosciences are also concerned with simulating subsurface flows. The work by Tang *et al.* (2020) applies a **residual U-Net** (Ronneberger *et al.*, 2015) coupled with a convolutional LSTM to surrogate the prediction of state maps (*i.e.*, subsurface flow states) from permeability maps. This architecture termed *recurrent R-U-Net* captures both local and global features via the U-Net component of the network, while the temporal evolution is addressed by the convolutional LSTM. The methodology is further extended to 3D subsurface flows in Tang *et al.* (2021b). Another notable work, Tang *et al.* (2021a), focuses on modeling pressure buildup and CO2 plume predictions for geologic carbon storage. Here, two networks - a residual convolutional network and U-Net - are trained and incorporated within an **Ensemble Smoother Multiple Data Assimilation** (ES-MDA) framework (as described in Emerick and Reynolds (2013)).

In the study of dynamical systems, LSTM networks, as discussed in 3.1.4, have been widely and effectively utilized as surrogate models. Their structure which includes a memory cell and several control gates, allows them to retain, add or release information over time. A pertinent example is the research conducted by Nadler *et al.* (2020) about the Covid-19 pandemic. In this study, an LSTM network learns the highly-nonlinear interactions between a few epidemiological variables in order to forecast COVID-19 cases.

Our knowledge and understanding of the physical phenomena involved in data assimilation can be incomplete, limited or imprecisely approximated: governing equations might be unknown, certain physical quantities neglected, or small-scale features unresolved. Therefore, there is a growing need to develop data-driven surrogate models that learn directly from sparse and noisy observations, rather than solely depending on physical simulation data. Pursuing this research direction, the studies by Gottwald and Reich (2021b,a) introduce a supervised learning strategy based on observations and random feature maps. Though not involving any stochastic gradient algorithms or back-propagation, random feature maps are comparable to shallow networks and also verify the universal approximation theorem. The approach, termed RAFDA - Random Feature maps Data Assimilation - encompasses both parameter and state estimation and demonstrates efficiency in a Lorenz63 (Lorenz, 1963) experimental setup. Likewise, the research by Brajard *et al.* (2020) proposes to alternate between deep learning training phases and data assimilation cycles to iteratively learn the dynamics of a Lorenz96 system (Lorenz, 1996) and thereby progressively produce more accurate analysis and predictions. The physical dynamics is assumed to be unknown and the neural network learns solely from the output of the data assimilation procedure (EnKF), which randomly observes half of the state vector at each time step. However, the convergence of this method highly depends on the chosen initialization strategy. Prior to the actual training stage, the authors resort to a preliminary phase of "light learning": in this phase, the network is trained using cubic interpolations of the observations across both space and time. This approach ensures convergence for the subsequent training phase. Additionally, the same research team renews the methodology in Bocquet *et al.* (2020) by relying on a Bayesian formalism.

Another significant advantage of deep learning in the context of variational data assimilation is its capability to derive tangent linear and adjoint models directly from the surrogate neural network, as discussed in Hatfield *et al.* (2021). In the study by Penny *et al.* (2022), the authors successfully perform data assimilation in the hidden/reservoir state of a RNN. This is achieved in scenarios with both full and sparse observation coverage, across varying noise levels. The effectiveness of this method is demonstrated on a Lorenz96 dynamics model (Lorenz, 1996), using both the ETKF and the 4D-Var algorithms. Unlike in deep learning, here only one matrix of the RNN - namely $\boldsymbol{W}_{out}$ - is trained through the resolution of regularized least squares problems, while other parameters, having been randomly selected initially, remain fixed throughout the process.

Performing data assimilation in a distinct space from the state space is an innovative approach that has attracted the attention of researchers within the data assimilation community. In particular, coupling **Reduced-Order Modelling** (ROM) techniques with data-driven surrogate models have successfully been investigated over the last few years (Amendola *et al.*, 2020; Tang *et al.*, 2021a; Amendola *et al.*, 2021; Cheng *et al.*, 2022b,a; Silva *et al.*, 2022). Section 4.1 reviews this area of research, providing a more in-depth discussion.

3.2.2 Enforcing physical constraints within the data assimilation process

Data assimilation and neural networks both encounter similar drawbacks when it comes to handling physical processes: typically, they are physics-agnostic. That is, they do not guarantee adherence to physical laws. For instance, while the background in a DA algorithm or the input in a neural network might satisfy a particular Partial Differential Equation (PDE), the resulting analysis or output may not. This disregard for physical laws can lead to inaccuracies and suboptimal solutions. For example, localization techniques, commonly used in EnKF algorithms to reduce sampling errors, can produce analyses that violate physical properties. Studies such as Janjić *et al.* (2014); Zeng and Janjić (2016); Zeng *et al.* (2017); Ruckstuhl and Janjić (2018) demonstrate how neglecting the conservation of mass, energy, or enstrophy can significantly diminish the quality of state analysis.

Similarly, the *a priori* physics-agnostic nature of neural networks poses risks, particularly in numerical simulations. While knowledge-based models can maintain physical properties, ensuring non-divergent runs, neural networks might accumulate errors, leading to diverging and non-physical outputs, rendering them operationally ineffective. Consequently, ensuring that neural networks adhere to physical laws has become a primary research focus in recent years, as seen in studies like Greydanus *et al.* (2019); Cranmer *et al.* (2020), and in the development of **Physics-Informed Neural Networks** (PINNs) (Raissi *et al.*, 2019).

In data assimilation, algorithms such as the **Quadratic Programming Ensemble** (QPEns) (Janjić *et al.*, 2014) have been designed to produce more physically realistic estimates. However, these methods often come with significant computational burdens, making them prohibitive for high-dimensional problems. Thus, leveraging deep learning to facilitate fast and accurate data assimilation analysis emerges as a key strategy to circumvent the limitations of traditional data assimilation algorithms.

The research conducted by Ruckstuhl *et al.* (2021) exemplifies this approach, using a convolutional neural network to learn the discrepancies between an EnKF analysis and its equivalent produced by the QPEns algorithm. The decision to utilize a deep learning architecture is driven by the natural capacity of convolutional kernels to perform localization. This study conducts numerical experiments using a shallow water model, focusing on two test scenarios: one with a sufficiently short model time step (here analogous to 5min) ensuring EnKF convergence, and another with a longer time step (analogous to 10min) leading to EnKF divergence.

In the former scenario, the CNN-assisted data assimilation achieves performances comparable to that of the QPEns. In the latter case, while the results with the CNN do not quite match the QPEns, they are significantly superior to the outcomes obtained solely with the EnKF.

In He *et al.* (2020), the authors utilize the PINN formulation to train a feed forward neural network and later assimilate multiphysics measurements in the context of sparse observations, heterogeneous porous media and high-computational costs.

3.2.3 Model error correction

In the context of data assimilation, multiple sources of uncertainties impact physical models and result in the so-called model errors. These errors typically fall into one of the following categories:

- incomplete understanding of real-world physical dynamics, that can lead to misrepresented physical phenomena.
- the high dimensionality of the problem (*e.g.*, in weather forecast or geosciences) coupled with limited computer resources that can translate into unresolved small-scale processes.
- inevitable numerical integration errors.

We can also mention the erroneous initial conditions that are evolved over time by the linearized model and contribute further to these sources of error as shown in equation (2.42), and reminded here-below:

$$\boldsymbol{P}_k^f = \boldsymbol{M}_k \boldsymbol{P}_{k-1}^a \boldsymbol{M}_k^T + \boldsymbol{Q}_k.$$

Model errors are commonly assumed to be additive, zero-mean, and white in time. However, these conditions are rarely met simultaneously in practice, and are often relaxed operationally: *e.g.*, at ECMWF, a multiplicative term is adopted in model error parameterization. Furthermore, the assumption of zero-mean can be detrimental, particularly when dealing with systematic model errors, also known as biases.

Recent developments in weak-constrained 4D-Var indicate that a significant portion of model errors can be estimated and corrected (Laloyaux *et al.*, 2020). This insight supports the view that model error correction is one of the most promising avenues for producing better forecasts in NWP.

In contrast to the previous subsection, where we discuss the opportunity to fully surrogate a physical model, here, we highlight the advantages of combining physical knowledge with deep learning. Numerical models benefit from a long history of modeling, as well as numerical improvements and parameterizations, making them a solid baseline for predictions. With the ever-growing availability of observational Earth data in NWP, it is legitimate to explore the development of a hybrid model. Such a model would integrate a knowledge-based part with a data-driven surrogate that addresses model errors.

The first insight to integrate deep learning into operational NWP were examined by Bonavita and Laloyaux (2020). Specifically, the study demonstrates that feedforward neural networks can achieve performances comparable to the weak-constrained 4D-Var system of the Integrated Forecasting System (IFS) at ECMWF. A notable advantage of neural networks is their ability to provide corrections across the entire atmospheric column, unlike the variational algorithm in the IFS, which is inactive below 100hPa — a limitation that has been a significant concern since the weak-constrained 4D-Var became operational, over a decade ago.

Model error estimation can be performed either by examining the discrepancies between observations and background (i.e., $observations - background$) or analysis (i.e., $observations - analysis$), or by assessing the increment between the background and the analysis (i.e., $analysis - background$). The latter, being more technically feasible, was the chosen method by the authors. This concept of accounting for the discrepancy to the analysis was initially proposed by Leith (1978) and subsequently adopted by Dee (2005) and Carrassi and Vannitsem (2011). In this study, three neural network architectures — *relu_one_layer*, *relu_two_layers*, and *relu_three_layers* — are considered and benchmarked against a linear regression baseline. The neural networks are trained using inputs that either solely consist of model variables or are supplemented with climatological predictors (latitude, longitude, time of day and month). The authors also investigate the potential benefits of data augmentation, by including past states in addition to the current predictors.

In line with the research efforts of Gottwald and Reich (2021b,a); Brajard *et al.* (2020), addressing the challenge of sparse and noisy observations remains a crucial direction in data assimilation studies. In Wikner *et al.* (2021), the authors introduce a machine learning approach (distinct from deep learning though) to correct model errors given sparse and noisy observations. In this method, they focus on optimizing the output weights of a reservoir computer to fit the analysis produced by an ETKF algorithm. This approach shares a similarity with deep learning in that it necessitates a training phase during which, the weights are fine-tuned to minimize a cost function. Afterwards, the model can be used for predictions. The authors further develop their algorithm by proposing an iterative version of it. This enhanced approach combine the imperfect model with the continually learning reservoir computer during the training stage, thereby improving the efficacy of the model error correction process. The success of the technique is demonstrated for the Lorenz63 (Lorenz, 1963) and the Kuramoto-Sivashinsky systems and the model error is introduced via a misspecified parameter in each case.

Building on the innovative method of Brajard *et al.* (2020), the study by Farchi *et al.* (2021b) focuses on correcting a knowledge-based model using sparse and noisy observations, employing a strong-constraint 4D-Var algorithm. Contrary to Brajard *et al.* (2020), the convergence of the approach when iterating between data assimilation and training stages is not a concern in this case. The weights are indeed initialized to zero, so that within the first data assimilation step, time propagation only relies on the knowledge-based model (no model error correction). While in Brajard *et al.* (2020), the fully data-driven surrogate may produce nonphysical states, here the forecasts are first generated with numerical solvers, which ensures convergence of the subsequent steps. When attempting to correct model error, two approaches can arise and are pointed out by the authors:

1. correcting the resolvent: $\mathcal{M}_k(\boldsymbol{p}, \boldsymbol{x}) = \mathcal{M}_k^o(\boldsymbol{x}) + \mathcal{M}_k^{ml}(\boldsymbol{p}, \boldsymbol{x})$, where $\mathcal{M}_k^o$ represents the resolvent of the knowledge-based model, and $\mathcal{M}_k^{ml}$ is the machine learning model, with $\boldsymbol{p}$ denoting its parameters.
2. correcting the tendencies: $\frac{\mathrm{d}\boldsymbol{x}}{\mathrm{dt}} = \mathcal{F}^o(\boldsymbol{x}) + \mathcal{F}^{ml}(\boldsymbol{p}, x)$, where $\mathcal{F}^o$ is the tendency of the original model and $\mathcal{F}^{ml}$ is the machine learning model. This method requires the integration of the tendency in order to obtain the corrected surrogate model.

In their research, the authors opt not to explore the second option due to its complexity in implementation. They test two neural network architectures on the Quasi-Geostrophic (QG) model of the ECMWF. In this setup, model error is introduced through a perturbed configuration, differing from the reference setup in four aspects: upper and lower layers' depths, integration time, and orography. The numerical results validate the neural networks' capability to enhance forecasts compared to the original model by correcting the model's resolvent. However, the authors identify some areas requiring further investigation:

- the potential benefits of implementing tendency correction in terms of the accuracy and quality of the analysis.
- the development of an online model error correction system.

The authors further discuss and address these two points in Farchi *et al.* (2021a). Within the framework of Brajard *et al.* (2020); Farchi *et al.* (2021b), they successfully perform resolvent and tendency corrections under sparse and noisy observation conditions by hybridizing a knowledge-based model with neural networks. The approach is tested on a Lorenz system (Lorenz, 1996) that exhibits large-scale and small-scale physical processes, and which is commonly referred to as L05III. To facilitate online learning, the authors introduce a novel formulation of the weak-constraint 4D-Var based on an augmented state method. This involves concatenating the current state $\boldsymbol{x}_k$ with the neural network parameters $\boldsymbol{p}$, leading to the following weak-constraint 4D-Var formulation:

$$\begin{aligned}\mathcal{J}(\mathbf{p}_k,\mathbf{x}_k) = \frac{1}{2}\left\|\mathbf{p}_k-\mathbf{p}_k^b\right\|^2_{\mathbf{B}_p^{-1}} + \frac{1}{2}\left\|\mathbf{x}_k-\mathbf{x}_k^b\right\|^2_{\mathbf{B}_x^{-1}} \\ + \frac{1}{2}\sum_{\ell=0}^{L-1}\left\|\mathbf{y}_{k+\ell}-\mathcal{H}\circ\mathcal{M}(\mathbf{p}_k,\mathbf{x}_k)\right\|^2_{\mathbf{R}^{-1}}.\end{aligned} \tag{3.15}$$

Other studies like Laloyaux *et al.* (2022) and Gregory *et al.* (2023), explore different applications of combining physical models with neural networks. In Laloyaux *et al.* (2022), CNNs learn to correct model biases from radio occultation measurements. Initially trained on 12 years of ERA5 reanalysis data (2008 - 2019), the authors subsequently leverage transfer learning to specifically target the years 2019-2020. Similarly, Gregory *et al.* (2023) employ CNNs to identify climatological biases in sea ice concentration forecasts.

3.2.4 Estimation of the background covariance matrix

One essential ingredient for successful data assimilation is an accurate background covariance matrix, $\boldsymbol{P}_t^b$. A wide range of DA applications like NWP are concerned with high-dimensional and nonlinear dynamical processes: *e.g.*, weather forecasting systems typically involve state variables with dimension n in the order of

$\mathcal{O}(10^6 - 10^8)$. Consequently, storing or explicitly computing the associated state error covariance matrix $\boldsymbol{P}_t^b$, of dimension n^2, is practically unfeasible.

To address this challenge, ensemble algorithms conduct data assimilation using small ensembles, typically in the order of $\mathcal{O}(10^1 - 10^2)$ members (Leutbecher, 2019). This approach allows for a computationally manageable low-rank approximation of $\boldsymbol{P}_t^b$. However, this rank-deficient matrix is prone to sampling errors, leading to spurious correlations. These errors necessitate the use of an *ad-hoc* localization term to mitigate them (Asch *et al.*, 2016). Nonetheless, such a corrective measure risks eliminating physically consistent correlations (Miyoshi *et al.*, 2014).

On the other hand, accurately approximating the background error covariance matrix theoretically requires a number of ensemble members comparable to the dimension of the unstable-neutral subspace (Strogatz, 1994; Legras and Vautard, 1996; Carrassi *et al.*, 2022), a requirement that often comes with a high computational burden in operational data assimilation: indeed, the number of unstable-neutral directions can possibly represent thousands up to millions degrees of freedom (Bocquet and Farchi, 2014), and most real-world physical models are too resource-intensive to process large ensembles swiftly.

As previously discussed in section 3.2.1, deep learning offers a viable solution for surrogating physical dynamics with significantly reduced computational costs. In line with this, the study by Chattopadhyay *et al.* (2023) introduces an innovative data-driven method for efficiently estimating the background covariance matrix. This approach involves using a neural network to learn the physical dynamics and leveraging its rapid inference capabilities to propagate a large ensemble of members. Specifically, the technique utilizes two sets of ensembles and conducts numerical experiments on the streamfunction, denoted by $\boldsymbol{\psi}$, of a Quasi-Geostrophic (QG) model.

Given the analysis state $\boldsymbol{\psi}_t^a$ at time t, a Gaussian noise $\mathcal{N}(0, \sigma_b^2)$ is used to generate the ensembles E_D and E_N whose number of members is in the order of $\mathcal{O}(1000)$ and $\mathcal{O}(10)$, respectively: the numerical model propagates the samples of E_N, while a pre-trained U-Net (Ronneberger *et al.*, 2015) handles the temporal propagation of the numerous members in E_D. The stochastic EnKF algorithm is then applied to the smaller ensemble E_N, but it benefits from the accurate background covariance matrix estimated using the larger sample set.

It is worth mentioning that in this context, the U-Net is not expected to achieve highly accurate long-term forecasts: indeed, the surrogate's predictions are only used to compute a reliable estimate of the background covariance matrix, while the numerical model is specifically utilized for accurate temporal propagations.

3.2.5 Improve observational knowledge

In the theoretical framework of data assimilation, the primary focus is on mathematically deriving accurate and reliable estimates from imperfect initial conditions and noisy observations. However, operational and practical considerations often have to be set aside, as they are case-dependent and require specific solutions. Specifically, efficiently processing observations is a source of concerns in real-world applications like weather forecasting. Despite the unprecedented volume of available observations and sensor measurements, these data are subject to various operational limitations:

- noise that is inherent to data acquisition
- sparsity of collected data
- heterogeneity of observations
- unevenly distributed data
- unstructured data
- placement and number of sensors that can dynamically evolve over time
- time delays among different observation sources
- indirect and possibly nonlinear mappings between observational space and state space

While previous subsections are dedicated to state variables and their covariance matrices, this section explores the potential of deep learning to enhance the utilization of observational data within the data assimilation framework.

As early as 2019, the study by Lin *et al.* (2019) highlighted the advantages of incorporating a neural network into a data assimilation algorithm to address the scarcity of observations. Data assimilation is relatively new to the field of chemical transport, having been in use for only about two decades. This research focuses on predicting PM_{10} – particulate matter of 10 micrometers or smaller in diameter – concentrations in the context of dust storms. One prerequisite for accurate forecasts is knowledge of local human emissions. However, in some cases, no real-time measurements of such emissions are available. To address this, the authors trained a LSTM network to estimate local non-dust aerosol levels using historical observation records from approximately 1000 ground-based stations across China. Operationally, this trained LSTM network can infer local non-dust PM_{10} concentrations from real-time observations. The dust concentration is then determined by subtracting the neural network's estimate from the raw PM_{10} observations.

The accuracy of acquiring observations is undeniably crucial, but equally important is determining the optimal locations for placing sensors (Deng *et al.*, 2018). In this context, the study by Deng *et al.* (2021) reveals an intriguing finding: using only the five most sensitive sensors can yield assimilation performances comparable to those obtained when utilizing observations from the entire flow field. This research employs a feed-forward neural network, augmented with a feature importance layer, to identify the most sensitive locations in the spatial field of a jet flow simulation. To achieve this, a set of model parameters are selected to generate associated velocity fields, enabling the neural network to learn the spatial sensitivity relative to these input parameters.

A significant challenge in data assimilation is handling sparse, unstructured, and time-varying observation data. Addressing this, the study by Cheng *et al.* (2024) introduces VIVID (Voronoi-tessellation Inverse operator for VariatIonal Data assimilation). This innovative approach involves training a neural network to function as an inverse operator. It maps from the observation space to the state space, accommodating unstructured and dynamically evolving sensor measurements over time in terms of both location and quantity. By nature, this defines an ill-posed problem and it thus requires to be integrated into a data assimilation framework (*e.g.*, a variational one).

The concept of Voronoi tessellation (Watson, 1981) involves partitioning space into Voronoi cells based on the locations of observations. A convolutional neural network is trained to reconstruct and estimate the state variable, taking the Voronoi tessellation as input. This methodology aligns with the steps described and implemented in Fukami *et al.* (2021). However, Cheng *et al.* (2024) differs by incorporating this approach into a variational framework to avoid an ill-defined problem. The cost function for the 3D-Var optimization problem is thus defined as:

$$J(\boldsymbol{x}) = \frac{1}{2}\|\boldsymbol{x} - \boldsymbol{x}_t^b\|^2_{\boldsymbol{B}_t^{-1}} + \frac{1}{2}\|\boldsymbol{y}_t - \mathcal{H}_t(\boldsymbol{x})\|^2_{\boldsymbol{R}_t^{-1}}. \tag{3.16}$$

In Cheng *et al.* (2024), an additional regularization term is introduced:

$$J(\boldsymbol{x}) = \frac{1}{2}\|\boldsymbol{x} - \boldsymbol{x}_t^b\|^2_{\boldsymbol{B}_t^{-1}} + \frac{1}{2}\|\boldsymbol{y}_t - \mathcal{H}_t(\boldsymbol{x})\|^2_{\boldsymbol{R}_t^{-1}} + \frac{1}{2}\|\boldsymbol{x} - \boldsymbol{x}_{v,t}\|^2_{\boldsymbol{P}_t^{-1}}, \tag{3.17}$$

where $\boldsymbol{x}_{v,t}$ represents the output of the learned operator when applied to the Voronoi tessellation of the observations at time t. The error covariance matrix of the CNN, $\boldsymbol{P}_t$, is empirically estimated using a validation set. Figure 3.2.1 provides a visual aid for understanding the operational workings of VIVID.

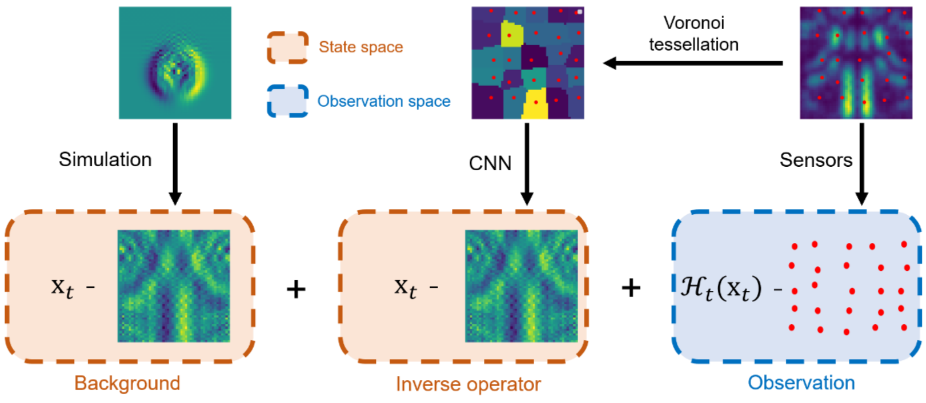


Figure 3.2.1: VIVID architecture. *This illustration is from Cheng* et al. *(2024).*

Furthermore, VIVID is compatible with Principal Component Analysis, enabling its application in reduced-order modeling contexts.

In data assimilation, the observation operator $\mathcal{H}$, which maps physical space to observation space, is non-invertible. This limitation can complicate the minimization process in data assimilation. Addressing this issue, the work by Frerix *et al.* (2021) showcases the potential benefits of a data-driven inverse observation operator, denoted by h_θ, in improving the 4D-Var optimization problem. Specifically, they aim to achieve two primary goals, as depicted in Figure 3.2.2:

- enhancing the initialization of the non-convex optimization problem.
- reformulating the cost function in terms of state variables rather than observational data.

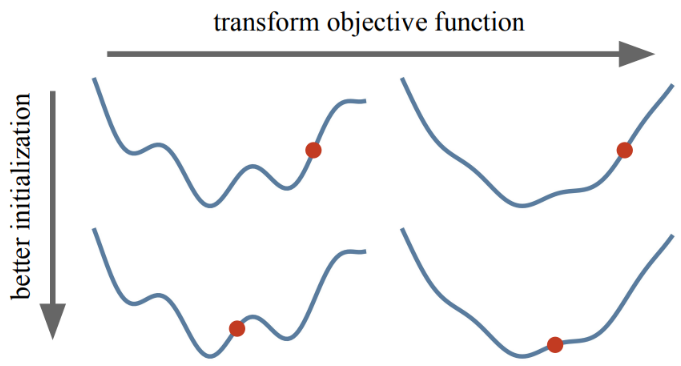


Figure 3.2.2: The two primary goals when learning an inverse observation operator. Credit to Frerix *et al.* (2021)

.

The authors propose an alternative to the standard 4D-Var objective function by neglecting the prior term and setting the error covariance matrix to the identity. The resulting formulation is:

$$J(\boldsymbol{x}_0) = \sum_{t=0}^{T} \|\boldsymbol{y}_t - \mathcal{H}(\boldsymbol{x}_t)\|_2^2, \tag{3.18}$$

where $\boldsymbol{x}_{t+1} = \mathcal{M}(\boldsymbol{x}_t)$. Incorporating the data-driven model h_θ, parameterized by θ, leads to the following reformulated objective:

$$\tilde{J}(\boldsymbol{x}_0) = \sum_{t=0}^{T} \|\boldsymbol{x}_t - h_\theta(\boldsymbol{y}_t)\|_2^2. \tag{3.19}$$

The authors demonstrate that minimizing equation (3.19) is more straightforward than tackling equation (3.18) directly. However, given the limitations of h_θ in accurately reconstructing the state field from observations, minimizing equation (3.19) serves as an intermediate step to establish an initial point for the minimization of equation (3.18).

This innovative method's effectiveness is validated using a Lorenz96 (Lorenz, 1996) system and a Kolmogorov flow (Chandler and Kerswell, 2013). The authors also state that their approach is applicable to the standard 4D-Var minimization problem.

In the same idea of transforming the objective function to get a smoother optimization space, we can mention the work of Fillion *et al.* (2018).

In their research, Liang *et al.* (2023) focus on applying deep learning techniques to directly learn the observation operator for satellite microwave brightness temperature. They define a reference setup that combines the nonhydrostatic icosahedral atmospheric model (NICAM) with the local ensemble transform Kalman filter (LETKF). in this setting, the observation operator is complemented with a bias correction step and is referred to as RTTOV-OO. With they framework, they run a one-month data assimilation and utilize the results along with the corresponding observations, to train a feed-forward neural network. This network named ML-OO is intended to surrogate the traditional observation operator. While the approach is shown to be functional, the neural network, in practice, exhibits slower prediction speeds and less accuracy in assimilating both conventional observations and brightness temperature data, compared to the traditional RTTOV model.

3.2.6 Parameter estimation

In real-world applications, data assimilation frequently deals with dynamics characterized by physical processes occurring at vastly different scales, which can hinder accurate forecasting. Phenomena at the largest and smallest scales exhibit such distinct spatio-temporal behaviors that a grid size or time step suitable for large-scale dynamics may be inadequate for smaller scales, and vice versa. For most real-world cases, the computational expense of resolving both large and small scale variables simultaneously is impractical. Consequently, not all multiscale physical processes are explicitly resolved, yet their interaction necessitates some level of representation. Commonly, *ad-hoc* parameterization schemes for the resolved variables are used to model subgrid scale processes (Stensrud, 2007; Duan and Nadiga, 2006; Randall, 1989). However, devising an interplay between the resolved variables that accurately represents the influence of small-scale dynamics remains a substantial challenge. Thus, incorporating deep learning into data assimilation emerges as a key approach for developing both qualitative and precise closure models: neural networks indeed have the capacity to discern complex interactions between multiscale variables by leveraging the large amount of observational data currently available.

In this vein, the study by Pawar and San (2021) develops a data-driven closure model by hybridizing neural networks with data assimilation. The authors trained three types of feedforward neural networks in a supervised framework to learn small-scale variables from large-scale ones. This method demonstrates effectiveness on the L05III system (Lorenz, 1996), and a Kraichnan turbulence model.

When dealing with parameter estimation, data assimilation often employs the augmented state approach (Jazwinski, 2007; Annan and Hargreaves, 2004; Smith *et al.*, 2013). This method consists in concatenating the state variable with the parameters, creating an extended new variable upon which data assimilation is performed. However, this approach has its limitations: *e.g.*, parameters typically lack direct observations, so their updates rely on cross-correlations, and the assumption of Gaussian distributions may not always hold true.

The study by Legler and Janjić (2022) successfully demonstrates the application of deep learning to parameter estimating within a data assimilation framework. The authors trained a neural network to estimate the parameters of a shallow-water model using sparse and noisy observations. Additionally, the work of Li *et al.* (2022) uses a deep residual neural network to learn the assimilation process in the context of parameter estimation.

3.3 Fully data-driven approaches to surpass traditional numerical weather prediction models

In the field of numerical weather prediction, the increasing availability of large datasets has opened up new opportunities for fully data-driven models. This shift is paving the way for end-to-end learning approaches that directly predict weather from observations. Over the past two years, several tech companies have published several outstanding deep learning-based weather forecast models (Keisler, 2022; Pathak *et al.*, 2022; Lam *et al.*, 2023; Bi *et al.*, 2022; Chen *et al.*, 2023; Price *et al.*, 2023; Nguyen *et al.*, 2023). These models, often utilizing transformers (Vaswani *et al.*, 2017) or graph neural networks (GNNs) (Wu *et al.*, 2023), are trained on decades of ERA5 reanalysis data from ECMWF and are competing with leading operational NWP systems in terms of performance.

These global, data-driven, high-resolution weather models demonstrate performances that are comparable to, or even surpass, those of cutting-edge operational systems. Their accuracy has extended from short-term (1-3 days) to medium-range forecasts (10 days). Their most notable advantage lies in their computational speed, ranging from seconds to a minute, significantly faster than traditional NWP processes that take about an hour. Additionally, such models hold the potential to enlarge current data assimilation ensembles and therefore provide more reliable forecasts. Also, some of these models can predict extreme weather events (Pathak *et al.*, 2022; Lam *et al.*, 2023; Bi *et al.*, 2022; Price *et al.*, 2023) like tropical or extra-tropical cyclones.

Whether utilizing graph networks (Keisler, 2022; Lam *et al.*, 2023; Price *et al.*, 2023) or transformers (Pathak *et al.*, 2022; Bi *et al.*, 2022; Chen *et al.*, 2023), these models commonly employ an autoencoder-type structure, underlining its effectiveness. Transformer architectures benefit from the attention mechanism (Vaswani *et al.*, 2017), while graph neural networks straightforwardly handle Earth's spherical geometry and are suitable for multi-mesh and multi-resolution computations. Figures 3.3.1 and 3.3.2 illustrate the neural network architectures used in Lam *et al.* (2023) and Chen *et al.* (2023), respectively.

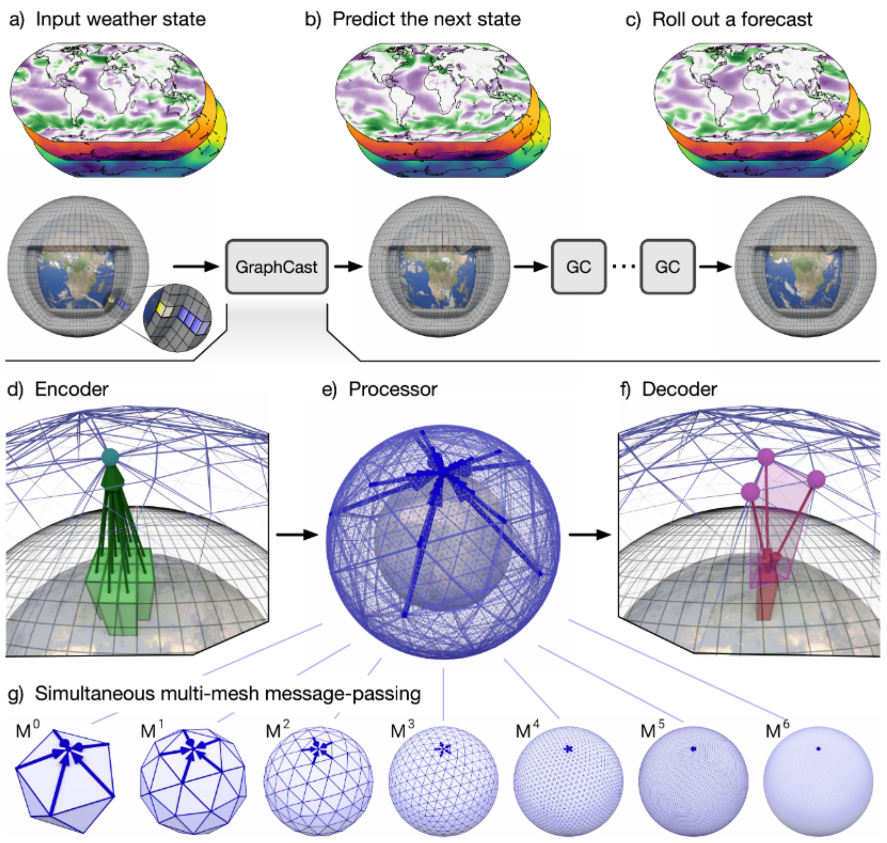


Figure 3.3.1: GNN-based autoencoder architecture of GraphCast (*illustration from* Lam *et al.* (2023)).

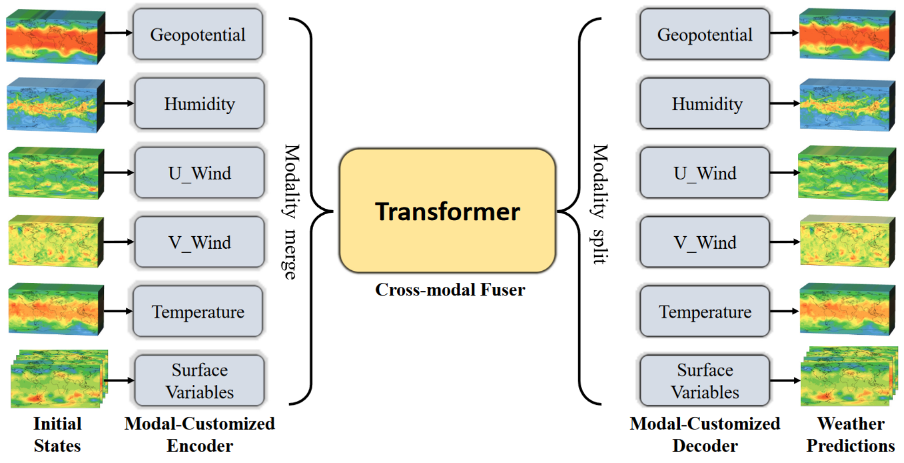


Figure 3.3.2: Transformer-based autoencoder architecture of FengWu model (*illustration from* Chen *et al.* (2023)).

A common practice in these models for efficient autoregressive forecasting is the use of a multi-step loss function, which encourages the network to mitigate error propagation over time. To circumvent the cost of accumulating gradients in memory when using a multi-step loss, Chen *et al.* (2023) introduces the replay buffer approach that consists in adding predicted states into the training dataset during the learning stage: it implicitly mimic the effect of the multi-step loss and thus enhances successful medium-range forecasts.

These fully data-driven models have shown results comparable or superior to current state-of-the-art operational systems (Lam *et al.*, 2023; Bi *et al.*, 2022; Chen *et al.*, 2023; Price *et al.*, 2023; Nguyen *et al.*, 2023). They are revolutionizing NWP by underscoring the benefits of integrating deep learning into operational weather systems. Weather centers like ECMWF are increasingly integrating data-driven approaches into their operational frameworks. Currently, models like FourCastNetv2-small, Graphcast, and Pangu-Weather are available at ECMWF.

The ultimate goal of the deep learning community is to explore the possibility of replacing an entire operational system like IFS with fully data-driven models. Such models would use prior forecasts and current observations to generate new forecasts. The advantages of this approach are numerous, including efficient handling of multi-resolution systems, improved accounting for unresolved physical processes, and significantly faster inference times that facilitate processing large ensembles to enhance forecast accuracy and reliability. Additionally, these models have the potential to more effectively correct model errors and relax Gaussian assumptions.

Although sometimes considered magical, these data-driven models suffer from important hurdles when viewed through the lens of weather forecasting. In particular, performance is tied to the metric used during the training. The widespread choice of the mean squared error (MSE) cost function therefore has smoothing consequences that can be detrimental to the predictions: by averaging over the uncertainties, they can lack fidelity and produce unrealistic and not physically consistent forecasts compared to physics-based models (Bonavita, 2023; Kochkov *et al.*, 2024). The price to pay for a better performance of deep learning model on the MSE criterion is a significantly lower forecast activity (Bouallegue *et al.*, 2023). The smoothing effect also degrades the variability and extremes of the forecast, making neural networks less likely to accurately predict rare weather events, compared to physics-based models. While ensembles provide statistical information and cover different weather scenarios, ML models are limited to generating what appears to be the most likely estimate, in a deterministic manner. In the context of climate modeling, fully data-driven models struggle to produce realistic forecasts over long time windows (Kochkov *et al.*, 2024). Bonavita (2023) provides interesting insight into the differences between deep learning and numerical models, by providing spectral decomposition analyses.

To address these limitations, Kochkov *et al.* (2024) published **NeuralGCM** (Neural General Circulation models for Weather and Climate), a hybrid and fully differentiable model trained on ERA5 reanalysis that couples deep learning components and governing equations, to produce physically consistent forecasts. Additionally, making these interactions between neural networks and physical equations, allows for online training.

Finally, in order to have a fair evaluation of the performance of weather prediction models, Google provides **WeatherBench 2** (Rasp *et al.*, 2024), an online service that evaluates the performance of major published models on several relevant metrics.

3.4 End-to-end learning of the data assimilation process

In the previous sections of this chapter, our focus has largely been on integrating deep learning tools into various aspects of the data assimilation process. Data assimilation, as a field, encompasses a range of mathematical operators, assumptions, and formulations, such as variational and ensemble data assimilation, along with practical challenges that necessitate innovative solutions. Deep learning emerges as a key and valuable technological resource in this context, offering effective and relevant solutions. We have explored numerous methods to cleverly and efficiently combine data assimilation with artificial intelligence, highlighting the potential of this synergistic approach.

Moving beyond mere hybridization, some recent studies have ventured into the realm of fully data-driven data assimilation. They are exploring the potential of what is known as end-to-end learning, aiming to transform the entire data assimilation process into a data-centric paradigm.

As early as 2010, Härter and de Campos Velho (2010, 2012) explored the potential of emulating the Kalman filter using a multilayer perceptron (Härter and de Campos Velho, 2010) and Elman Neural Networks (Härter and de Campos Velho, 2012). This research aimed to reduce computational costs associated with high-dimensional data. While a loss in accuracy compared to the traditional Kalman filter algorithm was observed, both MLP and ENN demonstrated success in a non-linear 1D shallow water problem known as DYNAMO.

The study by Fablet *et al.* (2021) establishes an end-to-end learning framework within a 4D-Var context, allowing for both supervised and unsupervised training strategies with noisy and unevenly distributed data. Tested on the Lorenz63 (Lorenz, 1963) and Lorenz96 (Lorenz, 1996) systems, this approach yields significant improvements over the standard 4D-Var algorithm. The method involves learning a representation of the physical dynamics with a neural network along with a data-driven model that learns to minimize the 4D-Var cost function. Such a minimization is possible thanks to using automatic differentiation tools available in deep learning packages like PyTorch or TensorFlow.

The context of their study is set by the following two equations:

$$\begin{cases} \frac{\partial \boldsymbol{x}}{\partial t}(t) & = \mathcal{M}(\boldsymbol{x}(t)) + \boldsymbol{\eta}(t), \\ \boldsymbol{y}(t,p) & = \boldsymbol{x}(t,p) + \boldsymbol{\varepsilon}(t). \end{cases} \tag{3.20}$$

In this case, the observation operator is the identity, so that the system is fully observed. The flow operator Φ is defined as:

$$\Phi(\boldsymbol{x})(t) = \boldsymbol{x}(t - \Delta t) + \int_{t-\Delta t}^{t} \mathcal{M}(\boldsymbol{x}(u))du. \tag{3.21}$$

For the 4D-Var problem, they do not consider the regularizing background term and assume spherical covariance matrices (i.e. matrices that are proportional to the identity) for both model and observation errors. The cost function therefore reads:

$$U_\theta(\boldsymbol{x}, \boldsymbol{y}) = \lambda_1 \sum_i \|\boldsymbol{x}(t_i) - \boldsymbol{y}(t_i)\|^2 + \lambda_2 \sum_i \|\boldsymbol{x}(t_i) - \Phi(\boldsymbol{x})(t_i)\|^2. \tag{3.22}$$

The authors define Φ via a neural network, either by directly learning this operator or by learning the resolvent $\mathcal{M}_\theta$. This allows the use of automatic differentiation tools to compute $\nabla_{\boldsymbol{x}} U_\theta$, facilitating the application of an iterative gradient scheme:

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} - \alpha \nabla_{\boldsymbol{x}} U_\theta(\boldsymbol{x}^{(k)}, \boldsymbol{y}). \tag{3.23}$$

Given an initial state $\boldsymbol{x}^{(0)}$ and a sequence of observations $\boldsymbol{y}^{(0)}, \dots, \boldsymbol{y}^{(T)}$, the authors demonstrate it is possible to combine two neural networks to solve the 4D-Var problem. Figure 3.4.1 illustrates this end-to-end learning architecture. The network which represents Φ alongside the one that iteratively minimizes the cost function (using residual units of LSTM or RNN cells) are jointly trained together. It is important to remind that the training can be unsupervised or supervised, according to whether ground truth states $\boldsymbol{x}^{(0)}, \dots, \boldsymbol{x}^{(T)}$ are provided.

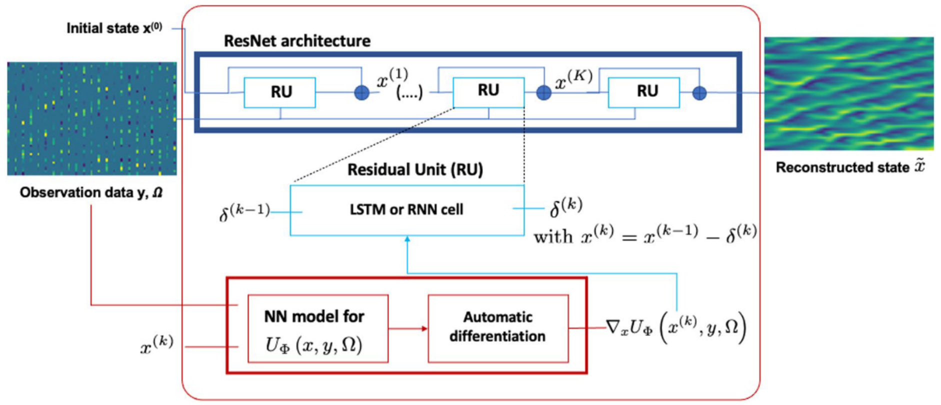

Initial state x⁽⁰⁾
ResNet architecture
RU
$x^{(1)}$ (....)
RU
$x^{(K)}$
RU
Observation data y, Ω
Reconstructed state $\tilde{x}$
Residual Unit (RU)
$\delta^{(k-1)}$
LSTM or RNN cell
$\delta^{(k)}$
with $x^{(k)} = x^{(k-1)} - \delta^{(k)}$
NN model for $U_\Phi(x, y, \Omega)$
Automatic differentiation
$x^{(k)}$
$\nabla_x U_\Phi\left(x^{(k)}, y, \Omega\right)$

Figure 3.4.1: End-to-end learning of the 4D-Var algorithm. *Illustration from* Fablet *et al.* (2021).

This methodology is further extended in Fablet *et al.* (2023) to include the case where both the observation operator and the prior are trainable, leading to the development of the so-called 4D-VarNet.

In the Bayesian data assimilation framework, the study by Boudier *et al.* (2023) introduces a novel end-to-end RNN-based training strategy for approximating prior and posterior probability density functions from noisy observations. This data-driven approach termed Data Assimilation Networks (DAN), optimizes a log-likelihood cost function, analogous to the Kullback-Leibler divergence, in a supervised manner. It is particularly adapted for nonlinear dynamics and non-Gaussian densities. The strategy is built around three key operators:

$$a\colon \mathbb{S} \times \mathbb{Y} \to \mathbb{S}, \text{ (analyzer)}, \tag{3.24}$$

$$b\colon \mathbb{S} \to \mathbb{S}, \text{ (propagator)}, \tag{3.25}$$

$$c\colon \mathbb{S} \to \mathbb{P}_{\mathbb{X}}, \text{ (procoder)}, \tag{3.26}$$

where $\mathbb{S}, \mathbb{Y}, \mathbb{X}$ represent the value spaces of random variables $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{s}$, respectively, and $\mathbb{P}_{\mathbb{X}}$ denotes the space of pdfs over $\mathbb{X}$. As suggested by the notations, $\mathbb{Y}$ corresponds to the observation space, $\mathbb{X}$ to the state space, and $\mathbb{S}$ to an internal representation of the state. Operators a, b, and c perform specific transformations: a assimilates within the memory space $\mathbb{S}$, b performs the time propagation within this space, and c maps to an actual pdf over $\mathbb{X}$. Numerical experiments on the Lorenz96 system (Lorenz, 1996) proves that the accuracy of DAN, without using any explicit regularization technique like localization and inflation, is comparable to the ones reached by IEnKF-Q (Sakov *et al.*, 2018) and LETKF algorithms (Hunt *et al.*, 2007).

CHAPTER 4

Latent Space Data Assimilation

After a detailed introduction to data assimilation in chapter 2, with particular emphasis on ensemble methods, we provided a comprehensive review in chapter 3 of scientific contributions that hybridize data assimilation algorithms with artificial intelligence techniques. Now, we turn our focus to our main research topic, what we have termed **latent space data assimilation**.

In the following, we present the key ingredients that theoretically motivate a data-driven reduced-space data assimilation approach. This methodology is articulated around three main points:

1. exploring the ability of deep learning to create an ℓ-dimensional reduced-space from a full dynamics of dimension n (with $\ell \ll n$).
2. defining a surrogate network within the latent space to perform the time propagation. We introduce an innovative iterative training approach to ensure the surrogate's temporal stability. Additionally, we propose a joint training strategy for the surrogate and the autoencoder, which yields better results compared to sequential training.
3. performing data assimilation within the learned ℓ-dimensional latent space using the trained autoencoder and surrogate.

The proposed methodology:

- offers significant reduction of the computational cost and memory needs.

- may improve the accuracy of the data assimilation analysis.

This chapter also guides the implementation of our novel approach within the framework of ensemble data assimilation. While latent data assimilation is a general concept, we will discuss the features of the ETKF-Q algorithm and feed-forward networks that are specific to our application cases. Nevertheless, it is important to note that this method could be adapted to other reduced rank methods (see section 2.2.2.2) such as RRSQRT (Verlaan and Heemink, 1997; Evensen, 1994), SEEK (Tuan Pham *et al.*, 1998), EnKF (Evensen, 1994; Burgers *et al.*, 1998; Houtekamer and Mitchell, 1998; Evensen, 2009a), or ETKF (Bishop *et al.*, 2001; Hunt *et al.*, 2007). It could also be further adapted to variational algorithms (Melinc and Zaplotnik, 2024).

The organization of this chapter is as follows:

- section 4.1 highlights the limitations encountered in operational data assimilation and details how, in practice, variational and sequential approaches perform computations within a lower-dimensional space. We review papers and algorithms that are used operationally and that fall into the category of reduced-space methods. Within this context, we introduce our latent data assimilation approach and position it in relation to similar scientific papers.
- the introduction of section 4.2 outlines the overall framework of the **latent ETKF-Q (ETKF-Q-L)** algorithm and presents the core concepts of this novel methodology. In the subsections of section 4.2, we first provide relevant and insightful developments about **Principal Component Analysis (PCA)** and chaos theory, which help determine appropriate latent space dimensions. We then derive mathematical connections between PCA and linear autoencoders to gain initial insights into how autoencoders shape the latent space. Following this, we present the role of the surrogate network in latent data assimilation, explaining how the encoder, decoder, and surrogate network work together to propagate the state vector forward in time through the reduced-space. We detail the block-residual architecture of the surrogate network and the loss function we use. Additionally, we discuss the importance of the temporal stability of the surrogate network and provide practical implementation techniques to ensure the trained surrogate network remains stable over time. Lastly, we describe the all-at-once training strategy we implemented.

- section 4.3 exposes how we switch from the ETKF-Q algorithm (see section 2.2.2.3) to its latent version. This section also addresses the challenges posed by the limited number of ensemble members inherent in the ETKF-Q algorithm (and therefore in the ETKF-Q-L as well) and discusses the solution implemented, named **SQRT-DEP** and introduced by Raanes *et al.* (2015).

4.1 Reduced space methods for data assimilation

As reviewed in chapter 2 and chapter 3, a major hurdle in data assimilation — both for sequential and variational approaches — is the significant computational burden involved in real-world applications. Specifically, operational data assimilation deals with high-dimensional, nonlinear, multi-scale, and potentially unstable and chaotic physical dynamics. Consequently, the demands for computational resources and memory far exceed the capabilities of standard computers, necessitating the use of high-performance computing clusters. In sequential data assimilation, the two main computational challenges are the propagation of the covariance matrix at each step and the need to solve large linear systems. On the variational side, the difficulties arise from the need to compute adjoint operators and the potential for the problem to be poorly conditioned. Additionally, both sequential and variational methods must produce real-time analysis to be operationally useful, as delays can render the analysis obsolete.

Several remedies have been proposed to alleviate the computational cost of operational data assimilation algorithms. The core common idea among these methods is to find the solution, namely the analysis, by performing computations within a lower-dimensional space compared to the original state space. These techniques are often referred to as **reduced-space data assimilation methods**.

In sequential data assimilation, ensemble algorithms compute a weighting term $\boldsymbol{w}^a$ lying in $\mathbb{R}^m$ (where m denotes the number of members in the ensemble) to adjust the background $\overline{\boldsymbol{x}}_k^f$ and thereby build the analysis $\boldsymbol{x}_k^a$, as shown in equation (2.49). Operationally, the number of ensemble members affordable is limited, so $m \ll n$. Consequently, the adjustment term added to the background is computed in the ensemble space, $\mathbb{R}^m$, which has a much smaller dimension than the state space $\mathbb{R}^n$. More precisely, given $\boldsymbol{X}_k^f \left(\boldsymbol{X}_k^f\right)^T$ a low-rank approximation of the forecast covariance matrix, the correcting term (expressed as $\boldsymbol{X}_k^f \boldsymbol{w}^a$ in equation (2.49)) lies in the column space of $\boldsymbol{X}_k^f$. However, a downside effect of ensemble approaches relates to the large sampling errors it introduces. They lead to spurious correlations in the covariance matrices and can also make the filter diverge in the absence of proper fixes (inflation and localization).

When considering variational data assimilation, the minimization of the nonlinear least squares problem is performed by iteratively minimizing a series of linear quadratic problems, thereby defining the incremental approach (see section 2.2.1). Similarly to the Kalman filter (see section 2.2.2.1) for which the gain $\boldsymbol{K}^*$ can be equivalently defined by equation (2.14) or equation (2.15) - according to whether the inversion occurs in $\mathbb{R}^n$ or $\mathbb{R}^p$ -, the incremental approach also exhibits dual properties. Courtier (2007) first introduced a dual approach, known as the **Physical Statistical Analysis System** (PSAS), which performs the minimization of the dual objective function in the observation space $\mathbb{R}^p$. As a result, whenever $p \ll n$, the dual minimization offers significant gains in terms of memory and computational costs compared to the primal one. In operational data assimilation, p is about two orders of magnitude smaller than n, making PSAS an appealing approach. However, Gauthier *et al.* (2008) and Gratton and Tshimanga (2009) reported non-monotone and chaotic behavior of PSAS algorithm, resulting in many inner iterations.

Gratton and Tshimanga (2009) proposed an alternative dual approach known as the Restricted Preconditioned Conjugate Gradient (RPCG) method. This method generates the same iterates in exact arithmetic as the primal approach (see section 2 of Gratton and Tshimanga (2009) for a definition of primal and dual approaches), but it significantly reduces memory and computational costs while maintaining the desired convergence properties, unlike the PSAS algorithm. However, the relationship between these two dual approaches and the development of efficient preconditioners are not addressed by Gratton and Tshimanga (2009). Gratton *et al.* (2013) tackles these unresolved issues by designing preconditioning techniques and a trust-region globalization that maintain the one-to-one correspondence between primal and dual iterates, ensuring cost-effective and globally convergent computations.

In practice, the introduction of the incremental approach enabled NWP centers to implement 4D-Var operationally. However, in its original form, incremental 4D-Var remains too computationally expensive for practical use, necessitating further improvements. To address this, it is common to run a simplified version of the linear model, often achieved by lowering the spatial resolution or performing spectral truncation. These methods, however, are driven purely by computational considerations and offer no guarantees regarding the extent of information loss due to these simplifications. Lawless *et al.* (2008) employed balanced truncation (Moore, 1981) to derive a low-order linear model that preserves the important features of the original system. This model is then used in the incremental 4D-Var algorithm to perform the minimization in a reduced-space of dimension r. The Hankel matrix of a dynamical system (Antoulas, 2005) captures its input-output behavior over time: it is a structured matrix that encodes information about the system's impulse response or transfer function. Balanced truncation ensures that the Hankel matrix of the reduced model retains the first r singular values of the full system Hankel matrix. Similar to PCA, balanced truncation provides a restriction operator $\boldsymbol{U}^T \in \mathbb{R}^{r\times n}$ and a prolongation matrix $\boldsymbol{V} \in \mathbb{R}^{n\times r}$ that allow mapping back and forth between $\mathbb{R}^r$ and $\mathbb{R}^n$. Consequently, the incremental 4D-Var minimization is performed within $\mathbb{R}^r$, with the exact linear model $\boldsymbol{M}$ being replaced by $\boldsymbol{U}^T\boldsymbol{M}\boldsymbol{V}$, and the observation operator $\boldsymbol{H}$ with $\boldsymbol{H}\boldsymbol{V}$. In the context of sequential data assimilation, Farrell and Ioannou (2001) employed balanced truncation in a Kalman filter algorithm.

Notably, Robert *et al.* (2005) and Cao *et al.* (2007) utilize PCA to perform a low-dimensional 4D-Var data assimilation.

Despite reducing the computational cost of data assimilation systems, these reduced-space methods still face significant limitations that cannot be overlooked, especially in operational data assimilation. For instance, ensemble algorithms can remain computationally expensive as they require a sufficient number of members to accurately represent the error covariance matrices. Propagating large ensembles can therefore be prohibitive, particularly if the numerical model $\boldsymbol{\mathcal{M}}$ is costly to apply. In large-scale data assimilation systems, $\boldsymbol{\mathcal{M}}$ is typically expensive, adding to the computational burden.

Dual approaches to incremental 4D-Var, such as PSAS or RPCG, share some of the same issues as incremental 4D-Var. Both dual algorithms require the computation of the tangent and the adjoint model and observation operators, which can represent a substantial computational burden in operational settings. Moreover, no solution is provided to reduce the cost of the numerical model $\boldsymbol{\mathcal{M}}$, and the linear systems to solve remain large, despite being smaller than those of the original state space. Similarly to the primal incremental 4D-Var approach, PSAS and RPCG can also struggle with highly nonlinear systems.

Balanced truncation does not alleviate the computational cost issue of incremental 4D-Var, as simplifications in the linear model already existed. Regardless of the choice of linear model approximation (e.g., balanced truncation, low-resolution model, or spectral truncation), incremental 4D-Var remains an expensive algorithm (see conclusion of (Lawless *et al.*, 2008)). Furthermore, balanced truncation is not yet suitable for real-life large-scale systems as pointed out by Lawless *et al.*. Beyond these issues, we can also question the relevance of using a linear mapping to represent the full state variable $\boldsymbol{x}$ into an r-dimensional space. When dealing with nonlinear or highly nonlinear dynamics, there is no guarantee that such a low-dimensional representation will be accurate. This concern also applies to the works of Robert *et al.* (2005) and Cao *et al.* (2007) where PCA is used.

Consequently, while reduced-space methods reviewed in this section overcome significant data assimilation challenges, they still leave room for further improvements. In particular, the use of nonlinear data reduction techniques such as autoencoders (see section 3.1.5) has not been explored. Also, the question of whether it might be possible to perform more accurate data assimilation within the low-dimensional space compared to the original has not been addressed as well.

With the advent of deep learning (see section 3.1) and the ever-increasing amount of available data, integrating neural networks into existing scientific frameworks has allowed researchers to overcome many practical and operational limitations. This integration has also opened new research horizons that were previously inaccessible. In this thesis, which sits at the intersection of data assimilation and deep learning, we aim to perform a data-driven latent space data assimilation, that exhibits significant computational speed-ups and offers the potential for more accurate analysis.

Peyron *et al.* (2021) is the first paper published in a peer-reviewed journal to implement data assimilation within the latent space of an autoencoder. This approach addresses some of the aforementioned issues that were not handled by the reduced-space methods reviewed in this section. Firstly, most real-life dynamics of interest are nonlinear, making PCA transformations or any linear mapping limited in their ability to accurately represent such dynamics within a low-dimensional space. Therefore, we rely on autoencoders (see section 3.1.5) and their nonlinear activation functions to outperform PCA in discovering a suitable latent space for the state variable $\boldsymbol{x}$. Additionally, we train a surrogate neural network to learn the temporal mapping between two successive latent states. In addition to allowing latent space data assimilation, this data-driven model is cheap to apply compared to its numerical full space counterpart. This novel data assimilation framework offers the formidable advantage of being computationally highly efficient while providing the potential for more accurate analysis compared to full-space assimilation. These points are theoretically supported and will be elaborated on later in this chapter.

Building on the work of Peyron *et al.* (2021), subsequent publications (Amendola *et al.*, 2021; Cheng *et al.*, 2022b,a; Yoon and Kadeethum, 2022; Silva *et al.*, 2022; Akbari *et al.*, 2023; Mücke *et al.*, 2024) have further explored these concepts across different applications.

Before exploring the specific features of these works, we present **RODDA** (a Reduced Order Deep Data Assimilation model), a methodology proposed by Casas *et al.* (2020), which innovatively combines reduced-space representations, data assimilation, and deep learning. While not being exactly aligned with the idea of latent space data assimilation, the work Casas *et al.* (2020) is a first step into this direction. The primary aim of RODDA is to enhance the accuracy of Computational Fluid Dynamics (CFD) simulations, by learning the misfit between outputs from the CFD model and the analysis returned by the 3D-Var algorithm (see section 2.1.2). Coupling the physical CFD model with a trained LSTM allows RODDA to produce accurate forecasts at a low cost. The computational burdens induced by the demanding numerical model, the DA algorithm and the neural network training, led the authors to introduce two distinct PCA-based reduced-spaces: one for the CFD model predictions and another for the DA output states. The combination of the reduced-spaces with the LSTM network yields a speed-up gain of 8×10^2, compared to the original data assimilation procedure. RODDA methodology is applied to a real-case air pollution scenario in South London. The CFD numerical simulations are carried out by the open-source computational fluid dynamics code Fluidity(1) (Davies *et al.*, 2011). Contrary to Peyron *et al.* (2021), the reduced-space solely serves the aggregation of the numerical model output with the neural network misfit inference, but the assimilation remains in the full space. Additionally, Casas *et al.* (2020) rely on PCA to create two distinct reduced-spaces, while Peyron *et al.* (2021) make the most of autoencoders to represent the input physical state into a unique low-dimensional space. This work therefore remains very distinct from our approach both in the objective and implementation strategy.

(1)`https://fluidityproject.github.io/`

Similarly to our approach, Amendola *et al.* (2021) performs data assimilation within the latent space of a convolutional autoencoder and train an LSTM network to propagate the latent states forward in time. Although the data assimilation framework and principles are similar to our method, there are some key differences worth mentioning. Specifically, in their approach, the observations are mapped to the latent space, introducing an additional source of errors compared to our methodology. In their process, the observations $\boldsymbol{y}_k \in \mathbb{R}^m$ are first interpolated to form $\hat{\boldsymbol{y}}_k \in \mathbb{R}^n$, which are then mapped into the latent space by the encoder. The observation error covariance matrix and model error covariance matrix are also empirically computed within the latent space, by considering samples of s model states and s observation vectors. Mapping the observations along with the error covariance matrices into the latent space introduces additional errors into the assimilation process compared to our approach. Moreover, our choice to employ an ensemble Kalman Filter with a model error is more challenging and closely aligns with operational data assimilation systems than the standard Kalman filter.

The work of Cheng *et al.* (2022b) builds strongly on Amendola *et al.* (2021), as they also map observations and error covariance matrices to the latent space. One distinctive element in their work is the combination of PCA with the autoencoder to produce an accurate low-dimensional representation of the input data. They employ a 3D-Var cost function, formulated within the latent space. Again, mapping the observations and error covariance matrices to the latent space leads to approximations and, consequently, errors compared to our latent data assimilation approach. The same authors extended this work to heterogeneous observations in Cheng *et al.* (2022a).

In the continuation of our work, Yoon and Kadeethum (2022) performs latent space data assimilation for real-time forecasting in the context of geologic carbon storage. They rely on a convolutional variational autoencoder (see section 3.1.5) to represent 2D distribution of static inputs and aggregate them with dynamic data to perform the time propagation with a LSTM model. Similarly, Akbari *et al.* (2023) train a recurrent surrogate model within a reduced-space obtained by PCA, to perform a latent space Kalman filter analysis, in the context of Boussinesq flows.

One important distinctive aspect of our approach lies in that our autoencoder and surrogate are jointly trained together, which has demonstrated superior results compared to sequential training. Notably, Amendola *et al.* (2021); Cheng *et al.* (2022b,a); Mücke *et al.* (2024) train the autoencoder first and the surrogate afterwards.

4.2 Latent space data assimilation methodology

As pointed out in chapter 2, achieving cost-effective data assimilation in Earth system modeling is crucial, especially in this era of big data where huge quantities of observations are available. Moreover, time integration models that propagate the analysis estimate forward in time represent major computational burdens in data assimilation. They are currently a strong limiting factor to increasing the number of ensemble members, especially in large-scale applications like weather forecasting. Resorting to a surrogate neural network that emulates the effects of the physical solver while leveraging rapid GPU computations offers a promising route to faster and potentially more accurate data assimilation forecasts. Besides, in the context of latent data assimilation, mathematically defining a physical model within the reduced-space that can propagate latent variables is far from trivial: the latent space indeed results from a stochastic optimization process, which precludes making assumptions about its physical/mathematical properties. Therefore, instead of relying on the knowledge-based model to propagate the analysis forward in time (by applying the decoder, the physical model and the encoder), it is advantageous to develop a surrogate network trained to learn the latent time-stepping operation. Our methodology thus involves coupling an autoencoder with a surrogate neural network to perform the assimilation along with time propagation within the latent space, eliminating the need to utilize the physical model.

We therefore leverage the capability of neural networks to discover reduced-space representations and to approximate physical dynamics, integrating these methods into our data assimilation framework. More precisely, we exploit the latent structure provided by autoencoders (see section 3.1.5) to design an ensemble transform Kalman filter (see section 2.2.2.2) operating within a reduced-space, while being faster and more accurate than its original full space version. The model dynamics is also propagated within this latent space using a surrogate neural network. An innovative iterative training approach enforces the surrogate to be stable over time. We adapt the ETKF-Q algorithm (refer to section 2.2.2.3) to include the trained encoder, decoder surrogate networks, thus establishing the **ETKF-Q Latent (ETKF-Q-L) algorithm** (Peyron *et al.*, 2021). Our approach is tested on a tailored instructional version of the Lorenz96 equations (Lorenz, 1996) and on the quasi-geostrophic model from the Object-Oriented Prediction System (OOPS[(2)]) framework, developed collaboratively by ECMWF and Météo-France.

A key aspect of our methodology involves utilizing an autoencoder to identify a low-dimensional representation of the state space data. This allows for data assimilation within a reduced-space, thereby speeding up computations and reducing memory storage usage, while offering the potential for accuracy improvements.

Figure 4.2.1 illustrates how an autoencoder combined with a surrogate neural network can represent a physical dynamics $\boldsymbol{x}$[(3)] into a latent space, and propagate the corresponding latent variable $\boldsymbol{z}$. The decoder ensures that any latent state $\boldsymbol{z}_k$ can be mapped back to the physical space. For the sake of simplicity, the autoencoder and surrogate are depicted as feed-forward neural networks, however, depending on the application, other choices of architectures are possible (convolutional networks, LSTM, graph neural networks, etc.).

[(2)] `https://www.ecmwf.int/en/elibrary/77561-oops-common-framework-research-and-operations`

[(3)] Unlike in chapter 2 and chapter 3, where distinct notations are used for the dimension of $\boldsymbol{x}$, we will now use n for the remainder of the manuscript. This choice is driven by the fact that neural networks now serve the purpose of data assimilation, which allows us to drop the notation N_x in favor of the common notation n to denote the dimension of vector $\boldsymbol{x}$.

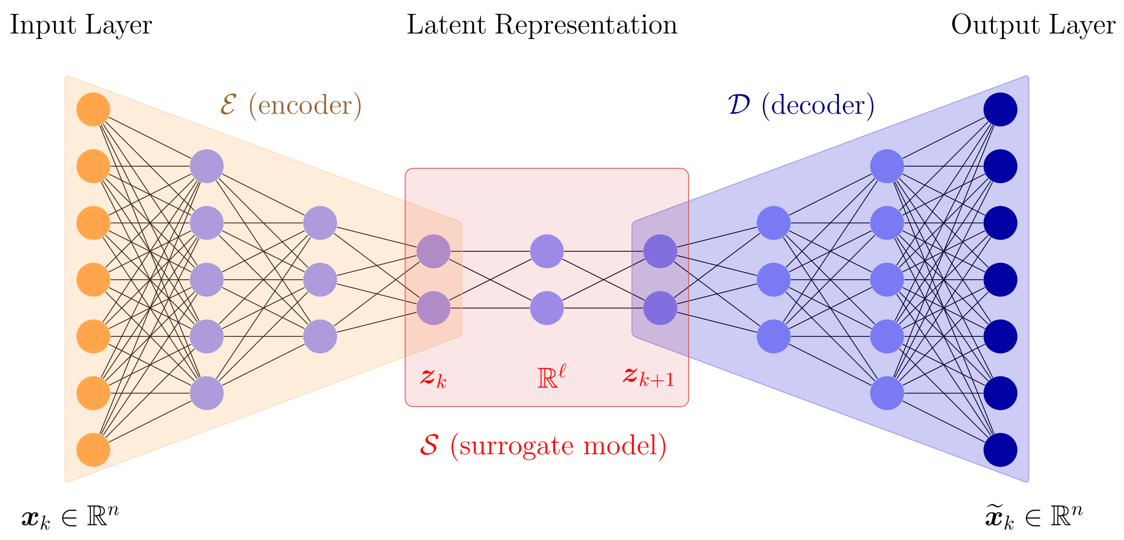


Figure 4.2.1: Illustration of the forward propagation of the physical state $\boldsymbol{x}_k$ through the latent space of an autoencoder. The encoder maps $\boldsymbol{x}_k$ to its latent representation, denoted by $\boldsymbol{z}_k$, which is then propagated one step in time with the surrogate network $\mathcal{S}$, yielding $\boldsymbol{z}_{k+1}$. This forecast latent variable is decoded to get the associated physical state vector $\widetilde{\boldsymbol{x}}_{k+1}$, which is an approximation of the ground truth $\boldsymbol{x}_{k+1}$.

4.2.1 Defining an appropriate latent dimension

By considering training an autoencoder on a physical process, we implicitly assume that the phenomenon possesses an underlying low-dimensional dynamics, allowing the full physical space to be accurately recovered from a limited number of latent dimensions.

Within the framework of this thesis, we consider a physical system of size n for which a latent space of lower dimension ℓ is deemed to exist and in which the observed dynamical system can be described. Our encoder maps from $\mathbb{R}^n$ to $\mathbb{R}^\ell$, whereas the decoder performs the reverse operation. We emphasize that there is absolutely no reason for the latent space of dimension ℓ produced by the autoencoder to be unique. The loss function used in the training promotes coherence between the triplet consisting of the decoder, encoder, and latent space on the one hand, and the data on the other hand. Whenever one particular latent space is discovered (the network is completely free in how it designs the latent space), other latent spaces exist as well and can be simply obtained by transformations such as rotations or changes of scale. We remind the expression of the MSE loss function for autoencoders, as firstly introduced in equation (3.10):

$$\boxed{\mathcal{L}_{AE}\left(\boldsymbol{x}_k; \theta_{\mathcal{E}}; \theta_{\mathcal{D}}\right) = \frac{1}{n}\|\boldsymbol{x}_k - \mathcal{D}(\mathcal{E}(\boldsymbol{x}_k))\|_2^2,} \tag{4.1}$$

where $\boldsymbol{x}_k$ denotes the state variable at time t_k, $\theta_{\mathcal{E}}$ is the encoder's parameters and $\theta_{\mathcal{D}}$ represents the weights of the decoder.

A key question that naturally arises when training an autoencoder, is whether the physical process under consideration can be represented in a lower dimensional space. There is indeed no guarantee that an arbitrarily chosen dynamical system can be accurately mapped back and forth from/to a smaller space.
A first relevant indicator to determine whether a reduced-space representation exists, and if so, to assess a suitable size for the latent space, is the principal component analysis, which is a widely used statistical technique for dimensionality reduction and feature extraction. In the context of latent space data assimilation, it is therefore possible to accurately represent the original data with only a few vectors as long as most variables are linearly correlated. For example, by selecting only the top eigenvectors that capture a cumulative explained variance threshold (e.g., 90% or 95%), we can project the full-space data onto these vectors without significant information loss. This approach is particularly useful for visualizing high-dimensional data, as often only the first two or three eigenvectors are needed for effective 2D/3D representations.

Plotting of cumulative explained variance against the number of eigenvectors can therefore provide a first insight into a possible latent dimension. However, since autoencoders take advantage of nonlinear transformations, their information compressing capability often go beyond the one of PCA. In chapter 5, we calculate the PCA components for the Lorenz96 and quasi-geostrophic datasets and derive the reconstruction scores for different reduced-space dimensions. Alongside these curves, we also plot the reconstruction performances of autoencoders trained for the same range of low dimensions. This comparison allows us to comprehensively assess the relative performances of PCA and autoencoders, enabling us to make an informed decision when selecting the optimal latent dimension for latent space data assimilation. For linearly dependent variables, a linear autoencoder (without activations) performs similarly to PCA (Hinton and Salakhutdinov, 2006; Bourlard and Kamp, 1988; Baldi and Hornik, 1989; Plaut, 2018). More specifically, the optimal weight matrices of a linear autoencoder, and interestingly, of an autoencoder with sigmoid activations as well, match those of the PCA solution.

Since our aim is to perform sequential data assimilation within the latent space of an autoencoder, it is also crucial to take into account DA theory aspects regarding possible important subspaces.

As developed in the introduction of chapter 2, data assimilation is particularly relevant when dealing with chaotic dynamical systems, as encountered in weather forecast or in geophysical applications. Understanding how physical chaotic behaviors can influence the outcome of a data assimilation algorithm is therefore of primary importance. A dynamical system $\boldsymbol{x} \in \mathbb{R}^n$ with an initial state $\boldsymbol{x}_0$, is said to be chaotic if for any perturbation $\boldsymbol{\delta x}_0$, however small, the resulting perturbed system describes a considerably different trajectory than the original variable $\boldsymbol{x}(t)$, after a finite number of time steps. This does not mean that the norm of the difference between the two dynamics keeps increasing over time, but rather that, at some point in time, the knowledge of the perturbed dynamics does not provide any clue about the state of the exact trajectory at the same time step. In particular, most studied chaotic systems have one or more attractor(s), so that the dynamics of various close initial conditions will describe very similar trajectories and cover much the same space regions, but without any global coherence or synchronicity (at least after a finite number of time steps). As a result, a chaotic dynamics does not allow for long-term predictions from any inexactly known initial state.

Contrary to what we could intuitively imagine, chaotic systems do not rely on stochasticity as they are fully deterministic, *e.g.*, the Lorenz96 equations (Lorenz, 1996).

Within the framework of data assimilation, studying and quantifying how the initial error $\boldsymbol{\delta x}_0$ will evolve over time is therefore a significant subject. The insight we provide are based on the detailed work of Carrassi *et al.* (2022). Let us consider an ordinary differential equation (ODE) of the form:

$$\boxed{\frac{\mathrm{d}\boldsymbol{x}}{\mathrm{dt}} = f(\boldsymbol{x}, \boldsymbol{\sigma}),} \tag{4.2}$$

where $\boldsymbol{x} \in \mathbb{R}^n$ is the state variable and $\boldsymbol{\sigma}$ is a set of parameters.

From equation (4.2), we can derive a subsequent ODE that approximates the evolution of the perturbation $\boldsymbol{\delta x}(t)$ over time, given that $\boldsymbol{\delta x}_0$ is sufficiently small and smooth:

$$\boxed{\frac{\mathrm{d}\boldsymbol{\delta x}}{\mathrm{dt}} \approx \frac{\partial \boldsymbol{F}}{\partial \boldsymbol{x}_{|\boldsymbol{x}(t)}} \boldsymbol{\delta x}.} \tag{4.3}$$

At any time t, computing $\boldsymbol{\delta x}(t)$ involves a matrix operator $\boldsymbol{M}(t, \boldsymbol{x}(t_0)) \in \mathbb{R}^{n \times n}$ - hereafter simply denoted by $\boldsymbol{M}$ - from which we can derive the so-called **Lyapunov vectors** and **Lyapunov exponents**, that play a major role in the understanding of error growth and error decay directions and magnitudes over time from $\boldsymbol{\delta x}_0$.

Given $t_0 \in \mathbb{R}^+$, the Oseledets theorem (Kuptsov and Parlitz, 2012) states that the following limit, denoted by $\boldsymbol{S}$, exists:

$$\boxed{\boldsymbol{S} = \lim_{t \to +\infty} \left(\boldsymbol{M}^T \boldsymbol{M} \right)^{1/(t-t_0)}.} \tag{4.4}$$

The eigenvectors of $\boldsymbol{S}$ are called the **forward Lyapunov vectors** (Legras and Vautard, 1996) and the logarithm of their eigenvalues are the **Lyapunov exponents**. While the forward Lyapunov vectors change over time - as matrix $\boldsymbol{M}$ depends on both t and $\boldsymbol{x}(t_0)$ -, and therefore define local properties of the dynamics, the Lyapunov exponents are a limit computed when $t \to +\infty$, independent of $\boldsymbol{x}(t_0)$, and are thus global properties of the flow, characterizing how much the dynamical system is asymptotically stable/unstable. More precisely, the Lyapunov exponents larger than 0 represent the average number of unstable directions, *i.e.*, the dimension of the space within which any error to the state dynamics will, in average, grows exponentially over time. Likewise, the negative Lyapunov exponents relate to the stable directions, namely the subspace for which perturbations asymptotically decay across time. Regarding the null Lyapunov exponents, they define the so-called neutral space. Chaotic systems are therefore characterized by having at least one positive Lyapunov exponent.

The multiplicative ergodic (MET) theorem as presented in (Barreira and Pesin, 2002, Theorem 2.1.2) states that we can also define an equivalent limit to equation (4.4), denoted by $\boldsymbol{S}^{'}$, by considering $\left(\boldsymbol{M}\boldsymbol{M}^T \right)^{1/(t-t_0)}$ when infinitely going backward in time:

$$\boxed{\boldsymbol{S}^{'} = \lim_{t_0 \to -\infty} \left(\boldsymbol{M}\boldsymbol{M}^T \right)^{1/(t-t_0)}.} \tag{4.5}$$

The MET demonstrates that $\boldsymbol{S}$ and $\boldsymbol{S}^{'}$ share the same eigenvalues, but that their eigenvectors are different and therefore named the **backward Lyapunov vectors**. While forward Lyapunov vectors are suitable to detect directions of exponential error growth over time, the backward Lyapunov vectors track the subspace of exponential error decay. Additionally, it is important to explicitly mention that the unstable, neutral and stable subspaces are not fixed, and therefore evolve across time. The **covariant Lyapunov vectors** - which span the Oseledets subspaces (Carrassi *et al.*, 2022), themselves defined from the forward and backward Lyapunov vectors - align with the unstable, neutral and stable subspaces according to their associated Lyapunov exponents.

Understanding the role of the unstable, neutral and stable susbpaces on the assimilation process, is therefore key to further improve current DA algorithms and achieve better accuracy and efficiency (Bocquet *et al.*, 2017; Bocquet and Carrassi, 2017; Gurumoorthy *et al.*, 2017; Grudzien *et al.*, 2018a,b; Carrassi *et al.*, 2022).

When assuming the observation and model operators to be linear, the model operator to perfectly describe the dynamics, and their associated noises to be Gaussian, theoretical results exist for both the Kalman filter and the EnKF algorithms. More precisely, we can prove that the forecast and posterior error covariance matrices asymptotically conform to the unstable-neutral subspace, *i.e.*, the column space of those matrices collapses to the directions of the forward Lyapunov vectors associated with the non-negative Lyapunov exponents. This means that the assimilation process, which essentially aims to correct erroneous forecasts, asymptotically occurs within the unstable-neutral subspace. In ensemble data assimilation, it is important that the number of members is at least equal to the dimension of this unstable-neutral space. Complementary results about the rate of converge of the eigenvalues of $\boldsymbol{P}_k$ (that is related to the Lyapunov exponents), the dependence of $\boldsymbol{P}_k$ to $\boldsymbol{P}_0$, and sufficient conditions for $\boldsymbol{P}_k$ to forget about $\boldsymbol{P}_0$ are also known (Carrassi *et al.*, 2022; Gurumoorthy *et al.*, 2017; Bocquet *et al.*, 2017; Bocquet and Carrassi, 2017). Mathematical results can also be derived when the model is not perfect anymore (Grudzien *et al.*, 2018a,b). When the dynamics is nonlinear, numerical evidences tend to confirm the results known for the linear case (Carrassi *et al.*, 2022).

As we will be referring to it in chapter 5, we briefly introduce the Kaplan and Yorke conjecture (Kaplan and Yorke, 1979). In chaos theory, the Kaplan-Yorke dimension, also known as the Lyapunov dimension quantifies the complexity of strange attractors. An attractor is said to be strange if it has a fractal structure, meaning its Hausdorff dimension (Hausdorff, 1918) is a non-integer. For example, the Lorenz63 system as defined in equation (5.1), has a strange attractor of dimension 2.06. The Kaplan-Yorke dimension D_{KY} is defined using the Lyapunov exponents of the system. For a system with n Lyapunov exponents $\lambda_1, \lambda_2, \ldots, \lambda_n$ sorted in descending order ($\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$), the Kaplan-Yorke dimension is given by:

$$D_{KY} = j + \frac{\sum_{i=1}^{j} \lambda_i}{|\lambda_{j+1}|}, \tag{4.6}$$

where j is the largest integer such that the sum of the sum of the first j Lyapunov exponents is non-negative, that is:

$$\sum_{i=1}^{j} \lambda_i \geq 0 \quad \text{and} \quad \sum_{i=j+1}^{n} \lambda_i < 0.$$

The conjecture has been proven for two-dimensional dynamical systems and for a substantial class of stochastic systems. Generally, the Kaplan-Yorke dimension is known to be an upper bound for the Hausdorff dimension. While it is possible to construct contrived counterexamples, it is widely believed that the conjecture holds true for most "generic" dynamical systems.

Principal component analysis and Lyapunov stability theory provide valuable insights and mathematical tools for defining the latent space dimension. PCA, along with autoencoder reconstruction curves, offers a visual method to select appropriate latent dimensions. This selection ensures a number of degrees of freedom that guarantee faithful data reconstruction while significantly enhancing computational speed and reducing memory costs.

The intersection of PCA and chaos theory thus allows us to define a latent dimension that harmoniously balances the requirements of accuracy and speed, potentially outperforming standard full space assimilation in both criteria.

4.2.2 Searching for a suitable latent space: PCA and autoencoder study

Before delving into the theoretical motivations and practical implementations of latent data assimilation, we first provide some insights into autoencoder learning.

Specifically, we consider a two-layer autoencoder with linear activation functions and no bias, and derive optimal weight matrices for the encoder and the decoder, under different scenarios. The cost function for our optimization problems is the MSE. We show that there are similarities between PCA and the optimal weights of linear autoencoders. The mathematical details are provided in appendix B. Recently, we noted that our study on the optimal solution for autoencoders' weights had already been investigated by Bourlard and Kamp (1988), Baldi and Hornik (1989), and more recently by Plaut (2018). However, we believe it is important and relevant to explicitly present the mathematical connections we derived between PCA and linear autoencoders in this manuscript.

4.2.2.1 An introductory case: PCA

Let us first properly define principal component analysis (PCA).

In 1901, the seminal work of Pearson (1901) early introduced what would later become PCA in 1933, when Hotelling (1933) extended Pearson's work and formalized PCA in its modern matrix decomposition form. PCA yields a basis of sorted orthogonal vectors along which the variance of the data is maximized. With this orthogonal coordinate system, PCA removes correlations, so that each basis vector defines a single feature of the total information that is uncorrelated with the others. By avoiding redundancies and disentangling correlated variables, PCA maximizes the spread of the points in the new basis, along with the amount of information held by each vector. The basis corresponds to the eigenvectors of the covariance matrix computed from the standardized data. They are sorted in descending order of their eigenvalues, so that we can quantify the percentage of information - here, the percentage of explained variance - carried by each vector of the basis.

Let us consider an ensemble $\boldsymbol{X} \in \mathbb{R}^{n\times m}$ containing m samples, each of dimension n: $\boldsymbol{X} = [\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_m]$. We assume $m \ll n$, which is typically encountered in data assimilation.

The covariance matrix associated with $\boldsymbol{X}$ is computed as follows:

$$\boldsymbol{C} = \left(\boldsymbol{X} - \frac{1}{m}\boldsymbol{X}\mathbf{1}\mathbf{1}^T\right)\left(\boldsymbol{X} - \frac{1}{m}\boldsymbol{X}\mathbf{1}\mathbf{1}^T\right)^T \tag{4.7}$$

We denote by $\boldsymbol{\Psi} \in \mathbb{R}^{n\times n}$ the matrix composed of the column-wise concatenation of the eigenvectors of $\boldsymbol{C}$ sorted in descending order of their eigenvalues. Given $r < n$, we denote by $\boldsymbol{\Psi}_r$ the restriction of $\boldsymbol{\Psi}$ to its first r columns. Thus, we have $\boldsymbol{\Psi}_r \in \mathbb{R}^{n\times r}$.

Let us consider ℓ such that $\ell \ll n$. PCA can provide an optimal ℓ-dimensional representation of the data by applying matrix $\boldsymbol{\Psi}_\ell^T$ to the centered input samples.

Let $\boldsymbol{X'}$ denote a validation dataset made up of m' samples. To represent $\boldsymbol{X'}$ in a ℓ-dimensional space, we compute the following:

$$\boldsymbol{X}'_\ell = \boldsymbol{\Psi}_\ell^T \left(\boldsymbol{X}' - \frac{1}{m'} \boldsymbol{X}' \mathbf{1}\mathbf{1}^T \right) \tag{4.8}$$

An approximation of the original data, denoted by $\widetilde{X}'$ can be obtained from $\boldsymbol{X}'_\ell$, by performing the reverse operation:

$$\widetilde{X}' = \boldsymbol{\Psi}_\ell \boldsymbol{X}'_\ell + \frac{1}{m'} \boldsymbol{X}' \mathbf{1}\mathbf{1}^T \tag{4.9}$$

PCA is considered an optimal linear transformation, defined as the solution of the following optimization problem:

$$\max_{\substack{\boldsymbol{\Psi}_\ell \in \mathbb{R}^{n\times\ell} \\ \boldsymbol{\Psi}_\ell^T \boldsymbol{\Psi}_\ell = \boldsymbol{I}_\ell}} \sum_{i=1}^{m} \left\| \boldsymbol{\Psi}_\ell^T \boldsymbol{x}_i - \frac{1}{m} \sum_{j=1}^{m} \boldsymbol{\Psi}_\ell^T \boldsymbol{x}_j \right\|_2^2 \tag{4.10}$$

which is also equivalent to the following one:

$$\max_{\substack{\boldsymbol{\Psi}_\ell \in \mathbb{R}^{n\times\ell} \\ \boldsymbol{\Psi}_\ell^T \boldsymbol{\Psi}_\ell = \boldsymbol{I}_\ell}} Tr\left[\boldsymbol{\Psi}_\ell^T \boldsymbol{X} \left(\boldsymbol{I}_m - \frac{1}{m} \mathbf{1}\mathbf{1}^T \right) \boldsymbol{X}^T \boldsymbol{\Psi}_\ell \right] \tag{4.11}$$

4.2.2.2 Linear autoencoder to find a reduced-subspace

1-dimensional case:

To better understand how autoencoders learn and determine the optimal solution matrices that lead to the best reconstruction score, we start with a simple case where the dimension of the reduced-space is one, and the unknown is a single vector (as opposed to regular autoencoders for which the encoder and decoder have distinct variables).

We aim to find a linear mapping $\boldsymbol{u}_1 : \mathbb{R}^n \to \mathbb{R}$ that best represents $\boldsymbol{X} \in \mathbb{R}^{n\times m}$ in a 1-dimensional space, under the constraint $\boldsymbol{u}_1^T \boldsymbol{u}_1 = 1$ (*i.e.*, $\boldsymbol{u}_1$ is a unit vector).

Mathematically, the problem we define can read:

$$\boldsymbol{u}_1^* = \underset{\boldsymbol{u}_1 \in \mathbb{R}^n \text{ s.t } \boldsymbol{u}_1^T \boldsymbol{u}_1 = 1}{\arg\min} \left\| \boldsymbol{u}_1 \boldsymbol{u}_1^T \boldsymbol{X} - \boldsymbol{X} \right\|_F^2 \tag{4.12}$$

where $\|.\|_F$ denotes the Frobenius norm.

The solution to equation (4.12) is the unit vector associated with the largest eigenvalue of matrix $\boldsymbol{C} = \boldsymbol{X}\boldsymbol{X}^T$. Therefore, it matches the PCA solution. the mathematical derivations leading to this result can be found in appendix B.1.

We now consider the scenario where we add one degree of freedom to the optimization problem defined in equation (4.12) by introducing variable $\boldsymbol{v}_1$. The aim here is to better fit the structure of an autoencoder, for which the encoder and the decoder weights are distinct matrices. We therefore define the following optimization problem, with $\boldsymbol{u}_1 \in \mathbb{R}^{n\times 1}$ and $\boldsymbol{v}_1 \in \mathbb{R}^{n\times 1}$:

$$\boldsymbol{u}_1^*, \boldsymbol{v}_1^* = \underset{\boldsymbol{u}_1, \boldsymbol{v}_1 \in \mathbb{R}^{n\times 1}}{\arg\min} \left\| \boldsymbol{u}_1 \boldsymbol{v}_1^T \boldsymbol{X} - \boldsymbol{X} \right\|_F^2 \tag{4.13}$$

We can mathematically derive that a solution of equation (4.13) is a pair $(\boldsymbol{u}_1^*, \boldsymbol{v}_1^*)$ such that $\boldsymbol{u}_1^*$ is the eigenvector of $\boldsymbol{C} = \boldsymbol{X}\boldsymbol{X}^T$ associated with the largest eigenvalue, and $\boldsymbol{v}_1^*$ is any vector of $\mathbb{R}^n$ such that $(\boldsymbol{v}_1^*)^T \boldsymbol{u}_1^* = 1$. Mathematical details are provided in appendix B.3.

ℓ–dimensional case

We extend our previous analysis from appendix B.1 to the more realistic case when the dimension of the reduced-space is an arbitrary value ℓ within the range $[\![1, n]\!]$. Let us consider $\boldsymbol{U}_\ell \in \mathbb{R}^{n\times \ell}$ and define the following minimization problem:

$$\boldsymbol{U}_\ell^* = \underset{\boldsymbol{U}_\ell \in \mathbb{R}^{n\times \ell} \, s.t \, \boldsymbol{U}_\ell^T \boldsymbol{U}_\ell = \boldsymbol{I}_\ell}{\arg\min} \left\| \boldsymbol{U}_\ell \boldsymbol{U}_\ell^T \boldsymbol{X} - \boldsymbol{X} \right\|_F^2 \tag{4.14}$$

If we denote by $\lambda_1^*, \dots, \lambda_\ell^*$ the ℓ largest eigenvalues of $\boldsymbol{C}$ and $u_1^*, \dots, u_\ell^*$ their associated eigenvectors, the solution of equation (4.14) is the following pair of matrices:

$$\boldsymbol{U}_\ell^*,\, \boldsymbol{\Lambda}^* = \begin{bmatrix} \vdots & \vdots & \dots & \vdots \\ u_1^* & u_2^* & \dots & u_\ell^* \\ \vdots & \vdots & \dots & \vdots \end{bmatrix},\ \begin{bmatrix} \lambda_1^* & 0 & \dots & 0 \\ 0 & \lambda_2^* & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \lambda_\ell^* \end{bmatrix} \tag{4.15}$$

Here again, we match the PCA solution. Mathematical details to derive this solution are provided in appendix B.2.

A standard autoencoder represents the input data within an ℓ-dimensional latent space, where ℓ is set between 1 and n, and is generally much lower than n. Unlike PCA, which uses a single mapping $\boldsymbol{\Psi}_\ell$ to convert physical space data into their latent versions and vice versa, an autoencoder relies on two distinct operators: the encoder and the decoder. We now mathematically define an optimization problem that fits the autoencoder structure, by setting $\boldsymbol{U}_\ell \in \mathbb{R}^{n\times\ell}$ and $\boldsymbol{V}_\ell \in \mathbb{R}^{n\times\ell}$. The optimization problem is given by:

$$\boldsymbol{U}_\ell^*, \boldsymbol{V}_\ell^* = \underset{\boldsymbol{U}_\ell, \boldsymbol{V}_\ell \in \mathbb{R}^{n\times\ell}}{\arg\min} \left\| \boldsymbol{U}_\ell \boldsymbol{V}_\ell^T \boldsymbol{X} - \boldsymbol{X} \right\|_F^2 \tag{4.16}$$

$\boldsymbol{U}_\ell \boldsymbol{X}$ comprises eigenvectors of $\boldsymbol{C}$ and $\boldsymbol{\Lambda}$ holds their related eigenvalues. Note that $\boldsymbol{W}$ and $\boldsymbol{C}$ share the same eigenvalues. Thus, the optimization problem is equivalent to searching for $\boldsymbol{U}_\ell^*$ and $\boldsymbol{V}_\ell^*$ such that $Tr\left((\boldsymbol{V}_\ell^*)^T \boldsymbol{C} \boldsymbol{U}_\ell^*\right)$ is the sum of the ℓ largest eigenvalues of $\boldsymbol{C}$.

4.2.3 Surrogate networks to propagate the latent dynamics

We aim to train a surrogate network to perform time propagation of the model dynamics within the latent space obtained by the autoencoder (AE). Therefore, our surrogate network is trained using encoded data as input and produces the corresponding latent forecasts as output.

In the architecture of the surrogate, we found it decisive to use so-called **skip-connections** (He *et al.*, 2016) which is now a common and good practice. This consists in adding the result of layer i to the one of layer $i+1$ in the form $\boldsymbol{z} = \boldsymbol{z} + layer_i(\boldsymbol{z})$. In this way, we predict the increment needed to reach $\boldsymbol{z}_{k+1}$ from $\boldsymbol{z}_k$ rather than the raw output directly. We go a step further in using the updated version of skipconnections proposed by Bachlechner *et al.* (2021) that performs better: $\boldsymbol{z} = \boldsymbol{z} + \alpha_i \times layer_i(\boldsymbol{z})$ where α_i are trainable parameters. The better results obtained when training the surrogate with versus without the modulation parameters α_i, clearly reveal that they are clearly beneficial. Figure 4.2.2 exposes the architecture of the surrogate network.

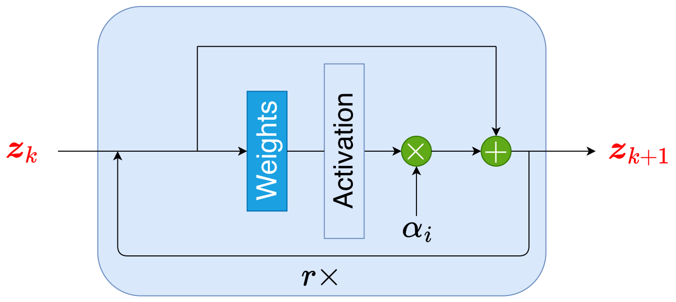


Figure 4.2.2: Architecture of the surrogate network based on residual blocks (He *et al.*, 2016), and more precisely on an updated version of them Bachlechner *et al.* (2021). The network consists of r layers of dimension ℓ, followed by an activation function: each activation is generally meant to be nonlinear, except for the last layer for which we use the identity function. Network's parameters $\boldsymbol{\alpha}_i$ are multiplicative coefficients that weights the residual vector at layer i, before it is summed to the input. Looping back after the summation operator allows a depiction of the network in a concise form.

4.2.3.1 Temporal stability of the surrogate networks

When performing time propagation with a numerical or data-driven model, ensuring the long-term stability of forecasts is crucial. In NWP, it is important to be able to generate medium and long-range predictions, not only to simply forecast the weather but also to provide insights into various possible weather scenarios through ensemble forecasts. Even though the quality of predictions naturally degrades over time, ensemble members can still indicate weather tendencies. Therefore, it is desirable to have stable data-driven models. In addition to accuracy, stability is thus a significant criterion for determining the relevance of a given model.

To achieve stable neural networks, we consider two primary approaches: a chained loss training and Lipschitz neural networks.

We assess the performance of our data-driven latent models, by computing and plotting different metrics and scores, later detailed in section 5.1.3.

Chained loss

Similar to the AE, a straightforward approach would be to train the surrogate network using an MSE loss function as follows:

$$\mathcal{L}_{Sur}(\boldsymbol{x}_k, \boldsymbol{x}_{k+1}; \theta_{\mathcal{S}}) = \frac{1}{n}\|\boldsymbol{x}_{k+1} - \mathcal{T}(\boldsymbol{x}_k)\|_2^2, \tag{4.17}$$

where $\boldsymbol{x}_k, \boldsymbol{x}_{k+1} \in \mathbb{R}^n$ are the state vectors at time t_k and t_{k+1}, respectively and the operator $\mathcal{T}$ is such that $\mathcal{T}(.) = \mathcal{D}(\mathcal{S}(\mathcal{E}(.)))$ with $\mathcal{E}$, $\mathcal{S}$ and $\mathcal{D}$ denoting the encoder, the surrogate and the decoder, respectively.

Nonetheless, as shown later in section 5.1.3, training our surrogate network with equation (4.17) may not yield a stable solution. This is especially easy to understand when the dynamics under consideration is chaotic, as is often encountered in data assimilation. In this case, if the nonvanishing components of the dynamics are not represented with enough accuracy, the surrogate dynamics is expected to be of insufficient quality. This effect is aggravated when the original dynamics exhibits conservative components; if the surrogate dynamics does not capture these components accurately enough, nonphysical unstable subspaces may occur, making the latent space time stepping with the surrogate inappropriate for DA.

Issues related to stable NNs approximation of time stepping methods have already been investigated in the literature, albeit outside of our DA context. They have been linked to exploding or vanishing gradients issues as well as the robustness of NNs. Haber and Ruthotto (2017); Haber *et al.* (2019) obtain some insight in this direction by proposing groundbreaking methods to make deep neural networks stable. However, the problem they address is not exactly the one we are looking at: they focus on the robustness of deep neural networks to input perturbations, on their capability to distinguish between two initial vector states, that is not to bring both of them to 0 nor cause them to diverge.

Within the framework of DA, the presence of non-physical unstable dynamics components is controlled by using a simple penalty approach involving a technique we now describe.

Our method relies on a chained loss function (Peyron *et al.*, 2021), meaning that we train the surrogate to predict c successive states to enforce stability. In practice, given $\boldsymbol{x}_k \in \mathbb{R}^n$, the encoder yields $\boldsymbol{z}_k \in \mathbb{R}^\ell$. Then, the surrogate outputs

$\boldsymbol{z}_{k+1}, \ldots \boldsymbol{z}_{k+c}$ which are all decoded afterwards and their distances to the ground-truth states are measured through a custom loss function defined as follows:

$$\boxed{\mathcal{L}_{Sur}(\boldsymbol{x}_k, \boldsymbol{x}_{k:k+C}; \theta_{\mathcal{S}}) = \sum_{c=1}^{C} \frac{1}{n} \|\boldsymbol{x}_{k+c} - \mathcal{T}^c(\boldsymbol{x}_k)\|_2^2,} \tag{4.18}$$

where $\mathcal{T}^c$ is a straightforward extension of operator $\mathcal{T}$ and is defined as $\mathcal{T}^c(.) = \mathcal{D}(\mathcal{S}^c(\mathcal{E}))$, and $\boldsymbol{x}_{k:k+C}$ denotes the sequence $[\boldsymbol{x}_k, \boldsymbol{x}_{k+1}, \ldots, \boldsymbol{x}_{k+C}]$. The notation $\mathcal{S}^c$ indicates that the surrogate is applied c times in a row over the given data.

One remaining question is the number of iterations C one must perform to achieve this stability criterion: according to our numerical experiments, just 2 consecutive predictions already guarantee a stable behaviour. In the numerical tests we performed, we picked this parameter in $\{2, 3, 4\}$.

Lipschitz neural networks

Lipschitz neural networks are particularly relevant for developing stable surrogate networks because the Lipschitz property ensures that the largest singular value of each weight matrix is bounded. This limitation reduces the risk of the network generating directions of exponential error growth. Moreover, the lower the Lipschitz constant, the more stable the resulting trained network becomes.

Let us consider a function g defined by:

$$\boxed{\begin{aligned} g\colon \mathcal{X} &\to \mathcal{Y} \\ \boldsymbol{x} &\mapsto g(\boldsymbol{x}), \end{aligned}} \tag{4.19a}$$

where $\mathcal{X}$ and $\mathcal{Y}$ are two metrics spaces, with metrics $d_{\mathcal{X}}$ and $d_{\mathcal{Y}}$ respectively.

For any $k \geq 0$, a function g is considered k-Lipschitz if, for all $\boldsymbol{x}_1, \boldsymbol{x}_2 \in \mathcal{X}$:

$$d_{\mathcal{Y}}(g(\boldsymbol{x}_1) - g(\boldsymbol{x}_2)) \leq k \times d_{\mathcal{X}}(\boldsymbol{x}_1 - \boldsymbol{x}_2), \tag{4.20}$$

with k as the Lipschitz constant.

A neural network, being a function between two vector spaces (see equation (3.4a)), can thus be defined as a k-Lipschitz network if it adheres to this condition (Anil *et al.*, 2019).

The building block of Lipschitz neural networks is the 1-Lipschitz network. By constructing a 1-Lipschitz network and scaling its output by any desired constant k, we obtain a k-Lipschitz network. Similarly, we define the extended Lipschitz network - abbreviated as ext-Lipschitz -, which consists in adding a regular (*i.e.*, not a 1-Lipschitz) layer at the end of a 1-Lipschitz network. The term "extended Lipschitz network" is used instead of "k-Lipschitz" to avoid confusion with the strict mathematical definition of a k-Lipschitz function or network as described in equation (4.20). This terminology also clarifies that, in our case, the exact Lipschitz constant is not predetermined.

1-Lipschitz networks are of interest in the context of temporally stable data-driven models because they ensure that the discrepancy between the predicted state and the ground truth is bounded in norm by the perturbation to the initial state $\boldsymbol{x}_k$. Specifically, the difference between $\mathcal{S}(\boldsymbol{x}_k)$ and $\mathcal{S}(\boldsymbol{x}_k + \boldsymbol{\delta}_k)$ is constrained by the norm of $\boldsymbol{\delta}_k$, ensuring bounded variations and enhancing model reliability (the use of index 'k' can be a bit misleading here, as it serves dual purposes in this paragraph, both as a temporal index and a Lipschitz constant). The advantage of the extended Lipschitz networks lies in the relaxation of the 1-Lipschitzness constraint, while encouraging the Lipschitz constant to remain low.

To achieve a 1-Lipschitz constraint, every layer of a neural network, including both affine transformations and nonlinear activations, must also be 1-Lipschitz. This requirement primarily affects the weights since standard activation functions like *Sigmoid*, *ReLU*, *Leaky ReLU*, and *Tanh* naturally adhere to the 1-Lipschitzness property. The constraint on the network weights is closely tied to their spectral norm, that is their largest singular value. The authors employ the Björck orthonormalization algorithm (Björck and Bowie, 1971) to the weight matrices, so that all their singular values are equal to 1: it consists in computing the closest orthonormal matrix of the input, in an iterative manner (Taylor expansion of the polar decomposition). The algorithm has the desirable property to be fully differentiable, making it particularly suitable for neural networks trainings. Besides, Björck method has the advantage to preserve the norm of the gradients across layers during back-propagation: Anil *et al.* (2019) indeed identified this property as key in order to obtain expressive neural networks. Since most activations do not guarantee the norm preservation of the gradient, the authors propose utilizing a specific activation, called **GroupSort** (Chernodub and Nowicki, 2016; Anil *et al.*, 2019).

Despite the constraints, the modified networks (especially those using GroupSort) can universally approximate Lipschitz functions. This is significant because it shows that adding these constraints does not prevent the networks from being able to learn a wide variety of functions.

In order to train 1-Lipschitz and ext-Lipschitz neural networks, we rely on the open source Python API **Deel-torchlip**[(4)], developed by Serrurier *et al.* (2020). Similarly to the chained loss, the utilization of 1-Lipschitz and ext-Lipschitz neural networks enables to enforce the stability of the surrogate network.

While the chained loss function strategy is a statistical method that can only affect certain directions of the latent trajectory, the 1-Lipschitz and extended Lipschitz networks impose global constraints, which makes them less flexible. However, the autoencoders can still adapt the shape of the latent space to better accommodate these stringent Lipschitz constraints.

4.2.4 An all-at-once training strategy

As is often the case when optimizing functions of several variables, performing sequential optimization by groups of variables may be appealing since it reduces the search space in each optimization step. However it generally leads to a suboptimal solution. In our case too, numerical experiments (not reported in this manuscript), showed that training both the AE and surrogate together gives better results than training the AE first and then the surrogate. Since the quality of the AE influences the performances of the surrogate, such combined training allows them to "communicate" and "share" information to learn more properly: the latent space is designed to fit the surrogate time-stepping operation and vice versa. To achieve this, we define a custom loss function with a weighting parameter ρ that balances between equation (4.17) and equation (4.18):

$$\boxed{\mathcal{L}(\boldsymbol{x}_k, \boldsymbol{x}_{k:k+C}; \theta_{\mathcal{E},\mathcal{D},\mathcal{S}}) = \mathcal{L}_{AE}(\boldsymbol{x}_{k:k+C}; \theta_{\mathcal{E},\mathcal{D}}) + \rho \times \mathcal{L}_{Sur}(\boldsymbol{x}_k, \boldsymbol{x}_{k:k+C}; \theta_{\mathcal{E},\mathcal{D},\mathcal{S}}),} \tag{4.21}$$

where $\theta_{\mathcal{E},\mathcal{D},\mathcal{S}}$ and $\theta_{\mathcal{E},\mathcal{D}}$ are shorts for $\theta_{\mathcal{E}}; \theta_{\mathcal{D}}; \theta_{\mathcal{S}}$ and $\theta_{\mathcal{E}}; \theta_{\mathcal{D}}$ respectively, and $\mathcal{L}_{AE}(\boldsymbol{x}_{k:k+C}; \theta_{\mathcal{E},\mathcal{D}})$ and $\mathcal{L}_{Sur}(\boldsymbol{x}_k, \boldsymbol{x}_{k:k+C}; \theta_{\mathcal{E},\mathcal{D},\mathcal{S}})$ are defined as follows:

[(4)] https://github.com/deel-ai/deel-torchlip

$$\boxed{\mathcal{L}_{AE}\left(\boldsymbol{x}_k;\theta_{\mathcal{E}};\theta_{\mathcal{D}}\right)=\sum_{c=0}^{C}\frac{1}{n}\|\boldsymbol{x}_{k+c}-\mathcal{D}(\mathcal{E}(\boldsymbol{x}_{k+c}))\|_2^2,} \tag{4.22}$$

and

$$\boxed{\mathcal{L}_{Sur}(\boldsymbol{x}_k,\boldsymbol{x}_{k:k+C};\theta_{\mathcal{E}};\theta_{\mathcal{D}};\theta_{\mathcal{S}})=\sum_{c=1}^{C}\frac{1}{n}\|\boldsymbol{x}_{k+c}-\mathcal{T}^c(\boldsymbol{x}_k)\|_2^2.} \tag{4.23}$$

4.3 Latent ETKF-Q (ETKF-Q-L)

Our goal with the ETKF-Q-L algorithm is to perform DA analysis within the latent space of our autoencoder. Indeed, we now assume the existence of a n-dimensional system possessing a latent representation of much lower dimension ℓ (*i.e.*, $\ell \ll n$). Algorithm 4.1 describes the changes we made to do so. When applied to an ensemble, $\boldsymbol{\mathcal{H}}_k$ is a columnwise operator. We highlight that ensemble $\boldsymbol{E}_0 \in \mathbb{R}^{n\times m}$ is first encoded into $\boldsymbol{Z}_0 \in \mathbb{R}^{\ell\times m}$ and then all computations happen within the latent space. To calculate the misfit vector $\boldsymbol{d}_k = \boldsymbol{y}_k - \overline{\boldsymbol{\mathcal{H}}_k(\boldsymbol{x}_k)}$, first the decoder $\mathcal{D}$ is used to map the ensemble from the latent space to the full space, then the observation operator $\boldsymbol{\mathcal{H}}_k$ maps the decoded ensemble to the observation space. Therefore, we do not need to perform any transformation to the observations: they remain in their original space. Since time propagation is performed in the latent space, we no longer refer to matrix $\boldsymbol{Q}$ but rather introduce $\boldsymbol{Q}_\ell$. Instead of using $\boldsymbol{\Delta}_k$ to represent a deviation matrix, we refer to it as $\boldsymbol{\Gamma}_k$ in the case of the latent algorithm. For simplicity, we assume that $\boldsymbol{R} = \boldsymbol{\sigma}_R^2\boldsymbol{I}_p$ and $\boldsymbol{Q}_\ell = \boldsymbol{\sigma}_{Q_\ell}^2\boldsymbol{I}_\ell$. Standard deviation $\boldsymbol{\sigma}_{Q_\ell}$ represents the error committed by the surrogate network within the latent space, and is a tuned parameter of the data assimilation process.

The overall architecture of our DA framework is illustrated in Figure 4.3.1.

Algorithm 4.1: Latent ETKF-Q algorithm

```
Inputs:
    Background ensemble $\boldsymbol{E}_0^f = \{\boldsymbol{x}_0^1, \ldots, \boldsymbol{x}_0^m\} \in \mathbb{R}^{n\times m}$ ;
    Observations $\{\boldsymbol{y}_0, \ldots, \boldsymbol{y}_T\} \in \mathbb{R}^{p\times(T+1)}$ ;
    Inflation parameter $\lambda \in \mathbb{R}$.
Initialization:
    Construct $\boldsymbol{U}_m$ matrix such that $\left[\frac{\mathbf{1}_m}{\sqrt{m}}\ \boldsymbol{U}_m\right]$ is orthonormal ;
    Define $\boldsymbol{\mathcal{U}} = \left[\frac{\mathbf{1}_m}{m}\ \frac{\boldsymbol{U}_m}{\sqrt{m-1}}\right]$ ;
    Define $\boldsymbol{Z}_0 = \mathcal{E}(\boldsymbol{E}_0)$ ;
1  for $k = 0, 1, \ldots, T$ do
                  Analysis step
       // Mean and deviation matrix of forecast ensemble
2      $\left[\overline{\boldsymbol{z}}_k^f\ \boldsymbol{\Gamma}_k\right] = \boldsymbol{Z}_k^f \times \boldsymbol{\mathcal{U}}$
       // Calculate eigenpairs of the error covariance matrix
3      $\left(\boldsymbol{\Gamma}_k\boldsymbol{\Gamma}_k^T + \boldsymbol{Q}_\ell\right)\boldsymbol{V}_k \approx \boldsymbol{V}_k\boldsymbol{\Lambda}_k$
4      where $\boldsymbol{V}_k \in \mathbb{R}^{\ell\times(m-1)}$ and $\boldsymbol{\Lambda}_k \in \mathbb{R}^{(m-1)\times(m-1)}$
       // Reassign $\boldsymbol{\Gamma}_k$ accordingly
5      $\boldsymbol{\Gamma}_k = \boldsymbol{V}_k\boldsymbol{\Lambda}_k^{1/2}$
       // Update ensemble with new statistics
6      $\boldsymbol{Z}_k^f = \left[\overline{\boldsymbol{z}}_k^f\ \boldsymbol{\Gamma}_k\right] \times \boldsymbol{\mathcal{U}}^{-1}$
       // Mean and deviation matrix of observation ensemble
7      $\left[\overline{\boldsymbol{y}}_k\ \boldsymbol{Y}_k^f\right] = \boldsymbol{\mathcal{H}}_k\left(\mathcal{D}\left(\boldsymbol{Z}_k^f\right)\right) \times \boldsymbol{\mathcal{U}}$
       // Compute transform matrix $\boldsymbol{T}_k \in \mathbb{R}^{(m-1)\times(m-1)}$
8      $\boldsymbol{T}_k\boldsymbol{T}_k^T = \left(\boldsymbol{I}_{m-1} + \left(\boldsymbol{Y}_k^f\right)^T \boldsymbol{R}^{-1}\boldsymbol{Y}_k^f\right)^{-1}$
       // Compute $\boldsymbol{w}^a$
9      $\boldsymbol{w}_k^a = \boldsymbol{T}_k\boldsymbol{T}_k^T\boldsymbol{Y}_k^f\boldsymbol{R}^{-1}(\boldsymbol{y}_k - \overline{\boldsymbol{y}}_k)$
       // Generate posterior ensemble
10     $\boldsymbol{Z}_k^a = \overline{\boldsymbol{z}}_k\mathbf{1}_m^T + \lambda \times \boldsymbol{\Gamma}_k\left(\boldsymbol{w}_k^a\mathbf{1}_m + \sqrt{m-1}\boldsymbol{T}_k\right)$
                  Propagation step
       // Forecast posterior ensemble
11     $\boldsymbol{Z}_{k+1}^f = \boldsymbol{\mathcal{M}}_{k+1}(\boldsymbol{Z}_k^a)$
12 end
```

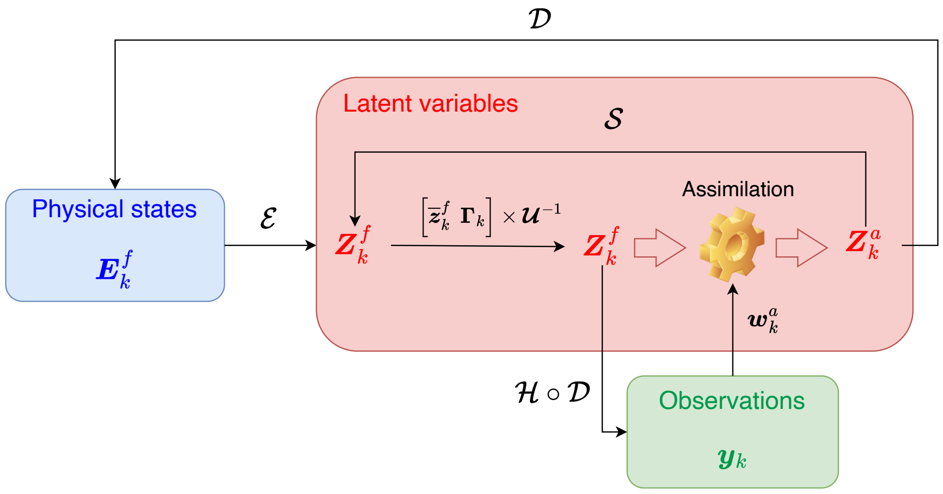


Figure 4.3.1: Outline of the latent DA framework, depicting the connections between the physical, latent and observational spaces.

The strong difference between algorithm 2.6 and algorithm 4.1, is the reduction of the computational space from $\mathbb{R}^n$ to $\mathbb{R}^\ell$, which straightforwardly reduces both the computational cost and the memory storage. In practice, our latent space is ten times smaller than our full space.

4.3.1 Extending ETKF-Q-L to larger ensemble sizes

When performing data assimilation the latent space of an autoencoder, as discussed in section 4.3, we have the flexibility to increase the number of ensemble members. This is feasible because the surrogate network facilitates low-cost time propagation (compared to the full space model), and the reduced dimensionality allows for computationally manageable linear algebra operations.

In practice, the dimension of the latent space can be small enough that the number of members m can equal or even exceed ℓ. However, the ETKF-Q algorithm (see Algorithm 2.6) from Fillion *et al.* (2020), and its latent variant ETKF-Q-L (see Algorithm 4.1), typically do not support more ensemble members than the dimension of the state space. This limitation stems from the model error correction step in these algorithms. Specifically, at line 3 of algorithms 2.6 and 4.1, a square root approximation of the updated error covariance matrix is computed. If $m \leq n+1$

(resp. $m \leq \ell + 1$), it is possible to compute the $(m-1)$ eigenpairs and define the square root approximation $\boldsymbol{V}_k \Lambda_k \in \mathbb{R}^{n\times(m-1)}$ (resp. $\in \mathbb{R}^{\ell\times(m-1)}$). However, when $m > n+1$ (resp. $m > \ell + 1$), at most n (resp. ℓ) eigenpairs can be computed, limiting the dimension of $\boldsymbol{V}_k \Lambda_k$ to $\mathbb{R}^{n\times n}$ (resp. $\mathbb{R}^{\ell\times\ell}$) and resulting in a dimension mismatch at the forecast ensemble update step at line 6.

We therefore modified the model error correction step of our algorithm, adopting a methodology that effectively handles cases where the number of members exceeds the state space dimension. Our research led us to consider the square root methods proposed by Raanes *et al.* (2015) that are part of the Python package DAPPER[(5)] (which stands for *Data Assimilation with Python: a Package for Experimental Research*). DAPPER is an open-source Python package designed for benchmarking the performance of data assimilation methods using synthetic twin experiments. It enables the testing of different data assimilation techniques by generating a synthetic truth from specified dynamic and observational models, and comparing how well various methods estimate this truth. DAPPER supports the development and assessment of data assimilation methods, providing valuable tools for research and education in the field.

In Raanes *et al.* (2015), the authors detail three variants of the square root method, namely **SQRT-CORE**, **SQRT-ADD-Z**, and **SQRT-DEP**, listed in increasing order of complexity and performance. We provide mathematical developments only for the **SQRT-CORE** methodology; the other two variants build upon the former.

The model is assumed linear and is therefore denoted by $\boldsymbol{M}$. We aim to remain consistent with the data assimilation notations introduced in chapter 2 as much as possible, adopting notations from Raanes *et al.* (2015) only for variables not defined in chapter 2. Notably, temporal subscripts are dropped as the analysis does not depend on time.

Under the linearity condition, the propagation equation reads:

$$\boldsymbol{E}^f = \boldsymbol{M}\boldsymbol{E}^a + \boldsymbol{D}, \tag{4.24}$$

$$\boldsymbol{D} = \boldsymbol{Q}^{1/2}\boldsymbol{\Xi}, \tag{4.25}$$

(5) https://github.com/nansencenter/DAPPER

where $\boldsymbol{\Xi} = [\boldsymbol{\xi}_1, \boldsymbol{\xi}_2, \ldots, \boldsymbol{\xi}_m]$ and each $\boldsymbol{\xi}_i$ is independently drawn from the Gaussian distribution $\mathcal{N}(0, \boldsymbol{I}_n)$. Thus, the columns of $\boldsymbol{D}$ follow $\mathcal{N}(0, \boldsymbol{Q})$.

Common practice is to center $\boldsymbol{D}$ by subtracting the mean from every column to eliminate the first-order sampling error. Consequently, we consider $\boldsymbol{D}$ to be centered, ensuring that $\boldsymbol{D}\mathbf{1} = 0$ (where $\mathbf{1} = [1, 1, \ldots, 1]^T$).

Given equation (4.24), we aim to derive the expression of $\boldsymbol{P}^f = \boldsymbol{X}^f \left(\boldsymbol{X}^f\right)^T$, the ensemble estimator of the second order moment of ground truth $\boldsymbol{x}^t$. To save calculation time, we point out that variable $\boldsymbol{X}^f$ introduced in section 2.2.2.2 and defined as $\left(\boldsymbol{E}^f - \overline{\boldsymbol{x}}^f \mathbf{1}^T\right)/\sqrt{m-1}$, can also be computed by $\boldsymbol{E}^f \left(\boldsymbol{I}_n - \mathbf{1}\mathbf{1}^T/m\right)/\sqrt{m-1}$. Also, we draw the reader's attention to the fact that our variable $\boldsymbol{X}^f$ corresponds to $\frac{1}{\sqrt{m-1}}\boldsymbol{A}^f$ in Raanes *et al.* (2015)'s notations. For simplicity and consistency with Raanes *et al.* (2015), we adopt the $\boldsymbol{A}$ notations in the following (which include $\boldsymbol{A}, \boldsymbol{A}^a$ and $\boldsymbol{A}^f$).

Derivation of $\boldsymbol{P}^f$ is as follows:

$$\begin{aligned}
\boldsymbol{P}^f &= \frac{1}{m-1}\boldsymbol{A}^f\left(\boldsymbol{A}^f\right)^T \\
&= \frac{1}{m-1}\boldsymbol{E}^f\left(\boldsymbol{I}_m - \frac{\mathbf{1}\mathbf{1}^T}{m}\right)\left(\boldsymbol{I}_m - \frac{\mathbf{1}\mathbf{1}^T}{m}\right)^T\left(\boldsymbol{E}^f\right)^T \\
&= \frac{1}{m-1}(\boldsymbol{M}\boldsymbol{E}^a + \boldsymbol{D})\left(\boldsymbol{I}_m - \frac{\mathbf{1}\mathbf{1}^T}{m}\right)\left(\boldsymbol{I}_m - \frac{\mathbf{1}\mathbf{1}^T}{m}\right)^T(\boldsymbol{M}\boldsymbol{E}^a + \boldsymbol{D})^T \\
&= \boldsymbol{M}\underbrace{\frac{1}{m-1}\boldsymbol{E}^a\left(\boldsymbol{I}_m - \frac{\mathbf{1}\mathbf{1}^T}{m}\right)\left(\boldsymbol{I}_m - \frac{\mathbf{1}\mathbf{1}^T}{m}\right)^T(\boldsymbol{E}^a)^T}_{=\boldsymbol{P}^a}\boldsymbol{M}^T + \\
&\quad \frac{1}{m-1}\boldsymbol{M}\boldsymbol{E}^a\underbrace{\left(\boldsymbol{I}_m - \frac{\mathbf{1}\mathbf{1}^T}{m}\right)\left(\boldsymbol{I}_m - \frac{\mathbf{1}\mathbf{1}^T}{m}\right)^T}_{=\left(\boldsymbol{I}_m - \frac{\mathbf{1}\mathbf{1}^T}{m}\right)}\boldsymbol{D}^T + \\
&\quad \frac{1}{m-1}\boldsymbol{D}\underbrace{\left(\boldsymbol{I}_m - \frac{\mathbf{1}\mathbf{1}^T}{m}\right)\left(\boldsymbol{I}_m - \frac{\mathbf{1}\mathbf{1}^T}{m}\right)^T}_{=\left(\boldsymbol{I}_m - \frac{\mathbf{1}\mathbf{1}^T}{m}\right)}(\boldsymbol{E}^a)^T\boldsymbol{M}^T + \\
&\quad \frac{1}{m-1}\boldsymbol{D}\underbrace{\left(\boldsymbol{I}_m - \frac{\mathbf{1}\mathbf{1}^T}{m}\right)\left(\boldsymbol{I}_m - \frac{\mathbf{1}\mathbf{1}^T}{m}\right)^T}_{=\left(\boldsymbol{I}_m - \frac{\mathbf{1}\mathbf{1}^T}{m}\right)}\boldsymbol{D}^T \\
&= \boldsymbol{M}\boldsymbol{P}^a\boldsymbol{M}^T + \frac{1}{m-1}\boldsymbol{M}\boldsymbol{A}^a\boldsymbol{D}^T - \frac{1}{m-1}\boldsymbol{M}\boldsymbol{E}^a\frac{\mathbf{1}\overbrace{\mathbf{1}^T\boldsymbol{D}^T}^{=0_{\mathbb{R}^n}}}{m} + \\
&\quad \frac{1}{m-1}\boldsymbol{D}(\boldsymbol{M}\boldsymbol{A}^a)^T - \frac{1}{m-1}\frac{\overbrace{\boldsymbol{D}\mathbf{1}}^{=0_{\mathbb{R}^n}}\mathbf{1}^T}{m} + \frac{1}{m-1}\boldsymbol{D}\boldsymbol{D}^T \\
&\boxed{= \boldsymbol{M}\boldsymbol{P}^a\boldsymbol{M}^T + \boldsymbol{Q} + (\overline{\boldsymbol{Q}} - \boldsymbol{Q}) + \frac{1}{m-1}\left(\boldsymbol{M}\boldsymbol{A}^a\boldsymbol{D}^T + \boldsymbol{D}(\boldsymbol{M}\boldsymbol{A}^a)^T\right),} \qquad (4.26)
\end{aligned}$$

where $\overline{\boldsymbol{Q}} - \frac{1}{m-1}\boldsymbol{D}\boldsymbol{D}^T$. Besides, we mention here that the derivation of equation (4.26) shows that (Raanes *et al.*, 2015, equation 26) holds a typo: there is no minus sign in front the fourth term of the sum.

Ideally, and as pointed out by (Raanes *et al.*, 2015, section 3.1), $\boldsymbol{P}^f$ should rather satisfy the following equation (which is the one verified by the estimators, see equation (2.66)):

$$\boxed{\boldsymbol{P}^f = \boldsymbol{M}\boldsymbol{P}^a\boldsymbol{M}^T + \boldsymbol{Q}.} \tag{4.27}$$

We illustrate the problem of the classic by assuming the model to be linear, but we aim to satisfy equation (4.27) whether the model is linear or not. In the following, we therefore consider the general case where the model is $\boldsymbol{\mathcal{M}}$.

In accordance with the notations of Raanes *et al.* (2015), we denote by $\boldsymbol{A}$ (no superscript) the anomalies related to the propagated ensemble **before** the model error correction step. Then, variable $\boldsymbol{A}^f$ represents the propagated ensemble **after** noise inclusion[6].

Given that $\boldsymbol{P}^a = \frac{1}{m-1}\boldsymbol{A}^a(\boldsymbol{A}^a)^T$ and $\boldsymbol{P}^f = \frac{1}{m-1}\boldsymbol{A}^f\left(\boldsymbol{A}^f\right)^T$, equation (4.27) is satisfied if $\boldsymbol{A}^f$ meets the condition:

$$\boxed{\boldsymbol{A}^f\boldsymbol{A}^f = \boldsymbol{A}\boldsymbol{A} + (m-1)\boldsymbol{Q}} \tag{4.28}$$

However, finding a matrix $\boldsymbol{A}$ that verifies equation (4.28) is an ill-defined problem, since the rank of the left-hand side is at most m, while the right-hand side has a rank of n (assuming $\boldsymbol{Q}$ is full-rank).

Raanes *et al.* (2015) thus defines the orthogonal projector onto the column space of $\boldsymbol{A}$, denoted by $\boldsymbol{\Pi}_{\boldsymbol{A}} = \boldsymbol{A}\boldsymbol{A}^+$, where $\boldsymbol{A}^+$ is the Moore-Penrose pseudoinverse of $\boldsymbol{A}$ (Moore, 1920; Penrose, 1955). The authors then introduce $\widehat{\boldsymbol{Q}} = \boldsymbol{\Pi}_{\boldsymbol{A}}\boldsymbol{Q}\boldsymbol{\Pi}_{\boldsymbol{A}}$, a symmetric orthogonal projection of $\boldsymbol{Q}$ onto the column space of $\boldsymbol{A}$. The SQRT-CORE algorithm aims to satisfy the following modified equation instead of equation (4.28):

$$\boxed{\boldsymbol{A}^f\boldsymbol{A}^f = \boldsymbol{A}\boldsymbol{A} + (m-1)\widehat{\boldsymbol{Q}}} \tag{4.29}$$

Equation (4.29) can be reformulated as:

$$\boxed{\boldsymbol{A}^f\boldsymbol{A}^f = \boldsymbol{A}\boldsymbol{G}^f\boldsymbol{A}^T,} \tag{4.30}$$

where $\boldsymbol{G}^f = \boldsymbol{I}_m + (m-1)\boldsymbol{A}^+\boldsymbol{Q}(\boldsymbol{Q})^T$.

As a result, in the SQRT-CORE algorithm, ones computes $\boldsymbol{T}^f$, a square root matrix of $\boldsymbol{G}^f$ so that $\boldsymbol{A} = \boldsymbol{A}^f\boldsymbol{T}^f$ fulfills equation (4.29). Regarding the choice

[6]in algorithm 2.6 (resp. algorithm 4.1), we have made the choice of simply overwriting variables $\boldsymbol{\Delta}_k$ (resp $\boldsymbol{\Gamma}_k$).

of $\boldsymbol{T}^f$, the authors recommend using a symmetric matrix for $\boldsymbol{T}^f$, as it has shown advantageous properties (see (Raanes *et al.*, 2015, sections 2.3 and 4)), such as preserving the ensemble mean (Wang *et al.*, 2004; Evensen, 2009b), confining the affine subspace (Evensen, 2003), satisfying equality constraints, or minimizing ensemble displacement (Ott *et al.*, 2004; Hunt *et al.*, 2007).

The difference between equation (4.28) and equation (4.29) lies in the term $(m-1)[\boldsymbol{Q} - \widehat{\boldsymbol{Q}}]$, called "residual noise," which the SQRT-CORE method does not account for. To address this, the authors propose the **SQRT-ADD-Z** algorithm, followed by **SQRT-DEP** to reintroduce statistical dependence.

Conclusion

Throughout this chapter, we have presented and discussed the most important aspects of our latent data assimilation approach. Specifically, we have reviewed the neural network architectures we employ, along with the adaption of the regular ETKF-Q algorithm to operate within a latent space. Beyond these critical elements, chaos theory and PCA are fundamental in underpinning our methodology and providing valuable insights into latent space data assimilation.

Dynamical systems theory emphasizes the importance of ensuring that the latent dimension is at least equal to the dimension of the unstable-neutral subspace. Ideally, an autoencoder should also capture some stable directions, as these are crucial for the long-term dynamics and overall behavior of the system. Accurately representing the unstable-neutral directions helps to correct the forecast state by mitigating errors in these subspaces. Furthermore, since autoencoders are trained to build a manifold that the state trajectory statistically adheres to, data assimilation corrections will likely target the most error-sensitive directions. Like PCA, autoencoders efficiently disentangle complex, nonlinearly related variables into more meaningful and simpler components. While full space data assimilation involves linear combinations within a high-dimensional, nonlinear, and possibly multi-scale dynamics, latent data assimilation, which operates on the core, potentially disentangled and simplified, dynamics, is more likely to result in impactful corrections. Data assimilation faces an inherent and insurmountable mathematical limitation in that it relies on linear computations. Latent data assimilation, however, offers a way to overcome this limitation by performing the assimilation directly within meaningful underlying structures of the data, as illustrated in Figure 4.3.2, and later demonstrated by numerical experiments in chapter 5.

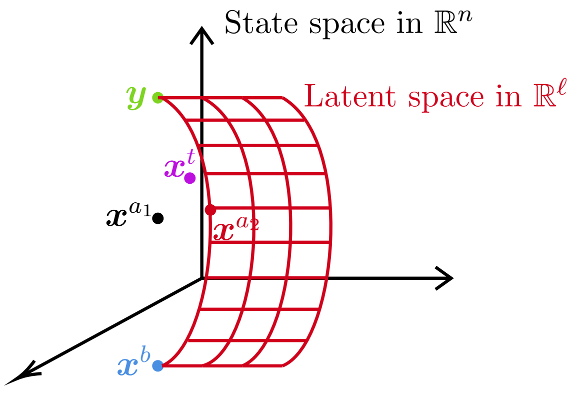


Figure 4.3.2: Comparison between full space DA and latent space DA (temporal subscript is dropped here). Variables $\boldsymbol{x}^b$, $\boldsymbol{x}^{a_1}$, $\boldsymbol{x}^{a_2}$ and $\boldsymbol{x}^t$ denote the background knowledge, the full space estimate, the latent estimate and the ground truth state, respectively. We also assume $\boldsymbol{\mathcal{H}}_k = \boldsymbol{I}_n$.

CHAPTER 5

Numerical experiments

In chapter 4, we presented the latent data assimilation approach and provided theoretical motivations for its computational efficiency and accuracy. In the following sections, we demonstrate that this novel methodology meets our expectations using two different instructive numerical models: in section 5.1, we present results obtained for a tailored version of the Lorenz96 system, and in section 5.2, we present those for the quasi-geostrophic model of the Object-Oriented Prediction System (OOPS), a weather forecast framework developed collaboratively by ECMWF and Météo-France[1].

In both section 5.1 and section 5.2, we begin by introducing and describing the numerical system under consideration. We then determine a suitable latent space dimension for the subsequent numerical experiments, based on PCA and the number of positive Lyapunov exponents. Following this, we provide detailed information about the neural network architectures, the dataset structure, the training settings, and the performance of the trained network (comprising the autoencoder and the surrogate). We then offer a comprehensive analysis of the surrogate's stability according to various criteria. Lastly, we define data assimilation benchmark experiments and draw conclusions for both latent space and physical space assimilation. Additionally, we present results when model error correction is performed using the SQRT-DEP method by Raanes *et al.* (2015), implemented in the DAPPER package.

[1]https://www.ecmwf.int/en/elibrary/77561-oops-common-framework-research-and-operations

5.1 An augmented version of the Lorenz96 system

To demonstrate the efficacy of our latent data assimilation approach, we define a dynamical system that is, by design, exactly representable within a lower-dimensional space. This is achieved by applying a nonlinear embedding to a chaotic 40-variable Lorenz96 system, mapping it into a 400-dimensional space.

Edward Lorenz (1917–2008), a mathematician and meteorologist, is renowned for his pioneering contributions to chaos theory. In 1963, he introduced what is now known as the Lorenz63 dynamics, a set of three coupled ordinary differential equations (ODEs):

$$\boxed{\begin{aligned}\frac{\mathrm{d}x}{\mathrm{d}t} &= \sigma(y-x), && (5.1a)\\ \frac{\mathrm{d}y}{\mathrm{d}t} &= x(\rho-z)-y, && (5.1b)\\ \frac{\mathrm{d}z}{\mathrm{d}t} &= xy-\beta z, && (5.1c)\end{aligned}}$$

where σ, β and ρ are system parameters typically assumed to be positive.

For certain parameter values, the system exhibits chaotic behavior: *e.g.*, Lorenz considered $\sigma = 10$, $\beta = \frac{8}{3}$ and $\rho = 28$.

In 1996, Lorenz proposed the following dynamical system of arbitrary dimension $N \geq 4$:

$$\boxed{\frac{\mathrm{d}[\boldsymbol{x}]_i}{\mathrm{d}t} = \big([\boldsymbol{x}]_{i+1} - [\boldsymbol{x}]_{i-2}\big)[\boldsymbol{x}]_{i-1} - [\boldsymbol{x}]_i + F,\ \forall i = 1, \dots, N} \tag{5.2}$$

with periodic boundary conditions $[\boldsymbol{x}]_{-1} = [\boldsymbol{x}]_{N-1}$, $[\boldsymbol{x}]_0 = [\boldsymbol{x}]_N$ and $[\boldsymbol{x}]_{N+1} = [\boldsymbol{x}]_1$. F is a forcing term, often set to 8 to induce chaos.

In equation (5.2), quadratic terms represent the advection that conserves the total energy, the linear term is the damping through which the energy decreases, and the constant term denotes the external forcing keeping the total energy away from zero. The N variables may be thought of as values of some atmospheric quantity in the N sectors of a latitude circle.

Both Lorenz63 and Lorenz96 systems serve as exemplary and educational models, often used within the scientific community to test new methods or algorithms. For instance, within the scope of our research, multiple papers cited in chapter 3 employ these models to validate their approaches, with Lorenz96 being more frequently used than Lorenz63: Gottwald and Reich (2021b,a); Brajard *et al.* (2020); Farchi *et al.* (2021b); Penny *et al.* (2022); Wikner *et al.* (2021); Frerix *et al.* (2021); Fablet *et al.* (2023, 2021); Boudier *et al.* (2023).

For the purpose of our research, we developed what we term the **augmented Lorenz96 system** based on the standard chaotic 40-dimensional model, ensuring by design the existence of a latent space where the dynamics can be exactly expressed. Our augmented Lorenz dynamics has therefore an inner latent dimensionality of 40.

Specifically, we consider a dataset of 40-dimensional Lorenz96 simulations generated with a forcing term F set to 8 to induce chaos, integrated using a fourth-order Runge-Kutta scheme. We define an orthonormal matrix $\boldsymbol{O} \in \mathbb{R}^{\ell \times n}$ that maps from $\mathbb{R}^{40}$ to $\mathbb{R}^{400}$, artificially expanding the number of variables from 40 to 400. Subsequently, we apply an element-wise nonlinear function, specifically an invertible 3rd-degree polynomial, to make the difficulty of discovering a 40-dimension latent space by an autoencoder more challenging. Python implementations of these two successive functions are provided in appendix C.1. We denote by $\mathcal{F}^{-1}$, the function that maps from the Lorenz96 system to the augmented system, and by $\mathcal{F}$ the reverse mapping.

Figure 5.1.1 shows a 40-dimensional Lorenz96 dynamics from our dataset, alongside its augmented counterpart. The upper plots depict the entire state vector across time, while the bottom graphs offer a detailed look at six selected variables from both systems, providing insight into the differences in dynamics and scale between the standard and augmented models.

5.1.1 Latent space dimension

From a theoretical perspective, it is quite straightforward and natural to look for a latent space of dimension 40, since the augmented dataset is built upon a 40-variable chaotic Lorenz96 system. Furthermore, the mapping between the original dynamics and the augmented one is bijective, meaning that all the information held in $\mathbb{R}^{400}$ can be fully represented in $\mathbb{R}^{40}$. However, in real-life problems, it is

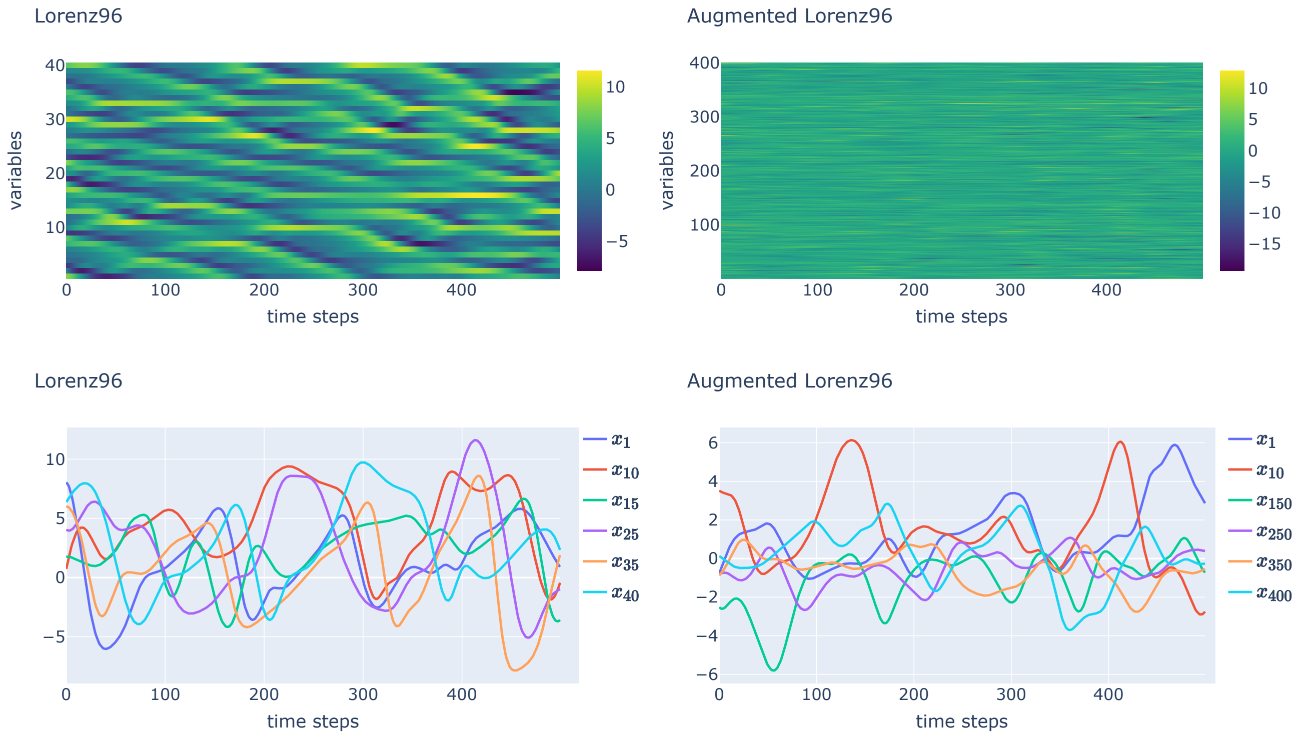


Figure 5.1.1: Comparative visualization of a 40-dimensional Lorenz96 dynamics alongside its augmented counterpart. In the top pair of images, the full state vector is represented over time, with the left depicting the original Lorenz96 and the right showing the augmented system. The bottom images offer a detailed look at six selected variables from both systems.

pretty unlikely that one would know in advance the appropriate latent dimension for properly representing the dynamics of interest.

Therefore, to validate this intuitive and natural choice, we demonstrate that $\ell = 40$ is consistent with PCA and autoencoder reconstruction analyses, as well as with the number of positive Lyapunov exponents (see section 4.2.1 for theoretical details).

First, PCA offers insightful information about the compression ratio achievable by a linear transformation of the data. We perform PCA over latent dimensions ranging from $\ell = 5$ to $\ell = 100$ with a step of 5. As we eventually aim to work with neural networks solely, we also train autoencoders section 3.1.5 on the augmented data using the same latent dimensions as those employed for PCA. The encoder layers sequentially reduce the dimension from $\mathbb{R}^{400}$ to $\mathbb{R}^{\ell}$ through intermediary sizes of 300, 200, and 150, with 0.2 slope *Leaky ReLU* activations, except for the last layer, whose activation function is *tanh*. The decoder performs the reverse operation, with the activation function of the last layer being the identity. This comparison between PCA and autoencoders allows us to evaluate their representation performances.

Figure 5.1.2 reports the results obtained for the two methods in terms of MSE scores on the training and test datasets. For both PCA and autoencoders, MSE values decrease sharply up to $\ell = 40$. Beyond this point, PCA scores continue to decrease but at a much slower rate, while those of autoencoders stagnate, indicating that more epochs would be required during training to achieve lower reconstruction errors. These results suggest that setting $\ell = 40$ is the most relevant choice in the context of our research: it indeed ensures the recovery of most of the original information, while reducing the physical space dimension by a factor of 10. Additionally, any latent dimension value larger than 40 does not offer significant improvement, especially for autoencoders, which far outperform PCA with much better reconstruction performances.

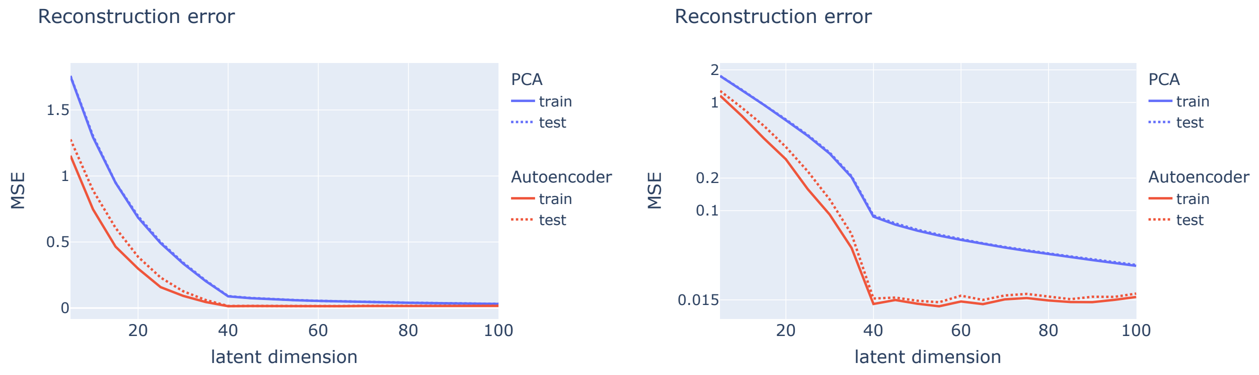


Figure 5.1.2: Left: MSE reconstruction curves over train (solid curve) and test (dashed curve) datasets for PCA (blue curve) and autoencoders (orange curve). Right: the same plot, but with a log-scale on the y-axis.

Regarding the number of unstable and neutral directions of the augmented data, they correspond to those of the underlying Lorenz96 dynamics, only the values of the Lyapunov exponents and the Kaplan-Yorke dimension differ. Therefore, by setting $\ell = 40$, we ensure that the autoencoder can represent the unstable, neutral and stable subspaces within the latent space. However, rather than directly moving on to the neural network architecture, we think it is insightful and relevant to provide some information about the chaotic behavior of the Lorenz96 system.

To compute the Lyapunov exponents, we use a dedicated module of the DAPPER package[(2)]. The method involves considering an initial state $\boldsymbol{x}_0$ and adding m orthonormal perturbations to it, thus defining an ensemble of m deviations (also known as members). For a system of n variables, this approach can compute at most m Lyapunov exponents, so that computing all of them requires setting $m = n$. The initial state $\boldsymbol{x}_0$ is propagated through time to define the ground truth trajectory. By also propagating the ensemble through time, we can compute the temporal evolution of the perturbations. At every time step, the set of perturbation vectors is reorthonormalized using the QR decomposition (Golub and Van Loan, 2013). This step is crucial to prevent numerical errors and ensure accurate computation of the Lyapunov exponents. The latter are tracked by storing the logarithm of the absolute values of the diagonal elements of the upper triangular matrix from the QR decomposition. To allow the method to converge, a sufficient number of time steps has to be defined (here 40000).

In the left plot of Figure 5.1.3, we represent the number of positive Lyapunov exponents along with the Kaplan-Yorke dimension (see section 4.2.1) of Lorenz96 systems whose dimensions vary from 5 up to 40. For each run, the number of members is set to the dimension of the system, so that all the Lyapunov exponents can be computed. We observe that the dimension of the attractor is always greater than the number of chaotic directions. In the right plot of Figure 5.1.3, we show the number of positive Lyapunov exponents with respect to the number of members in the perturbed ensemble. We have that for a 40-dimensional Lorenz96 system, the dimension of the unstable subspace is 13 and the Kaplan-Yorke dimension is equal to 27.

(2) `https://github.com/nansencenter/DAPPER/blob/cb357afed32cf81aad008d792fd55bf9c2e6a5c3/dapper/mods/explore_props.py`

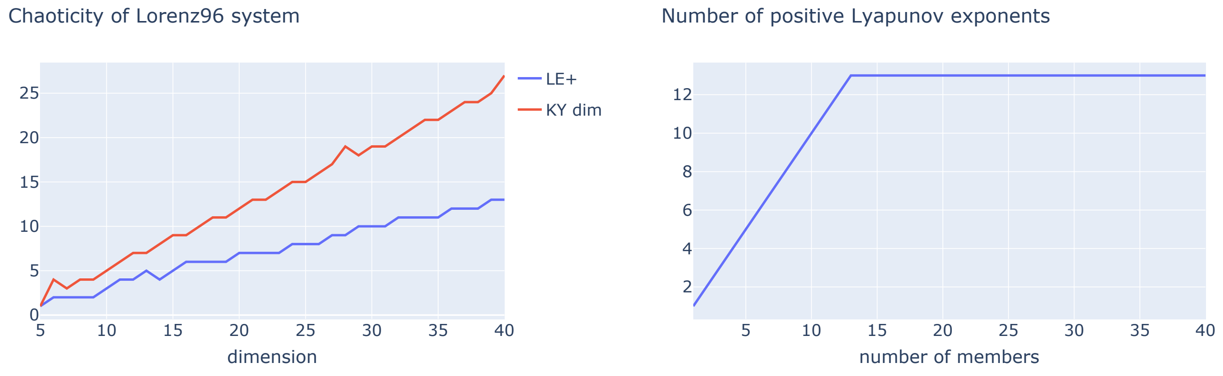


Figure 5.1.3: Left: number of positive Lyapunov exponents (blue curve) and Kaplan-Yorke dimension (orange curve) with respect to the dimension of the Lorenz96 system. Right: for a 40-dimensional Lorenz96 system, the number of positive Lyapunov exponents as a function of the number of members.

In Figure 5.1.4, we plot the values of the 40 Lyapunov exponents, in decreasing order. Below, we also provide the first 16 Lyapunov exponents (with a 2-digit precision): 1.68, 1.48, 1.31, 1.17, 1.01, 0.875, 0.743, 0.625, 0.488, 0.376, 0.262, 0.143, 0.0326, −0.0002, −0.0848. The dimension of the neutral subspace is therefore 0, as there are no Lyapunov exponents equal or close enough to 0.

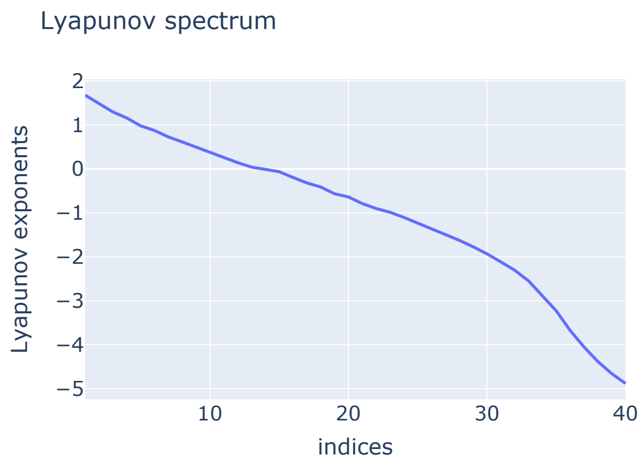


Figure 5.1.4: Lyapunov spectrum of the 40-dimensional Lorenz96 system.

The signs of the Lyapunov exponents remain the same from the Lorenz96 system to the augmented one; however, their values do change. In Figure 5.1.5, we plot the first 40 Lyapunov exponents in decreasing order. We also provide the values of the first 16 Lyapunov exponents with a 2-digit precision (except for the fourteenth exponent, for which we give a 3-digit round-off): 0.65, 0.56, 0.50, 0.45, 0.40, 0.34, 0.29, 0.25, 0.20, 0.14, 0.09, 0.06, 0.01, -0.003, -0.04, -0.09. As mentioned earlier, for the augmented dynamics, the dimension of the unstable subspace is 13, the one of the neutral space is 0, and the stable space comprises the remaining 387 dimensions. The Kaplan-Yorke dimension differs between Lorenz96 and augmented Lorenz, being 27 in the former case and 25.3 in the latter.

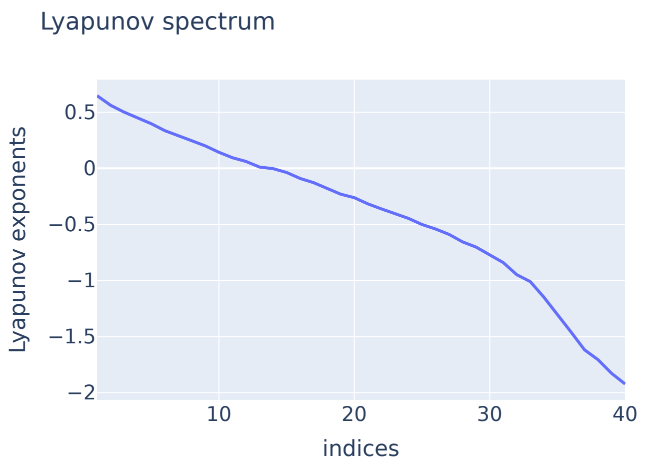


Figure 5.1.5: Lyapunov spectrum of the 400-dimensional augmented Lorenz96 system.

In summary, the PCA and autoencoder analyses, along with the Lyapunov spectrum, confirm that setting $\ell = 40$ offers an excellent latent representation of the physical dynamics. This choice also allows to capture the unstable and the attractor subspaces, while reducing the dimensionality by a factor of 10.

5.1.2 Neural networks architectures and training settings

In the following, we present the exact neural architecture we train and subsequently use for latent space data assimilation. We remind that the chained loss function we use for the trainings is defined in equation (4.21). Before detailing the specific features we set, it is important to note that we tuned numerous hyperparameters in order to find the architecture and the training settings that work best: the number of hidden layers, the choice of activation functions, whether to normalize the data, the batch size, the noise magnitude, the learning rate, the loss weighting parameter ρ, and the number of forward steps C.

The encoder is made of four fully connected layers, each one followed by a 0.2 slope *Leaky ReLU* activation, except for the last layer, whose activation function is *tanh*. These layers sequentially reduce the dimension from $\mathbb{R}^{400}$ to $\mathbb{R}^{40}$ through intermediary sizes of 300, 200, and 150. The decoder performs the reverse operation, with the activation function of the last layer being the identity. Several variants have been tested, and this configuration has proven to yield the best results. Figure 5.1.6 helps visualize the autoencoder architecture.

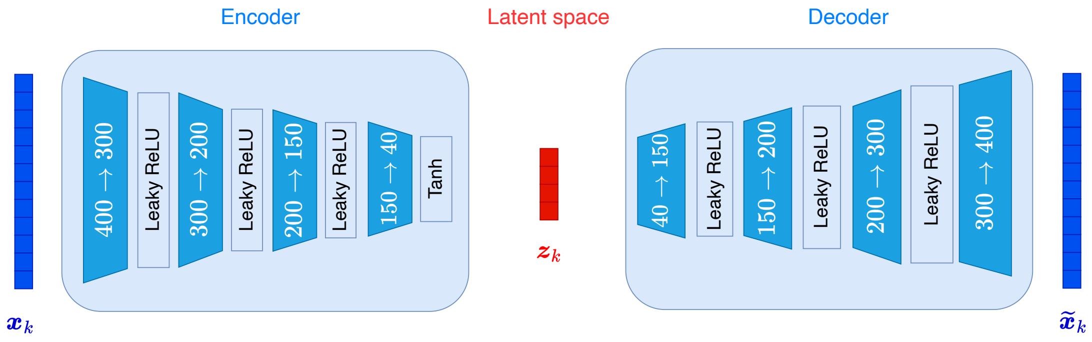


Figure 5.1.6: Autoencoder architecture used for numerical experiments: $\boldsymbol{x}_k \in \mathbb{R}^{400}$ refers to the input vector, $\boldsymbol{z}_k \in \mathbb{R}^{40}$ to the latent variable, and $\widetilde{\boldsymbol{x}}_k \in \mathbb{R}^{400}$ to the reconstructed vector. Trapeziums denote dense layers, while rectangles represent activation functions.

The surrogate network consists of six fully connected layers, maintaining a consistent dimension of $\mathbb{R}^{40}$, with each layer followed by a *Leaky ReLU* activation with a 0.2 slope, except the last. The structure and residual blocks (Bachlechner *et al.*, 2021) of this network are detailed in Figure 5.1.7.

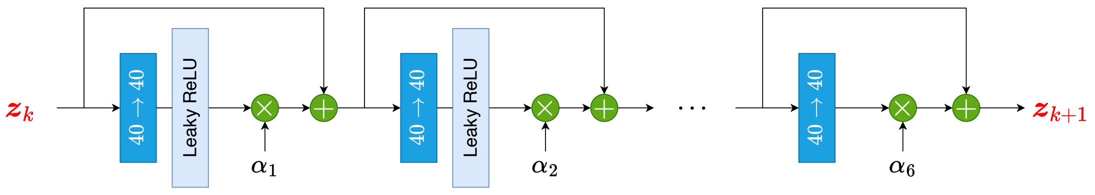


Figure 5.1.7: Our surrogate network is composed of six residual blocks mapping from $\mathbb{R}^{40}$ to $\mathbb{R}^{40}$. Latent variable z_k is the input, and z_{k+1} refers to the latent vector at time t_{k+1} yielded by the network. Also, we have that $\alpha_i \in \mathbb{R}^{40}$, $\forall i \in [\![1, 6]\!]$. Multiplicative and additive operators are also represented.

Our dataset comprises 1000 Lorenz96 simulations, each with 500 time steps across 40 dimensions, producing 1000 instances of 500×40 matrices. Each simulation represents the temporal evolution of the 40 Lorenz96 variables under a specific initial condition, formatted as a 500×40 matrix.

We then transform the 1000 Lorenz96 simulations into their augmented versions, generating 1000 instances of 400-dimensional data. The dataset is normalized (mean and standard deviation normalization), and split into 80% for training, with the remaining 20% equally divided between validation and testing. A white Gaussian noise with a magnitude of 0.01 is added to the input data during the training stage. The batch size is set to 32, and we chose the Adam optimizer with a learning rate of 10^{-3}. The number of epochs is fixed at 200, and early stopping regularization (see section 3.1.6) with a patience of 15 epochs is implemented to save computational time. Network weights are saved each time a lower loss score on the validation set is reached. The weighting parameter ρ of our custom loss function defined in equation (4.21) is set to 5, and the number of iterations C is set to 2.

Our network, comprising the autoencoder and the surrogate, has $443,580$ trainable parameters. The training takes 2 hours and 52 minutes over 109 epochs on a 2021 MacBook Pro equipped with the Apple M1 Pro chip. Figure 5.1.8 shows the training and validation loss curves, along with the test loss value. The test MSE is computed at the end of the training after loading the best model - *i.e.*, after loading the weights for which the network reaches the lowest MSE on the validation dataset - and is therefore not related to the training phase (nor to the epochs, contrary to what Figures 5.1.8 and 5.1.9 might suggest at first glance). However, to make this test score visible in these two figures, we represent it similarly to the training and validation curves along the x-axis.

The loss values in Figure 5.1.8 are computed according to equation (4.21). The three curves validate the effective learning of the network: indeed, the smooth exponential decrease of the loss function indicates that it performs its assigned task increasingly better over the course of training.

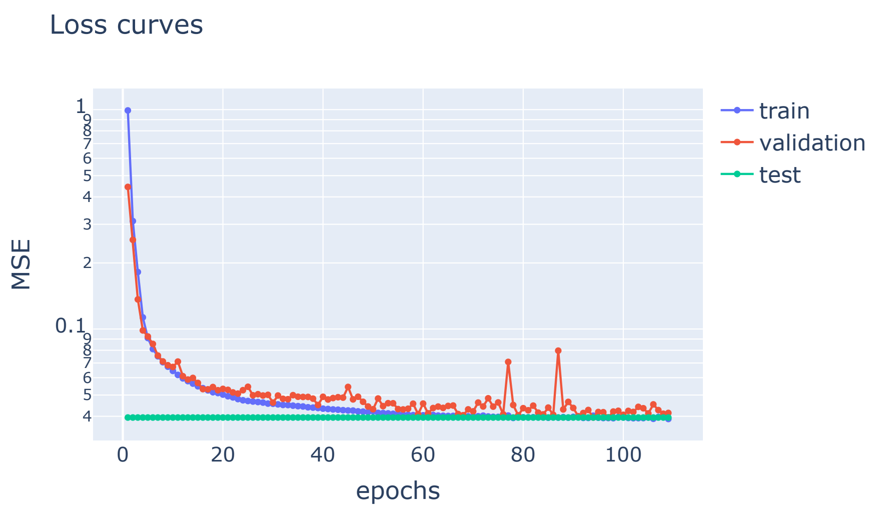


Figure 5.1.8: Loss values (see equation (4.21)). We remind that the score on the test dataset is repeated along the x-axis with a dotted line for visual convenience.

Figure 5.1.9 reports the reconstruction loss as expressed in equation (4.22) (left plot), and the chained loss given by equation (4.23) (right plot).

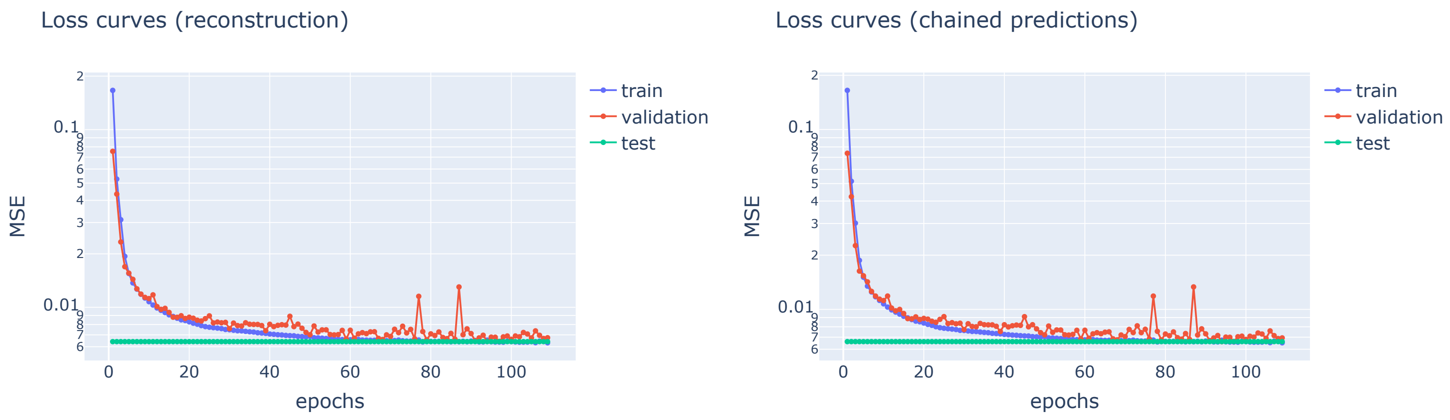


Figure 5.1.9: Left: reconstruction loss (see equation (4.22)) with a log-scale on the y-axis. Right: chained loss (see equation (4.23) with a log-scale on the y-axis. We remind that the score on the test dataset is repeated along the x-axis with a dotted line for visual convenience.

The left and right plots confirm that both the AE and the surrogate effectively learn; that is, neither of them is left behind during the training stage. One can also notice that $\mathcal{L}_{AE}$ and $\mathcal{L}_{Sur}$ have almost the same values throughout the learning process: this might suggest setting ρ to 1 rather than 5 in order to define a fair loss function. However, it turns out that increasing the weighting on $\mathcal{L}_{Sur}$ yields better scores, meaning that more effort is needed for the surrogate to properly learn than for the AE. Along with the stability issues presented in section 4.2.3.1, this confirms that the surrogate network is the more challenging to train. The best model is saved at epoch 94, and achieves the following loss scores:

- validation loss score: 0.0395
- test loss score: 0.0395
- validation reconstruction loss score: 0.0064
- test reconstruction loss score: 0.0064
- validation chained loss score: 0.0066
- test chained loss score: 0.0066

We draw the reader's attention to the fact that the scores are actually different between the validation and test sets, but rounding may make them appear strictly equal here.

In the following, we provide some insights into the performances of our trained neural network. The loss scores on the training, validation and test sets already suggest that the network is effective in representing the input data into a 40-dimensional latent space as well as propagating within it. However, we are interested in going one step further by seeing how this performance actually translates into our augmented dynamics in terms of reconstructed simulations and variables.

We first consider an entire simulation from the test dataset. We encode and decode all the 500 states of this simulation and compute the differences in absolute value between the reconstruction and the original data. The results are shown in Figure 5.1.10: we observe that errors range from approximately 0 to about 2.5 and that most states appear to be correctly reconstructed. We notice a few outliers, such as the ones located at variable 263 from approximately the 40th time step to the 80th one.

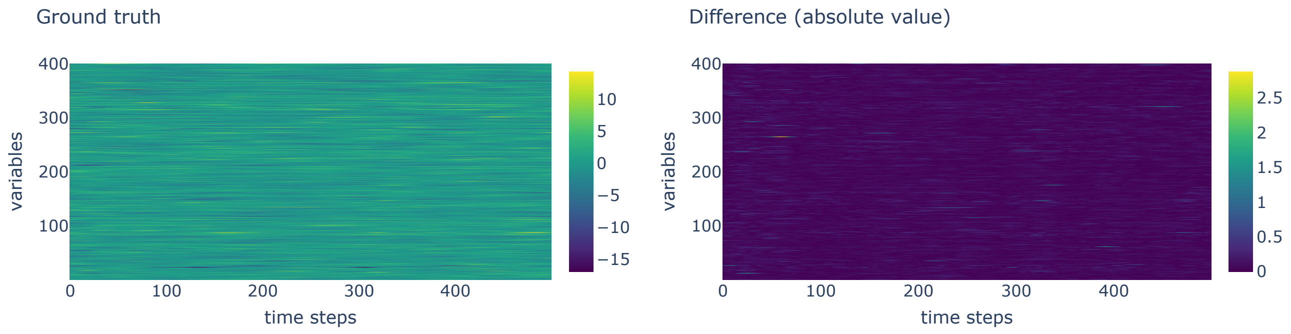


Figure 5.1.10: Left: ground truth simulation from the test dataset. Right: differences in absolute value between the ground truth and the reconstructed simulations (using autoencoder only).

We zoom in and plot the temporal evolution of six variables out of 400 to locally compare the discrepancies between the ground truth and the reconstruction. In addition to the autoencoder reconstruction, we also include another form of reconstruction, called one latent step propagation reconstruction. The idea here is to encode the initial state $\boldsymbol{x}_0$ into $\boldsymbol{z}_0$, propagate it once with the surrogate to obtain $\boldsymbol{z}_1$ and then decode it to get $\widetilde{\boldsymbol{x}}_{\mathbf{1}}$. We repeat the process of encoding, propagating once and decoding until the end of the simulation, resulting in the following approximated trajectories $[\widetilde{\boldsymbol{x}}_{\boldsymbol{1}}, \widetilde{\boldsymbol{x}}_{\boldsymbol{2}}, \ldots, \widetilde{\boldsymbol{x}}_{\boldsymbol{499}}]$. To produce a full simulation of 500 temporal states, we also encode and decode $\boldsymbol{x}_0$, so that we actually get $[\widetilde{\boldsymbol{x}}_{\boldsymbol{0}}, \widetilde{\boldsymbol{x}}_{\boldsymbol{1}}, \widetilde{\boldsymbol{x}}_{\boldsymbol{2}}, \ldots, \widetilde{\boldsymbol{x}}_{\boldsymbol{499}}]$. We therefore consider the same simulation as the one depicted in Figure 5.1.10, and perform both the regular reconstruction and the one latent step propagation reconstruction. The results are shown in Figure 5.1.11. Visually, the two plots appear to be exactly the same, indicating that the one latent step propagation does not introduce any noticeable error: in both plots, the ground truth and reconstruction curves almost perfectly superimpose.

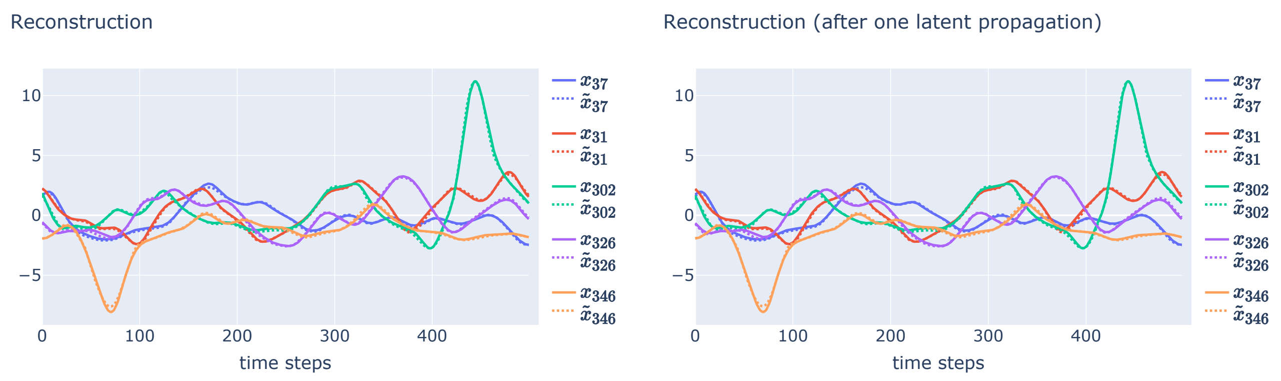


Figure 5.1.11: Left: reconstruction of six out of 400 variables from a simulation in the test dataset (the same one as in Figure 5.1.10) using only the autoencoder. Right: the same reconstruction as in the left plot, but with one-step latent propagation also performed. Ground truths are represented solid lines, and reconstructions with dashed lines.

We are also interested in how the autoencoder represents the latent information in a 40-dimensional space. We know that the original Lorenz96 system evolves smoothly and continuously over time, as depicted in Figure 5.1.1. Figure 5.1.12 shows the latent representations of two simulations from the test datasets: we observe that the encoding stage preserves the smoothness and continuity of the variables across time. It is important to note that one cannot expect the autoencoder to produce a latent representation that is very similar to the top left plot of Figure 5.1.1. The adjacent variables of the Lorenz96 system are indeed tightly linked through the governing equations, and since they are plotted side by side in the same order as they appear in the equations, this visually results in continuous and smooth wavy patterns. However, the autoencoder arranges the latent variables randomly, making such patterns very unlikely to be produced.

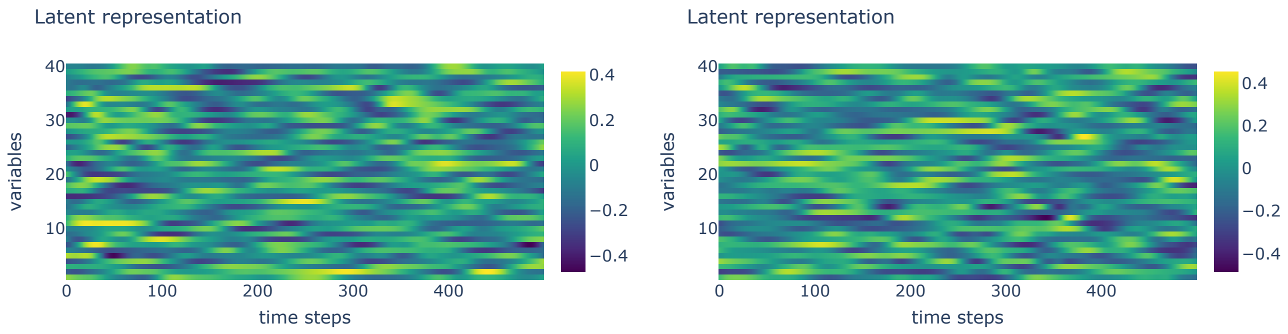


Figure 5.1.12: Left: latent space representation of an entire simulation from the test dataset. Right: another latent space representation from the test dataset, with the same network.

Subsequently in this chapter, we will present the reconstruction, stability, and numerical data assimilation results obtained with the trained 1-Lipschitz and ext-Lipschitz surrogate networks. We remind that we refer to an extended Lipschitz network (abbreviated as ext-Lipschitz) as a data-driven model in which all layers are 1-Lipschitz, except for the final layer, which remains unconstrained. Since the latent spaces produced by the combination of autoencoders with a Lipschitz surrogate show a different structure compared to those depicted in Figure 5.1.12, we believe it is relevant to offer the reader a visual comparison here, rather than waiting for these two Lipschitz networks to be introduced in the next subsection. Figure 5.1.13 shows the latent space representations of two simulations from the test datasets. Unlike Figure 5.1.12, where the representations are smooth and visually homogeneous, Figure 5.1.13 reveals that latent variable 38 has a nearly constant value over time. This behavior is not specific to a single simulation, as both plots in Figure 5.1.13 display the same constant band at variable 38.

A similar pattern is observed when the surrogate is not restricted to 1-Lipschitzness but is ext-Lipschitz, as shown in Figure 5.1.14. In this case, variable 13 appears constant over time, while variable 34 shows minor but still visible changes. An interesting observation from Figures 5.1.12 to 5.1.14 is that the autoencoder architecture is identical in all three cases; the only difference lies in the surrogate network. This indicates that the autoencoder's weights are optimized to produce a latent space adapted to the specific surrogate network it is paired with.

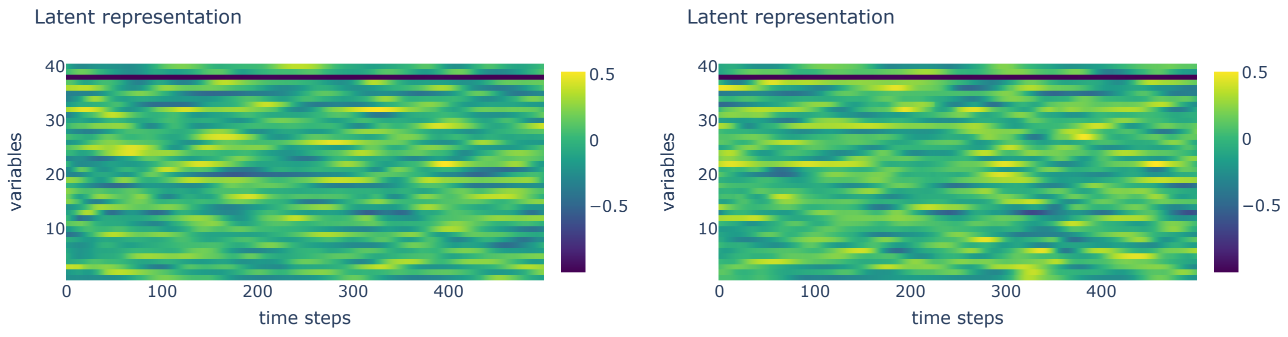


Figure 5.1.13: Left: latent space representation of an entire simulation from the test dataset obtained with our trained 1-Lipschitz network (later introduced in this section). Right: another latent space representation from the test dataset with the same network.

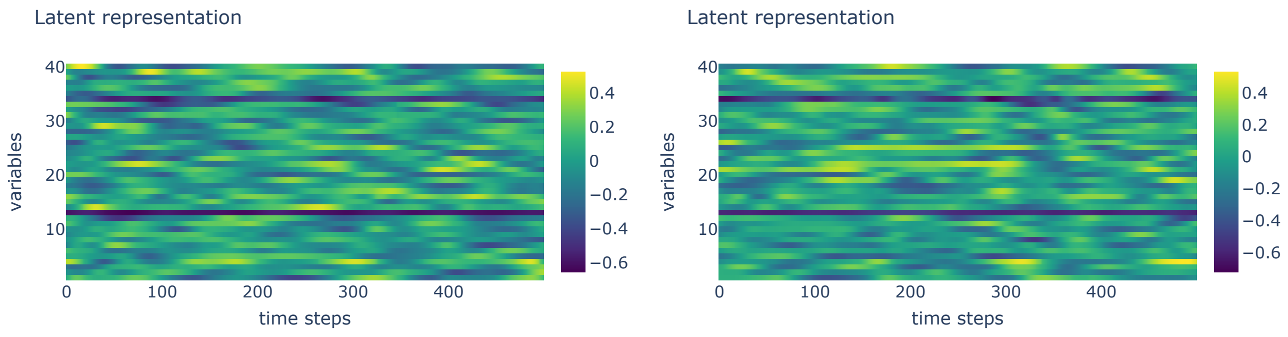


Figure 5.1.14: Left: latent space representation of an entire simulation from the test dataset obtained with our trained ext-Lipschitz network (later introduced in this section). Right: another latent space representation from the test dataset with the same network.

5.1.3 Enforcing stability of the surrogate network

As mentioned in section 4.2.3.1, we initially aim to learn the latent temporal propagation process using a regular, non-chained loss function. Therefore, we first trained the neural network presented in section 5.1.2 with the loss function parameter C (see equation (4.18)) set to 1, meaning the surrogate is applied only once to the encoded data. We refer to this network as the "one-step neural network", as opposed to the regular model (*i.e.*, the one trained with $C = 2$) which is now referred to as the "two-step neural network". We first present the numerical results for the two-step neural network before moving on to the stability study of the 1-Lipschitz and ext-Lipschitz network.

Stability analysis of the one-step neural network

First, when comparing the reconstruction performance of the one-step and two-step networks by examining Figure 5.1.15 and Figure 5.1.10, as well as Figure 5.1.16 and Figure 5.1.11, we do not observe any significant visual differences between these pairs of plots.

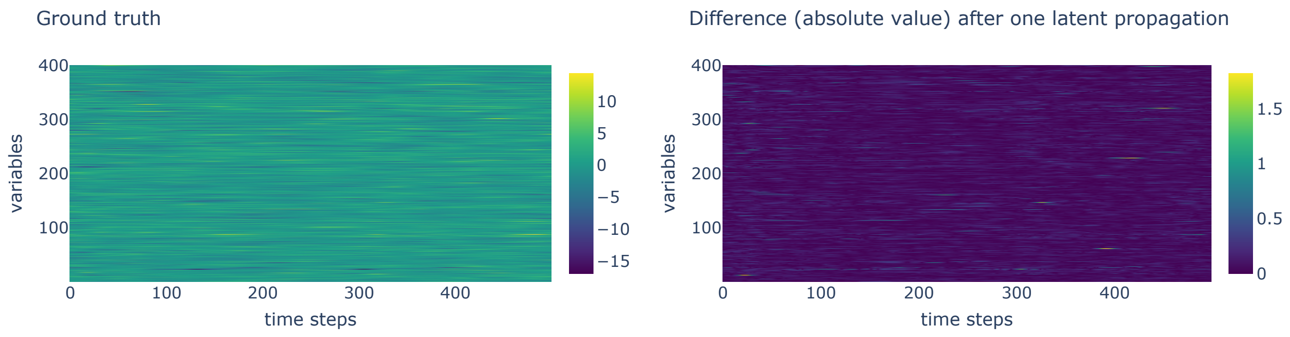


Figure 5.1.15: Left: ground truth simulation from the test dataset. Right: difference in absolute value between the ground truth and the reconstructed simulations (using autoencoder only) achieved by the one-step neural network.

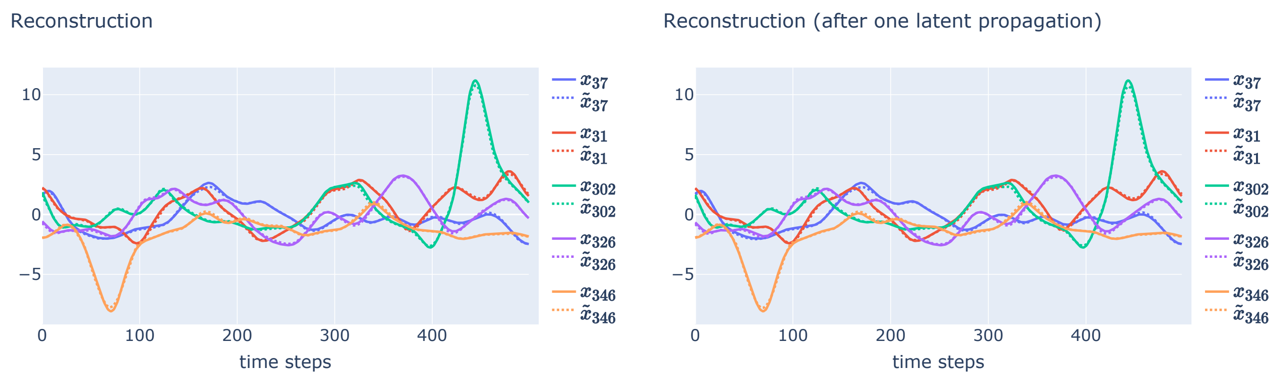


Figure 5.1.16: Left: reconstruction of five out of 400 variables from a simulation in the test dataset (the same one as in Figure 5.1.10) using only the autoencoder, produced by the one-step neural network. Right: the same reconstruction as in the left plot, but with one-step latent propagation also performed. Ground truths are represented by solid lines, and reconstructions by dashed lines.

Despite showing satisfactory results, the performance of the one-step neural network significantly degrades when it comes to temporal stability, unlike the two-step network (to be discussed subsequently). As detailed in section 4.2.3.1, one way to quantify a neural network's temporal stability is to apply it recursively a sufficient number of times from a given initial state. In our experiments with the augmented Lorenz96 system, we assess the stability of the neural networks separately on the training and test datasets. For the initial state, we consider two cases: either it is set to $\boldsymbol{x}_0$ or to $\boldsymbol{x}_{250}$. Since $\boldsymbol{x}_0$ is the first state of the simulation, it is less likely to be well represented in the data distribution compared to subsequent states (such as $\boldsymbol{x}_{250}$). Consequently, a full reconstruction starting from $\boldsymbol{x}_0$ is more likely to diverge than one starting from $\boldsymbol{x}_{250}$.

Figure 5.1.17 and Figure 5.1.18 compare the reconstruction performances of the one-step neural network in terms of dataset type and choice of the initial point, for two simulations (one from the training dataset and one from the test dataset). At first glance, the plots reveal that in every case, the one-step neural network fails to produce a stable reconstruction, as predictions (and therefore RMSE scores) quickly diverge to extremely large values. Interestingly, the errors are larger on the training simulation than on the test one. However, we cannot yet draw any general conclusions since we are currently considering only two distinct elements of the training and test datasets. One point previously mentioned and that tends to be verified here (at least with these two simulations) is that the reconstruction is worse when starting at $\boldsymbol{x}_0$ rather than $\boldsymbol{x}_{250}$.

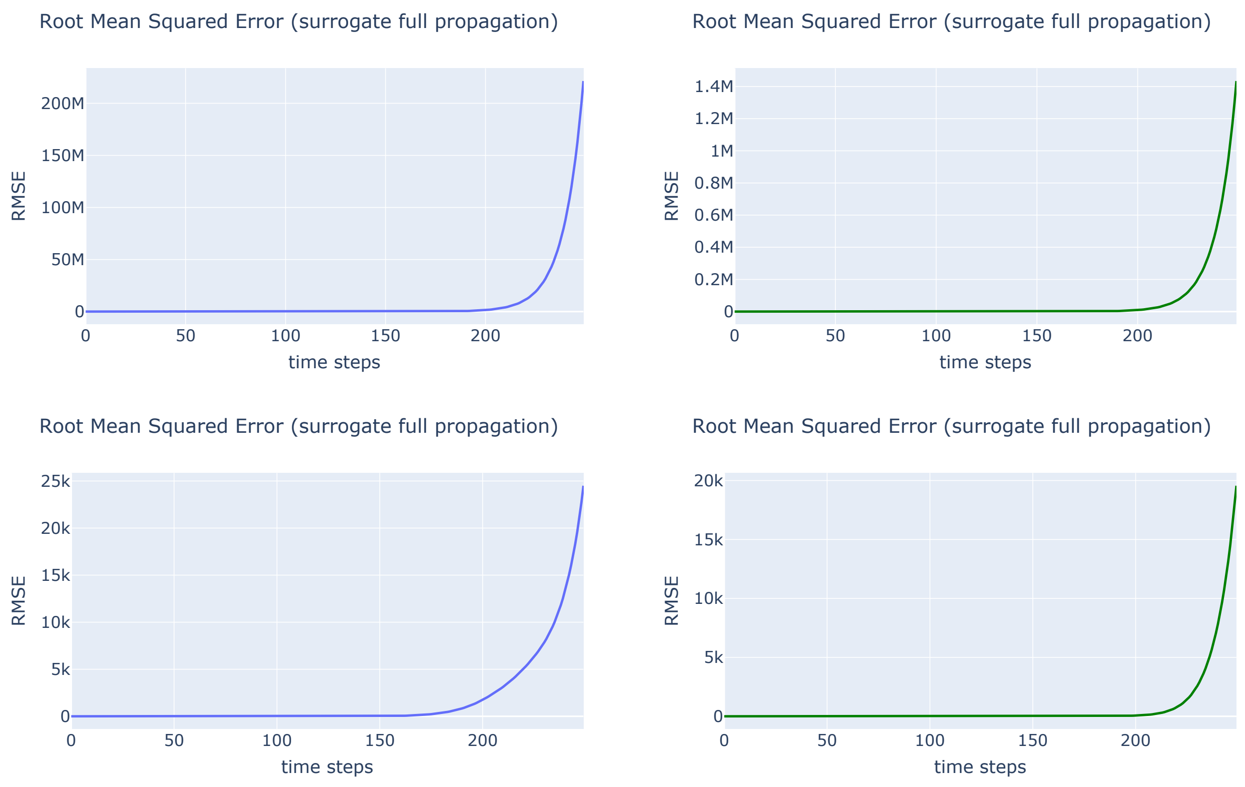


Figure 5.1.17: Top left: RMSE between a ground truth simulation from the training dataset and its full reconstruction by the one-step trained neural network over 250 time steps from $\boldsymbol{x}_0$. Top right: RMSE between a ground truth simulation from the test dataset and its full reconstruction by the one-step trained neural network over 250 time steps from $\boldsymbol{x}_0$. Bottom left: same as top left but with the initial state set as $\boldsymbol{x}_{250}$. Bottom right: same as top right, but the initial state set as $\boldsymbol{x}_{250}$. Blue and green colors are used to differentiate between training (blue) and test (green) data.

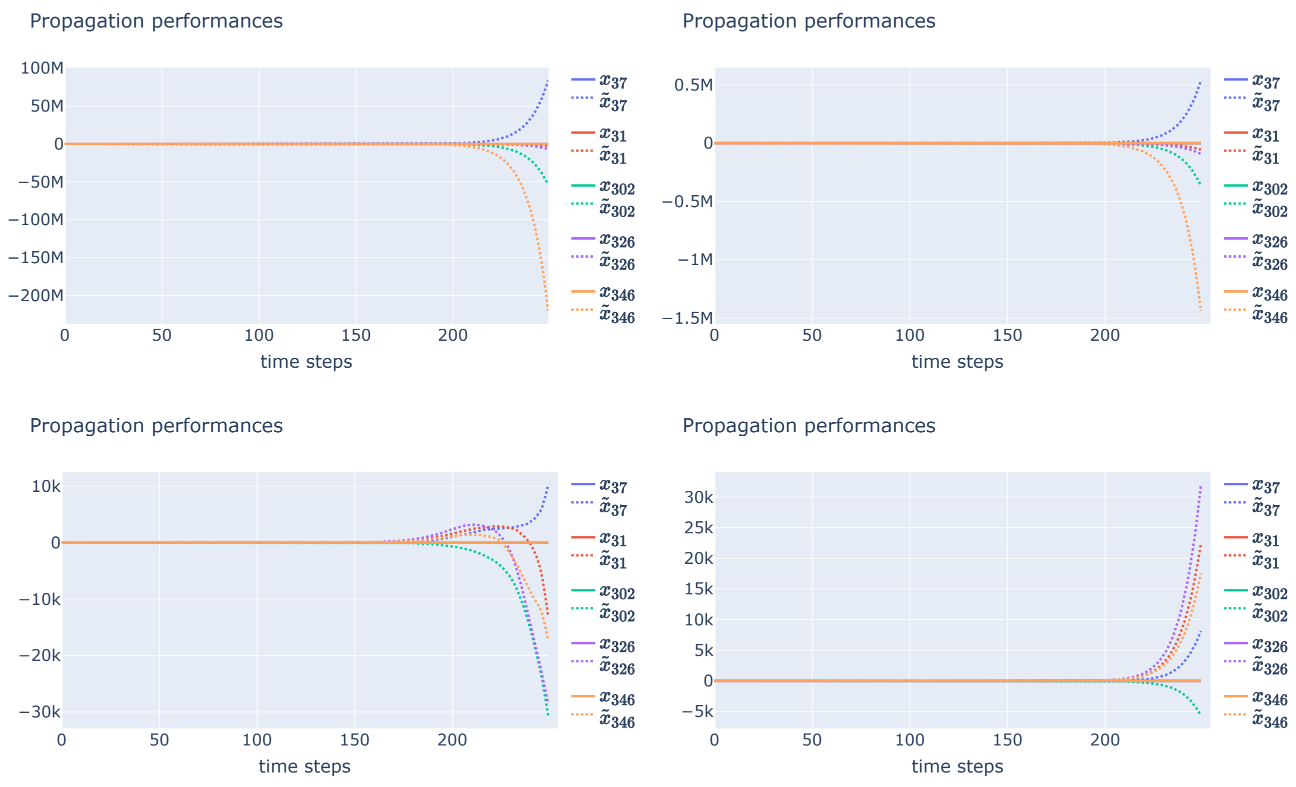
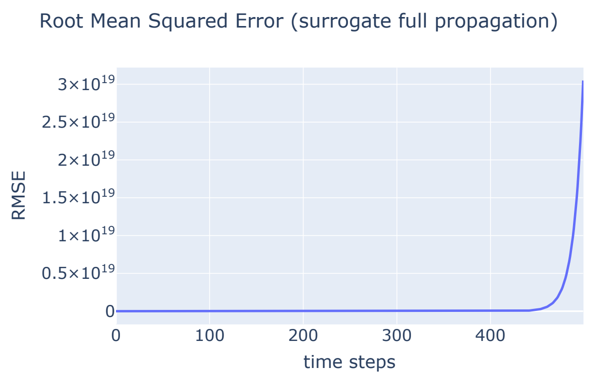


Figure 5.1.18: Top left: full temporal reconstruction of five out of 400 variables by the one-step trained network from a simulation in the training dataset (initial state $\boldsymbol{x}_0$). Top right: full temporal reconstruction of five out of 400 variables by the one-step trained network from a simulation in the test dataset (initial state $\boldsymbol{x}_0$). Bottom left: same as top left but with $\boldsymbol{x}_{250}$ as the initial state. Bottom right: same as top right, but with $\boldsymbol{x}_{250}$ as the initial state. Ground truths are represented by a solid line, and reconstructions with a dashed one.

We also consider another reconstruction metric, which is the reconstruction of the entire simulation. This involves applying the surrogate network 500 times consecutively over the encoded initial state $\boldsymbol{z}_0$. The results are shown in Figure 5.1.19 and Figure 5.1.20. As expected, given the one-step neural network's poor performance over 250 time steps, the same plots extended to 500 time steps confirm its unstable and diverging behavior.



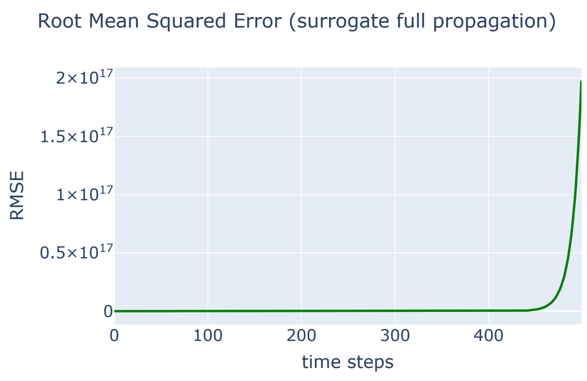
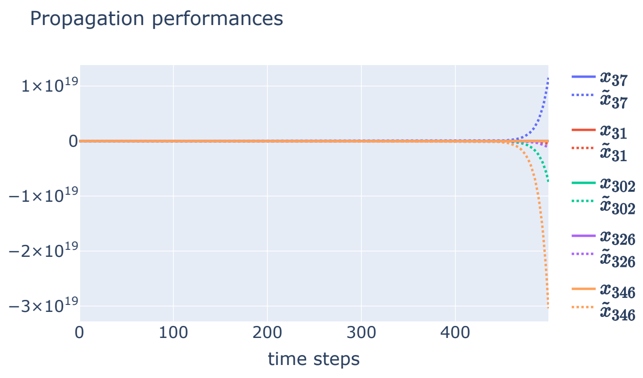


Figure 5.1.19: Plots similar to the top left and top right ones in Figure 5.1.17, but with the surrogate applied over a total of 500 time steps.



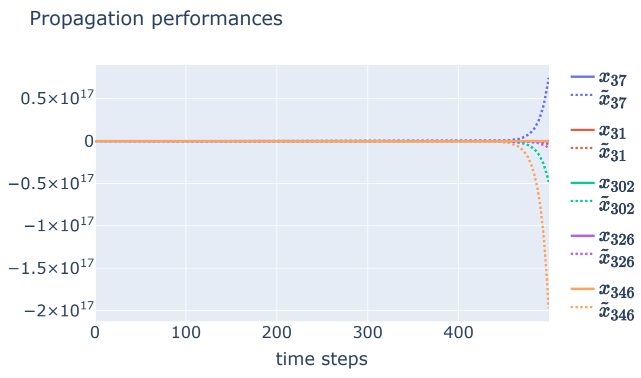


Figure 5.1.20: Same plots as the top left and top right ones of Figure 5.1.18, but with the surrogate applied over 500 time steps in total.

However, we cannot draw global conclusions about the behavior of the one-step neural network based on two simulations only. Therefore, we chose to consider RMSE statistics over both the training and test datasets rather than selecting one simulation from each. Our strategy is as follows:

1. Compute the RMSE scores for every simulation in both the training and test datasets. This produces time series RMSE values like those shown in Figure 5.1.17 and Figure 5.1.19. As before, we distinguish between the full reconstructions of 500 time steps and those involving only 250 time steps.

2. For each RMSE time series, compute its mean temporal value.
3. For each mean temporal value, compute its base 10 logarithm.
4. Plot the RMSE distributions (in percentage) of the base 10 logarithm values using bar charts. Results are reported in Figure 5.1.21 and Figure 5.1.22.

Contrary to what we might have concluded from Figure 5.1.17 and Figure 5.1.18, the global statistics shown in Figure 5.1.21 indicate that starting from $\boldsymbol{x}_0$ or $\boldsymbol{x}_{250}$ does not make much of a difference in the mean RMSE values achieved. When comparing the ratios per bin between the upper and lower plots, we notice that they are almost identical for the training dataset simulation and still quite similar for the test dataset one. However, a closer look at these ratios shows that the results are very slightly better when the initial state is $\boldsymbol{x}_{250}$. As expected, extending the reconstructions for another 250 time steps results in much higher mean RMSE values, causing a significant shift in the RMSE's power of ten from Figure 5.1.21 to Figure 5.1.22.

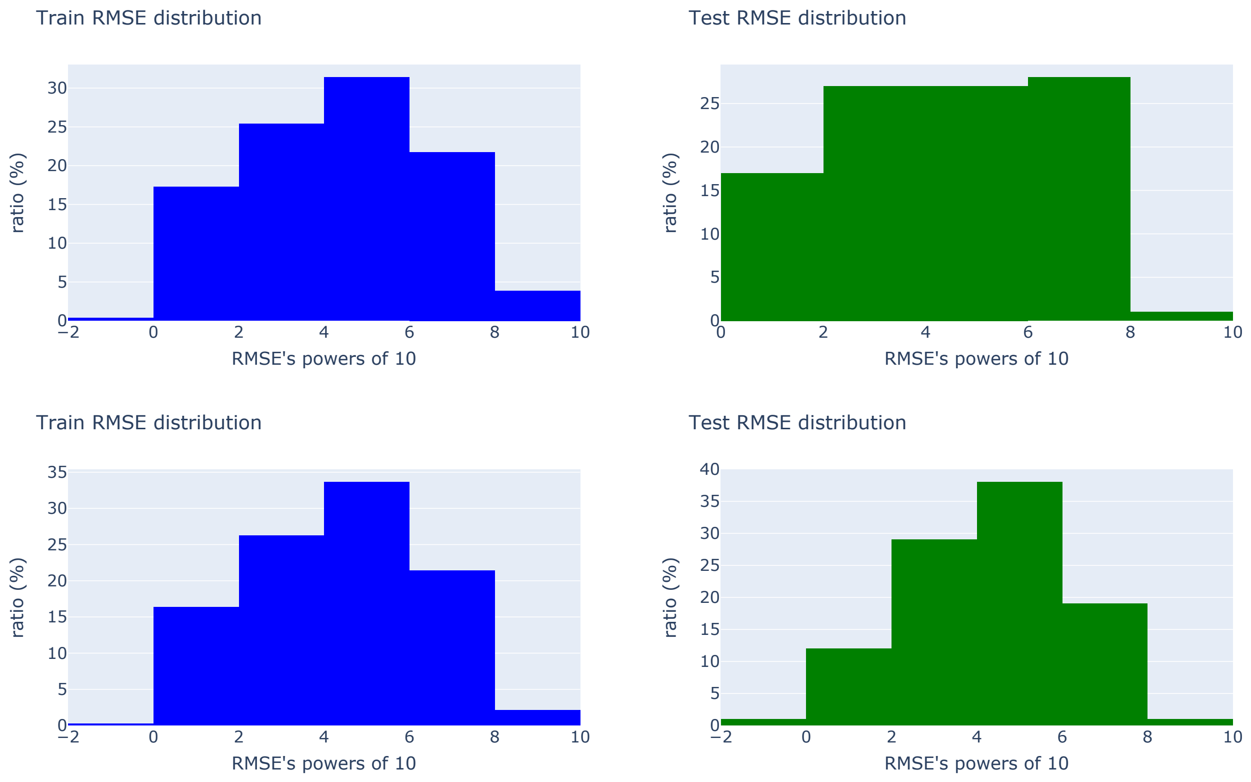


Figure 5.1.21: Top left: distribution of the mean RMSE values when reconstructing the entire training dataset simulations over 250 time steps with the one-step network (initial state is $\boldsymbol{x}_0$). Top right: distribution of the mean RMSE values when reconstructing the entire test dataset simulations with the one-step network (initial state $\boldsymbol{x}_0$). Bottom left: same as top left but with $\boldsymbol{x}_{250}$ as the initial state. Bottom right: same as top right, but with $\boldsymbol{x}_{250}$ as the initial state.

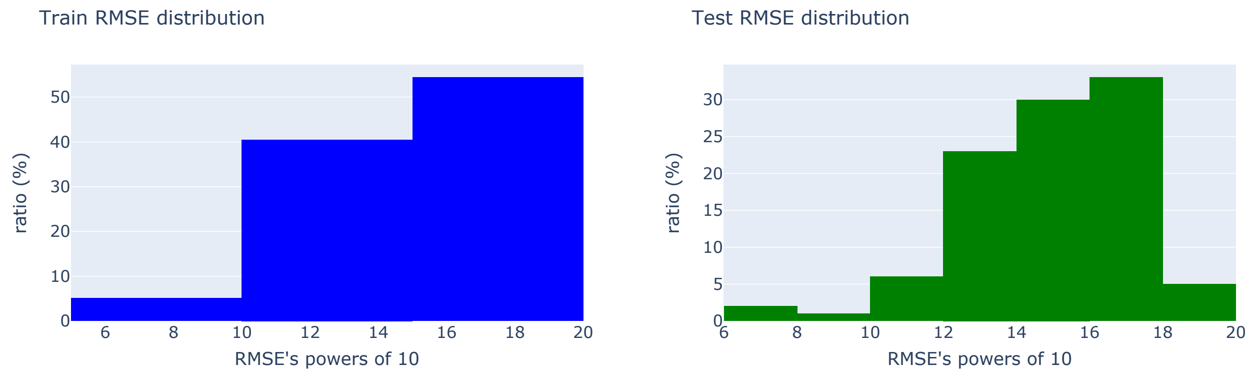


Figure 5.1.22: Plots similar to the top left and top right ones in Figure 5.1.21, but with the surrogate applied over a total of 500 time steps.

In conclusion, despite the impressive reconstruction capabilities of the one-step neural network, our thorough analysis of its stability performance demonstrates that setting C to 1 leads to unstable dynamics and an inability to perform long-term predictions. This also confirms that the accuracy of a neural network can be completely uncorrelated with its temporal stability.

Stability analysis of the two-step neural network

In the following, we present the results achieved with the two-step neural network. We demonstrate that introducing recursivity in the training by simply setting C to 2 results in outstanding temporal stability of the network. We use exactly the same metrics and plot the same variables as those reported for the one-step network to show the significant positive change brought by the $C = 2$ setting. Figure 5.1.23 and Figure 5.1.24 show that the two-step neural network is completely stable over 250 time steps for the two reported simulations, and exhibits relatively accurate forecasts up to about the first 50 time steps. As opposed to the one-step network, for which the RMSE over 250 time steps explode to values between 20.10^3 and 200.10^6, the RMSE scores here seem to stabilize below 1.5.

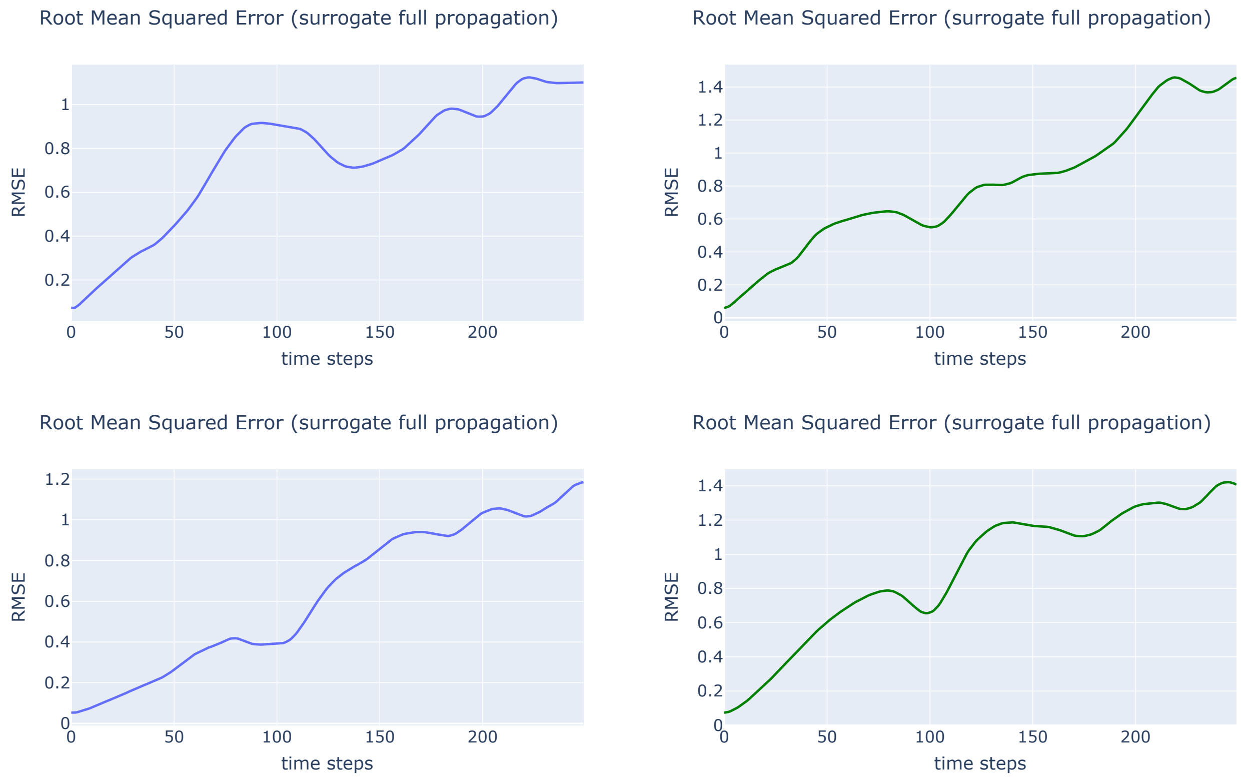


Figure 5.1.23: Top left: RMSE between a ground truth simulation from the training dataset and its full reconstruction by the two-step trained neural network over 250 time steps from $\boldsymbol{x}_0$. Top right: RMSE between a ground truth simulation from the test dataset and its full reconstruction by the two-step trained neural network over 250 time steps from $\boldsymbol{x}_0$. Bottom left: same as top left but with the initial state set as $\boldsymbol{x}_{250}$. Bottom right: same as top right, but the initial state set as $\boldsymbol{x}_{250}$. Blue and green colors are used to differentiate between training (blue) and test (green) data.

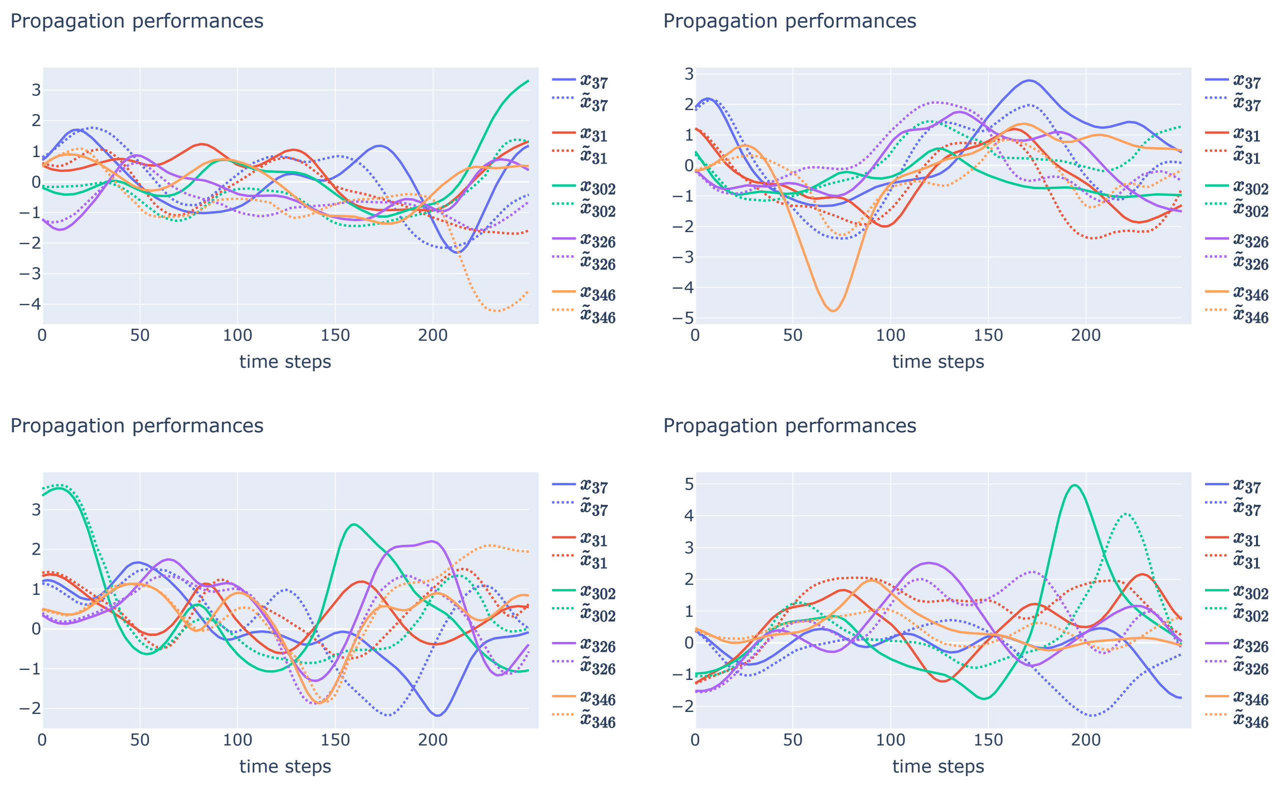


Figure 5.1.24: Top left: full temporal reconstruction of five out of 400 variables by the two-step trained network from a simulation in the training dataset (initial state $\boldsymbol{x}_0$). Top right: full temporal reconstruction of five out of 400 variables by the two-step trained network from a simulation in the test dataset (initial state $\boldsymbol{x}_0$). Bottom left: same as top left but with $\boldsymbol{x}_{250}$ as the initial state. Bottom right: same as top right, but with $\boldsymbol{x}_{250}$ as the initial state. Ground truths are represented by a solid line, and reconstructions with a dashed one.

When extending the full reconstruction of Figure 5.1.23 for another 250 time steps, the RMSE slightly increases but remains stable around 1.5 on the training dataset, and fluctuates around 1.4 on the test dataset, as shown in Figure 5.1.25. Regarding the predictions themselves, as depicted in Figure 5.1.26, we can draw the same conclusion as in Figure 5.1.24: from time step 50 onward, the forecasts no longer fit the ground truth. Over the time window $[250, 500]$, they continue to deviate from reality but remain approximately in the same range as the ground truth in terms of variable values (at least for the five variables shown in Figure 5.1.25). However, since we have only examined two simulations (one from the training dataset and another from the test dataset), we cannot draw general conclusions about the network's stability. Therefore, we now consider global statistics, similar to those presented in Figure 5.1.21 and Figure 5.1.22 for the one-step network.

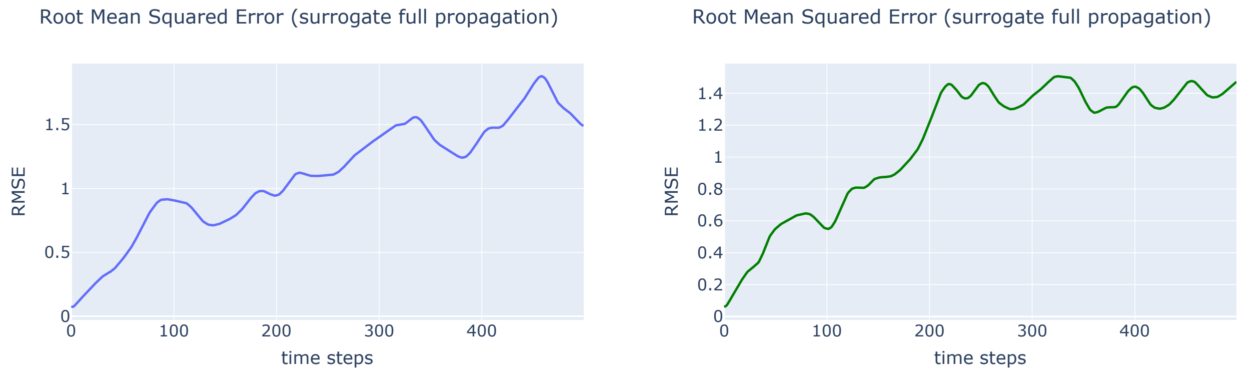


Figure 5.1.25: Plots similar to the top left and top right ones in Figure 5.1.23, but with the surrogate applied over a total of 500 time steps.

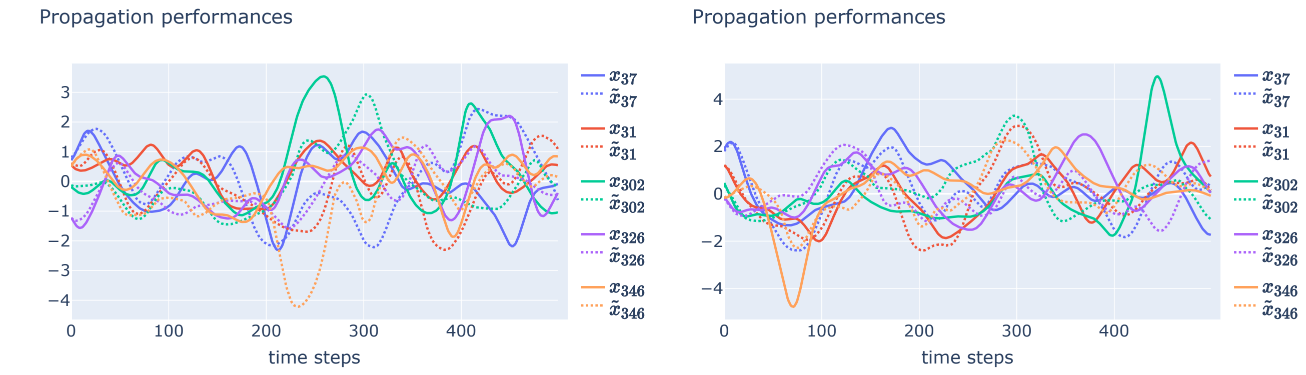


Figure 5.1.26: Plots similar to the top left and top right ones in Figure 5.1.24, but with the surrogate applied over a total of 500 time steps.

In the case of a full reconstruction over 250 consecutive time steps, the mean RMSE values are remarkably lower (compared to those of the one-step network) and are mostly concentrated between $10^{-0.2}$ and $10^{0.2}$ as shown in Figure 5.1.27. Notably, there is not a single simulation in either the training or test dataset for which the reconstruction diverges, with the largest mean RMSE being around 10. Additionally, the statistics are very similar whether the initial point is set to $\boldsymbol{x}_0$ or $\boldsymbol{x}_{250}$.

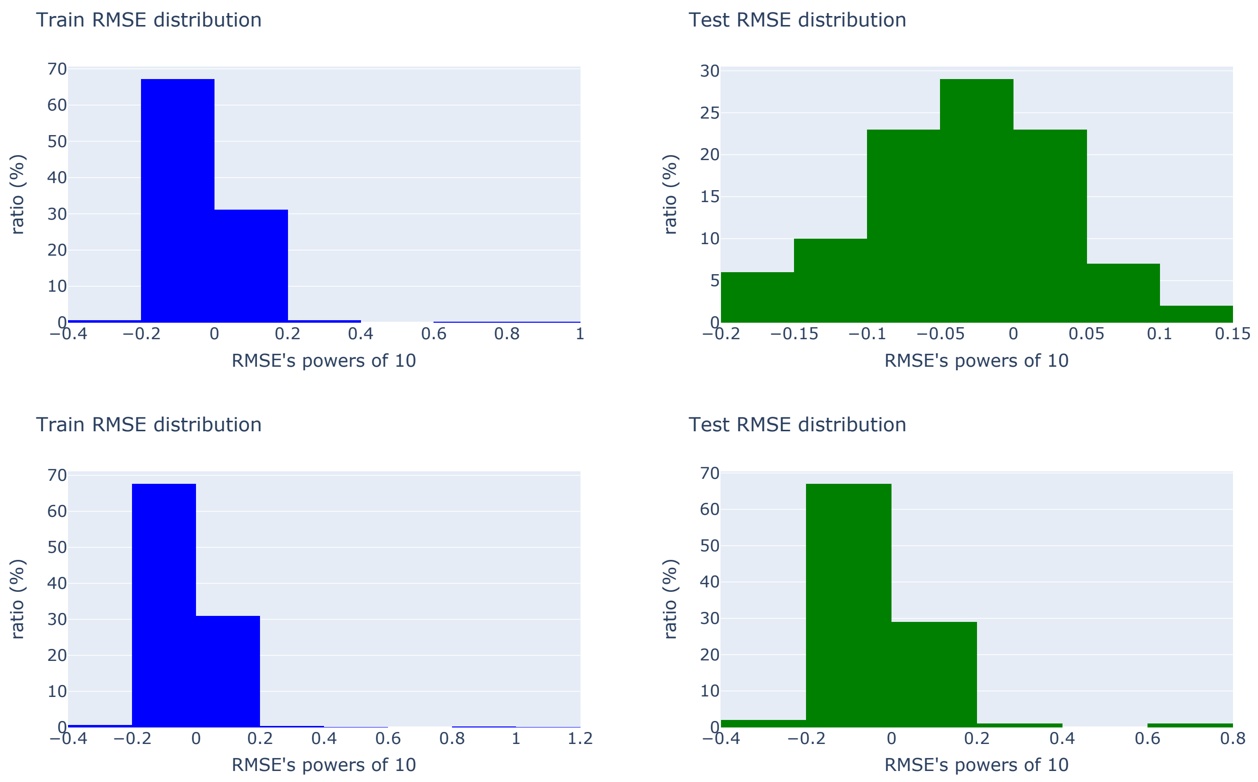


Figure 5.1.27: Top left: distribution of the mean RMSE values when reconstructing the entire training dataset simulations over 250 time steps with the two-step network (initial state is $\boldsymbol{x}_0$). Top right: distribution of the mean RMSE values when reconstructing the entire test dataset simulations with the two-step network (initial state $\boldsymbol{x}_0$). Bottom left: same as top left but with $\boldsymbol{x}_{250}$ as the initial state. Bottom right: same as top right, but with $\boldsymbol{x}_{250}$ as the initial state.

When considering a full reconstruction of 500 time steps from $\boldsymbol{x}_0$, the statistics remain impressive as 95% of the mean RMSE values are smaller than 10 on both the training and test datasets, as shown by Figure 5.1.28. Unlike Figure 5.1.27, we notice here in Figure 5.1.28 that the surrogate can lead to errors larger than 10^3, even if this occurs in less than 1.5% of the data.

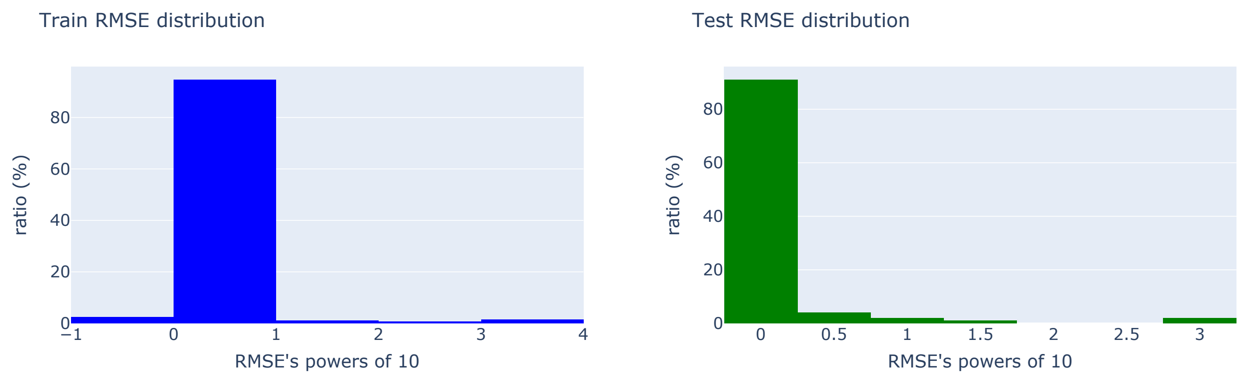


Figure 5.1.28: Plots similar to the top left and top right ones in Figure 5.1.27, but with the surrogate applied over a total of 500 time steps.

In conclusion, the mere introduction of recursivity when performing latent propagation with the surrogate during training has a significant impact on the stability of the resulting trained neural network. While the one-step model is unable to show any stable behavior on a single simulation from either the training or test datasets, the two-step network demonstrates remarkable performance with very low RMSE values over all the considered simulations and over at least 500 consecutive calls of the surrogate.

Stability analysis of Lipschitz surrogate network:

As presented in section 4.2.3.1, we also train 1-Lipschitz and ext-Lipschitz surrogate networks to achieve stable data-driven models. As the stability of the surrogate networks is intended to be ensured by the Lipschitzness of the surrogates, parameter C is set to 1. When training the two Lipschitz surrogates with a patience parameter set to 15, the results are not satisfactory, showing high loss values, with the 1-Lipschitz network performing better than the ext-Lipschitz one (even though being ext-Lipschitz offers more flexibility to the network than being 1-Lipschitz). Therefore, we remove early stopping and train these two networks for 200 epochs, which allows them to demonstrate equivalent performance. The loss values over the validation and test datasets are very similar and are reported below in table 5.1. We observe that the loss scores of the Lipschitz networks are clearly below those of the one-step network: this significant difference can be attributable to the strong constraints imposed by the Lipschitzness of the two surrogate networks. The loss scores for the two-step network are not included here: they are already provided in section 5.1.2, and since the parameter C is different in this case, we cannot fairly compare them with those of the one-step, 1-Lipschitz, and ext-Lipschitz networks.

Dataset	Loss type	One-step network	1-Lipschitz network	ext-Lipschitz
Validation	reconstruction	0.0068	0.0141	0.0142
	chained	0.0068	0.0138	0.0141
	total	0.0407	0.0833	0.0844
Test	reconstruction	0.0069	0.0146	0.0143
	chained	0.0069	0.0143	0.0143
	total	0.0416	0.0860	0.0858

Table 5.1: Loss scores of training involving the one-step, 1-Lipschitz and ext-Lipschitz neural networks over the validation and test datasets.

Lipschitz surrogate networks prove to be very stable over time. For the sake of conciseness, we only show the reconstruction curves of the five selected variables (see Figure 5.1.29) and the bar charts over 500 time steps (see Figure 5.1.30). Despite the high reconstruction loss values of table 5.1, Figure 5.1.29 shows that the autoencoders trained with the Lipschitz surrogate networks perform well visually (at least for the five selected variables shown here), even though the performance is clearly inferior to the results depicted in Figure 5.1.16 and Figure 5.1.11. Specifically, in Figure 5.1.29, the curves do not perfectly superimpose, as seen in the extreme yellow and green peaks at time steps 70 and 445. Additionally, as suggested by table 5.1, the reconstructions (both regular and one-step latent propagation) are so similar that we can barely distinguish between the 1-Lipschitz and ext-Lipschitz reconstructions.

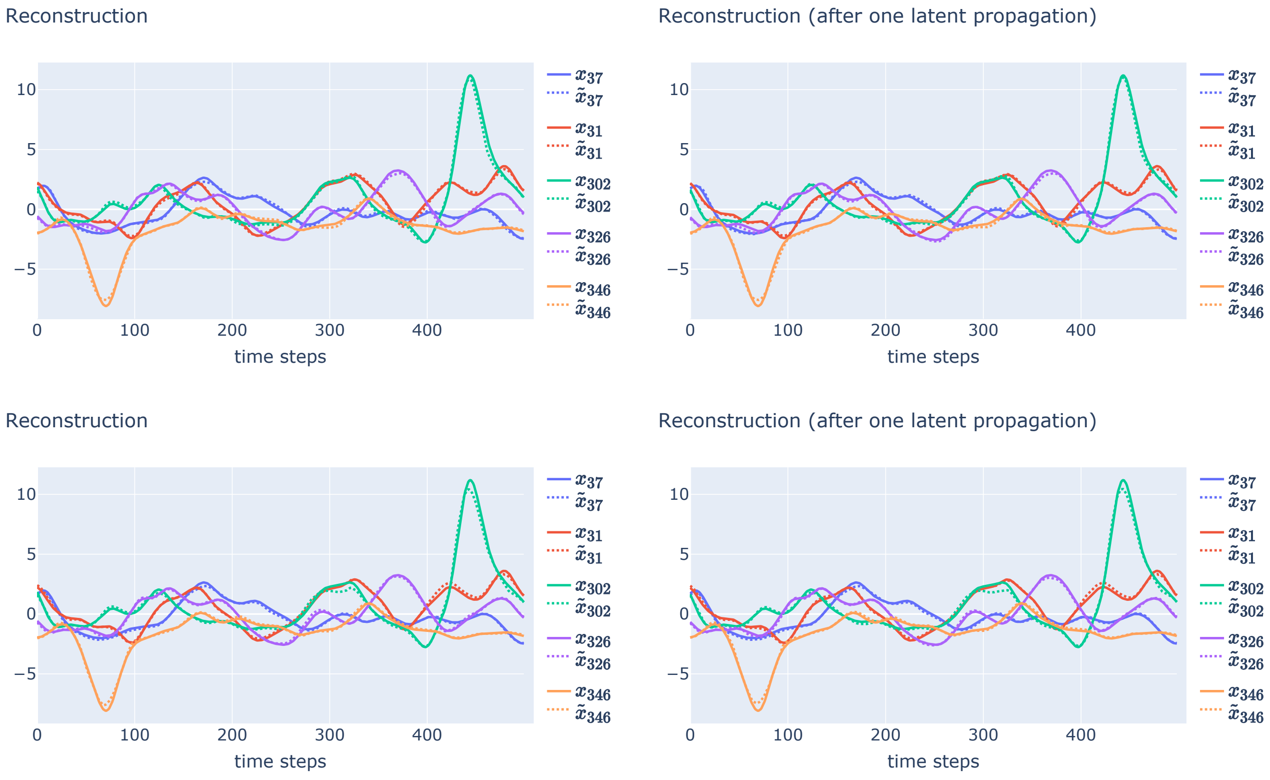


Figure 5.1.29: Top left: reconstruction of five out of 400 variables by the 1-Lipschitz trained network from a simulation in the test dataset, using only the autoencoder. Top right: same as top left plot, but with one latent propagation. Bottom left: reconstruction of five out of 400 variables by the ext-Lipschitz trained network from a simulation in the test dataset, using only the autoencoder. Bottom right: same as bottom left plot, but with one latent propagation. Ground truths are represented by a solid line, and reconstructions with a dashed one.

The primary advantage of the 1-Lipschitz and ext-Lipschitz neural networks is their impressive stability performance, as reported in Figure 5.1.30, which is comparable to that of the two-step network (see Figure 5.1.28 for comparison). Lipschitz networks might be slightly better at enforcing stability than the two-step trained model. Specifically, the 1-Lipschitz network does not produce a single diverging simulation over the entire dataset, and the ext-Lipschitz network has only 5 simulations with a mean RMSE larger than 100, compared to 32 for the two-step network (these figures are derived from our computations but are not directly visible from Figure 5.1.30 or Figure 5.1.28). Therefore, Lipschitz networks are a highly effective way to enforce long-term stability. However, upon closer examination of the computed data, we also found that the two-step network achieves a lower mean RMSE over 500 time steps compared to the Lipschitz networks.

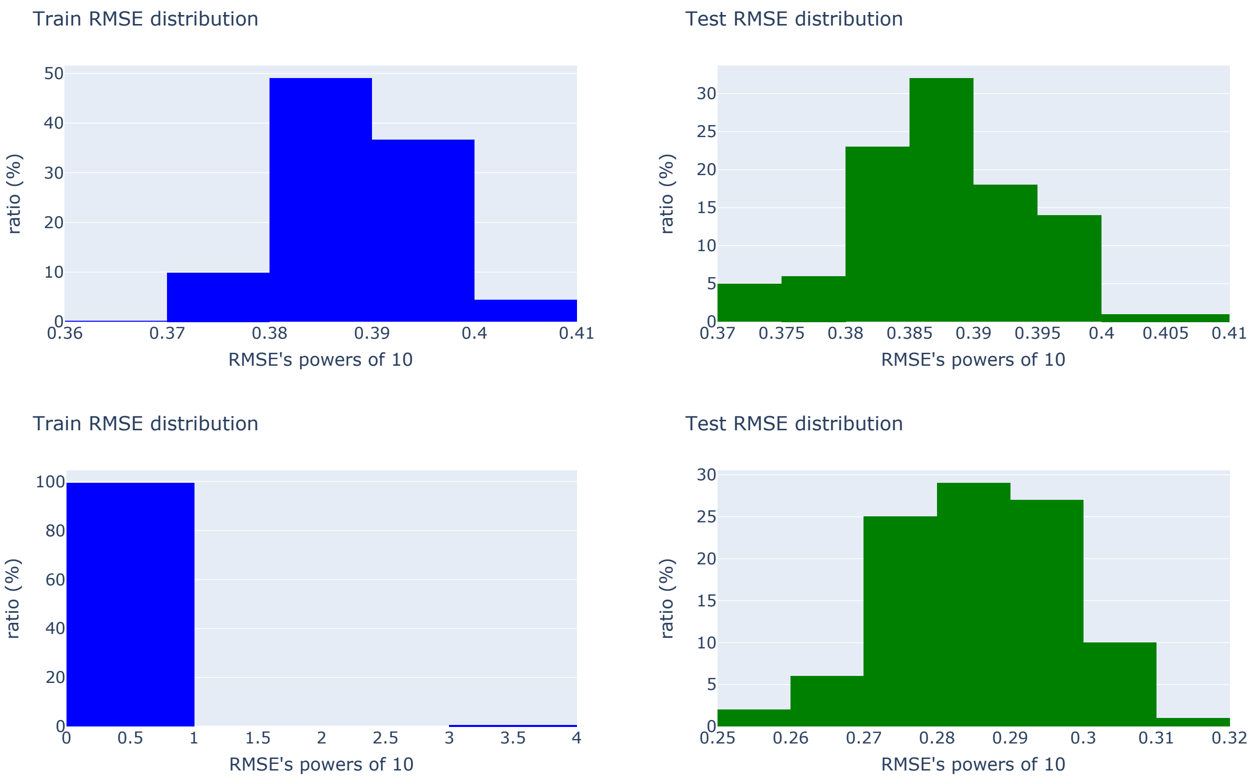


Figure 5.1.30: Top left: distribution of the mean RMSE values when fully reconstructing the entire training dataset simulations over 500 time steps with the 1-Lipschitz network. Top right: same as top left, but with the test dataset. Bottom left: distribution of the RMSE values when fully reconstructing the entire train dataset simulations over 500 time steps with the ext-Lipschitz network. Bottom right: same as bottom left, but with the test dataset.

5.1.4 Assessing the performance of our data assimilation framework

In this section, we present the data assimilation experiments conducted to assess and benchmark the efficiency of our ETKF-Q-L algorithm, as described in section 4.3. The comparison criterion used is the mean temporal RMSE, computed in the physical space of dimension 400. To objectively and fairly evaluate our ETKF-Q-L approach, we define the following benchmark experiments:

- **ETKF-Q**: this is the standard ETKF-Q algorithm (Fillion *et al.*, 2020). Assimilation is performed within the augmented Lorenz96 space without using any neural network. Temporal propagation involves mapping the augmented data into the original 40-dimensional space, applying the regular Lorenz96 propagator (using the fourth-order Runge-Kutta scheme), and then mapping this forecast back to the augmented space.
- **ETKF-Q-L**: this is our proposed method. Data assimilation is performed within the latent space of the two-step network's autoencoder. Temporal propagation is directly performed on the latent analysis $\boldsymbol{z}_a$, by the trained surrogate network.
- **ETKF-Q-L-1-step**: data assimilation is performed within the latent space of the one-step network's autoencoder. Temporal propagation is directly performed on the latent analysis $\boldsymbol{z}_a$, by the trained surrogate network.
- **ETKF-Q-L-1-Lipschitz**: data assimilation is performed within the latent space of the 1-Lipschitz network's autoencoder. Temporal propagation is directly performed on the latent analysis $\boldsymbol{z}_a$, by the 1-Lipschitz surrogate.
- **ETKF-Q-L-ext-Lipschitz**: data assimilation is performed within the latent space of the ext-Lipschitz network's autoencoder. Temporal propagation is directly performed on the latent analysis $\boldsymbol{z}_a$, by the ext-Lipschitz surrogate.
- **ETKF-Q-L-PCA**: data assimilation is performed within the latent space obtained by PCA on the augmented Lorenz96 data. Temporal propagation is directly performed on the latent analysis $\boldsymbol{z}_a$, by the trained surrogate network.
- **ETKF-Q-L-PCA-LinReg**: data assimilation is performed within the latent space obtained by PCA. Temporal propagation is directly performed on the latent analysis $\boldsymbol{z}_a$, by a linear regression operator. This operator is first fitted on the low-dimensional data produced by PCA.

In addition to the six latent DA benchmark experiments, we define their **physical-latent data assimilation** counterparts. For each experiment involving latent data assimilation, we define a new experiment where assimilation is performed in the physical (*i.e.*, augmented Lorenz96) space using the regular ETKF-Q algorithm, but time propagation occurs in the latent space. Specifically, at every DA propagation step, the analysis $\boldsymbol{x}_a$ is encoded (either by the trained encoder or PCA), propagated forward in time (either by the trained surrogate or the linear regression operator), and then decoded (either by the trained decoder or the inverse PCA operation). We add the character string "**-P-**" to the name of each experiment to define the following new ones: **ETKF-Q-L-P**, **ETKF-Q-P-L-1-step**, **ETKF-Q-P-L-1-Lipschitz**, **ETKF-Q-P-L-ext-Lipschitz**, **ETKF-Q-P-L-PCA**, and **ETKF-Q-P-L-PCA-LinReg**. The only difference between the latent DA and physical-latent DA versions is the assimilation space. This approach aims to demonstrate that performing assimilation within a reduced-space can lead to faster and more accurate results.

We summarize the details of each data assimilation experiment in table 5.2 and table 5.3. For ease of comparison, both tables include the results of the reference ETKF-Q experiment, which performs data assimilation in the physical space.

Experiment name	Data assimilation space	Propagation operator
ETKF-Q-L	Latent space	$\mathcal{S}$
ETKF-Q-L-1-step	Latent space	$\mathcal{S}_{\text{1-step}}$
ETKF-Q-L-1-Lipschitz	Latent space	$\mathcal{S}_{\text{1-Lip}}$
ETKF-Q-L-ext-Lipschitz	Latent space	$\mathcal{S}_{\text{ext-Lip}}$
ETKF-Q-L-PCA	Latent space	$\mathcal{S}_{\text{PCA}}$
ETKF-Q-L-PCA-LinReg	Latent space	LinReg

Table 5.2: Latent data assimilation benchmark experiments. Data assimilation analyses are performed within the latent space of each respective experiment. Subscript notation for the surrogate networks distinguishes between the different latent propagators. Plain notation $\mathcal{S}$ is used exclusively for our ETKF-Q-L method, and refers to the two-step network's surrogate, as described in section 5.1.2 and section 5.1.3.

As indicated by the common notations for the surrogate networks in table 5.2 and table 5.3, they denote the same trained networks in both tables. For the experiments **ETKF-Q-L-PCA** and **ETKF-Q-P-L-PCA**, which combine PCA and a surrogate network, we specifically trained a neural network to propagate forward in time within the reduced-space obtained via PCA. In this context, the parameter C, defining the recursivity of the surrogate network, is set to 2. This ensures that the trained surrogate exhibits strong temporal stability, similar to the stability reported for the two-step network in section 5.1.2 and section 5.1.3. The reduced-space itself

Experiment name	Data assimilation space	Propagation operator
ETKF-Q	Physical space	$\mathcal{F} \circ \text{L96} \circ \mathcal{F}^{-1}$
ETKF-Q-P-L	Physical space	$\mathcal{E} \circ \mathcal{S} \circ \mathcal{D}$
ETKF-Q-P-L-1-step	Physical space	$\mathcal{E}_{\text{1-step}} \circ \mathcal{S}_{\text{1-step}} \circ \mathcal{D}_{\text{1-step}}$
ETKF-Q-P-L-1-Lip	Physical space	$\mathcal{E}_{\text{1-Lip}} \circ \mathcal{S}_{\text{1-Lip}} \circ \mathcal{D}_{\text{1-Lip}}$
ETKF-Q-P-L-ext-Lip	Physical space	$\mathcal{E}_{\text{ext-Lip}} \circ \mathcal{S}_{\text{ext-Lip}} \circ \mathcal{D}_{\text{ext-Lip}}$
ETKF-Q-P-L-PCA	Physical space	$\text{PCA} \circ \mathcal{S}_{\text{PCA}} \circ \text{PCA}^{-1}$
ETKF-Q-P-L-PCA-LinReg	Physical space	$\text{PCA} \circ \text{LinReg} \circ \text{PCA}^{-1}$

Table 5.3: Benchmark experiments where assimilation is performed within the physical space, *i.e.*, within the augmented Lorenz96 space. Recall that $\mathcal{F}$ is defined in section 5.1 and denotes the function that maps from the augmented system to the Lorenz96 system, while $\mathcal{F}^{-1}$ represents the reverse mapping. Subscript notation for the encoder, surrogate, and decoder distinguishes between the different networks. Plain notations $\mathcal{E}, \mathcal{S}, \mathcal{D}$ are used exclusively for our ETKF-Q-L method, denoting the two-step network's components, as described in section 5.1.2 and section 5.1.3.

is generated by applying PCA to the training dataset.

We establish a common data assimilation framework to assess the relative performances of the ETKF-Q-L method and the benchmark experiments listed in table 5.2 and table 5.3:

- **40** ensemble members.
- **1000** cycles.
- observation operator $\boldsymbol{\mathcal{H}}$ is the identity matrix, i.e., $\boldsymbol{I}_{400}$.
- observation error covariance matrix $\boldsymbol{R} = \boldsymbol{\sigma}_R^2 \boldsymbol{I}_{400}$, with $\boldsymbol{\sigma}_R = 1.0$. This matrix is used at every cycle.
- background error covariance matrix $\boldsymbol{B} = \boldsymbol{\sigma}_B^2 \boldsymbol{I}_{400}$, with $\boldsymbol{\sigma}_B = 0.3$.

In the case of the ETKF-Q experiment, we have to define a model error covariance matrix, denoted by $\boldsymbol{Q}$ (see section 2.2.2.3). We set $\boldsymbol{Q}$ as a diagonal matrix weighted by variance $\boldsymbol{\sigma}_Q^2$. When performing data assimilation on the Lorenz96 system, typical values range from about 0.1 to 0.5, but can also exceed this interval. For instance, Brajard *et al.* (2020) compute statistics over a broader spectrum of values, ranging from 0.0 (no model error) up to 10. In the ETKF-Q experiment, the model error is directly applied within the Lorenz96 space, so relevant values can be considered within the range of $[0.1, 0.5]$, we picked $\boldsymbol{\sigma}_Q = 0.13$.

Additionally, we are interested in understanding how model errors in the Lorenz96 dynamics translate into the augmented system. Since the augmented dynamics are built upon the Lorenz96 equations, we can use our knowledge of typical model errors used for data assimilation in Lorenz96 to infer the corresponding errors in the augmented system.

To achieve this, we take Lorenz96 data and add white Gaussian noise with a standard deviation varying from 0.1 to 10 in increments of 0.05. We then apply the operator $\mathcal{F}^{-1}$ to both the regular and perturbed Lorenz96 data and compute the standard deviations of the differences. The results are shown in Figure 5.1.31. We observe that the standard deviations of the augmented data are approximately half the magnitude of the input perturbations, except for input standard deviations larger than 5, where input and output errors tend to converge to similar values.

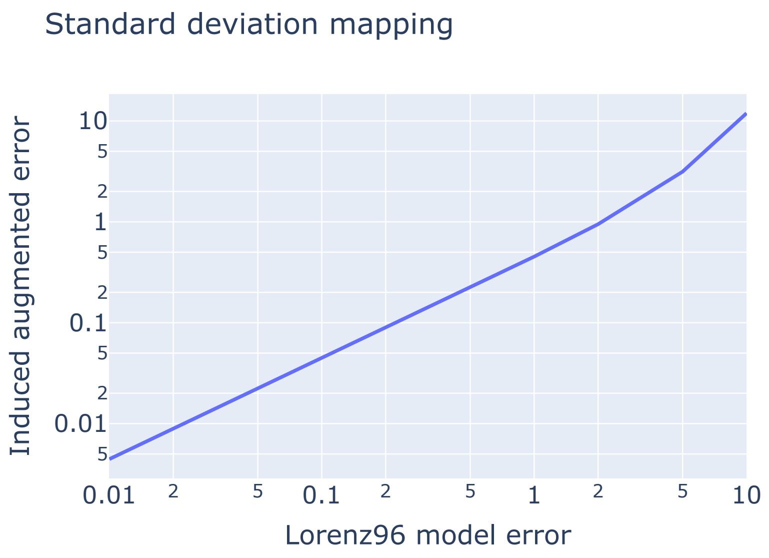


Figure 5.1.31: Standard deviations on the augmented Lorenz96 system, when adding a white Gaussian noise to the underlying Lorenz 96, with standard deviations ranging from 0.01 up to 10. We use log-scale for both x and y axis.

For all the experiments listed in table 5.2 and table 5.3, we have to tune two parameters: the inflation parameter λ (see section 2.2.2.2) and the model error correction term denoted by $\boldsymbol{Q}_\ell$ (see section 4.3). We consider $\boldsymbol{Q}_\ell$ to be a diagonal matrix weighted by $\boldsymbol{\sigma}^2_{Q_\ell}$. We perform a grid search over inflation values ranging from 1.01 up to 1.9, and over $\boldsymbol{\sigma}_{Q_\ell}$ values lying in [1e-4, 1]. These ranges have been fine-tuned experimentally and are therefore appropriate to optimize over λ and $\boldsymbol{\sigma}_{Q_\ell}$.

The numerical results of the grid search runs, along with computational times, are reported in table 5.4 and table 5.5. Mean temporal RMSE values are computed between the ground truth and the data assimilation analyses, omitting the first fifth of the total number of cycles to ensure RMSE scores are not affected by the stabilization period of the data assimilation system. Computational time statistics, that is means and standard deviations, are computed over the 414 runs resulting from the grid search strategy.

Our ETKF-Q-L algorithm outperforms all other benchmark experiments, both for latent and physical assimilations. It also shows one of the shortest computational times, completing 1000 cycles in about 2 seconds compared to about 22 seconds for the regular ETKF-Q algorithm. Notably, latent space assimilation experiments ETKF-Q-L, ETKF-Q-L-1-step, ETKF-Q-L-1-Lipschitz, ETKF-Q-L-ext-Lipschitz, ETKF-Q-L-PCA, and ETKF-Q-L-PCA-LinReg are between 25 to about 60 times faster than their physical counterparts.

The one-step network also demonstrates strong performance, with a mean temporal RMSE value of 0.189 for latent DA, but it still remains below the 0.159 mean RMSE score achieved by the two-step network in the same context. The one-step surrogate only slightly surpasses the two-step network in physical data assimilation, with a difference of 0.008. Despite being the least costly models in latent space data assimilation, both the 1-Lipschitz and ext-Lipschitz neural networks perform poorly compared to the simple combination of PCA with linear regression. However, they achieve lower mean RMSE in physical space assimilation, outperforming the ETKF-Q-P-L-PCA and ETKF-Q-P-L-PCA-LinReg benchmark experiments. For both latent and physical assimilation, we observe that the combination of a surrogate network and an autoencoder consistently surpass a surrogate network trained within a reduced-space obtained by PCA.

In the current configuration, the standard ETKF-Q algorithm achieves the second overall performance behind ETKF-Q-L, with a score of 0.169, reached for $\lambda = 1.04$ and $\boldsymbol{\sigma}_{Q_\ell} = 0.5$.

As theoretically motivated in section 4.2.1, latent space data assimilation as the potential to be more accurate than physical space data assimilation. This observation holds true for all benchmark experiments that run assimilation in both spaces, with the exception of those involving Lipschitz surrogate networks or PCA-based dimensionality reduction. It is possible that the Lipschitz constraint limits the latent space's ability to accurately align with the unstable-neutral subspace or to capture other critical directions. Similarly, PCA may not be the optimal method for reduced-space representation when it comes to accounting for the most important subspaces. Nonetheless, the results presented in table 5.4 and table 5.5 tend to confirm that there exists latent space representations of the original data for which the ETKF-Q assimilation process is more accurate than the standard full space algorithm.

Notably, performing the assimilation in the physical space proves to be very costly when using neural networks or PCA combined with linear regression, compared to data-driven latent space data assimilation. This loss of time relates to the convergence of scipy and numpy eigendecomposition functions: it is therefore likely that neural networks lead to poorly-conditioned matrices when performing the assimilation within the physical space.

In conclusion, our ETKF-Q-L algorithm proves to be both more accurate and faster than all other benchmark experiments, validating the relevance of the proposed methodology, as theoretically motivated in section 4.2.1.

Experiment name	Mean temporal RMSE	Inflation	$\boldsymbol{\sigma}_{Q_\ell}$	Time Avg.	Time Std.
ETKF-Q	0.169	1.04	0.5	21.79s	7.88s
ETKF-Q-L	0.159	1.03	0.006	2.043s	0.263s
ETKF-Q-L-1-step	0.189	1.09	0.006	2.038s	0.258s
ETKF-Q-L-1-Lip	0.602	1.16	0.07	4.01s	0.818s
ETKF-Q-L-ext-Lip	0.422	1.04	0.08	3.40s	0.804s
ETKF-Q-L-PCA	0.357	1.18	0.15	1.52s	0.216s
ETKF-Q-L-PCA-LinReg	0.413	1.01	0.5	1.44s	0.211s

Table 5.4: Data assimilation results when performing the assimilation within the latent space, along with the reference experiment ETKF-Q (first row) which performs the assimilation in the physical space. Along with the mean temporal RMSE (mean over the DA cycles, with omission of the first fifth), we indicate the inflation and model error correction values found by grid search. Additionally, we report the average computational times and their associated standard deviations.

Experiment name	Mean temporal RMSE	Inflation	$\boldsymbol{\sigma}_{Q_\ell}$	Time Avg.	Time Std.
ETKF-Q	0.169	1.04	0.5	21.79s	7.88s
ETKF-Q-P-L	0.224	1.08	0.35	1m 31s	9.98s
ETKF-Q-P-L-1-step	0.216	1.03	0.45	1m 25s	17.36s
ETKF-Q-P-L-1-Lip	0.240	1.03	0.6	1m 34s	5.7s
ETKF-Q-P-L-ext-Lip	0.243	1.1	0.45	1m 35s	17s
ETKF-Q-P-L-PCA	0.343	1.07	0.3	1m 20s	6.97s
ETKF-Q-P-L-PCA-LinReg	0.409	1.01	0.5	1m 21s	7.1s

Table 5.5: Data assimilation results when performing the assimilation within the physical space. Along with the mean temporal RMSE (mean over the DA cycles, with omission of the first fifth), we indicate the inflation and model error correction values found by grid search. Additionally, we report the average computational times and their associated standard deviations.

In section 4.3.1, we introduced the DAPPER package[(3)], which implements three variants of the square root method proposed by Raanes *et al.* (2015). This package allows for model error correction when the number of ensemble members is larger than the assimilation state space, a scenario not handled by the current ETKF-Q algorithm. We conducted latent space data assimilation experiments using the ETKF-Q-L method, varying the number of ensemble members to 60, 80, and 100. The numerical results are reported in table 5.6. The addition of more ensemble members does not result in significant differences. Only when the number of ensemble members is set to 60, we observe a minor improvement in the mean temporal RMSE score. This can be attributed to the fact that the augmented Lorenz96 system is built upon 40-dimensional Lorenz96 data, and having 40 ensemble members already correctly represents the data distribution and the error covariance matrix.

We also observe that the computational efficiency of our latent assimilation method is significantly impacted by the use of the SQRT-DEP model error correction algorithm from the DAPPER package. Whereas the regular ETKF-Q-L methodology allows the assimilation process to complete within a few seconds, as shown in table 5.4, the integration of the SQRT-DEP algorithm increases the computational times to levels comparable to those of the physical latent assimilation experiments reported in table 5.5. This increase is primarily due to the costly linear algebra computations of the SQRT-DEP approach.

(3) https://github.com/nansencenter/DAPPER

Number of members	RMSE	Inflation	$\boldsymbol{\sigma}_{Q_\ell}$	Time Avg.	Time Std.
60	0.1567	1.03	0.006	1m 54s	3.99s
80	0.1588	1.03	0.006	1m 55s	3.16s
100	0.1573	1.05	0.005	2m 3s	4.76s

Table 5.6: Data assimilation results of the ETKF-Q-L algorithm when using the SQRT-DEP model error correction method as described by Raanes *et al.* (2015) and implemented in the DAPPER package. Along with the Root Mean Squared Error, we indicate the inflation and model error correction values found by grid search. Additionally, we report the average computational times and their associated standard deviations.

5.2 The quasi-geostrophic (QG) model

The quasi-geostrophic (QG) model is used to describe the horizontal dynamics of atmospheric or oceanic motion on the synoptic scale in middle latitudes. In the following, we are specifically interested in the QG model applied to the atmosphere. Figure 5.2.1 highlights the mid-latitudes areas of the globe. The synoptic scale refers to large-scale atmospheric phenomena that typically span hundreds to thousands of kilometers and have timescales of several days. The dynamics at this scale are primarily governed by the balance between the Coriolis force and the pressure gradient force. The geostrophic approximation assumes that the horizontal pressure gradient force is exactly balanced by the Coriolis force. By relaxing this hypothesis, we achieve a better representation of real-world atmospheric motions, defining what is known as the quasi-geostrophic approximation. A visualization of global weather conditions can be found at `https://earth.nullschool.net/`, with updates made every three hours.

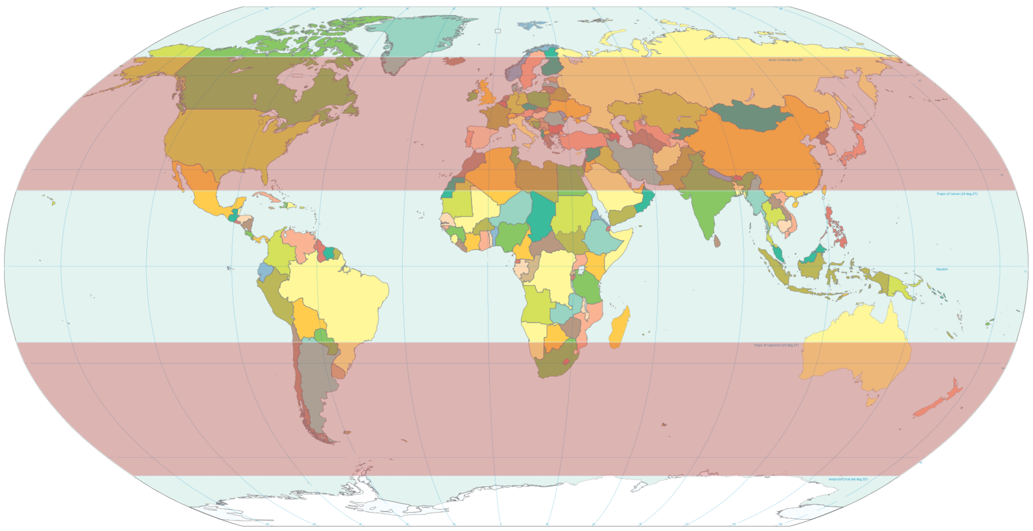


Figure 5.2.1: World map highlighting the middle latitudes regions with two red bands. Credit to `https://en.wikipedia.org/wiki/Middle_latitudes`.

The quasi-geostrophic model simplifies the full equations of motion (Wallace and Hobbs, 2006) by focusing on the dominant forces in geophysical fluid dynamics: the Coriolis force and the pressure gradient force. QG is particularly valuable for understanding the dynamics of large-scale weather systems, such as cyclones and anticyclones, and is an essential tool in both theoretical and applied meteorology.

The QG model is based on several key assumptions that simplify the atmospheric equations of motion:

1. Geostrophic Balance: the model assumes that the atmospheric flow is predominantly in geostrophic balance, meaning the Coriolis force nearly balances the horizontal pressure gradient force.
2. Hydrostatic Balance: the vertical pressure gradient force is balanced by the gravitational force, a reasonable assumption for large-scale atmospheric motions.
3. Small Rossby Number: the Rossby number, which measures the ratio of inertial to Coriolis forces, is assumed to be small. This allows the simplification of the momentum equations.

The atmosphere is stratified, meaning that properties like temperature, pressure, and wind can vary significantly with altitude. To capture these variations, the QG model can include multiple layers, each representing a distinct level or layer in the atmosphere. These layers can interact with each other, allowing for a more comprehensive representation of the vertical structure of atmospheric motions.

A common simplification is the two-layer QG model, which divides the atmosphere into two discrete layers:

1. the upper layer, which typically represents the upper troposphere or lower stratosphere.
2. the lower layer, which usually describes the lower to mid-troposphere.

For greater accuracy, the atmosphere can be divided into more than two layers, creating a multi-layer QG model. This allows for finer resolution of vertical structures and dynamics, but also increases computational complexity.

For further information about the quasi-geostrophic model and meteorology in general, we recommend Pedlosky (1987); Holton and Hakim Gregory (2004); Vallis (2017). Francophone readers might also refer to Malardel (2022).

5.2.1 A two-layer weather forecast model of the OOPS framework

To further assess the performance of our latent data assimilation approach, we consider the QG model implemented within OOPS. We remind that the Object-Oriented Prediction System (OOPS) is a collaborative project developed by ECMWF and Météo-France. OOPS aims to create a flexible, modular, and extensible framework for developing, testing, and implementing numerical weather prediction models and data assimilation systems. The primary goal of OOPS is to facilitate the integration of various components of NWP systems, making it easier to update and improve forecasting capabilities.

The two-layer quasi-geostrophic model is described by two variables: the streamfunction $\boldsymbol{\psi}$ and the potential vorticity $\boldsymbol{q}$. Each variable is labeled with subscripts 1 and 2, indicating whether it represents the dynamics of the upper layer (subscript 1) or the lower layer (subscript 2). The two-layer QG equations, given by

Fandry and Leslie (1984) (see also Pedlosky (1987)), are expressed in terms of non-dimensionalized variables and connect $\boldsymbol{q}$ and $\boldsymbol{\psi}$ as follows:

$$\begin{cases} \boldsymbol{q}_1 = \nabla^2 \boldsymbol{\psi}_1 - F_1 \left(\boldsymbol{\psi}_1 - \boldsymbol{\psi}_2\right) + \beta y \\ \boldsymbol{q}_2 = \nabla^2 \boldsymbol{\psi}_2 - F_2 \left(\boldsymbol{\psi}_2 - \boldsymbol{\psi}_1\right) + \beta y + R_s, \end{cases} \tag{5.3}$$

where ∇^2 denotes the two-dimensional Laplacian, β is the (non-dimensionalized) northward derivative of the Coriolis force parameter, R_s represents orography, and y is a normalized meridional distance (along the north-south axis). F_1 and F_2 are parameters that couple the two layers together (see Fisher and Gürol (2017) for further details).

The model domain is assumed to be cyclic in the zonal direction (i.e., along the x-axis or latitudinal lines), and the meridional velocity is assumed to vanish one grid space to the north and south of the domain. More physical and technical details can be found in Fandry and Leslie (1984); Pedlosky (1987); Fisher and Gürol (2017).

In our numerical experiments, the QG model runs over two 40×20 rectangular grids, one grid per atmospheric layer, representing atmospheric motions at 4 km (lower layer) and 6 km (upper layer) of altitude. Thus, the streamfunction $\boldsymbol{\psi}$ and the potential vorticity $\boldsymbol{q}$ each amount to $2 \times (40 \times 20) = 1600$ scalar values. Figure 5.2.2 provides a visual representation of these two grids.

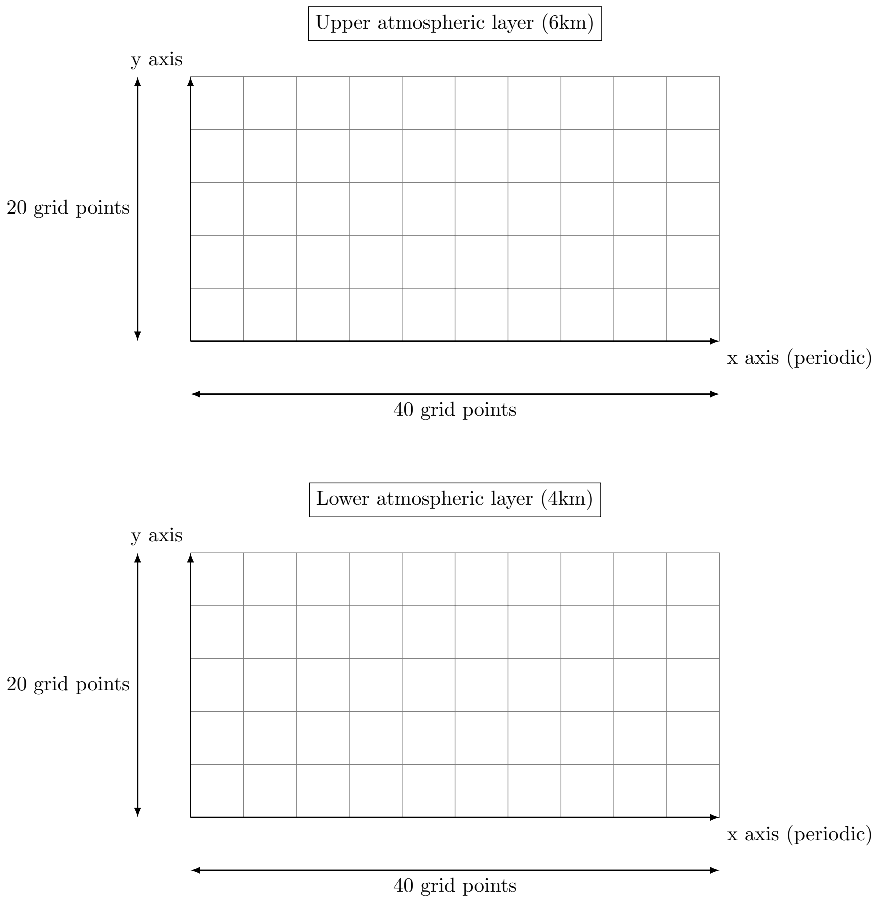


Figure 5.2.2: Representation of the atmospheric grids for the two-layer quasi-geostrophic model.

The numerical model runs with a time step of 8 minutes, and data are saved every two hours. From our numerical experiments, we concluded that a two-hour interval offers sufficient changes for a data-driven model to properly learn. Running a 20-day simulation takes about 2 seconds on our 2021 Macbook Pro equipped with the Apple M1 Pro chip. Figure 5.2.3 shows the contour lines of the streamfunction and potential vorticity for the upper and lower atmospheric layers at day 20 of the simulation. It is observed that the streamfunction represents the low frequencies of the dynamics, while the potential vorticity contains the high-frequency information. Since data assimilation is focused on forecasting the streamfunction variable, we do not further discuss potential vorticity in the following.

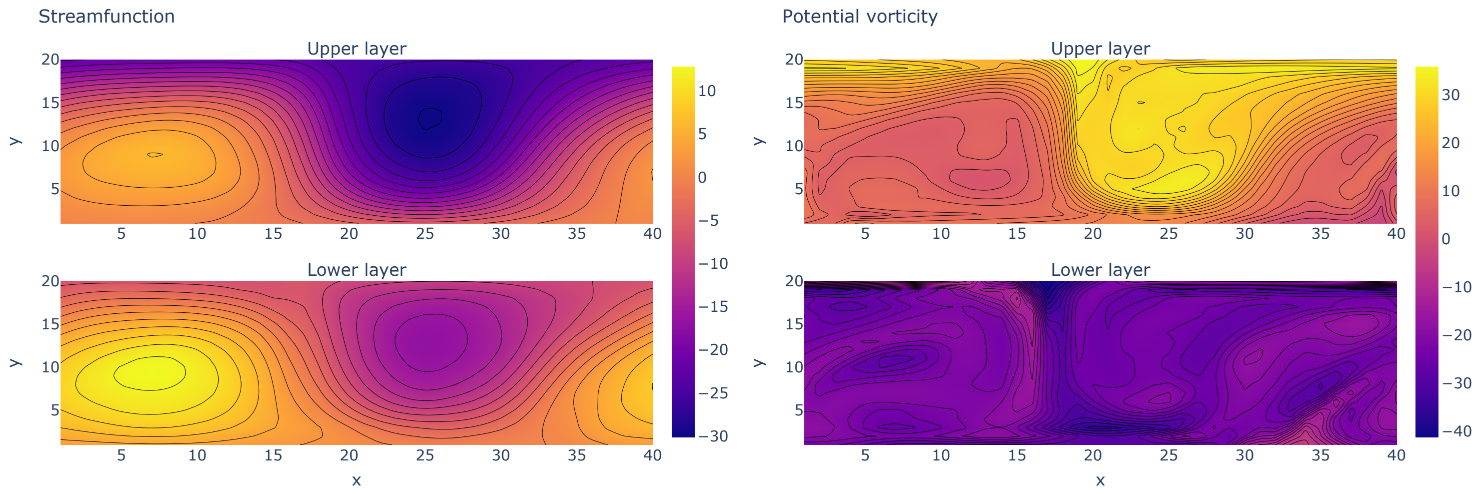


Figure 5.2.3: Left: contour lines of the streamfunction $\boldsymbol{\psi}$ in the upper and lower layers at day 20. Right: contour lines of the potential vorticity $\boldsymbol{q}$ in the upper and lower layers at day 20.

The initial state of the QG simulation for both the streamfunction and the potential vorticity consists of horizontally constant contour lines. The model is known to have a relaxation period of several days and day 16 is commonly considered as a physically meaningful day to start data assimilation with (Farchi *et al.*, 2021b). Figure 5.2.4 shows the initial state of the QG model, along with the streamfunction states at day 4 and day 16.

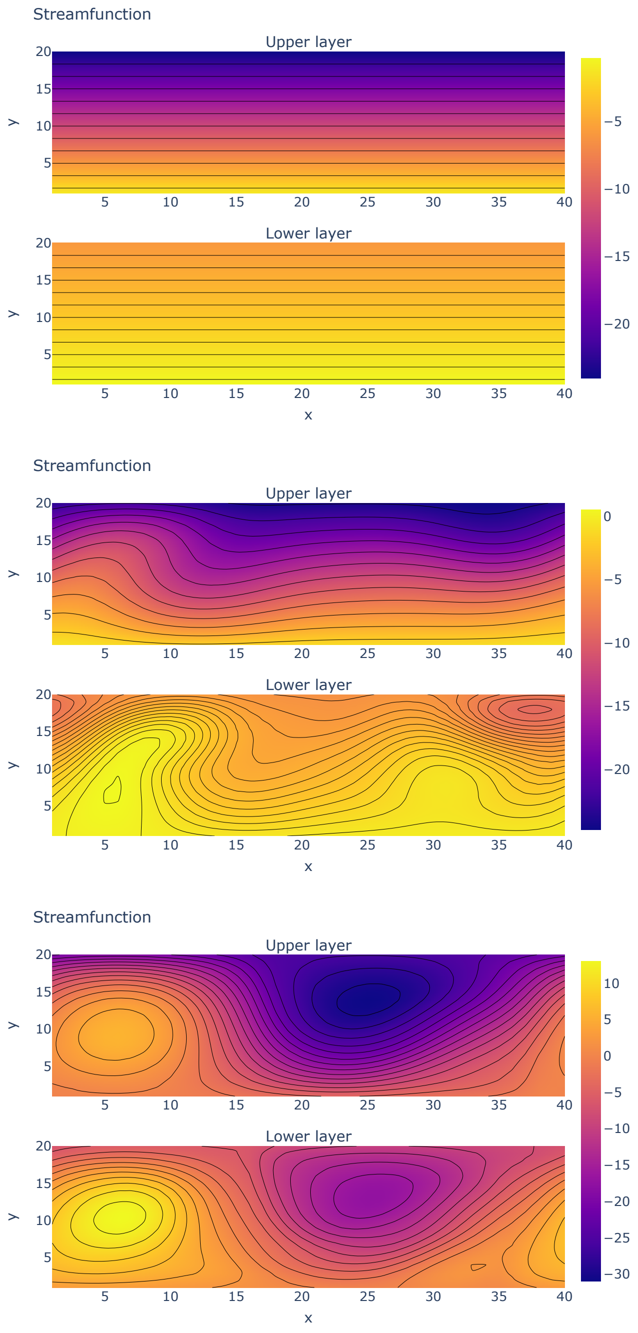


Figure 5.2.4: Top: contour lines of the initial streamfunction state for the upper and lower layers. Middle: contour lines of the streamfunction state at day 4 for the upper and lower layer, simulated from the initial state. Bottom: same as middle plot, but for the streamfunction state at day 16.

The QG model of the OOPS framework implements ensemble forecasts. Given an initial ground truth state and perturbation statistics, we can generate as many members as desired. This allows us to create our dataset for subsequently training our autoencoder and surrogate networks. However, we first need to determine appropriate values for the following parameters:

1. number of ensemble members. It must be large enough to accurately represent the variability of the perturbed initial state over the considered period.
2. temporal window. We need to ensure that the dynamics' statistics (*i.e.*, mean and standard deviation) indicate a stable regime

To determine the optimal number of ensemble members, we compute mean and standard deviation values over time, considering ensemble sizes ranging from 1 (a single vector) to 400. The total temporal window covers a 100-day period, starting from day 16 of the simulation (referred to as “day 0” in the legend) to day 116. We refer to day 16 as day 0 because it represents the first physically meaningful state.

We separate the computations of the mean and standard deviation values for the upper and lower layers. The results are reported in Figure 5.2.5: the x-axis indicates the ensemble size, while the legend helps visualize which curves correspond to which part of the time window. We observe that the standard deviation values stabilize with relatively few members for both layers, with about 25 to 30 samples being sufficient. However, the mean values require significantly more members to stabilize, with around 150 samples being necessary to achieve steady values. The number of ensemble members is also guided by deep learning requirements. To ensure a sufficiently large dataset for training, we therefore had to set the ensemble size to 400. This choice not only meets the deep learning needs but also ensures that we can faithfully represent the variability of the generated simulations.

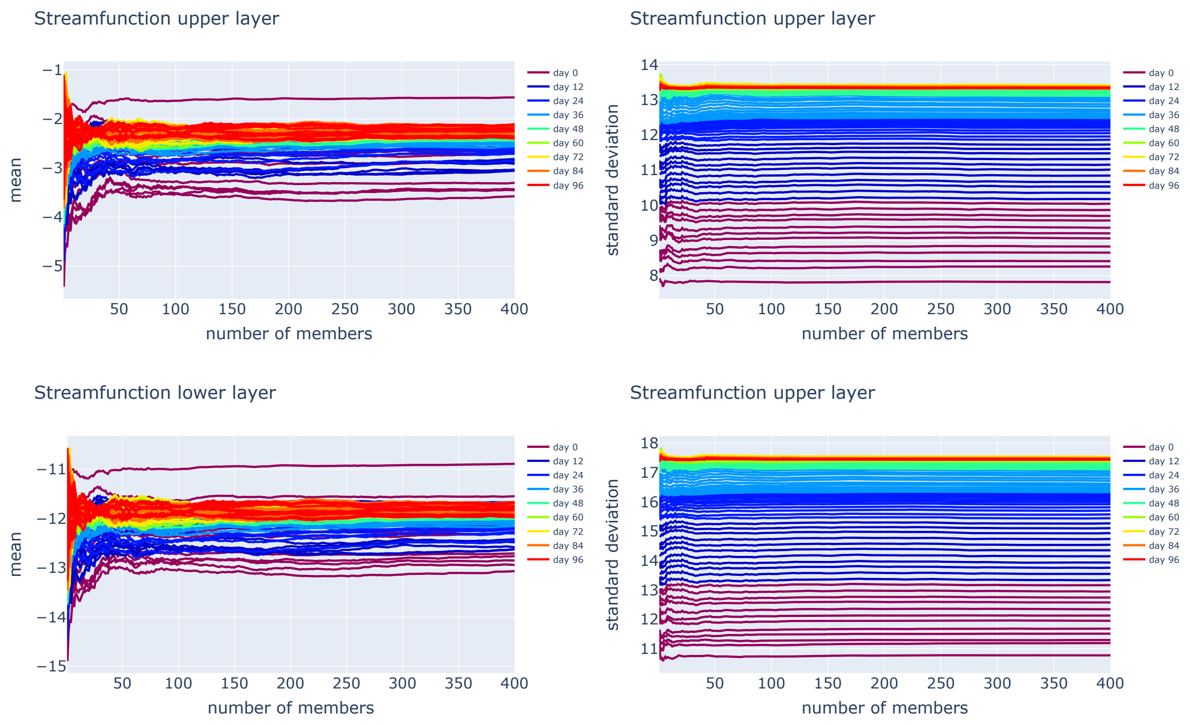


Figure 5.2.5: Top left: mean values for the upper layer with ensemble sizes ranging between 1 (a single vector) to 400, computed at different times over a 100-day period. Top right: standard deviation values for the upper layer with ensemble sizes ranging between 1 (a single vector) to 400, computed at different times over a 100-day period. Bottom left: same as top left, but for the lower layer. Bottom right: same as top right, but for the lower layer.

With the number of ensemble members set, we next need to select an appropriate time window to define our dataset. To do this, we compute the mean and standard deviation values for each layer over the 400-member ensemble and across the grid. These results are plotted with respect to time, as shown in Figure 5.2.6. In the top plots, we observe that the mean value for both layers reaches convergence earlier than the standard deviation, around day 60 for the former and day 80 for the latter. Since positive and negative values can sometimes offset each other, the mean over the grid might suggest stabilization while individual values can still vary significantly. To confirm our initial observations, the bottom plots show the same statistics, but computed in absolute values. Similar conclusions can be drawn, leading us to set the start of the temporal window at day 80. This ensures that the

QG dynamics is in a stable regime at this time. We chose a 10-day time window to provide a sufficient amount of data for training while ensuring the neural networks are exposed to and learn different behaviors of the QG dynamics.

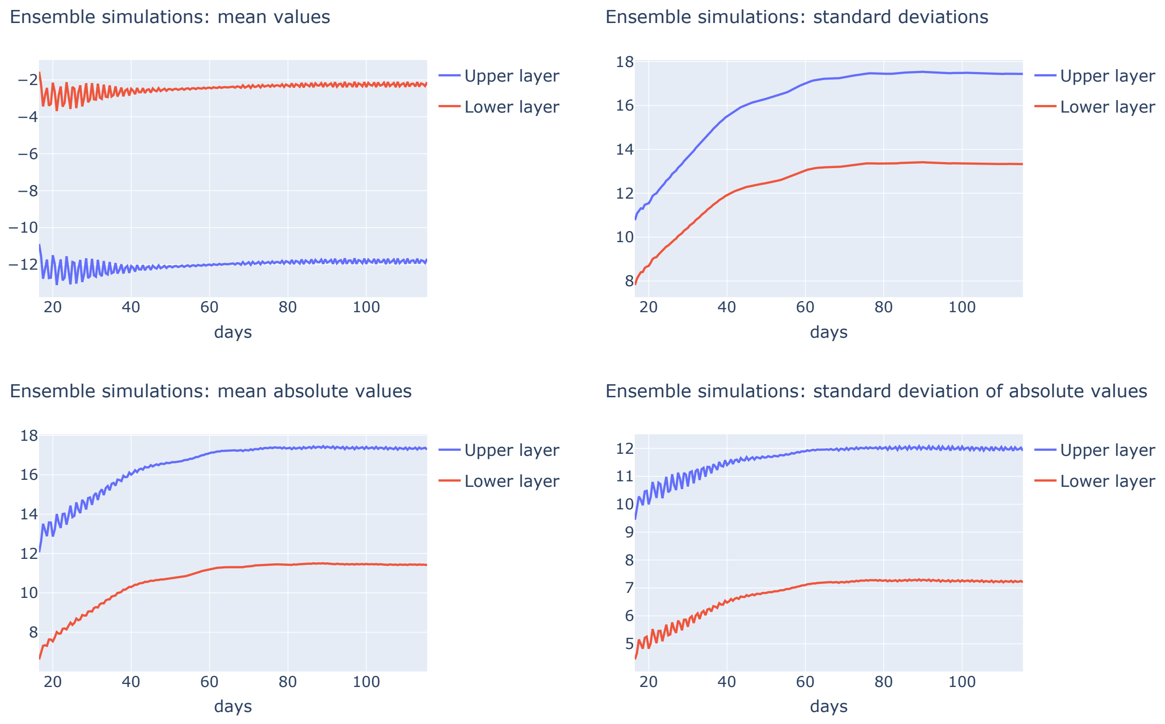


Figure 5.2.6: Top left: mean values for the upper layer with ensemble sizes ranging between 1 (a single vector) to 400, computed at different times over a 100-day period. Top right: standard deviation values for the upper layer with ensemble sizes ranging between 1 (a single vector) to 400, computed at different times over a 100-day period. Bottom left: same as top left, but for the lower layer. Bottom right: same as top right, but for the lower layer.

5.2.2 Determining a suitable latent dimension

As with the augmented Lorenz96 system (see section 5.1), we rely on PCA and Lyapunov stability theory to determine an appropriate latent space dimension for the two-layer QG model. However, unlike the augmented dynamics, where 40 was known to be a good candidate for the reduced-space dimension, we have no initial clue for our QG model. We only know from Figure 5.2.3 that the streamfunction exhibits low-frequency patterns, suggesting that PCA is likely to be very effective in reducing the 1,600 variables of the system.

We first perform PCA on our QG data, exploring latent dimensions ranging from $\ell = 5$ to $\ell = 200$ with increments of 5 within the interval $[\![5, 100]\!]$ and increments of 10 within $[\![120, 200]\!]$. Similarly to the augmented system, we also compare PCA with deep learning by training autoencoders over a common range of latent space dimensions. The encoder layers sequentially reduce the dimension from $\mathbb{R}^{1600}$ to $\mathbb{R}^{\ell}$ through an intermediary size of 800, using *Leaky ReLU* and identity activation functions. The choice of an almost fully linear autoencoder was guided by the highly linear nature of the QG dynamics (as demonstrated by PCA results in Figure 5.2.7), and the inability of more complex nonlinear autoencoders to outperform this simpler design.

Figure 5.2.7 shows the reconstruction performances of PCA and autoencoders in terms of MSE scores on the training and test datasets. We observe that PCA achieves impressively accurate reconstructions with very limited information from the latent space. For instance, the MSE with only 5 latent variables is about 0.05 and dramatically decreases to 0.005 when considering a latent space dimension of 10. From $\ell = 10$ up to $\ell = 200$, the MSE continues to decrease at a sustained pace, reaching 121e-6 with $\ell = 50$, 12e-6 with $\ell = 100$, and finally 689e-9 for a latent space of 200 dimensions. These figures confirm that the streamfunction dynamics are composed of a few low-frequency linear patterns that hold most of the total information, allowing the dynamics to be accurately recovered with only a few latent dimensions. Since the dynamics is highly linear, we also notice that the autoencoder can leverage its nonlinear capabilities when $\ell < 40$, outperforming PCA in this range. However, for larger latent dimensions, autoencoders cannot compress the information better than PCA. Based on this analysis and given the accurate reconstructions with very few latent variables, a latent dimension between 10 and 30 would be appropriate. More precisely, since autoencoders perform best at $\ell = 20$, this would be the ideal setting.

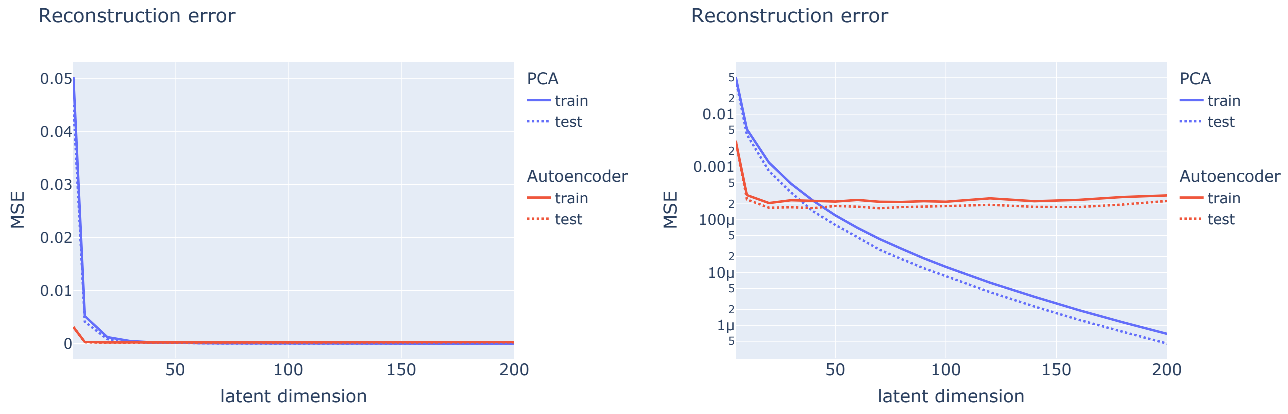


Figure 5.2.7: Left: MSE reconstruction curves over train (solid curve) and test (dashed curve) datasets for PCA (blue curve) and autoencoders (orange curve). Right: the same plot, but with a log-scale on the y-axis.

Lyapunov stability theory (see section 4.2.2) also provides valuable information to define a sufficiently large latent space, ensuring that the unstable-neutral subspace along with the attractors are well-represented. Mathematical details about the computation of the Lyapunov exponents are given in section 5.1.2.

Figure 5.2.8 shows the number of unstable dimensions as a function of the ensemble size. By considering a maximum ensemble size of 100, we can detect at most one hundred chaotic directions, here we notice that we only have three. In Figure 5.2.9, we plot the values of the first 100 Lyapunov exponents, in decreasing order. Although it is not apparent from the plot, we numerically find that the dimension of the neutral subspace is 0. Additionally, we computed the dimension of the attractor, which is 4.7. Consequently, this stability analysis suggests that an appropriate latent space dimension should be at least 8.

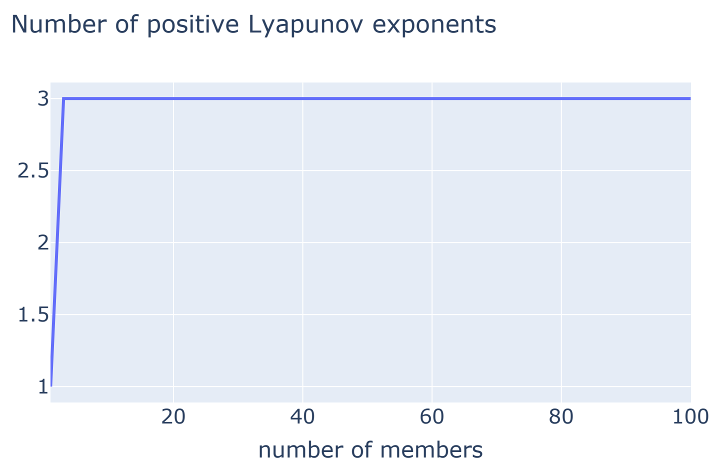


Figure 5.2.8: Number of positive Lyapunov exponents as a function of the number of members.

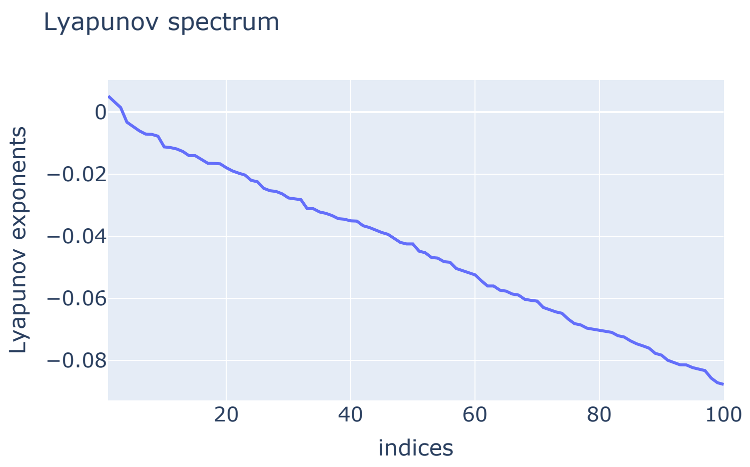


Figure 5.2.9: Lyapunov spectrum of our two-layer QG model.

In summary, combining insights from PCA and Lyapunov theory encourages us to consider a latent dimension between 10 and 30. Given that the trained autoencoders achieve the most accurate results at $\ell = 20$, this is our chosen dimension for performing latent space data assimilation.

5.2.3 Neural networks architectures and training setting

In this section, we present the neural network architectures of the autoencoder and the surrogate, and provide details about the training settings. The chained loss function we use for training is defined in equation (4.21). Similar to the augmented system, we have tuned numerous hyperparameters to find the neural architecture and training settings that yield the best results. These include the number of hidden layers, choice of activation functions, data normalization, batch size, noise magnitude, learning rate, the loss weighting parameter ρ, and the number of forward steps C.

A single element from our dataset corresponds to the streamfunction at a specific time for a particular ensemble member. This element can be plotted over two 2D grids, as shown in Figure 5.2.4, and is numerically represented by a 5-dimensional tensor. The default format of this tensor's shape within the OOPS framework is (nl, ny, nx), where nl denotes the number of layers (here 2), ny the number of grid points along the y-axis (here 20), and nx those along the x-axis (here 40). Since convolutional networks are often preferred over traditional MLPs when learning from images or multi-dimensional data, they are a good candidate for our dataset. While convolutional autoencoders can be directly fed the input streamfunction data, feed-forward neural networks first require a flattening layer to convert the input tensor of shape (2, 20, 40) into a one-dimensional vector of size 1,600. After training both types of autoencoders, results turned out to be better with the MLP.

The encoder consists of two layers: the first compresses the 1,600 input variables into 800, followed by a 0.5 slope *Leaky ReLU* activation, while the second layer maps from $\mathbb{R}^{800}$ to $\mathbb{R}^{20}$ with no activation function. The decoder performs the reverse operation. Several variants have been tested, and this configuration yielded the best results. Figure 5.2.10 depicts the autoencoder architecture.

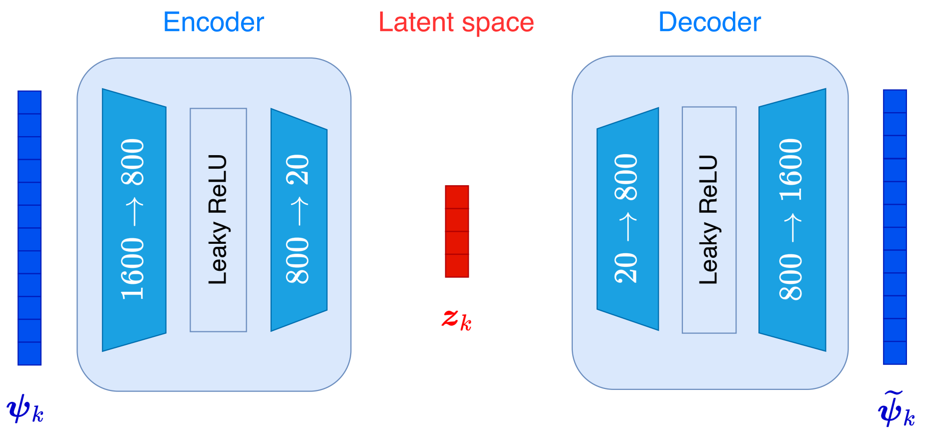


Figure 5.2.10: Autoencoder architecture used for numerical experiments: $\psi_k \in \mathbb{R}^{1600}$ refers to the input vector, $z_k \in \mathbb{R}^{20}$ to the latent variable, and $\widetilde{\psi}_k \in \mathbb{R}^{1600}$ to the reconstructed vector. Trapeziums denote dense layers, while rectangles represent activation functions.

The surrogate network is composed of six fully connected layers, each mapping from $\mathbb{R}^{20}$ to $\mathbb{R}^{20}$. The first four layers are followed by a *Tanh* activation, the fifth by a 0.5 slope *Leaky ReLU* activation, and the last layer has no activation. Similar to the augmented Lorenz96 system, our surrogate also leverages the residual block structure (Bachlechner *et al.*, 2021), as detailed in Figure 5.2.11.

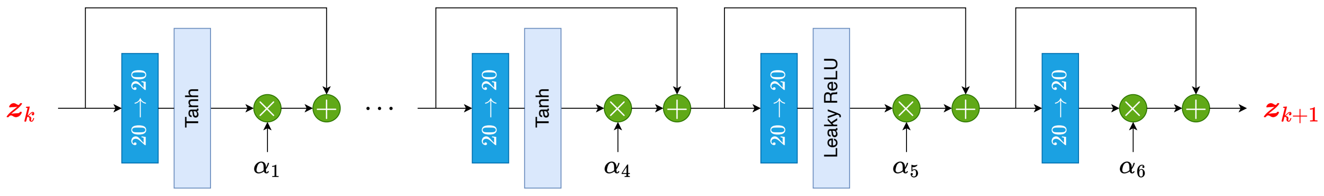


Figure 5.2.11: Our surrogate network is composed of six residual blocks mapping from $\mathbb{R}^{20}$ to $\mathbb{R}^{20}$. Latent variable z_k is the input, and z_{k+1} refers to the latent vector at time t_{k+1} yielded by the network. Also, we have that $\alpha_i \in \mathbb{R}^{20}$, $\forall i \in [\![1, 6]\!]$. Multiplicative and additive operators are also represented. For the second and third layer, the activation functions is also *Tanh*.

We remind that our dataset consists of 400 simulations generated with OOPS, each covering a 10-day time window with a 2-hour interval between successive snapshots. This represents a total of 120 time steps. Consequently, we have 400 instances of $120 \times 2 \times 20 \times 40$ matrices to train our neural networks.

The dataset is normalized using mean and standard deviation normalization and then split into 80% for training, with the remaining 20% equally divided between validation and testing. During the training stage, white Gaussian noise with a magnitude of 0.01 is added to the input data. The batch size is set to 16, and we use the Adam optimizer with a learning rate of 10^{-4}. Unlike the augmented Lorenz system, for which we defined an early stopping criterion, we set the number of epochs to 300 for the QG model. This choice is motivated by the fact that training the QG model takes more time than training the Lorenz model, and using early stopping can lead to a significantly different number of effective epochs between two networks, making it difficult to draw fair comparisons. Network weights are saved each time a lower loss score on the validation set is reached. The weighting parameter ρ of our custom loss function defined in equation (4.21) is set to 0.5, and the number of iterations C is set to 2.

Training the autoencoder and surrogate, as defined earlier, involves optimizing a total of 2,597,740 parameters. The training process is completed in approximately 3 hours and 56 minutes on a 2021 MacBook Pro equipped with the Apple M1 Pro chip. Figure 5.2.12 shows the training and validation loss curves, along with the test loss value. We remind that the test MSE is a single value computed at the end of the training on the test dataset, after loading the best model's weights. While the test score is not related to the training phase, it is represented alongside the training and validation curves for visibility.

The loss values in Figure 5.2.12 are computed according to equation (4.21). We observe that the loss scores decrease by a factor of about 100 between the first epoch and the end of the training. The learning process appears to be somewhat erratic, with small deviations from the current optimal weights leading to significant changes in the loss scores. However, this does not prevent the network from performing its task increasingly better over the course of the training, ultimately achieving impressively low MSE scores.

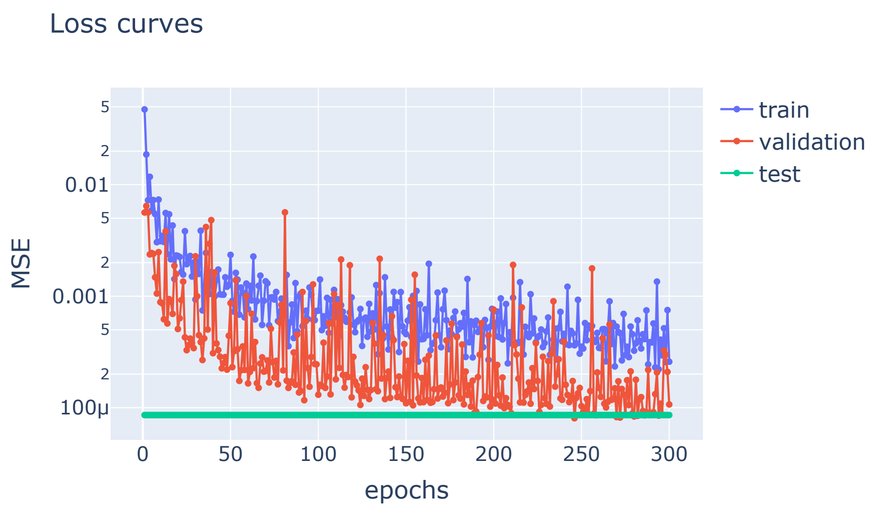
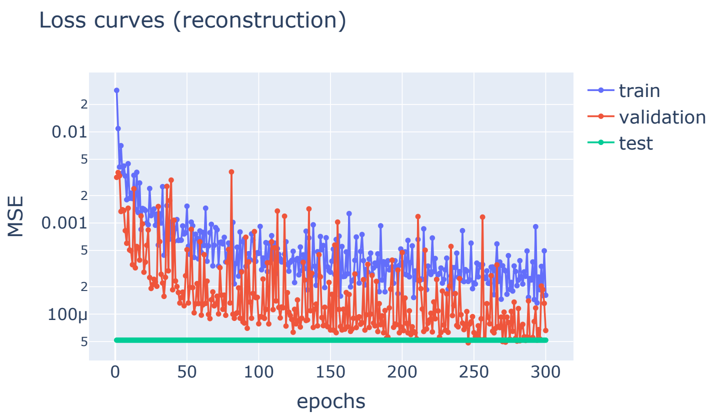


Figure 5.2.12: Loss values (see equation (4.21)). We remind that the score on the test dataset is repeated along the x-axis with a dotted line for visual convenience.

Figure 5.2.13 reports the reconstruction loss as expressed in equation (4.22) (left plot) and the chained loss given by equation (4.23) (right plot). These plots confirm that the autoencoder and surrogate properly learn, despite the rapid changes in loss scores over a few successive epochs.



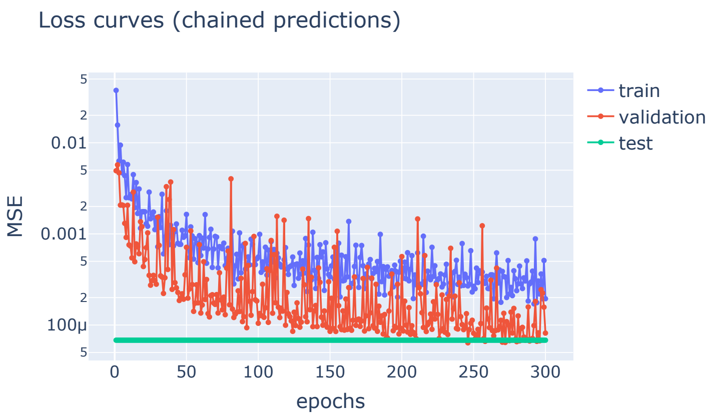


Figure 5.2.13: Left: reconstruction loss (see equation (4.22)) with a log-scale on the y-axis. Right: chained loss (see equation (4.23)) with a log-scale on the y-axis. We remind that the score on the test dataset is repeated along the x-axis with a dotted line for visual convenience.

The best model is saved at epoch 246, and achieves the following loss scores (rounded):

- validation loss score: 8.04e-05
- test loss score: 8.59e-05
- validation reconstruction loss score: 4.84e-05
- test reconstruction loss score: 5.18e-05
- validation chained loss score: 6.38e-05
- test chained loss score: 6.82e-05

At this stage of the analysis, we know that the autoencoder and the surrogate have successfully learned their respective tasks. In the following section, we provide insights into the performance of our trained neural network. The loss scores on the training, validation, and test sets suggest that the network effectively represents the input data within a 20-dimensional latent space and accurately propagates within it. However, we are interested to know how this loss scores visually translate in terms of reconstructed two-layer images and variables.

To do this, we consider a full simulation from the test dataset and apply the autoencoder to the 120 states it comprises. We then compute the absolute differences between the reconstruction and the original data. The results are shown in Figure 5.2.14: the top plot represents the ground truth, the middle plot shows the reconstruction obtained with the autoencoder, and the bottom image plots the computed absolute differences. We observe that the streamfunction is reconstructed so accurately that it is visually impossible to discern any differences between the two top images. This is confirmed by the bottom plot, where errors range from approximately 0 to about 0.02, which is very low compared to the magnitude of the streamfunction values. Therefore, the original information contained in a 1,600-dimensional space can be almost exactly represented in a latent space consisting of only 20 variables, representing a reduction by a factor of 80.

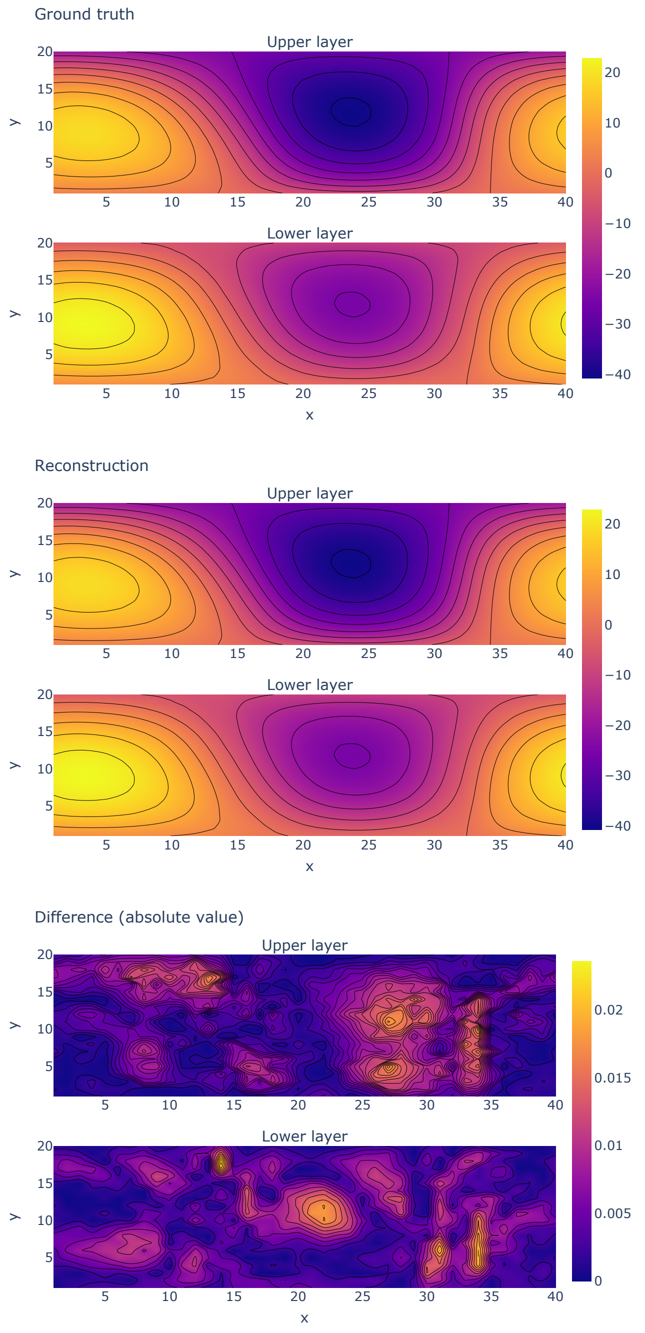


Figure 5.2.14: Top: contour lines of the streamfunction state for the upper and lower layers at day 85 of a simulation from the test dataset. Middle: reconstruction of the top image with the trained autoencoder. Bottom: differences in absolute value between the top and middle plot.

While Figure 5.2.14 demonstrates the impressive performance of the autoencoder, we also want to examine how this accuracy holds up when focusing on a subset of variables. To do this, we randomly select 5 variables out of the 1,600 and plot their reconstructions. Additionally, we assess the performance of the surrogate by plotting the reconstruction of these variables after one latent propagation, meaning both the autoencoder and surrogate's performances are evaluated. The results are shown in Figure 5.2.15. Notably, we observe a perfect superposition of the reconstruction curves and ground truth trajectories for both the regular reconstruction and the reconstruction after one latent propagation. Unlike the augmented Lorenz96 system, where small discrepancies could be discerned between the solid and dashed lines, here, they perfectly coincide. Another remarkable observation is the visual similarity of the two plots: for these 5 variables in this test dataset simulation, no differences can be detected between the two reconstruction metrics. This indicates that the surrogate does not introduce any visually detectable errors.

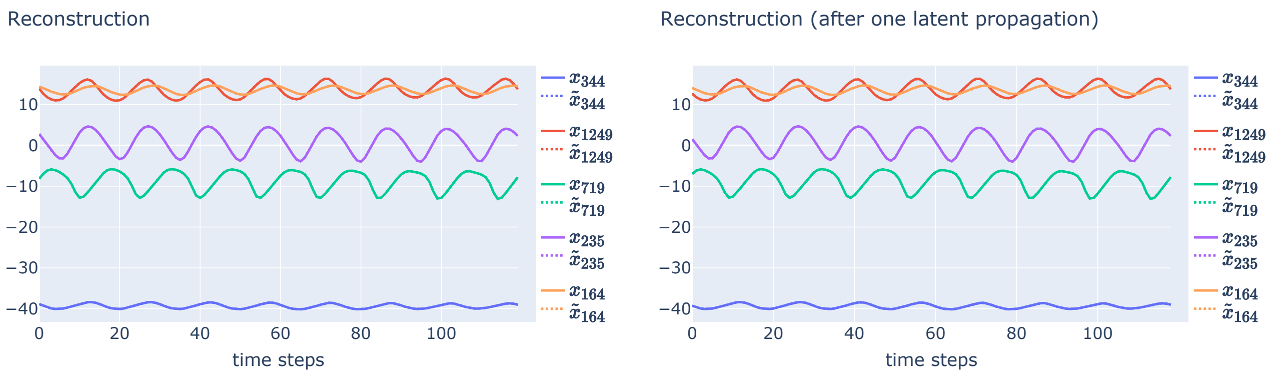


Figure 5.2.15: Left: reconstruction of five out of 1,600 variables from a simulation in the test dataset (the same one as in Figure 5.2.14) using only the autoencoder. Right: the same reconstruction as in the left plot, but with one-step latent propagation also performed. Ground truths are represented by solid lines, and reconstructions with dashed lines.

We are also interested in how the autoencoder represents the latent information in a 20-dimensional space. Figure 5.2.16 shows the latent representations of two simulations from the test dataset. Remarkably, each latent variable seems to represent a pattern that cycles over time. At any given time t_k, the right combination of these cycling patterns allows $\boldsymbol{z}_k$ to faithfully represent $\boldsymbol{\psi}_k$ with only 20 variables. Since the autoencoder has two layers but only one activation function, a

0.5 slope *Leaky ReLU*, it performs an almost linear transformation when encoding and decoding data. Thus, the information contained within the latent space and represented in Figure 5.2.16 almost corresponds to a linear decomposition of the full space data, similar to what PCA does. This is a distinctive feature of our autoencoder applied to the QG model, which is not shared by the latent space obtained in the augmented Lorenz system with a deeper and more nonlinear autoencoder.

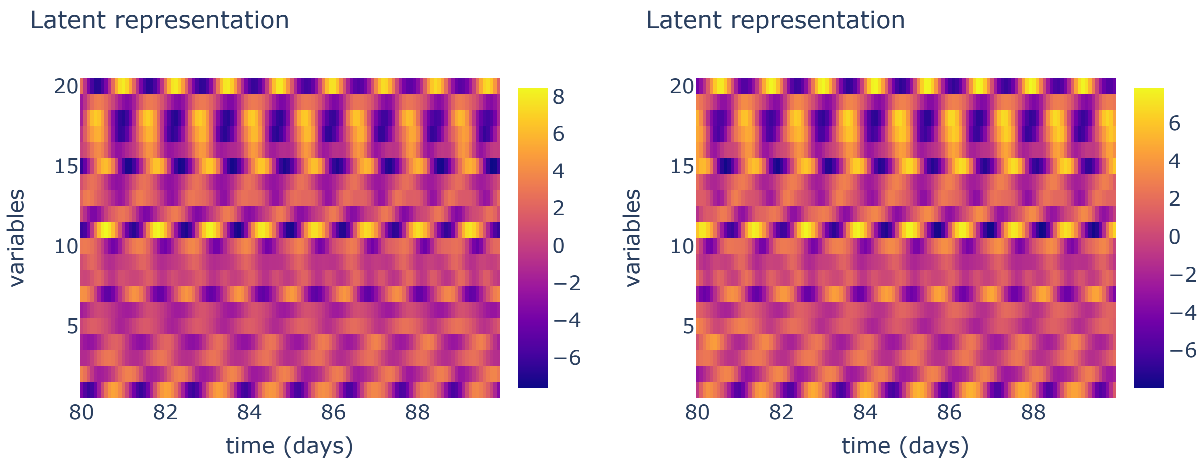


Figure 5.2.16: Left: latent space representation of an entire simulation from the test dataset. Right: another latent space representation from the test dataset with the same network.

We also provide the latent spaces of the same simulation produced when combining the autoencoder with a 1-Lipschitz surrogate (see Figure 5.2.17) and with a ext-Lipschitz surrogate (see Figure 5.2.18). We remind that we refer to an extended Lipschitz network (abbreviated as ext-Lipschitz) as a data-driven model in which all layers are 1-Lipschitz, except for the final layer, which remains unconstrained. Unlike the augmented Lorenz96 system, where introducing Lipschitz networks resulted in different latent space representations characterized by one or more constant variables over time, here, we cannot visually detect any differences resulting from using a Lipschitz surrogate during the training.

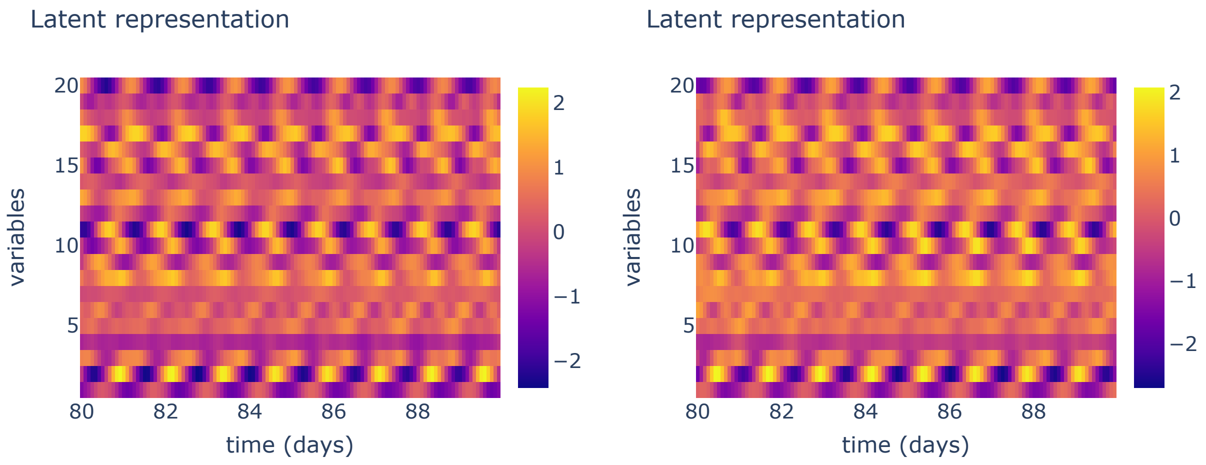


Figure 5.2.17: Left: latent space representation of an entire simulation from the test dataset obtained with our trained 1-Lipschitz network (later introduced in this section). Right: another latent space representation from the test dataset with the same network.

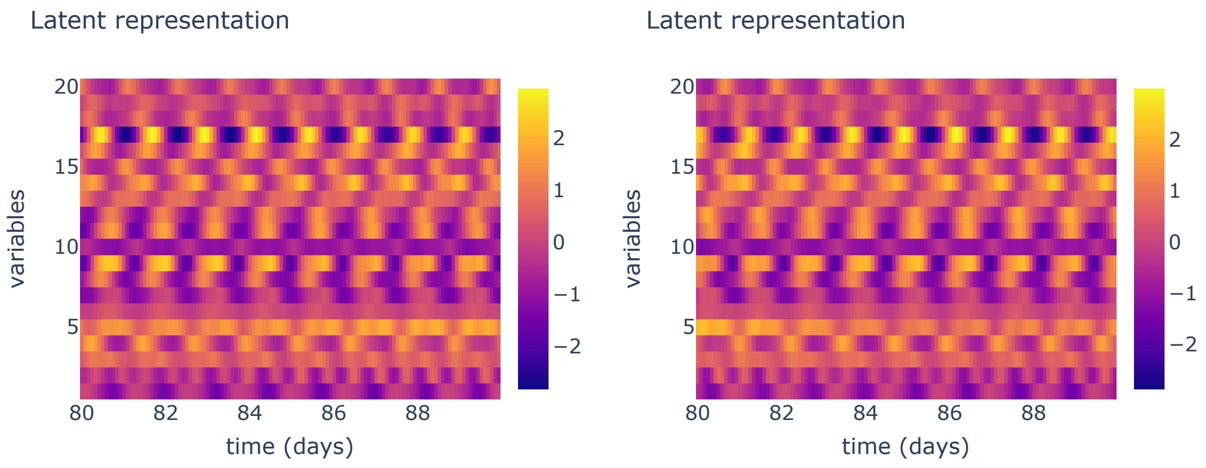


Figure 5.2.18: Left: latent space representation of an entire simulation from the test dataset obtained with our trained ext-Lipschitz network (later introduced in this section). Right: another latent space representation from the test dataset with the same network.

5.2.4 Stability of the surrogate network

Ensuring the stability of the surrogate neural network over time is crucial, particularly for medium-range and long-term predictions. In this section, we compare the stability of four autoencoders, each one being trained with a different surrogate network: either the one-step, two-step, 1-Lipschitz, or ext-Lipschitz neural network. The one-step and two-step networks are defined in section 5.1.3, while the Lipschitz networks are introduced in section 4.2.3.1. We remind that the one-step network refers to the surrogate trained with the parameter C of equation (4.23) set to 1 (i.e., no iterative training), whereas the two-step network is trained with C set to 2, thereby implementing a chained loss function.

We first compare the reconstruction, chained, and total loss scores for three different network combinations: the autoencoder with the one-step surrogate, the autoencoder with the 1-Lipschitz network, and the autoencoder with the ext-Lipschitz network. The corresponding figures are provided in table 5.7. It is important to note that we cannot include the loss scores for the autoencoder paired with the two-step surrogate in this comparison, as the parameters used—specifically, setting C to 2 and the loss parameter ρ to 0.5 (compared to $\rho = 1$ when $C = 1$)—make a direct comparison unfair. For all networks, the loss scores on both the validation and test datasets are quite similar. While the training results for the 1-Lipschitz and ext-Lipschitz networks yield comparable loss scores, the experiment with the one-step network achieves values that are approximately five times smaller, making it the most effective combination among the three.

Dataset	Loss type	One-step network	1-Lipschitz network	ext-Lipschitz
Validation	reconstruction	57.25e-6	195.98e-6	168.67e-6
	chained	61.88e-6	391.05e-6	414.1e-6
	total	119.13e-6	587.03e-6	582.80e-6
Test	reconstruction	61.32e-6	198.07e-6	170.52e-6
	chained	65.77e-6	374.96e-6	412.01e-6
	total	127.10e-6	573.03e-6	582.53e-6

Table 5.7: Loss scores of trainings involving the one-step, 1-Lipschitz and ext-Lipschitz neural networks over the validation and test datasets.

The chaotic nature of the augmented Lorenz96 system necessitates either an iterative training strategy or the use of Lipschitz networks to ensure the stability of the surrogate (as discussed in section 4.2.3.1). In contrast, the QG model exhibits significantly less chaotic behavior, with only 3 chaotic directions out of 1,600 variables, compared to 13 out of 40 for the augmented Lorenz system. Consequently, the one-step surrogate network for the QG model already demonstrates stability over the 120 time steps of the dataset, as we will show.

We begin by briefly evaluating the reconstruction performances of the four autoencoders, as we also did in section 5.1.3. We select a simulation from the test dataset and plot the absolute differences between the ground truth state and the reconstructions produced by each of the autoencoders. Since one state at a given time is represented as a 2D figure, we cannot display the entire simulation. We therefore focus on the state at day 85, which lies in the middle of the time window. The results are shown in Figure 5.2.19.

The colorbars in the plots indicate that the reconstruction error ranges are quite similar for the autoencoders associated with the one-step, two-step, and 1-Lipschitz neural networks, with minimum values around 0.005 and maximum values between 0.02 and 0.04, depending on the specific plot. The reconstruction errors produced by the autoencoder associated with the ext-Lipschitz network range from about 0.01 to 0.06, which are higher than those of the other three autoencoders. Visually, the experiment involving the two-step network appears to produce smaller errors across both the upper and lower layers, followed closely by the one-step surrogate experiment. Despite the higher maximum values in its colorbar, the ext-Lipschitz surrogate seems to induce fewer significant reconstruction errors than the 1-Lipschitz network, as evidenced by the darker areas observed in the bottom right plot compared to the bottom left plot.

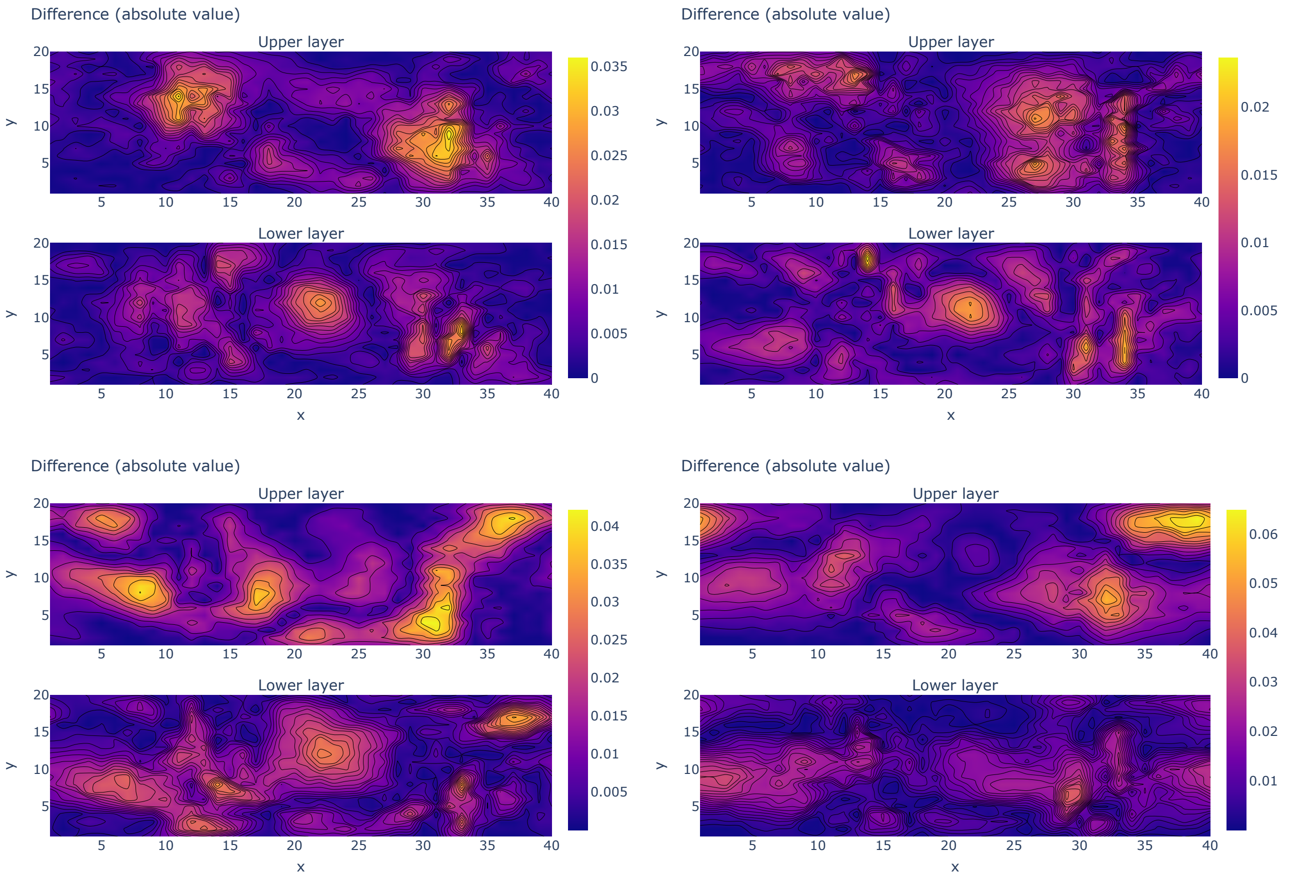


Figure 5.2.19: Top left: contour lines of the differences in absolute values between the ground truth state of a test dataset simulation at day 85 and its reconstruction by the autoencoder trained with the one-step surrogate network. Top right: same as top left, but for the autoencoder trained with the two-step surrogate network. Bottom left: same as top left, but for the autoencoder trained with the 1-Lipschitz surrogate network. Bottom right: same as top left, but for the autoencoder trained with the ext-Lipschitz surrogate network.

We now assess the performance of the four surrogate networks by evaluating the full temporal reconstruction of a simulation from the test dataset (the same simulation referenced in Figure 5.2.19). Starting from the initial state $\boldsymbol{\psi}_0$, we encode it into $\boldsymbol{z}_0$ for each surrogate network and then propagate it forward in time across 120 steps, generating latent variables $\boldsymbol{z}_1, \boldsymbol{z}_2, \ldots, \boldsymbol{z}_{119}$. These are subsequently decoded into the reconstructed states $\widetilde{\boldsymbol{\psi}}_1, \widetilde{\boldsymbol{\psi}}_2, \ldots, \widetilde{\boldsymbol{\psi}}_{119}$. We then compute the RMSE between the ground truth states and their reconstructed counterparts at each time step. The resulting temporal RMSE curves for the one-step, two-step, 1-Lipschitz, and ext-Lipschitz surrogate networks are shown in Figure 5.2.20. The two-step neural network exhibits the best performance, closely followed by the one-step surrogate, with both networks delivering stable and accurate forecasts over time. In contrast, the 1-Lipschitz and ext-Lipschitz surrogate networks diverge rapidly from the ground truth, with RMSE values oscillating between approximately 0.8 and 1.8 after the first 10 epochs. Despite their lower accuracy, the Lipschitz networks demonstrate a strong capacity to remain stable over time, a characteristic already verified in the augmented Lorenz96 system (see section 5.1.3).

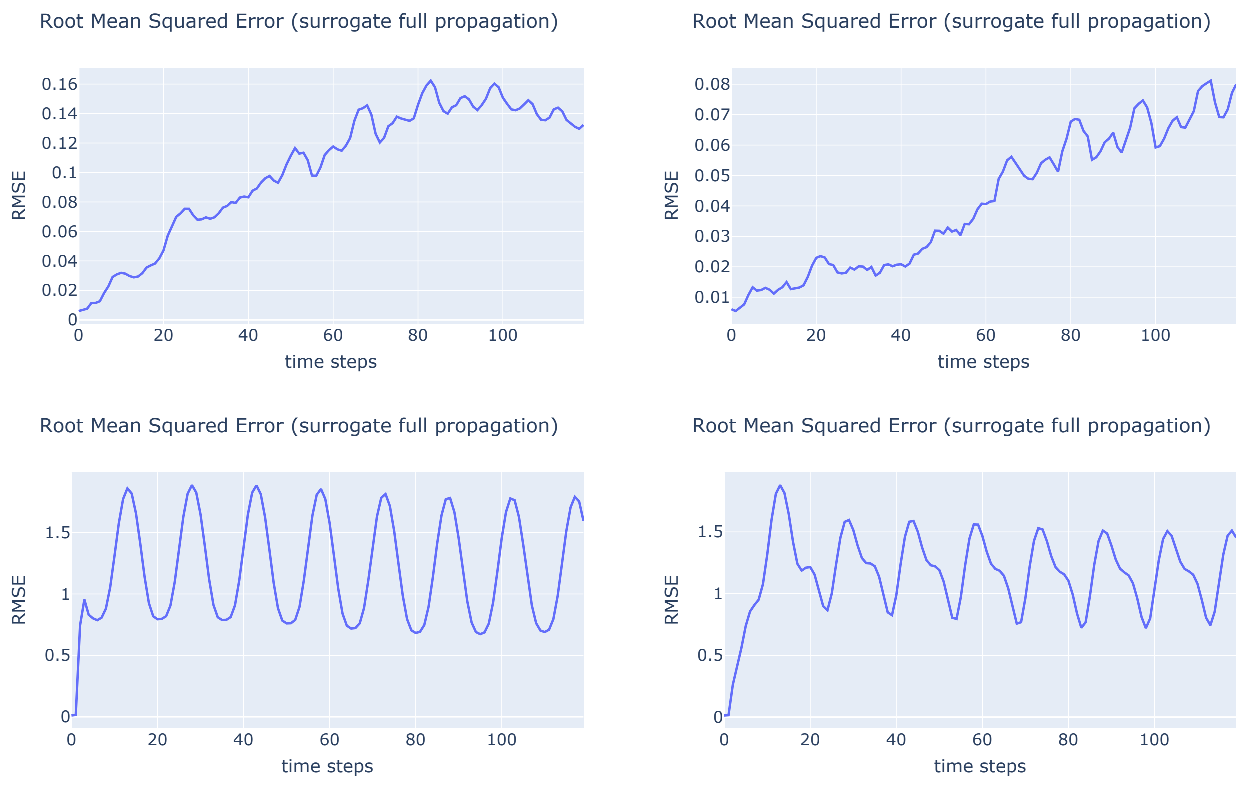


Figure 5.2.20: Top left: RMSE between a ground truth simulation from the training dataset and its full reconstruction by the one-step trained neural network over 120 time steps. Top right: same as top left, but for the experiment involving the two-step surrogate network. Bottom left: same as top left, but for the experiment involving the 1-Lipschitz surrogate network. Bottom right: same as top left, but for the experiment involving the ext-Lipschitz surrogate network.

To gain a deeper understanding of the stability and accuracy of the different surrogate networks, we closely examine the full temporal reconstructions of five selected variables out of the 1,600 total. The reconstruction curves are presented in Figure 5.2.21. The two-step surrogate network clearly outperforms the other data-driven models, with reconstruction curves that nearly perfectly align with the ground truth values. In the first half of the simulation, the dashed lines and solid lines coincide exactly, while in the second half, only very slight differences can be discerned between them. The one-step surrogate network also delivers highly accurate predictions, nearly matching the performance of the two-step network. As indicated by Figure 5.2.20, the Lipschitz neural networks, although not capable of making medium-range or long-term accurate predictions, maintain stability over time by predicting nearly constant values throughout the simulation, which explains the periodic nature of the RMSE curves in Figure 5.2.20

Overall, the analysis of the results shown in Figures 5.2.19 to 5.2.21 suggests that all four surrogate networks are capable of maintaining stability over the 120 time steps of the simulation. However, only the one-step and two-step networks demonstrate the ability to provide accurate long-term predictions.

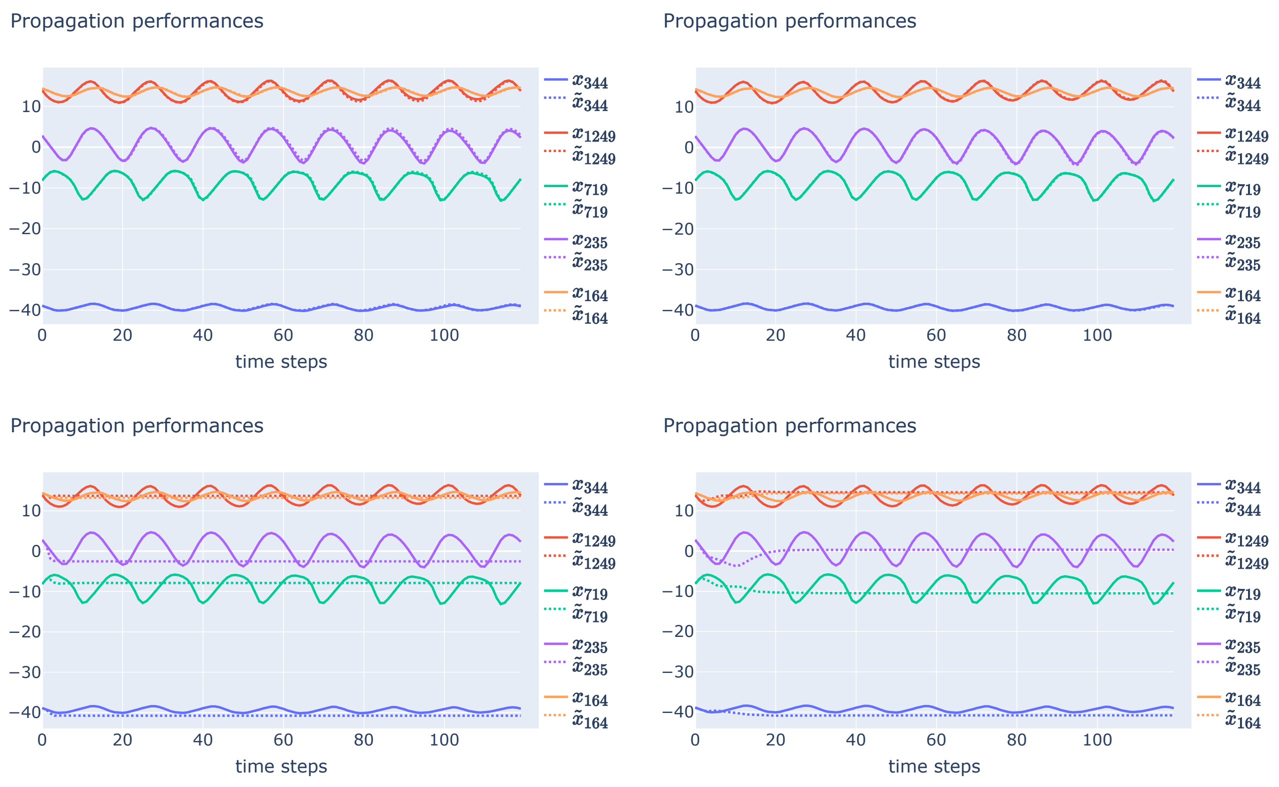


Figure 5.2.21: Top left: full temporal reconstruction of five out of 1,600 variables from a simulation in the test dataset (the same as one as in Figure 5.2.19) using the autoencoder and the one-step surrogate network. Top right: same as top left, but for the experiment involving the two-step surrogate network. Bottom left: same as top left, but for the experiment involving the 1-Lipschitz surrogate network. Bottom right: same as top left, but for the experiment involving the ext-Lipschitz surrogate network. Ground truths are represented with solid lines, and reconstructions with dashed lines.

Despite the insightful information provided by Figures 5.2.19 to 5.2.21, we cannot draw global and definitive conclusions about the behavior of the surrogate networks from these plots. Therefore, we consider RMSE statistics over both the training and test datasets rather than selecting one simulation from each. We remind our strategy introduced in section 5.1.3:

1. Compute the RMSE scores for every simulation in both the training and test datasets.

2. For each RMSE time series, compute its mean temporal value.

3. For each mean temporal value, compute its base 10 logarithm.

4. Plot the RMSE distributions (in percentage) of the base 10 logarithm values using bar charts.

The results shown in Figure 5.2.22 corroborate the preliminary conclusions drawn from Figures 5.2.19 to 5.2.21, confirming that the two-step network is the most stable and accurate data-driven model for long-term predictions, closely followed by the one-step surrogate, which exhibits comparable performance. Both the two-step and one-step networks significantly outperform the Lipschitz neural networks, which, despite their lower accuracy, still demonstrate strong stability and a narrow spread in their mean RMSE values. Additionally, in the augmented Lorenz96 system, the one-step neural network exhibits complete instability, whereas in this context, it demonstrates the opposite behavior. This difference may be attributed to the significantly less chaotic nature of the QG model compared to the augmented Lorenz dynamics, as well as the difference in the number of time steps in the reconstruction plots (500 for the augmented Lorenz96 versus 120 for the QG model). Across all the networks, the mean RMSE scores are consistent between the training and test datasets, though the training set may include outlier simulations that increase the spread of the distribution.

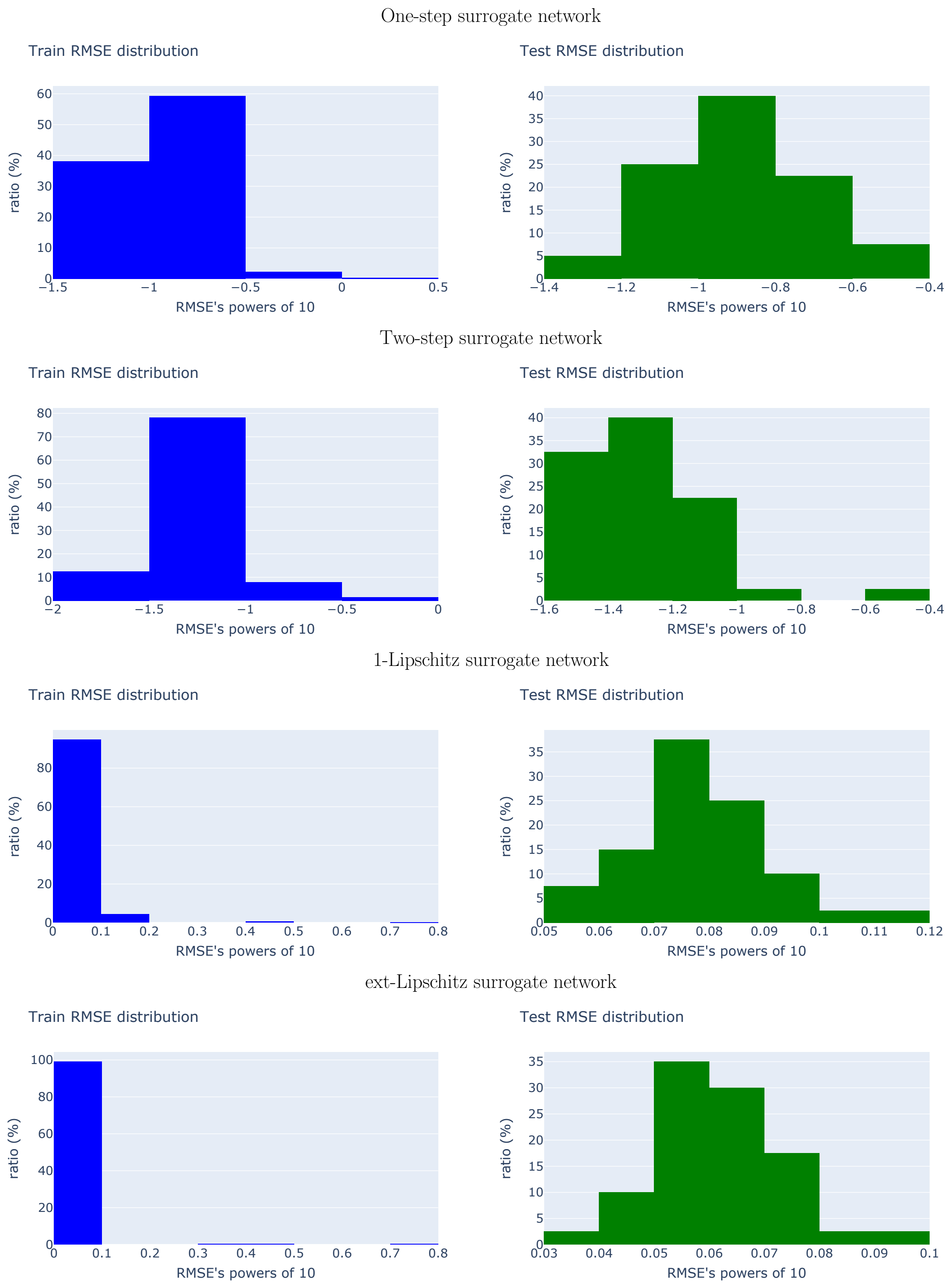


Figure 5.2.22: Distributions of the mean RMSE values when fully reconstructing the entire training (left column plots) and test (right column plots) datasets simulations over 120 times steps, with different surrogate networks.

5.2.5 Data assimilation results

After presenting the quasi-geostrophic model, the neural network architectures, and the stability of the trained models, we now assess the performance of the ETKF-Q algorithm (see section 4.3) in a benchmark context similar to that of the augmented Lorenz96 system discussed in section 5.1.4. The comparison metric we use is also the mean temporal RMSE, computed in the physical space of dimension 1,600. The naming conventions and definitions for the QG experiments are those provided for the augmented dynamics in section 5.1.4, with one exception: the definition of the ETKF-Q experiment is adjusted to account for the use of the QG model for state propagation, instead of the Lorenz96 equations to solve. The revised definition is as follows:

- **ETKF-Q:** this is the standard ETKF-Q algorithm (Fillion *et al.*, 2020). Assimilation is conducted within the 1,600-dimensional physical space of the QG dynamics, without involving any neural network. Temporal propagation is performed using the regular model from the OOPS package.

Latent data assimilation experiments are therefore also denoted by ETKF-Q-L, ETKF-Q-L-1-Lipschitz, ETKF-Q-L-ext-Lipschitz, ETKF-Q-L-PCA and ETKF-Q-L-PCA-LinReg, as summarized in table 5.2 for the augmented Lorenz96 system.

We also define their physical-latent data assimilation counterparts, with the same naming convention as that in section 5.1.4. For each experiment involving latent data assimilation, a corresponding experiment is defined where the assimilation is performed in the physical space using the regular ETKF-Q algorithm, while the time propagation occurs in the latent space. Further details are provided in section 5.1.4. Information on the physical-latent experiments is summarized in table 5.8.

We also remind that in all experiments utilizing PCA as a reduced-space method, the fitting is conducted on the training dataset. Similarly, the linear regression operator is calibrated to represent the mapping between successive latent variables that are obtained by a PCA transformation.

To assess the performance of each method, we define the following data assimilation setting:

- **20** ensemble members.

Experiment name	Data assimilation space	Propagation operator
ETKF-Q	Physical space	QG model
ETKF-Q-P-L	Physical space	$\mathcal{E} \circ \mathcal{S} \circ \mathcal{D}$
ETKF-Q-P-L-1-step	Physical space	$\mathcal{E}_{\text{1-step}} \circ \mathcal{S}_{\text{1-step}} \circ \mathcal{D}_{\text{1-step}}$
ETKF-Q-P-L-1-Lip	Physical space	$\mathcal{E}_{\text{1-Lip}} \circ \mathcal{S}_{\text{1-Lip}} \circ \mathcal{D}_{\text{1-Lip}}$
ETKF-Q-P-L-ext-Lip	Physical space	$\mathcal{E}_{\text{ext-Lip}} \circ \mathcal{S}_{\text{ext-Lip}} \circ \mathcal{D}_{\text{ext-Lip}}$
ETKF-Q-P-L-PCA	Physical space	$\text{PCA} \circ \mathcal{S}_{\text{PCA}} \circ \text{PCA}^{-1}$
ETKF-Q-P-L-PCA-LinReg	Physical space	$\text{PCA} \circ \text{LinReg} \circ \text{PCA}^{-1}$

Table 5.8: Benchmark experiments where assimilation is performed within the physical space, *i.e.*, within the 1,600-dimensional space of the QG dynamics. Subscript notation for the encoder, surrogate, and decoder distinguishes between the different networks. Plain notations $\mathcal{E}, \mathcal{S}, \mathcal{D}$ are used exclusively for our ETKF-Q-L method, denoting the two-step network's components, as described in section 5.2.3 and section 5.2.4.

- **48 cycles** (equivalent to a 4-day analysis).
- observation operator $\boldsymbol{\mathcal{H}}$ is the identity matrix, i.e., $\boldsymbol{I}_{1600}$.
- observation error covariance matrix $\boldsymbol{R} = \boldsymbol{\sigma}_R^2 \boldsymbol{I}_{1600}$, with $\boldsymbol{\sigma}_R = 0.4$. This matrix is used at every cycle.
- background error covariance matrix $\boldsymbol{B} = \boldsymbol{\sigma}_B^2 \boldsymbol{I}_{1600}$, with $\boldsymbol{\sigma}_B = 0.8$.

For the ETKF-Q experiment, we also define a model error covariance matrix, denoted by $\boldsymbol{Q}$ (see section 2.2.2.3). Here, $\boldsymbol{Q}$ is set as a diagonal matrix weighted by variance σ_Q^2 such that $\sigma_Q = 0.1$.

Similar to the approach used for the augmented system, the inflation parameter λ (see section 2.2.2.2) and the model error correction term $\boldsymbol{Q}_\ell$ (see section 4.3) need to be optimized. Specifically, $\boldsymbol{Q}_\ell$ is considered as a diagonal matrix weighted by $\sigma_{Q_\ell}^2$, meaning that the standard deviation of $\boldsymbol{Q}_\ell$ is the only parameter requiring optimization. We conduct a grid search over inflation values ranging from 0.8 to 1.9, and over σ_{Q_ℓ} values in the range [1e-3, 1]. These ranges have been experimentally fine-tuned to effectively optimize both λ and σ_{Q_ℓ}.

The numerical results of the grid search, along with computational times, are presented in table 5.9 and table 5.10. For ease of comparison, both tables include the results of the reference ETKF-Q experiment, which performs data assimilation in the physical space. The mean RMSE is computed over the 48 cycles of the assimilation. Computational time statistics, that is means and standard deviations, are computed over the 576 runs resulting from the grid search strategy.

As shown by table 5.9 and table 5.10, the latent ETKF-Q-L algorithm combined with the one-step neural network achieves the best overall mean RMSE score, reaching 0.194 with $\lambda = 1.0$ (which is equivalent to no inflation) and $\sigma_{Q_\ell} = 0.55$. The superior performance of the one-step surrogate over the two-step network could be attributed to the relatively low number of assimilation cycles (*i.e.*, 48) compared to the 1,000 cycles used in the augmented Lorenz experiments. Additionally, the low chaotic nature and linearity of the QG model may also foster the trained one-step network to be more stable in this case than for the augmented Lorenz96 system.

The surrogate within the PCA-derived latent space, as well as the combination of PCA with a linear regression operator, demonstrates impressive performance in both latent and physical assimilations. This is particularly noteworthy given the simplicity of implementing these methods. While the ETKF-Q-L experiment does yield better results as those of the aforementioned models, its mean RMSE of 0.239 still surpasses that of the two experiments with Lipschitz surrogate networks. Furthermore, as observed with the latent assimilation of the augmented Lorenz96 system (see table 5.4), the 1-Lipschitz surrogate network yields a higher mean RMSE than its ext-Lipschitz counterpart in table 5.9, likely due to the strong constraint imposed by the 1-Lipschitz condition.

Comparing the mean RMSE values between table 5.9 and table 5.10 reveals that for the ETKF-Q-L-1-step, ETKF-Q-L-PCA, and ETKF-Q-L-PCA-LinReg experiments, latent space assimilation is more accurate than physical space assimilation. This supports the hypothesis presented in section 4.2.1, which suggests that performing data assimilation within a reduced-space can enhance the correction process by focusing on directions of maximum error growth. However, this is not guaranteed, as demonstrated by the ETKF-Q-P-L, ETKF-Q-P-L-1-Lip, and ETKF-Q-P-L-ext-Lip experiments, where physical assimilation yields better results than latent assimilation.

Interestingly, the ETKF-Q-P-L-1-Lip experiment surpasses its ext-Lipschitz counterpart, which is the opposite of what we observe when the assimilation is performed within a latent space rather than the physical space. This highlights the potential for significant differences in outcomes between latent and physical space assimilations.

In the current setup, the ETKF-Q-L-1-step algorithm outperforms the standard ETKF-Q experiment in both accuracy and computational time. Notably, the ETKF-Q approach is the slowest to complete the 48 data assimilation cycles, indicating that the numerical model of OOPS is significantly more computationally demanding than the data-driven models.

In terms of computational time, the latent experiments are, on average, 280 times faster than the full-space data assimilation ETKF-Q experiment that utilizes the standard numerical model from the OOPS framework. This remarkable speedup is made possible by the autoencoder's ability to accurately represent the 1,600-variable dynamics within a 20-dimensional space. Among the quickest methods, the ETKF-Q-L-PCA stands out with an average runtime of 0.06 seconds for 48 cycles, followed by the ETKF-Q-L-PCA with 0.10 seconds, and the ETKF-Q-L-1-step with a mean runtime of 0.12 seconds.

In conclusion, the iterative training approach of the ETKF-Q-L method does not provide additional benefits for the QG case - in contrast to the augmented Lorenz system -, with the ETKF-Q-1-step outperforming all benchmark experiments. As theorized in section 4.2.1 and initially verified in section 5.1.4, latent assimilation has the potential to surpass physical assimilation both in terms of accuracy and computational efficiency. Importantly, the QG model illustrates that data-driven latent space assimilation can be 280 times faster than its physical space counterpart that relies on the traditional numerical solver, while also producing more accurate results.

As with the augmented Lorenz96 system, we also replace the model error correction step in the latent ETKF-Q algorithm with the SQRT-DEP approach proposed by Raanes *et al.* (2015) from the DAPPER package[4]. More details on the methodology of Raanes *et al.* (2015) are provided in section 4.3.1. This method enables model error correction when the number of ensemble members exceeds the dimensionality of the assimilation state space, a scenario that the standard ETKF-Q algorithm cannot handle. We conducted latent space data assimilation experiments using the ETKF-Q-L-1-step method — chosen since it achieves the lowest mean RMSE score —, while varying the number of ensemble members to 40, 60, 80, and 100. The numerical results are presented in table 5.11.

(4) https://github.com/nansencenter/DAPPER

Experiment name	Mean temporal RMSE	Inflation	$\boldsymbol{\sigma}_{Q_\ell}$	Time Avg.	Time Std.
ETKF-Q	0.200	1.08	0.4	39.60s	2.45s
ETKF-Q-L	0.239	1.01	0.55	0.13s	0.04s
ETKF-Q-L-1-step	0.194	1.0	0.55	0.12s	0.04s
ETKF-Q-L-1-Lip	0.584	1.0	0.25	0.26s	0.07s
ETKF-Q-L-ext-Lip	0.378	1.05	0.3	0.18s	0.056s
ETKF-Q-L-PCA	0.204	1.05	0.7	0.10s	0.04s
ETKF-Q-L-PCA-LinReg	0.231	1.15	0.9	0.06s	0.03s

Table 5.9: Data assimilation results when performing the assimilation within the latent space, along with the reference experiment ETKF-Q (first row) which performs the assimilation in the physical space. Along with the mean temporal RMSE (mean over the DA cycles), we indicate the inflation and model error correction values found by grid search. Additionally, we report the average computational times and their associated standard deviations.

Experiment name	Mean temporal RMSE	Inflation	$\boldsymbol{\sigma}_{Q_\ell}$	Time Avg.	Time Std.
ETKF-Q	0.200	1.08	0.4	39.60s	2.45s
ETKF-Q-P-L	0.237	1.1	0.75	7.67s	2.16s
ETKF-Q-P-L-1-step	0.206	1.15	0.7	8.13s	2.82s
ETKF-Q-P-L-1-Lip	0.287	1.04	1.0	7.84s	2.22s
ETKF-Q-P-L-ext-Lip	0.331	1.05	1.0	7.45s	2.19s
ETKF-Q-P-L-PCA	0.208	1.2	0.55	7.75s	2.31s
ETKF-Q-P-L-PCA-LinReg	0.235	1.15	0.9	7.20s	2.16s

Table 5.10: Data assimilation results when performing the assimilation within the physical space. Along with the mean temporal RMSE (mean over the DA cycles), we indicate the inflation and model error correction values found by grid search. Additionally, we report the average computational times and their associated standard deviations.

First, all experiments listed in table 5.11 yield mean RMSE values between 0.191 and 0.193, which are very slightly lower than the 0.194 mean RMSE achieved by the standard 20-member ETKF-Q-L approach. This suggests that replacing the model error correction step in the latent ETKF-Q methodology with the SQRT-DEP approach may improve the RMSE scores, but not significantly. Therefore, it appears that no significant improvement can be expected from the current implementation of this extended algorithm. Similar to the augmented dynamics, we also notice for QG a substantial increase in computational cost when integrating the SQRT-DEP code from DAPPER. While latent algorithms typically complete the 48 cycles in under a second, as reported in table 5.9, the running times here range from approximately 16 to 50 seconds. This increase in the computational time compared to table 5.9 is largely due to the high computational cost associated with the linear algebra operations introduced by the SQRT-DEP method.

Number of members	RMSE	Inflation	$\boldsymbol{\sigma}_{Q_\ell}$	Time Avg.	Time Std.
40	0.191	1.04	0.5	16.15s	1.12s
60	0.193	1.02	0.55	21.18s	0.76s
80	0.193	1.02	0.45	24.32s	1.44s
100	0.193	1.06	0.45	25.36s	0.58s

Table 5.11: Data assimilation results of the ETKF-Q-L-1-step algorithm when using the SQRT-DEP model error correction method as described by Raanes *et al.* (2015) and implemented in the DAPPER package. Along with the Root Mean Squared Error, we indicate the inflation and model error correction values found by grid search. Additionally, we report the average computational times and their associated standard deviations.

CHAPTER 6

Conclusions and perspectives

In this thesis, we introduced and explored latent space data assimilation, a novel data-driven framework designed to achieve low-cost and accurate data assimilation. Importantly, this approach can be adapted to other data assimilation algorithms (Melinc and Zaplotnik, 2024).

In chapter 3, we provided a comprehensive review of research that demonstrates how deep learning can address various limitations of traditional data assimilation methods. We specifically highlighted scientific publications that illustrate the integration of deep learning into data assimilation, offering a broad perspective on the evolving literature in this field.

We explored the potential of rethinking the assimilation process itself through a data-driven framework, extending the concepts of traditional reduced-space methods. As demonstrated in chapter 4, our latent space data assimilation methodology is theoretically grounded in Lyapunov stability theory (Carrassi *et al.*, 2022).

Our methodology is based on the joint training of an autoencoder and a surrogate neural network. The autoencoder iteratively learns to accurately represent the physical dynamics of interest within a low-dimensional space, while the surrogate is simultaneously trained to propagate the latent variables through time. To ensure stability, we proposed a chained loss function strategy, and alternatively, the implementation of 1-Lipschitz and extended Lipschitz surrogate networks.

We validated our methodology on two distinct systems: a 400-dimensional system derived from a 40-variable chaotic Lorenz96 dynamics and the quasi-geostrophic model of the OOPS framework. These systems present unique challenges, the augmented Lorenz96 is highly chaotic and nonlinear, while QG is less chaotic and characterized by linear combinations of simple patterns. For these tests, we utilized the ETKF-Q algorithm (Fillion *et al.*, 2020), with MLP networks for both the autoencoder and surrogate.

The augmented Lorenz96 system allowed us to demonstrate the effectiveness of the iterative chained loss approach. By adding just one iteration of the surrogate network within the loss function, we transformed a highly unstable one-step network (which reached RMSE values up to 10e19 after 500 time steps) into a two-step network that achieved RMSE values of about 1.5 over the same time window. While the chained loss function strategy is a statistical method affecting specific directions of the latent trajectory, the 1-Lipschitz and extended Lipschitz networks impose global constraints, making them less flexible. Consequently, the chained loss method not only performs on par with the Lipschitz network in terms of stability but also significantly outperforms it in accuracy. Additionally, the two-step network proved superior to the one-step network over 1,000 assimilation cycles, with lower mean RMSE values. Importantly, our latent experiments were approximately 10 times faster than the regular ETKF-Q approach and more than five times faster than other physical assimilation experiments.

The QG dynamics further highlighted the substantial computational gains provided by latent space data assimilation. On average, the latent experiments were 280 times faster than the full-space data assimilation ETKF-Q experiment that utilizes the standard numerical model from the OOPS framework. This remarkable speedup was made possible by the autoencoder's ability to accurately represent the 1,600-variable dynamics within a 20-dimensional space. Unlike with the augmented Lorenz96 system, we did not need to employ a chained loss strategy or Lipschitz neural networks to ensure the surrogate's stability. The QG model's low chaotic nature and linearity allowed the one-step network to perform comparably to the two-step network.

For both the augmented Lorenz96 and QG systems, our experiments confirmed that there exists a low-dimensional space where the assimilation process is not only faster but also more accurate than traditional full-space data assimilation, thereby validating our theoretical expectations.

However, it is important to note that the training process does not always yield a latent space where assimilation is more accurate than in the full space. This was particularly evident for data assimilation experiments using Lipschitz surrogate networks. Despite demonstrating impressive stability in long-term predictions, Lipschitz networks increased training time due to the Björck orthonormalization algorithm and provided less accurate forecasts compared to regular MLP networks. Our numerical results suggest that the iterative training strategy we introduced, using a chained loss function, outperforms Lipschitz networks in both stability and accuracy.

We also proposed an extension of the ETKF-Q algorithm to accommodate scenarios where the number of ensemble members exceeds the state space dimension — a situation not addressed by the standard ETKF-Q algorithm. This was achieved by incorporating the SQRT-DEP model error correction method of Raanes *et al.* (2015) in place of the ETKF-Q's standard model error correction approach.

The work presented in this thesis is foundational, however, it opens up several questions and research directions that remain to be explored. For the augmented Lorenz96 system and the QG model, we employed simple MLP networks, but real-world data assimilation problems may necessitate the use of more advanced, state-of-the-art neural network architectures, such as transformers Vaswani *et al.* (2017) or recurrent surrogate networks.

Additionally, in our numerical experiments, we assumed the observation operator to be an identity matrix. Future work should test the latent data assimilation methodology under more challenging operational scenarios, where the observation operator is more complex, nonlinear, and provides partial or indirect information relative to the state variables.

A crucial aspect of our approach is the neural networks' ability to capture the unstable-neutral subspace. Enhancing the latent space of the autoencoder to adhere to specific mathematical or physical properties could further improve the accuracy of the results. More broadly, the ability to endow the latent space with targeted features or structures is highly desirable. Notably, the unstable-neutral subspace is not static; it evolves over time. Ideally, we would like the autoencoder to capture this temporal evolution to better represent the state information at each time step.

In our numerical experiments, increasing the ensemble size did not yield significant improvements. This warrants further investigation into the potential benefits of using larger ensembles in latent space data assimilation.

APPENDIX A

Data Assimilation methods: mathematical details

A.1 BLUE: minimization of $\boldsymbol{P}^a$ with respect to the gain $\boldsymbol{K}$

Minimizing the trace of the posterior error covariance matrix $\boldsymbol{P}^a$ with respect to $\boldsymbol{K}$, represents an optimization problem to solve. Let us introduce function f so that this minimization problem reads:

$$\boldsymbol{K}^* = \underset{\boldsymbol{K}\in\mathbb{R}^{n\times p}}{\arg\min} \begin{cases} f\colon & \mathbb{R}^{n\times p} \to \mathbb{R} \\ & \boldsymbol{K} \mapsto Tr\Big((\boldsymbol{I}_n - \boldsymbol{K}\boldsymbol{H})\boldsymbol{B}(\boldsymbol{I}_n - \boldsymbol{K}\boldsymbol{H})^T + \boldsymbol{K}\boldsymbol{R}\boldsymbol{K}^T\Big). \end{cases} \tag{A.1}$$

For the sake of clarity and simplicity in the subsequent mathematical derivations, we express f as the sum of two functions, namely f_1 and f_2. Let us therefore define f_1 and f_2 as follows:

$$\begin{aligned} f_1\colon & \mathbb{R}^{n\times p} \to \mathbb{R} \\ & \boldsymbol{K} \mapsto Tr\Big((\boldsymbol{I}_n - \boldsymbol{K}\boldsymbol{H})\boldsymbol{B}(\boldsymbol{I}_n - \boldsymbol{K}\boldsymbol{H})^T\Big), \end{aligned} \tag{A.2}$$

$$\begin{aligned} f_2 \colon \mathbb{R}^{n\times p} &\to \mathbb{R} \\ \boldsymbol{K} &\mapsto Tr\Big(\boldsymbol{K}\boldsymbol{R}\boldsymbol{K}^T\Big), \end{aligned} \tag{A.3}$$

such that $\forall\, \boldsymbol{K} \in \mathbb{R}^{n\times p}$, $f(\boldsymbol{K}) = f_1(\boldsymbol{K}) + f_2(\boldsymbol{K})$.

Functions f_1, f_2 are differentiable as sums and products of matrices, and by the linearity of the trace operator. Therefore f is also differentiable. Let us denote by $\mathcal{V}$, a neighbourhood of $0_{\mathbb{R}^{n\times p}}$, and by $\|\cdot\|$, a norm operator in $\mathbb{R}^{n\times p}$. According to the Taylor series expansion formula, we therefore have that:

$\forall\, \boldsymbol{K} \in \mathbb{R}^{n\times p}$, $\forall\, \boldsymbol{\delta K} \in \mathcal{V}$:

1. $f(\boldsymbol{K} + \boldsymbol{\delta K}) = f(\boldsymbol{K}) + df(\boldsymbol{K})(\boldsymbol{\delta K}) + o(\|\boldsymbol{\delta K}\|)$,
2. $f_1(\boldsymbol{K} + \boldsymbol{\delta K}) = f_1(\boldsymbol{K}) + df_1(\boldsymbol{K})(\boldsymbol{\delta K}) + o(\|\boldsymbol{\delta K}\|)$,
3. $f_2(\boldsymbol{K} + \boldsymbol{\delta K}) = f_2(\boldsymbol{K}) + df_2(\boldsymbol{K})(\boldsymbol{\delta K}) + o(\|\boldsymbol{\delta K}\|)$,

where df, df_1 and df_2 are the differential functions of f, f_1 and f_2, respectively.

We want to determine these differential functions. We will be using the following property of the trace operator:

$$\forall\, A \in \mathbb{R}^{s\times s},\, Tr(A) = Tr\big(A^T\big). \tag{A.4}$$

Let us consider $\boldsymbol{K} \in \mathbb{R}^{n\times p}$ and $\boldsymbol{\delta K} \in \mathcal{V}$:

- Determining $df_1(\boldsymbol{K})(\boldsymbol{\delta K})$:

$$\begin{aligned}
f_1(\boldsymbol{K}+\boldsymbol{\delta K}) &= Tr\Big((\boldsymbol{I}_n-(\boldsymbol{K}+\boldsymbol{\delta K})\boldsymbol{H})\boldsymbol{B}(\boldsymbol{I}_n-(\boldsymbol{K}+\boldsymbol{\delta K})\boldsymbol{H})^T\Big)\\
&= Tr\Big(((\boldsymbol{I}_n-\boldsymbol{KH})-\boldsymbol{\delta KH})\boldsymbol{B}(\boldsymbol{I}_n-\boldsymbol{KH})^T-\boldsymbol{H}^T(\boldsymbol{\delta K})^T\Big)\\
&= Tr\Big(((\boldsymbol{I}_n-\boldsymbol{KH})\boldsymbol{B}-\boldsymbol{\delta KHB})(\boldsymbol{I}_n-\boldsymbol{KH})^T-\boldsymbol{H}^T(\boldsymbol{\delta K})^T\Big)\\
&= \underbrace{Tr\Big((\boldsymbol{I}_n-\boldsymbol{KH})\boldsymbol{B}(\boldsymbol{I}_n-\boldsymbol{KH})^T\Big)}_{=f_1(\boldsymbol{K})}-Tr\Big((\boldsymbol{I}_n-\boldsymbol{KH})\boldsymbol{BH}^T(\boldsymbol{\delta K})^T\Big)\\
&\quad -Tr\Big(\boldsymbol{\delta KHB}(\boldsymbol{I}_n-\boldsymbol{KH})^T\Big)+\underbrace{Tr\Big(\boldsymbol{\delta KHBH}^T(\boldsymbol{\delta K})^T\Big)}_{o(\|\boldsymbol{\delta K}\|)}\\
&= f_1(\boldsymbol{K})-Tr\Big((\boldsymbol{I}_n-\boldsymbol{KH})\boldsymbol{BH}^T(\boldsymbol{\delta K})^T\Big)-Tr\Big(\Big((\boldsymbol{I}_n-\boldsymbol{KH})\boldsymbol{BH}^T(\boldsymbol{\delta K})^T\Big)^T\Big)\\
&\quad +o(\|\boldsymbol{\delta K}\|)\\
&= f_1(\boldsymbol{K})-2Tr\Big((\boldsymbol{I}_n-\boldsymbol{KH})\boldsymbol{BH}^T(\boldsymbol{\delta K})^T\Big)+o(\|\boldsymbol{\delta K}\|).
\end{aligned}$$

Therefore, by the uniqueness of Taylor series expansion at point $\boldsymbol{K}$, we obtain:

$$df_1(\boldsymbol{K})(\boldsymbol{\delta K}) = -2Tr\Big((\boldsymbol{I}_n-\boldsymbol{KH})\boldsymbol{BH}^T(\boldsymbol{\delta K})^T\Big). \tag{A.5}$$

- Determining $df_2(\boldsymbol{K})(\boldsymbol{\delta K})$. Similar calculations lead to:

$$df_2(\boldsymbol{K})(\boldsymbol{\delta K}) = 2Tr\Big(\boldsymbol{KR}(\boldsymbol{\delta K})^T\Big). \tag{A.6}$$

- Determining $df(\boldsymbol{k})(\boldsymbol{\delta K})$. From equations (A.5) and (A.6), we get:

$$\begin{aligned}
df(\boldsymbol{K})(\boldsymbol{\delta K}) &= 2Tr\Big(\boldsymbol{KR}(\boldsymbol{\delta K})^T\Big)-2Tr\Big((\boldsymbol{I}_n-\boldsymbol{KH})\boldsymbol{BH}^T(\boldsymbol{\delta K})^T\Big)\\
&= 2Tr\Big(\boldsymbol{KR}(\boldsymbol{\delta K})^T-(\boldsymbol{I}_n-\boldsymbol{KH})\boldsymbol{BH}^T(\boldsymbol{\delta K})^T\Big)\\
&\boxed{= 2Tr\Big(\Big(\boldsymbol{KR}-(\boldsymbol{I}_n-\boldsymbol{KH})\boldsymbol{BH}^T\Big)(\boldsymbol{\delta K})^T\Big).}
\end{aligned} \tag{A.7}$$

As shown in equation (A.1), function f is a quadratic function with respect to $\boldsymbol{K}$. Also, by considering the $o(\|\boldsymbol{\delta K}\|)$ terms in the previous derivations, we can prove that $\mathrm{Hess(f)}_{\boldsymbol{K}}(\delta\boldsymbol{K}) = \mathrm{Tr}\Big(\delta\boldsymbol{K}\Big(\boldsymbol{HBH}^{\mathrm{T}}+\boldsymbol{R}\Big)(\delta\boldsymbol{K})^{\mathrm{T}}\Big)$, which is positive definite. Therefore, f has a minimum, denoted $\boldsymbol{K}^*$, such that $df(\boldsymbol{K}^*) = 0_{\mathcal{L}(\mathbb{R}^{n\times p},\mathbb{R}^{n\times p})}$, where $\mathcal{L}(\mathbb{R}^{n\times p},\mathbb{R}^{n\times p})$ denotes the set of all linear mappings of $\mathbb{R}^{n\times p}$. We thus have:

$$\begin{aligned}
\boldsymbol{K}^*\boldsymbol{R} - (\boldsymbol{I}_n - \boldsymbol{K}^*\boldsymbol{H})\boldsymbol{B}\boldsymbol{H}^T &= 0_{\mathbb{R}^{n\times p}} \\
\boldsymbol{K}^*\boldsymbol{R} - \boldsymbol{B}\boldsymbol{H}^T + \boldsymbol{K}^*\boldsymbol{H}\boldsymbol{B}\boldsymbol{H}^T &= 0_{\mathbb{R}^{n\times p}} \\
\boldsymbol{K}^*\left(\boldsymbol{H}\boldsymbol{B}\boldsymbol{H}^T + \boldsymbol{R}\right) &= \boldsymbol{B}\boldsymbol{H}^T \\
\boxed{\boldsymbol{K}^* = \boldsymbol{B}\boldsymbol{H}^T\left(\boldsymbol{R} + \boldsymbol{H}\boldsymbol{B}\boldsymbol{H}^T\right)^{-1}.}
\end{aligned} \tag{A.8}$$

A.2 BLUE: derive a second expression for the gain $\boldsymbol{K}$

Let us introduce $\boldsymbol{G} = \boldsymbol{B}^{-1} + \boldsymbol{H}^T\boldsymbol{R}^{-1}\boldsymbol{H}$, so that:

$$\begin{aligned}
\boldsymbol{B}\boldsymbol{H}^T\left(\boldsymbol{H}\boldsymbol{B}\boldsymbol{H}^T + \boldsymbol{R}\right)^{-1} &= \boldsymbol{G}^{-1}\boldsymbol{G}\boldsymbol{K}^* \\
&= \boldsymbol{G}^{-1}\left(\boldsymbol{B}^{-1} + \boldsymbol{H}^T\boldsymbol{R}^{-1}\boldsymbol{H}\right)\boldsymbol{B}\boldsymbol{H}^T\left(\boldsymbol{H}\boldsymbol{B}\boldsymbol{H}^T + \boldsymbol{R}\right)^{-1} \\
&= \boldsymbol{G}^{-1}\left(\underbrace{\boldsymbol{B}^{-1}\boldsymbol{B}}_{=\boldsymbol{I}_n}\boldsymbol{H}^T + \boldsymbol{H}^T\boldsymbol{R}^{-1}\boldsymbol{H}\boldsymbol{B}\boldsymbol{H}^T\right)\left(\boldsymbol{H}\boldsymbol{B}\boldsymbol{H}^T + \boldsymbol{R}\right)^{-1} \\
&= \boldsymbol{G}^{-1}\left(\boldsymbol{H}^T + \boldsymbol{H}^T\boldsymbol{R}^{-1}\boldsymbol{H}\boldsymbol{B}\boldsymbol{H}^T\right)\left(\boldsymbol{H}\boldsymbol{B}\boldsymbol{H}^T + \boldsymbol{R}\right)^{-1} \\
&= \boldsymbol{G}^{-1}\left(\boldsymbol{H}^T\boldsymbol{R}^{-1}\boldsymbol{R} + \boldsymbol{H}^T\boldsymbol{R}^{-1}\boldsymbol{H}\boldsymbol{B}\boldsymbol{H}^T\right)\left(\boldsymbol{H}\boldsymbol{B}\boldsymbol{H}^T + \boldsymbol{R}\right)^{-1} \\
&= \boldsymbol{G}^{-1}\boldsymbol{H}^T\boldsymbol{R}^{-1}\underbrace{\left(\boldsymbol{R} + \boldsymbol{H}\boldsymbol{B}\boldsymbol{H}^T\right)\left(\boldsymbol{H}\boldsymbol{B}\boldsymbol{H}^T + \boldsymbol{R}\right)^{-1}}_{=\boldsymbol{I}_p} \\
&\boxed{= \left(\boldsymbol{B}^{-1} + \boldsymbol{H}^T\boldsymbol{R}^{-1}\boldsymbol{H}\right)^{-1}\boldsymbol{H}^T\boldsymbol{R}^{-1}.}
\end{aligned} \tag{A.9}$$

A.3 BLUE: deriving two formula for $\boldsymbol{P}^a$ under optimal $\boldsymbol{K}$

Let us first derive a simplified expression of $\boldsymbol{P}^a$ when when $\boldsymbol{K} = \boldsymbol{K}^*$:

$$
\begin{aligned}
\boldsymbol{P}^a &= (\boldsymbol{I}_n - \boldsymbol{K}^*\boldsymbol{H})\boldsymbol{B}(\boldsymbol{I}_n - \boldsymbol{K}^*\boldsymbol{H})^T + \boldsymbol{K}^*\boldsymbol{R}(\boldsymbol{K}^*)^T \\
&= (\boldsymbol{I}_n - \boldsymbol{K}^*\boldsymbol{H})\boldsymbol{B}\left(\boldsymbol{I}_n - \boldsymbol{H}^T\boldsymbol{K}^T\right) + \boldsymbol{K}^*\boldsymbol{R}(\boldsymbol{K}^*)^T \\
&= (\boldsymbol{I}_n - \boldsymbol{K}^*\boldsymbol{H})\boldsymbol{B} - (\boldsymbol{I}_n - \boldsymbol{K}^*\boldsymbol{H})\boldsymbol{B}\boldsymbol{H}^T\boldsymbol{K}^T + \boldsymbol{K}^*\boldsymbol{R}(\boldsymbol{K}^*)^T \\
&= (\boldsymbol{I}_n - \boldsymbol{K}^*\boldsymbol{H})\boldsymbol{B} - \underbrace{\left(\boldsymbol{K}^*\boldsymbol{R} - (\boldsymbol{I}_n - \boldsymbol{K}^*\boldsymbol{H})\boldsymbol{B}\boldsymbol{H}^T\right)}_{\boldsymbol{C}}\boldsymbol{K}^T.
\end{aligned}
$$

Here-below, we can show that $\boldsymbol{C} = 0_{\mathbb{R}^{n\times n}}$, by using equation (2.15):

$$
\begin{aligned}
\boldsymbol{C} &= \left(\boldsymbol{B}^{-1} + \boldsymbol{H}^T\boldsymbol{R}^{-1}\boldsymbol{H}\right)^{-1}\boldsymbol{H}^T\underbrace{\boldsymbol{R}^{-1}\boldsymbol{R}}_{=\boldsymbol{I}_p} - \left(\boldsymbol{I}_n - \left(\boldsymbol{B}^{-1} + \boldsymbol{H}^T\boldsymbol{R}^{-1}\boldsymbol{H}\right)^{-1}\boldsymbol{H}^T\boldsymbol{R}^{-1}\boldsymbol{H}\right)\boldsymbol{B}\boldsymbol{H}^T \\
&= \left(\boldsymbol{B}^{-1} + \boldsymbol{H}^T\boldsymbol{R}^{-1}\boldsymbol{H}\right)^{-1}\boldsymbol{H}^T - \left(\boldsymbol{B}^{-1} + \boldsymbol{H}^T\boldsymbol{R}^{-1}\boldsymbol{H}\right)^{-1}\left(\left(\boldsymbol{B}^{-1} + \boldsymbol{H}^T\boldsymbol{R}^{-1}\boldsymbol{H}\right) - \boldsymbol{H}^T\boldsymbol{R}^{-1}\boldsymbol{H}\right)\boldsymbol{B}\boldsymbol{H}^T \\
&= \left(\boldsymbol{B}^{-1} + \boldsymbol{H}^T\boldsymbol{R}^{-1}\boldsymbol{H}\right)^{-1}\boldsymbol{H}^T - \left(\boldsymbol{B}^{-1} + \boldsymbol{H}^T\boldsymbol{R}^{-1}\boldsymbol{H}\right)^{-1}\left(\boldsymbol{B}^{-1} + \underbrace{\boldsymbol{H}^T\boldsymbol{R}^{-1}\boldsymbol{H} - \boldsymbol{H}^T\boldsymbol{R}^{-1}\boldsymbol{H}}_{=0_{\mathbb{R}^{p\times p}}}\right)\boldsymbol{B}\boldsymbol{H}^T \\
&= \left(\boldsymbol{B}^{-1} + \boldsymbol{H}^T\boldsymbol{R}^{-1}\boldsymbol{H}\right)^{-1}\boldsymbol{H}^T - \left(\boldsymbol{B}^{-1} + \boldsymbol{H}^T\boldsymbol{R}^{-1}\boldsymbol{H}\right)^{-1}\underbrace{\boldsymbol{B}^{-1}\boldsymbol{B}}_{=\boldsymbol{I}_n}\boldsymbol{H}^T \\
&= \left(\boldsymbol{B}^{-1} + \boldsymbol{H}^T\boldsymbol{R}^{-1}\boldsymbol{H}\right)^{-1}\boldsymbol{H}^T - \left(\boldsymbol{B}^{-1} + \boldsymbol{H}^T\boldsymbol{R}^{-1}\boldsymbol{H}\right)^{-1}\boldsymbol{H}^T \\
&= 0_{\mathbb{R}^{n\times n}}.
\end{aligned}
$$

Therefore, by substituting $\boldsymbol{K}$ in equation (2.13) by $\boldsymbol{K}^*$ as expressed in equation (2.14), we get the following first formula for $\boldsymbol{P}^a$:

$$
\begin{aligned}
\boldsymbol{P}^a &= (\boldsymbol{I}_n - \boldsymbol{K}^*\boldsymbol{H})\boldsymbol{B} && \text{(A.10)} \\
&= \left(\boldsymbol{I}_n - \boldsymbol{B}\boldsymbol{H}^T\left(\boldsymbol{R} + \boldsymbol{H}\boldsymbol{B}\boldsymbol{H}^T\right)^{-1}\boldsymbol{H}\right)\boldsymbol{B} \\
&\boxed{= \boldsymbol{B} - \boldsymbol{B}\boldsymbol{H}^T\left(\boldsymbol{R} + \boldsymbol{H}\boldsymbol{B}\boldsymbol{H}^T\right)^{-1}\boldsymbol{H}\boldsymbol{B}.} && \text{(A.11)}
\end{aligned}
$$

By using equation (2.15), we obtain a second expression of $\boldsymbol{P}^a$:

$$\begin{aligned}
\boldsymbol{P}^a &= \left(\boldsymbol{I}_n - \left(\boldsymbol{B}^{-1} + \boldsymbol{H}^T\boldsymbol{R}^{-1}\boldsymbol{H}\right)^{-1}\boldsymbol{H}^T\boldsymbol{R}^{-1}\boldsymbol{H}\right)\boldsymbol{B} \\
&= \left(\boldsymbol{B}^{-1} + \boldsymbol{H}^T\boldsymbol{R}^{-1}\boldsymbol{H}\right)^{-1}\left(\boldsymbol{B}^{-1} + \boldsymbol{H}^T\boldsymbol{R}^{-1}\boldsymbol{H} - \boldsymbol{H}^T\boldsymbol{R}^{-1}\boldsymbol{H}\right)\boldsymbol{B} \\
&\boxed{= \left(\boldsymbol{B}^{-1} + \boldsymbol{H}^T\boldsymbol{R}^{-1}\boldsymbol{H}\right)^{-1}.} \qquad \text{(A.12)}
\end{aligned}$$

APPENDIX B

Latent space data assimilation: relation between PCA and autoencoders

B.1 Minimization in a 1-dimensional reduced space

In the following, we solve the optimization problem defined in equation (4.12).

To mathematically decompose equation (4.12), we first remind the definition of the Frobenius norm operator:

$$\forall \boldsymbol{A} \in \mathbb{R}^{k\times p}, \|\boldsymbol{A}\|_F^2 = Tr\left(\boldsymbol{A}^T\boldsymbol{A}\right). \tag{B.1}$$

Given equation (B.1), we can reformulate equation (4.12) in terms of matrix traces. Since the optimization problem in equation (4.12) is subject to an equality constraint, we introduce the Lagrange multiplier $\lambda \in \mathbb{R}$ and define the following:

$$\boldsymbol{u}_1^*, \lambda^* = \underset{\boldsymbol{u}_1\in\mathbb{R}^n,\, \lambda\in\mathbb{R}}{\arg\min}\Big[Tr\left(\left(\boldsymbol{u}_1\boldsymbol{u}_1^T\boldsymbol{X}-\boldsymbol{X}\right)^T\left(\boldsymbol{u}_1\boldsymbol{u}_1^T\boldsymbol{X}-\boldsymbol{X}\right)\right)+\lambda\big(\boldsymbol{u}_1^T\boldsymbol{u}_1-1\big)\Big] \tag{B.2}$$

We define $\boldsymbol{C} = \boldsymbol{X}\boldsymbol{X}^T$ and recall following trace property:

$$\forall \boldsymbol{A} \in \mathbb{R}^{k\times p}, \forall \boldsymbol{B} \in \mathbb{R}^{p\times k},\ Tr\left(\boldsymbol{A}\boldsymbol{B}\right) = Tr\left(\boldsymbol{B}\boldsymbol{A}\right) \tag{B.3}$$

$$\boldsymbol{u}_1^*, \lambda^* = \underset{\boldsymbol{u}_1 \in \mathbb{R}^n,\, \lambda \in \mathbb{R}}{\arg\min} \Big[Tr\left(\big(\boldsymbol{u}_1 \boldsymbol{u}_1^T \boldsymbol{X} - \boldsymbol{X}\big)\big(\boldsymbol{u}_1 \boldsymbol{u}_1^T \boldsymbol{X} - \boldsymbol{X}\big)^T \right) + \lambda\big(\boldsymbol{u}_1^T \boldsymbol{u}_1 - 1\big) \Big]$$
$$\boldsymbol{u}_1^*, \lambda^* = \underset{\boldsymbol{u}_1 \in \mathbb{R}^n,\, \lambda \in \mathbb{R}}{\arg\min} \Big[Tr\left(\big(\boldsymbol{u}_1 \boldsymbol{u}_1^T - \boldsymbol{I}_n\big) \boldsymbol{X}\boldsymbol{X}^T \big(\boldsymbol{u}_1 \boldsymbol{u}_1^T - \boldsymbol{I}_n\big) \right) + \lambda\big(\boldsymbol{u}_1^T \boldsymbol{u}_1 - 1\big) \Big]$$

Then, we have:

$$\begin{aligned}
\boldsymbol{u}_1^*, \lambda^* &= \underset{\boldsymbol{u}_1 \in \mathbb{R}^n,\, \lambda \in \mathbb{R}}{\arg\min} \big[Tr\left(\boldsymbol{C}\big(\boldsymbol{u}_1 \boldsymbol{u}_1^T - \boldsymbol{I}_n\big)\big(\boldsymbol{u}_1 \boldsymbol{u}_1^T - \boldsymbol{I}_n\big) \right) + \lambda\big(\boldsymbol{u}_1^T \boldsymbol{u}_1 - 1\big) \big] \\
\boldsymbol{u}_1^*, \lambda^* &= \underset{\boldsymbol{u}_1 \in \mathbb{R}^n,\, \lambda \in \mathbb{R}}{\arg\min} \big[Tr\left(\boldsymbol{C}\big(\boldsymbol{u}_1 \boldsymbol{u}_1^T \boldsymbol{u}_1 \boldsymbol{u}_1^T - \boldsymbol{u}_1 \boldsymbol{u}_1^T - \boldsymbol{u}_1 \boldsymbol{u}_1^T + \boldsymbol{I}_n\big) \right) + \lambda\big(\boldsymbol{u}_1^T \boldsymbol{u}_1 - 1\big) \big] \\
\boldsymbol{u}_1^*, \lambda^* &= \underset{\boldsymbol{u}_1 \in \mathbb{R}^n,\, \lambda \in \mathbb{R}}{\arg\min} \big[Tr\left(\boldsymbol{C}\big(\boldsymbol{I}_n - \boldsymbol{u}_1 \boldsymbol{u}_1^T\big) \right) + \lambda\big(\boldsymbol{u}_1^T \boldsymbol{u}_1 - 1\big) \big] \\
\boldsymbol{u}_1^*, \lambda^* &= \underset{\boldsymbol{u}_1 \in \mathbb{R}^n,\, \lambda \in \mathbb{R}}{\arg\min} \big[Tr\left(\big(\boldsymbol{I}_n - \boldsymbol{u}_1 \boldsymbol{u}_1^T\big)\boldsymbol{C} \right) + \lambda\big(\boldsymbol{u}_1^T \boldsymbol{u}_1 - 1\big) \big] \\
\boldsymbol{u}_1^*, \lambda^* &= \underset{\boldsymbol{u}_1 \in \mathbb{R}^n,\, \lambda \in \mathbb{R}}{\arg\min} \big[Tr\left(\boldsymbol{C} \right) - Tr\left(\boldsymbol{u}_1 \boldsymbol{u}_1^T \boldsymbol{C} \right) + \lambda\big(\boldsymbol{u}_1^T \boldsymbol{u}_1 - 1\big) \big] \\
\boldsymbol{u}_1^*, \lambda^* &= \underset{\boldsymbol{u}_1 \in \mathbb{R}^n,\, \lambda \in \mathbb{R}}{\arg\min} \big[-Tr\left(\boldsymbol{u}_1 \boldsymbol{u}_1^T \boldsymbol{C} \right) + \lambda\big(\boldsymbol{u}_1^T \boldsymbol{u}_1 - 1\big) \big] \qquad \text{(B.4)}
\end{aligned}$$

Let us define the Lagrangian function $\mathcal{L}_1$ such that:

$$\begin{aligned}
\mathcal{L}_1 \colon & \mathbb{R}^n \times \mathbb{R} \to \mathbb{R} \\
& (\boldsymbol{u}_1, \lambda) \mapsto -Tr\left(\boldsymbol{u}_1 \boldsymbol{u}_1^T \boldsymbol{C} \right) + \lambda\big(\boldsymbol{u}_1^T \boldsymbol{u}_1 - 1\big)
\end{aligned} \qquad \text{(B.5)}$$

In order to find the solution of equation (4.12), the following optimality conditions have to be satisfied:

$$\begin{cases} \frac{\partial \mathcal{L}_1}{\partial \boldsymbol{u}_1}(\boldsymbol{u}_1^*, \lambda^*) = 0 \\ \\ \frac{\partial \mathcal{L}_1}{\partial \lambda}(\boldsymbol{u}_1^*, \lambda^*) = 0 \end{cases} \qquad \text{(B.6)}$$

Using standard trace derivative rules[(1)], we get:

$$\begin{cases} \frac{\partial \mathcal{L}_1}{\partial \boldsymbol{u}_1}(\boldsymbol{u}_1^*, \lambda^*) = 0 \iff -\Big(\boldsymbol{C}\boldsymbol{u}_1^* + \boldsymbol{C}^T \boldsymbol{u}_1^*\Big) + 2\lambda \boldsymbol{u}_1^* = 0 \\ \frac{\partial \mathcal{L}_1}{\partial \lambda}(\boldsymbol{u}_1^*, \lambda^*) = 0 \iff (\boldsymbol{u}_1^*)^T \boldsymbol{u}_1^* - 1 = 0 \end{cases}$$

Since $\boldsymbol{C}$ is symmetric we have:

(1) `https://www.math.uwaterloo.ca/~hwolkowi/matrixcookbook.pdf`

$$\begin{cases} \dfrac{\partial \mathcal{L}_1}{\partial \boldsymbol{u}_1}(\boldsymbol{u}_1^*, \lambda^*) = 0 \iff \boldsymbol{C}\boldsymbol{u}_1^* = \lambda^* \boldsymbol{u}_1^* & \text{(B.7)} \\ \dfrac{\partial \mathcal{L}_1}{\partial \lambda}(\boldsymbol{u}_1^*, \lambda^*) = 0 \iff (\boldsymbol{u}_1^*)^T \boldsymbol{u}_1^* = 1 & \text{(B.8)} \end{cases}$$

From equations equation (B.7), and equation (B.7), we know that $\boldsymbol{u}_1^*$ is an eigenvector of $\boldsymbol{C}$ and that λ^* is its related eigenvalue.

Let us consider $\bar{\boldsymbol{u}}_1 \in \mathbb{R}^{n\times 1}$ that satisfies equation (B.7) and equation (B.7): $\bar{\boldsymbol{u}}_1$ is an eigenvector of $\boldsymbol{C}$ and its related eigenvalue is denoted $\bar{\lambda}$. One could note that since $\boldsymbol{C}$ is a semi-definite positive matrix ($i.e. \forall \boldsymbol{x} \in \mathbb{R}^n, \boldsymbol{x}^T\boldsymbol{C}\boldsymbol{x} \geq 0$), all eigenvalues are positive. Including $\bar{\boldsymbol{u}}_1$ into equation (B.4):

$$\begin{aligned} & \boldsymbol{u}_1^* = \arg\min -Tr\left(\bar{\boldsymbol{u}}_1 \bar{\boldsymbol{u}}_1^T \boldsymbol{C}\right) \\ \iff & \boldsymbol{u}_1^* = \arg\min -Tr\left(\bar{\boldsymbol{u}}_1^T \boldsymbol{C} \bar{\boldsymbol{u}}_1\right) \\ \iff & \boldsymbol{u}_1^* = \arg\max Tr\left(\bar{\boldsymbol{u}}_1^T \boldsymbol{C} \bar{\boldsymbol{u}}_1\right) \\ \iff & \boldsymbol{u}_1^* = \arg\max \bar{\lambda} \end{aligned}$$

The solution that maximizes $\bar{\lambda}$ and thus minimizes our optimization problem is the eigenpair $(\boldsymbol{u}_1^*, \lambda^*)$ such that λ^* is the largest eigenvalue of the covariance matrix $\boldsymbol{C} = \boldsymbol{X}\boldsymbol{X}^T$.

B.2 Minimization in a ℓ-dimensional reduced space

In the following, we solve the optimization problem of equation (4.14). Since it is subject to an equality constraint, we introduce the Lagrange multipliers $\lambda_1, \lambda_2, \ldots, \lambda_\ell$ via the diagonal matrix $\boldsymbol{\Lambda} \in \mathbb{R}^{\ell\times\ell}$:

$$\boldsymbol{U}_\ell^*, \boldsymbol{\Lambda}^* = \underset{\boldsymbol{U}_\ell \in \mathbb{R}^{n\times\ell}}{\arg\min} \left[Tr\left(\left(\boldsymbol{U}_\ell \boldsymbol{U}_\ell^T \boldsymbol{X} - \boldsymbol{X}\right)\left(\boldsymbol{U}_\ell \boldsymbol{U}_\ell^T \boldsymbol{X} - \boldsymbol{X}\right)^T \right) + Tr\left(\boldsymbol{\Lambda}\left(\boldsymbol{U}_\ell^T \boldsymbol{U}_\ell - \boldsymbol{I}_\ell\right)\right)\right] \tag{B.9}$$

Similarly to the upper calculations of appendix B.1, we get:

$$\boldsymbol{U}_\ell^*, \boldsymbol{\Lambda}^* = \underset{\boldsymbol{U}_\ell \in \mathbb{R}^{n\times\ell}}{\arg\min} \left[-Tr\left(\boldsymbol{U}_\ell \boldsymbol{U}_\ell^T \boldsymbol{C}\right) + Tr\left(\boldsymbol{\Lambda}\left(\boldsymbol{U}_\ell^T \boldsymbol{U}_\ell - \boldsymbol{I}_\ell\right)\right)\right] \tag{B.10}$$

We define function the Lagrangian function $\mathcal{L}_2$ such that:

$$\begin{aligned}\mathcal{L}_2 \colon & \mathbb{R}^{n\times\ell} \times \mathbb{R}^{\ell\times\ell} \to \mathbb{R} \\ & (\boldsymbol{U}_\ell, \boldsymbol{\Lambda}) \mapsto \left[-Tr\left(\boldsymbol{U}_\ell \boldsymbol{U}_\ell^T \boldsymbol{C}\right) + Tr\left(\boldsymbol{\Lambda}\left(\boldsymbol{U}_\ell^T \boldsymbol{U}_\ell - \boldsymbol{I}_\ell\right)\right)\right]\end{aligned} \tag{B.11}$$

Following optimality conditions have to be satisfied:

$$\begin{cases} \frac{\partial \mathcal{L}_2}{\partial \boldsymbol{U}_\ell}(\boldsymbol{U}_\ell^*, \boldsymbol{\Lambda}^*) = 0 \\ \\ \frac{\partial \mathcal{L}_2}{\partial \boldsymbol{\Lambda}}(\boldsymbol{U}_\ell^*, \boldsymbol{\Lambda}^*) = 0 \end{cases} \tag{B.12}$$

Then, using the trace derivative rules we get:

$$\begin{cases} \dfrac{\partial \mathcal{L}_2}{\partial \boldsymbol{U}_\ell}(\boldsymbol{U}_\ell^*, \boldsymbol{\Lambda}^*) = 0 \iff \boldsymbol{C}\boldsymbol{U}_\ell^* = \boldsymbol{U}_\ell^* \boldsymbol{\Lambda}^* & \text{(B.13)} \\ \dfrac{\partial \mathcal{L}_2}{\partial \boldsymbol{\Lambda}}(\boldsymbol{U}_\ell^*, \boldsymbol{\Lambda}^*) = 0 \iff \boldsymbol{U}_\ell^T \boldsymbol{U}_\ell = \boldsymbol{I}_\ell & \text{(B.14)} \end{cases}$$

From equation (B.13), we infer:

$$\frac{\partial \mathcal{L}_2}{\partial \boldsymbol{U}_\ell}(\boldsymbol{U}_\ell^*, \boldsymbol{\Lambda}^*) = 0 \iff \boldsymbol{C} \begin{bmatrix} u_{11}^* & u_{12}^* & \dots & u_{1\ell}^* \\ u_{21}^* & u_{22}^* & \dots & u_{2\ell}^* \\ \vdots & \vdots & \ddots & \vdots \\ u_{n1}^* & u_{n2}^* & \dots & u_{n\ell}^* \end{bmatrix} = \begin{bmatrix} \lambda_1^* u_{11}^* & \lambda_2^* u_{12}^* & \dots & \lambda_\ell^* u_{1\ell}^* \\ \lambda_1^* u_{21}^* & \lambda_2^* u_{22}^* & \dots & \lambda_\ell^* u_{2\ell}^* \\ \vdots & \vdots & \ddots & \vdots \\ \lambda_1^* u_{n1}^* & \lambda_2^* u_{n2}^* & \dots & \lambda_\ell^* u_{n\ell}^* \end{bmatrix}$$

$$\frac{\partial \mathcal{L}_2}{\partial \boldsymbol{U}_\ell}(\boldsymbol{U}_\ell^*, \boldsymbol{\Lambda}^*) = 0 \iff \boldsymbol{U}_\ell^* \text{ is made of eigenvectors of } \boldsymbol{C} \text{ and } \boldsymbol{\Lambda}^* \text{ comprises their related eigenvalues.}$$

Let us consider $\bar{\boldsymbol{U}}_\ell \in \mathbb{R}^{n\times\ell}$ that verifies equations B.13 and B.14: $\bar{\boldsymbol{U}}_\ell$ is made of ℓ arbitrarily picked eigenvectors of $\boldsymbol{C}$ and is such that $\bar{\boldsymbol{U}}_\ell{}^T \bar{\boldsymbol{U}}_\ell = \boldsymbol{I}_\ell$. Their corresponding eigenvalues are denoted $\bar{\lambda}_1, \bar{\lambda}_2, \dots, \bar{\lambda}_\ell$. Incorporating $\bar{\boldsymbol{U}}_\ell$ into equation (B.10) gives:

$$
\begin{aligned}
&\boldsymbol{U}_\ell^* = \arg\min -Tr\left(\bar{\boldsymbol{U}}_\ell \bar{\boldsymbol{U}}_\ell^T \boldsymbol{C}\right) \\
\iff &\boldsymbol{U}_\ell^* = \arg\min -Tr\left(\bar{\boldsymbol{U}}_\ell^T \boldsymbol{C} \bar{\boldsymbol{U}}_\ell\right) \\
\iff &\boldsymbol{U}_\ell^* = \arg\max Tr\left(\bar{\boldsymbol{U}}_\ell^T \boldsymbol{C} \bar{\boldsymbol{U}}_\ell\right) \\
\iff &\boldsymbol{U}_\ell^* = \arg\max \sum_{i=1}^{\ell} \bar{\lambda}_i
\end{aligned}
$$

This sum is maximized when the eigenvectors horizontally stacked in $\bar{\boldsymbol{U}}_\ell$ are related to the largest eigenvalues of $\boldsymbol{C}$ (which are all positive since $\boldsymbol{C}$ is semi-definite positive). Therefore, if we denote $\lambda_1^*, \dots, \lambda_\ell^*$ the ℓ largest eigenvalues of $\boldsymbol{C}$ and $u_1^*, \dots, u_\ell^*$ their related eigenvectors, the solution we are looking for is:

$$
\boldsymbol{U}_\ell^*,\, \boldsymbol{\Lambda}^* = \begin{bmatrix} \vdots & \vdots & \dots & \vdots \\ u_1^* & u_2^* & \dots & u_\ell^* \\ \vdots & \vdots & \dots & \vdots \end{bmatrix},\ \begin{bmatrix} \lambda_1^* & 0 & \dots & 0 \\ 0 & \lambda_2^* & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \lambda_\ell^* \end{bmatrix} \tag{B.15}
$$

B.3 Minimization in a 1-dimensional reduced space: a more flexible scenario with two mapping operators

In the following, we solve the optimization problem defined in equation (4.13).

Similarly to the calculations made in appendix B.1, we have:

$$
\begin{aligned}
\boldsymbol{u}_1^*, \boldsymbol{v}_1^* &= \underset{\boldsymbol{u}_1, \boldsymbol{v}_1 \in \mathbb{R}^n}{\arg\min}\ Tr\left(\left(\boldsymbol{u}_1 \boldsymbol{v}_1^T - \boldsymbol{I}_n\right) \boldsymbol{X}\boldsymbol{X}^T \left(\boldsymbol{v}_1 \boldsymbol{u}_1^T - \boldsymbol{I}_n\right)\right) \\
\boldsymbol{u}_1^*, \boldsymbol{v}_1^* &= \underset{\boldsymbol{u}_1, \boldsymbol{v}_1 \in \mathbb{R}^n}{\arg\min}\ Tr\left(\boldsymbol{u}_1 \boldsymbol{v}_1^T \boldsymbol{C} \boldsymbol{v}_1 \boldsymbol{u}_1^T - \boldsymbol{u}_1 \boldsymbol{v}_1^T \boldsymbol{C} - \boldsymbol{C} \boldsymbol{v}_1 \boldsymbol{u}_1^T + \boldsymbol{C}\right) \\
\boldsymbol{u}_1^*, \boldsymbol{v}_1^* &= \underset{\boldsymbol{u}_1, \boldsymbol{v}_1 \in \mathbb{R}^n}{\arg\min}\ Tr\left(\boldsymbol{C}\right) + Tr\left(\boldsymbol{u}_1 \boldsymbol{v}_1^T \boldsymbol{C} \boldsymbol{v}_1 \boldsymbol{u}_1^T\right) - Tr\left(\boldsymbol{u}_1 \boldsymbol{v}_1^T \boldsymbol{C}\right) - Tr\left(\boldsymbol{C} \boldsymbol{v}_1 \boldsymbol{u}_1^T\right) \\
\boldsymbol{u}_1^*, \boldsymbol{v}_1^* &= \underset{\boldsymbol{u}_1, \boldsymbol{v}_1 \in \mathbb{R}^n}{\arg\min}\ Tr\left(\boldsymbol{u}_1 \boldsymbol{v}_1^T \boldsymbol{C} \boldsymbol{v}_1 \boldsymbol{u}_1^T\right) - Tr\left(\boldsymbol{u}_1 \boldsymbol{v}_1^T \boldsymbol{C}\right) - Tr\left(\boldsymbol{C} \boldsymbol{v}_1 \boldsymbol{u}_1^T\right)
\end{aligned}
$$

Let us define function h such that:

$$\begin{aligned} h\colon \mathbb{R}^n \times \mathbb{R}^n &\to \mathbb{R} \\ \boldsymbol{u}_1, \boldsymbol{v}_1 &\mapsto Tr\left(\boldsymbol{u}_1 \boldsymbol{v}_1^T \boldsymbol{C} \boldsymbol{v}_1 \boldsymbol{u}_1^T\right) - Tr\left(\boldsymbol{C} \boldsymbol{u}_1 \boldsymbol{v}_1^T\right) - Tr\left(\boldsymbol{C} \boldsymbol{v}_1 \boldsymbol{u}_1^T\right) \end{aligned} \tag{B.16}$$

Following optimality conditions have to be satisfied:

$$\begin{cases} \frac{\partial h}{\partial \boldsymbol{u}_1}(\boldsymbol{u}_1^*, \boldsymbol{v}_1^*) = 0 \\ \\ \frac{\partial h}{\partial \boldsymbol{v}_1}(\boldsymbol{u}_1^*, \boldsymbol{v}_1^*) = 0 \end{cases} \tag{B.17}$$

From which we derive:

$$\begin{cases} \frac{\partial h}{\partial \boldsymbol{u}_1}(\boldsymbol{u}_1^*, \boldsymbol{v}_1^*) &= 0 \iff \boldsymbol{u}_1\left(\boldsymbol{v}_1^T \boldsymbol{C} \boldsymbol{v}_1\right)^T + \boldsymbol{u}_1\left(\boldsymbol{v}_1^T \boldsymbol{C} \boldsymbol{v}_1\right) - \boldsymbol{C}\boldsymbol{v}_1 - \boldsymbol{C}\boldsymbol{v}_1 = 0 \\ \\ \frac{\partial h}{\partial \boldsymbol{v}_1}(\boldsymbol{u}_1^*, \boldsymbol{v}_1^*) &= 0 \iff \boldsymbol{C}\boldsymbol{v}_1 \boldsymbol{u}_1^T \boldsymbol{u}_1 + \boldsymbol{C}\boldsymbol{v}_1 \boldsymbol{u}_1^T \boldsymbol{u}_1 - \boldsymbol{C}\boldsymbol{u}_1 - \boldsymbol{C}\boldsymbol{u}_1 = 0 \end{cases}$$

$$\begin{cases} \frac{\partial h}{\partial \boldsymbol{u}_1}(\boldsymbol{u}_1^*, \boldsymbol{v}_1^*) &= 0 \iff \boldsymbol{C}\boldsymbol{v}_1 = \boldsymbol{u}_1 \boldsymbol{v}_1^T \boldsymbol{C} \boldsymbol{v}_1 \\ \\ \frac{\partial h}{\partial \boldsymbol{v}_1}(\boldsymbol{u}_1^*, \boldsymbol{v}_1^*) &= 0 \iff \boldsymbol{C}\boldsymbol{u}_1 = \boldsymbol{C}\boldsymbol{v}_1 \boldsymbol{u}_1^T \boldsymbol{u}_1 \end{cases} \tag{B.18}$$

We can simplify equation (B.18) by introducing scalars α and β such that:

$$\begin{cases} \boldsymbol{C}\boldsymbol{v}_1 = \boldsymbol{u}_1 \overbrace{\boldsymbol{v}_1^T \boldsymbol{C} \boldsymbol{v}_1}^{\alpha \in \mathbb{R}^+} \\ \boldsymbol{C}\boldsymbol{u}_1 = \boldsymbol{C}\boldsymbol{v}_1 \underbrace{\boldsymbol{u}_1^T \boldsymbol{u}_1}_{\beta \in \mathbb{R}^+} \end{cases}$$

$$\begin{cases} \boldsymbol{C}\boldsymbol{v}_1 = \alpha \boldsymbol{u}_1 \\ \boldsymbol{C}\boldsymbol{u}_1 = \beta \boldsymbol{C}\boldsymbol{v}_1 \end{cases}$$

$$\begin{cases} \boldsymbol{C}\boldsymbol{v}_1 = \alpha \boldsymbol{u}_1 \\ \boldsymbol{C}\boldsymbol{u}_1 = \alpha\beta \boldsymbol{u}_1 \end{cases} \tag{B.19}$$

By definition, $\beta \in \mathbb{R}^+$. A first thing we can tell is that $\beta \neq 0$:

1. Let us assume $\beta = 0$.
2. Then, $\|\boldsymbol{u}_1\|_2^2 = \boldsymbol{u}_1^T \boldsymbol{u}_1 = \beta = 0$ and by definition of a norm $\boldsymbol{u}_1 = 0_{\mathbb{R}^n}$.

3. By assigning $0_{\mathbb{R}^n}$ to $\boldsymbol{u}_1$, the cost of the function to be minimized (see equation (4.13)) is $\|\boldsymbol{X}\|_F^2$ which is not optimal: indeed, by simply choosing $\boldsymbol{u}_1 = \boldsymbol{v}_1 = [1, 0, \ldots, 0]$, we get:

$$\boldsymbol{u}_1\boldsymbol{v}_1^T = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix}$$

$$\iff \boldsymbol{u}_1\boldsymbol{v}_1^T\boldsymbol{X} - \boldsymbol{X} = \begin{bmatrix} 0 & 0 & \cdots & 0 \\ -x_{21} & -x_{22} & \cdots & -x_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ -x_{n1} & -x_{n2} & \cdots & -x_{nm} \end{bmatrix}$$

$$\left\|\boldsymbol{u}_1\boldsymbol{v}_1^T\boldsymbol{X} - \boldsymbol{X}\right\|_F^2 \leq \|\boldsymbol{X}\|_F^2$$

4. In all likelihood, we will not have $x_{11} = x_{12} = \cdots = x_{1m} = 0$ since $x_1, \ldots x_m$ are randomly perturbed samples. We can therefore claim that $\left\|\boldsymbol{u}_1\boldsymbol{v}_1^T\boldsymbol{X} - \boldsymbol{X}\right\|_F^2 < \|\boldsymbol{X}\|_F^2$ and thus $\beta > 0$.

Likewise, $\alpha \in \mathbb{R}^+$ and we can prove that $\alpha \neq 0$:

1. Let us assume that $\alpha = 0$ and denote $\widetilde{\boldsymbol{v}_1}$ a vector such that $\widetilde{\boldsymbol{v}_1}^T\boldsymbol{C}\widetilde{\boldsymbol{v}_1} = 0$.

2. Then, considering the cost function as expressed in equation (B.16), we have $\forall\, \boldsymbol{u}_1 \in \mathbb{R}^n$:

$$h(\boldsymbol{u}_1, \widetilde{\boldsymbol{v}_1}) = Tr\left(\boldsymbol{u}_1\widetilde{\boldsymbol{v}_1}^T\boldsymbol{C}\widetilde{\boldsymbol{v}_1}\boldsymbol{u}_1^T\right) - Tr\left(\boldsymbol{C}\boldsymbol{u}_1\widetilde{\boldsymbol{v}_1}^T\right) - Tr\left(\boldsymbol{C}\widetilde{\boldsymbol{v}_1}\widetilde{u}^T\right)$$
$$h(\boldsymbol{u}_1, \widetilde{\boldsymbol{v}_1}) = Tr\left(\boldsymbol{u}_1 \times \alpha \times \boldsymbol{u}_1^T\right) - Tr\left(\boldsymbol{C}\boldsymbol{u}_1\widetilde{\boldsymbol{v}_1}^T\right) - Tr\left(\boldsymbol{C}\widetilde{\boldsymbol{v}_1}\widetilde{u}^T\right)$$
$$h(\boldsymbol{u}_1, \widetilde{\boldsymbol{v}_1}) = 0 - Tr\left(\boldsymbol{C}\boldsymbol{u}_1\widetilde{\boldsymbol{v}_1}^T\right) - Tr\left(\boldsymbol{C}\widetilde{\boldsymbol{v}_1}\widetilde{u}^T\right)$$

From first line of equation (B.18), we have $\boldsymbol{C}\widetilde{\boldsymbol{v}_1} = \alpha\boldsymbol{u}_1$, thus $\boldsymbol{C}\widetilde{\boldsymbol{v}_1} = 0$. Then, by replacing $\boldsymbol{C}\widetilde{\boldsymbol{v}_1}$ by 0 in the second line of equation (B.18), we also get $\boldsymbol{C}\boldsymbol{u}_1 = \beta\boldsymbol{C}\widetilde{\boldsymbol{v}_1} = 0$. Therefore:

$$h(\boldsymbol{u}_1, \widetilde{\boldsymbol{v}_1}) = 0 - 0 - 0 = 0$$

3. $\boldsymbol{C}$ is a semi-definite positive matrix, then $\forall\, \boldsymbol{w} \in \mathbb{R}^n$, $\boldsymbol{w}^T\boldsymbol{C}\boldsymbol{w} \geq 0$.

4. Since $\boldsymbol{C} \neq 0_{\mathbb{R}^{n\times n}}$, it exists a unitary vector $\bar{\boldsymbol{v}_1}$ such that $\bar{\boldsymbol{v}_1}^T\boldsymbol{C}\bar{\boldsymbol{v}_1} > 0$.

5. Let us denote $\bar{\alpha} = \bar{\boldsymbol{v}_1}^T \boldsymbol{C} \bar{\boldsymbol{v}_1}$.

6. Applying function h to the pair $(\bar{\boldsymbol{v}_1}, \bar{\boldsymbol{v}_1})$ gives:

$$\begin{aligned}
h(\bar{\boldsymbol{v}_1}, \bar{\boldsymbol{v}_1}) &= Tr\left(\bar{\boldsymbol{v}_1}\bar{\boldsymbol{v}_1}^T \boldsymbol{C} \bar{\boldsymbol{v}_1}\bar{\boldsymbol{v}_1}^T\right) - Tr\left(\boldsymbol{C}\bar{\boldsymbol{v}_1}\bar{\boldsymbol{v}_1}^T\right) - Tr\left(\boldsymbol{C}\bar{\boldsymbol{v}_1}\bar{\boldsymbol{v}_1}^T\right)\\
h(\bar{\boldsymbol{v}_1}, \bar{\boldsymbol{v}_1}) &= Tr\left(\bar{\boldsymbol{v}_1} \times \bar{\alpha} \times \bar{\boldsymbol{v}_1}^T\right) - Tr\left(\bar{\boldsymbol{v}_1}^T \boldsymbol{C} \bar{\boldsymbol{v}_1}\right) - Tr\left(\bar{\boldsymbol{v}_1}^T \boldsymbol{C} \bar{\boldsymbol{v}_1}\right)\\
h(\bar{\boldsymbol{v}_1}, \bar{\boldsymbol{v}_1}) &= \bar{\alpha} \times Tr\left(\bar{\boldsymbol{v}_1}\bar{\boldsymbol{v}_1}^T\right) - 2\bar{\alpha}\\
h(\bar{\boldsymbol{v}_1}, \bar{\boldsymbol{v}_1}) &= \bar{\alpha} \times \|\bar{\boldsymbol{v}_1}\|_2^2 - 2\bar{\alpha}\\
h(\bar{\boldsymbol{v}_1}, \bar{\boldsymbol{v}_1}) &= \bar{\alpha} \times 1^2 - 2\bar{\alpha}\\
h(\bar{\boldsymbol{v}_1}, \bar{\boldsymbol{v}_1}) &= -\bar{\alpha}\\
h(\bar{\boldsymbol{v}_1}, \bar{\boldsymbol{v}_1}) &< 0\\
h(\bar{\boldsymbol{v}_1}, \bar{\boldsymbol{v}_1}) &< h(\boldsymbol{u}_1, \widetilde{\boldsymbol{v}_1}),\ \forall \boldsymbol{u}_1 \in \mathbb{R}^n
\end{aligned}$$

7. Finally, if we consider pair solutions $(\boldsymbol{u}_1^*, \boldsymbol{v}_1^*)$ to our minimization problem, they lead to $\alpha \neq 0$.

Let us consider a pair solution $(\boldsymbol{u}_1^*, \boldsymbol{v}_1^*)$, then from equation (B.19), we get:

$$\begin{cases} \boldsymbol{C}\boldsymbol{v}_1^* = \alpha \boldsymbol{u}_1^*,\ \alpha \neq 0 \\ \boldsymbol{C}\boldsymbol{u}_1^* = \alpha\beta \boldsymbol{u}_1^*,\ \beta \neq 0 \end{cases} \tag{B.20}$$

We then infer that $\boldsymbol{u}_1^*$ is an eigenvector of $\boldsymbol{C}$ and its related eigenvalue is $\alpha\beta = \|\boldsymbol{u}_1^*\|_2^2 (\boldsymbol{v}_1^*)^T \boldsymbol{C}\boldsymbol{v}_1^*$.

We also have the following:

$$\begin{aligned}
\alpha \boldsymbol{u}_1^* &= \boldsymbol{C}\boldsymbol{v}_1^*\\
\alpha (\boldsymbol{v}_1^*)^T \boldsymbol{u}_1^* &= (\boldsymbol{v}_1^*)^T \boldsymbol{C}\boldsymbol{v}_1^*\\
\alpha (\boldsymbol{v}_1^*)^T \boldsymbol{u}_1^* &= \alpha\\
(\boldsymbol{v}_1^*)^T \boldsymbol{u}_1^* &= 1 \text{ (since } \alpha \neq 0)
\end{aligned}$$

By considering $h(\boldsymbol{u}_1^*, \boldsymbol{v}_1^*)$, we get:

$$
\begin{aligned}
h(\boldsymbol{u}_1^*, \boldsymbol{v}_1^*) &= Tr\left(\boldsymbol{u}_1^*(\boldsymbol{v}_1^*)^T \boldsymbol{C} \boldsymbol{v}_1^*(\boldsymbol{u}_1^*)^T\right) - Tr\left(\boldsymbol{u}_1^*(\boldsymbol{v}_1^*)^T \boldsymbol{C}\right) - Tr\left(\boldsymbol{C}\boldsymbol{v}_1(\boldsymbol{u}_1^*)^T\right)\\
h(\boldsymbol{u}_1^*, \boldsymbol{v}_1^*) &= Tr\left(\boldsymbol{C}\boldsymbol{v}_1^*(\boldsymbol{u}_1^*)^T\right) - Tr\left(\boldsymbol{u}_1^*(\boldsymbol{v}_1^*)^T \boldsymbol{C}\right) - Tr\left(\boldsymbol{C}\boldsymbol{v}_1(\boldsymbol{u}_1^*)^T\right) \text{ (since } \boldsymbol{u}_1\boldsymbol{v}_1^T\boldsymbol{C}\boldsymbol{v}_1 = \boldsymbol{C}\boldsymbol{v}_1\text{)}\\
h(\boldsymbol{u}_1^*, \boldsymbol{v}_1^*) &= Tr\left(\boldsymbol{C}\boldsymbol{v}_1^*(\boldsymbol{u}_1^*)^T\right) - Tr\left(\left(\boldsymbol{u}_1^*(\boldsymbol{v}_1^*)^T \boldsymbol{C}\right)^T\right) - Tr\left(\boldsymbol{C}\boldsymbol{v}_1(\boldsymbol{u}_1^*)^T\right)\\
h(\boldsymbol{u}_1^*, \boldsymbol{v}_1^*) &= Tr\left(\boldsymbol{C}\boldsymbol{v}_1^*(\boldsymbol{u}_1^*)^T\right) - Tr\left(\boldsymbol{C}\boldsymbol{v}_1(\boldsymbol{u}_1^*)^T\right) - Tr\left(\boldsymbol{C}\boldsymbol{v}_1(\boldsymbol{u}_1^*)^T\right)\\
h(\boldsymbol{u}_1^*, \boldsymbol{v}_1^*) &= -Tr\left(\boldsymbol{C}\boldsymbol{v}_1(\boldsymbol{u}_1^*)^T\right)\\
h(\boldsymbol{u}_1^*, \boldsymbol{v}_1^*) &= -Tr\left((\boldsymbol{u}_1^*)^T\boldsymbol{C}\boldsymbol{v}_1\right)\\
h(\boldsymbol{u}_1^*, \boldsymbol{v}_1^*) &= -Tr\left((\boldsymbol{u}_1^*)^T\alpha\boldsymbol{u}_1\right)\\
h(\boldsymbol{u}_1^*, \boldsymbol{v}_1^*) &= -\alpha\beta
\end{aligned}
$$

In order to minimize h, we have to maximize the positive scalar $\alpha\beta$ and therefore to choose $\boldsymbol{u}_1^*$ to be the eigenvector of $\boldsymbol{C}$ related to the largest eigenvalue $\lambda_{max} = (\alpha\beta)_{max}$. Then, $\boldsymbol{v}_1^*$ can be any vector of $\mathbb{R}^n$ such that $(\boldsymbol{v}_1^*)^T\boldsymbol{u}_1^* = 1$.

B.4 Minimization in a ℓ-dimensional reduced space: a more flexible scenario with two mapping operators

In the following, we derive properties that the solution of equation (4.16) must satisfy.

Similarly to the calculations made in appendix B.3, we have:

$$
\begin{aligned}
\boldsymbol{U}_\ell^*, \boldsymbol{V}_\ell^* &= \underset{\boldsymbol{U}_\ell, \boldsymbol{V}_\ell \in \mathbb{R}^{n\times\ell}}{\arg\min}\; Tr\left(\left(\boldsymbol{U}_\ell\boldsymbol{V}_\ell^T - \boldsymbol{I}_n\right)\boldsymbol{X}\boldsymbol{X}^T\left(\boldsymbol{V}_\ell\boldsymbol{U}_\ell^T - \boldsymbol{I}_n\right)\right)\\
\boldsymbol{U}_\ell^*, \boldsymbol{V}_\ell^* &= \underset{\boldsymbol{U}_\ell, \boldsymbol{V}_\ell \in \mathbb{R}^{n\times\ell}}{\arg\min}\; Tr\left(\boldsymbol{U}_\ell\boldsymbol{V}_\ell^T\boldsymbol{C}\boldsymbol{V}_\ell\boldsymbol{U}_\ell^T - \boldsymbol{U}_\ell\boldsymbol{V}_\ell^T\boldsymbol{C} - \boldsymbol{C}\boldsymbol{V}_\ell\boldsymbol{U}_\ell^T + \boldsymbol{C}\right)\\
\boldsymbol{U}_\ell^*, \boldsymbol{V}_\ell^* &= \underset{\boldsymbol{U}_\ell, \boldsymbol{V}_\ell \in \mathbb{R}^{n\times\ell}}{\arg\min}\; Tr\left(\boldsymbol{C}\right) + Tr\left(\boldsymbol{U}_\ell\boldsymbol{V}_\ell^T\boldsymbol{C}\boldsymbol{V}_\ell\boldsymbol{U}_\ell^T\right) - Tr\left(\boldsymbol{U}_\ell\boldsymbol{V}_\ell^T\boldsymbol{C}\right) - Tr\left(\boldsymbol{C}\boldsymbol{V}_\ell\boldsymbol{U}_\ell^T\right)\\
\boldsymbol{U}_\ell^*, \boldsymbol{V}_\ell^* &= \underset{\boldsymbol{U}_\ell, \boldsymbol{V}_\ell \in \mathbb{R}^{n\times\ell}}{\arg\min}\; Tr\left(\boldsymbol{U}_\ell\boldsymbol{V}_\ell^T\boldsymbol{C}\boldsymbol{V}_\ell\boldsymbol{U}_\ell^T\right) - Tr\left(\boldsymbol{U}_\ell\boldsymbol{V}_\ell^T\boldsymbol{C}\right) - Tr\left(\boldsymbol{C}\boldsymbol{V}_\ell\boldsymbol{U}_\ell^T\right)
\end{aligned}
$$

Let us define function ϕ such that:

$$\begin{aligned}\phi\colon &\mathbb{R}^{n\times\ell}\times\mathbb{R}^{n\times\ell}\to\mathbb{R}\\ &\boldsymbol{U}_\ell,\boldsymbol{V}_\ell\mapsto Tr\left(\boldsymbol{U}_\ell\boldsymbol{V}_\ell^T\boldsymbol{C}\boldsymbol{V}_\ell\boldsymbol{U}_\ell^T\right)-Tr\left(\boldsymbol{C}\boldsymbol{U}_\ell\boldsymbol{V}_\ell^T\right)-Tr\left(\boldsymbol{C}\boldsymbol{V}_\ell\boldsymbol{U}_\ell^T\right)\end{aligned} \tag{B.21}$$

Following optimality conditions have to be satisfied:

$$\begin{cases}\frac{\partial\phi}{\partial\boldsymbol{U}_\ell}(\boldsymbol{U}_\ell^*,\boldsymbol{V}_\ell^*)=0\\ \\ \frac{\partial\phi}{\partial\boldsymbol{V}_\ell}(\boldsymbol{U}_\ell^*,\boldsymbol{V}_\ell^*)=0\end{cases} \tag{B.22}$$

From which we derive:

$$\begin{cases}\frac{\partial\phi}{\partial\boldsymbol{U}_\ell}(\boldsymbol{U}_\ell^*,\boldsymbol{V}_\ell^*) &=0\iff \boldsymbol{U}_\ell\Big(\boldsymbol{V}_\ell^T\boldsymbol{C}\boldsymbol{V}_\ell\Big)^T+\boldsymbol{U}_\ell\Big(\boldsymbol{V}_\ell^T\boldsymbol{C}\boldsymbol{V}_\ell\Big)-\boldsymbol{C}\boldsymbol{V}_\ell-\boldsymbol{C}\boldsymbol{V}_\ell=0\\ \\ \frac{\partial\phi}{\partial\boldsymbol{V}_\ell}(\boldsymbol{U}_\ell^*,\boldsymbol{V}_\ell^*) &=0\iff \boldsymbol{C}\boldsymbol{V}_\ell\boldsymbol{U}_\ell^T\boldsymbol{U}_\ell+\boldsymbol{C}\boldsymbol{V}_\ell\boldsymbol{U}_\ell^T\boldsymbol{U}_\ell-\boldsymbol{C}\boldsymbol{U}_\ell-\boldsymbol{C}\boldsymbol{U}_\ell=0\end{cases}$$

$$\begin{cases}\frac{\partial\phi}{\partial\boldsymbol{U}_\ell}(\boldsymbol{U}_\ell^*,\boldsymbol{V}_\ell^*) &=0\iff \boldsymbol{C}\boldsymbol{V}_\ell=\boldsymbol{U}_\ell\boldsymbol{V}_\ell^T\boldsymbol{C}\boldsymbol{V}_\ell\\ \\ \frac{\partial\phi}{\partial\boldsymbol{V}_\ell}(\boldsymbol{U}_\ell^*,\boldsymbol{V}_\ell^*) &=0\iff \boldsymbol{C}\boldsymbol{U}_\ell=\boldsymbol{C}\boldsymbol{V}_\ell\boldsymbol{U}_\ell^T\boldsymbol{U}_\ell\end{cases} \tag{B.23}$$

Let us consider $\boldsymbol{U}_\ell\in\mathbb{R}^{n\times\ell}$ and $\boldsymbol{V}_\ell\in\mathbb{R}^{n\times\ell}$ that verify equation (B.23). Let us inject $\boldsymbol{U}_\ell$ and $\boldsymbol{V}_\ell$ in equation (B.21):

$$\begin{aligned}\boldsymbol{U}_\ell^*,\boldsymbol{V}_\ell^* &= \underset{\boldsymbol{U}_\ell,\boldsymbol{V}_\ell\in\mathbb{R}^{n\times\ell}}{\arg\min}\; Tr\left(\boldsymbol{U}_\ell\boldsymbol{V}_\ell^T\boldsymbol{C}\boldsymbol{V}_\ell\boldsymbol{U}_\ell^T\right)-Tr\left(\boldsymbol{U}_\ell\boldsymbol{V}_\ell^T\boldsymbol{C}\right)-Tr\left(\boldsymbol{C}\boldsymbol{V}_\ell\boldsymbol{U}_\ell^T\right)\\ \boldsymbol{U}_\ell^*,\boldsymbol{V}_\ell^* &= \underset{\boldsymbol{U}_\ell,\boldsymbol{V}_\ell\in\mathbb{R}^{n\times\ell}}{\arg\min}\; Tr\left(\boldsymbol{C}\boldsymbol{V}_\ell\boldsymbol{U}_\ell^T\right)-Tr\left(\boldsymbol{C}\boldsymbol{V}_\ell\boldsymbol{U}_\ell^T\right)-Tr\left(\boldsymbol{C}\boldsymbol{V}_\ell\boldsymbol{U}_\ell^T\right)\\ \boldsymbol{U}_\ell^*,\boldsymbol{V}_\ell^* &= \underset{\boldsymbol{U}_\ell,\boldsymbol{V}_\ell\in\mathbb{R}^{n\times\ell}}{\arg\min}\; -Tr\left(\boldsymbol{C}\boldsymbol{V}_\ell\boldsymbol{U}_\ell^T\right)\\ \boldsymbol{U}_\ell^*,\boldsymbol{V}_\ell^* &= \underset{\boldsymbol{U}_\ell,\boldsymbol{V}_\ell\in\mathbb{R}^{n\times\ell}}{\arg\max}\; Tr\left(\boldsymbol{C}\boldsymbol{V}_\ell\boldsymbol{U}_\ell^T\right)\\ \boldsymbol{U}_\ell^*,\boldsymbol{V}_\ell^* &= \underset{\boldsymbol{U}_\ell,\boldsymbol{V}_\ell\in\mathbb{R}^{n\times\ell}}{\arg\max}\; Tr\left(\boldsymbol{V}_\ell^T\boldsymbol{C}\boldsymbol{U}_\ell\right)\end{aligned} \tag{B.24}$$

Also, we have:

$$\begin{aligned}\boldsymbol{C}\boldsymbol{U}_\ell &= \boldsymbol{U}_\ell \boldsymbol{V}_\ell^T \boldsymbol{C}\boldsymbol{V}_\ell \boldsymbol{U}_\ell^T \boldsymbol{U}_\ell \\ \boldsymbol{C}\boldsymbol{U}_\ell &= \boldsymbol{U}_\ell \Big(\boldsymbol{V}_\ell^T \boldsymbol{C}\boldsymbol{V}_\ell \boldsymbol{U}_\ell^T \boldsymbol{U}_\ell\Big) \\ \boldsymbol{C}\boldsymbol{U}_\ell &= \boldsymbol{U}_\ell \Big(\boldsymbol{V}_\ell^T \boldsymbol{C}\boldsymbol{U}_\ell\Big)\end{aligned}$$

$$\boldsymbol{C}\boldsymbol{U}_\ell = \boldsymbol{U}_\ell \boldsymbol{W} \tag{B.25}$$

Let us denote $\boldsymbol{X}$ the eigenvectors of $\boldsymbol{W}$. Then, it exists a diagonal matrix $\boldsymbol{\Lambda} \in \mathbb{R}^{\ell\times\ell}$ comprising the eigenvalues of $\boldsymbol{W}$ and such that $\boldsymbol{W}\boldsymbol{X} = \boldsymbol{W}\boldsymbol{\Lambda}$.

From equation (B.25) we derive:

$$\begin{aligned}\boldsymbol{C}\boldsymbol{U}_\ell \boldsymbol{X} &= \boldsymbol{U}_\ell \boldsymbol{W}\boldsymbol{X} \\ \boldsymbol{C}\boldsymbol{U}_\ell \boldsymbol{X} &= \boldsymbol{U}_\ell \boldsymbol{X}\boldsymbol{\Lambda} \\ \boldsymbol{C}(\boldsymbol{U}_\ell \boldsymbol{X}) &= (\boldsymbol{U}_\ell \boldsymbol{X})\boldsymbol{\Lambda}\end{aligned}$$

Therefore, $\boldsymbol{U}_\ell \boldsymbol{X}$ comprises eigenvectors of $\boldsymbol{C}$ and $\boldsymbol{\Lambda}$ holds their related eigenvalues. One could note that $\boldsymbol{W}$ and $\boldsymbol{C}$ share the same eigenvalues. Thus, the optimization problem is equivalent to searching for $\boldsymbol{U}_\ell^*$ and $\boldsymbol{V}_\ell^*$ such that $Tr\left((\boldsymbol{V}_\ell^*)^T \boldsymbol{C}\boldsymbol{U}_\ell^*\right)$ is the sum of the ℓ largest eigenvalues of $\boldsymbol{C}$.

APPENDIX C

Numerical Experiments: implementation details

C.1 Augmented Lorenz96

We provide the Python implementation of the Lorenz96 class we specifically defined to represent the Lorenz96 system of equation (5.2):

```python
import numpy as np

class Lorenz96:

    def __init__(self, dim: int, time_step: float, forcing_term:float=8.0) -> None:
        self.n = dim
        self.F = forcing_term
        self.dt = time_step

    def l96_ode(self, x: np.ndarray) -> np.ndarray:
        d = np.zeros(x.shape)

        # Calculate derivatives for the 3 edge cases: i=1, 2, n
        d[0] = (x[1] - x[self.n - 2]) * x[self.n - 1] - x[0]
        d[1] = (x[2] - x[self.n - 1]) * x[0] - x[1]
        d[self.n - 1] = (x[0] - x[self.n - 3]) * x[self.n - 2] - x[self.n - 1]

        # Calculate derivatives for the general case
        d[2:self.n - 1] = (x[3:self.n] - x[0:self.n - 3]) * x[1:self.n - 2] - x[2:self.n - 1]

        return d + self.F

    def RKstep(self, x: np.ndarray) -> np.ndarray:
        k1 = self.l96_ode(x)
        k2 = self.l96_ode(x + (self.dt/2.0) * k1)
        k3 = self.l96_ode(x + (self.dt/2.0) * k2)
        k4 = self.l96_ode(x + self.dt * k3)
        return x + (self.dt/6.0) * (k1 + 2.0 * k2 + 2.0 * k3 + k4)


    def traj(self, x: np.ndarray , nb_time_steps: int=1) -> np.ndarray:
        assert x.shape[0] == self.n
        x = x.copy()

        for _ in range(nb_time_steps):
            x = self.RKstep(x)
        return x

    def trajT(self, x: np.ndarray , nb_time_steps: int=1) -> np.ndarray:
        return self.traj(x.T, nb_time_steps=nb_time_steps).T
```

Listing C.1: Implementation of the Lorenz96 class. This class is instantiated in listing C.2.

The Python code provided in listing C.2 details the generation of our 40-dimensional Lorenz96 dataset. The variable *time_step* specifies the time increment for the integration, set at 0.01, using a fourth-order Runge-Kutta scheme. The forcing term is fixed at 8 to induce chaotic behavior, and a seed ensures the reproducibility of the simulations.

```python
import numpy as np
import random
from tqdm import tqdm

def generate_lorenz96_dataset(num_simulations, dim, nb_time_steps, time_step,
                              burning_period, forcing_term=8.0, seed=26):

    model = Lorenz96(dim, time_step, forcing_term=forcing_term)
    dataset = np.zeros((nb_time_steps, num_simulations, dim))

    np.random.seed(seed)
    random.seed(seed)

    initial_states = np.random.randn(num_simulations, dim)

    dataset[0] = model.trajT(initial_states, nb_time_steps=burning_period)

    for i in tqdm(range(1, nb_time_steps)):
        dataset[i] = model.trajT(dataset[i-1], nb_time_steps=1).astype('float32')

    return dataset
```

Listing C.2: Python code to generate the 40-dimensional Lorenz96 dataset.

The conversion of the Lorenz96 dataset into its 400-dimensional augmented version is performed in two steps. First, an orthonormal matrix maps the original data from $\mathbb{R}^{40}$ to $\mathbb{R}^{400}$. The Python class performing this linear dimension augmentation is provided below in listing C.3. Second, a nonlinear invertible 3rd-degree polynomial is applied element-wise on the linearly augmented data. The code for this second transformation is provided in listing C.4.

```
import numpy as np
import random
from scipy.stats import ortho_group

class LinearAugmentation:
    def __init__(self, in_dim, out_dim, seed=26):
        self.in_dim = in_dim
        self.out_dim = out_dim

        np.random.seed(seed)
        random.seed(seed)
        self.ortho = ortho_group.rvs(out_dim)[:in_dim]

    def convert(self, x):
        return x @ self.ortho

    def invert(self, xx):
        return xx @ self.ortho.T
```

```python

    def traj(self, xx, inner_model, nb_time_steps=1):
        x = self.invert(xx)
        x_ = inner_model.traj(x, nb_time_steps=nb_time_steps)
        return self.convert(x_)
```

Listing C.3: Python code of the LinearAugmentation class, used to generate the 400-dimensional augmented Lorenz96 dataset.

```python
import numpy as np
import random

class PolynomialAugmentation(LinearAugmentation):
    def __init__(self, in_dim, out_dim, seed=26):
        super().__init__(in_dim, out_dim, seed=seed)

        np.random.seed(seed)
        random.seed(seed)
        sign = np.sign(np.random.random(out_dim) - 0.5)
        self.A = np.random.random(out_dim) / 10 * sign
        self.B = (1 + (np.random.random(out_dim) - 0.5) * 0.2) * sign
        self.C = np.random.random(out_dim) * 2 - 1

    def cardan_solver(self, xx):
        p = self.B / self.A
        q = (self.C - xx) / self.A
        delta = -(4 * p * p * p + 27 * q * q)
        v = np.cbrt(1 / 2 * (-q + np.sqrt(-delta / 27)))
        u = np.cbrt(1 / 2 * (-q - np.sqrt(-delta / 27)))
        return v + u

    def convert(self, x):
        xx = super().convert(x)
        return self.A * xx ** 3 + self.B * xx + self.C

    def invert(self, xx: np.array):
        x = self.cardan_solver(xx)
        return super().invert(x)
```

Listing C.4: Python code of the PolynomialAugmentation class, used to generate the 400-dimensional augmented Lorenz96 dataset.

Bibliography

Abarbanel HDI, Rozdeba PJ, Shirman S. 2018. Machine Learning: Deepest Learning as Statistical Data Assimilation Problems. *Neural Computation* **30**(8): 2025–2055, doi:10.1162/neco_a_01094, URL https://doi.org/10.1162/neco_a_01094.

Akbari S, Dabaghian PH, San O. 2023. Blending machine learning and sequential data assimilation over latent spaces for surrogate modeling of boussinesq systems. *Physica D: Nonlinear Phenomena* **448**: 133 711, doi:https://doi.org/10.1016/j.physd.2023.133711, URL https://www.sciencedirect.com/science/article/pii/S0167278923000659.

Amari S. 1967. A theory of adaptive pattern classifiers. *IEEE Transactions on Electronic Computers* **3**: 299–307.

Amendola M, Arcucci R, Mottet L, Casas CQ, Fan S, Pain C, Linden P, Guo YK. 2021. Data assimilation in the latent space of a convolutional autoencoder. In: *Computational Science – ICCS 2021*, Paszynski M, Kranzlmüller D, Krzhizhanovskaya VV, Dongarra JJ, Sloot PM (eds). Springer International Publishing: Cham, ISBN 978-3-030-77977-1, pp. 373–386.

Amendola M, Arcucci R, Mottet L, Casas CQ, Fan S, Pain CC, Linden P, Guo Y. 2020. Data assimilation in the latent space of a neural network. *CoRR* **abs/2012.12056**, URL https://arxiv.org/abs/2012.12056.

Amezcua J, Goodliff M, Leeuwen PJV. 2017. A weak-constraint 4densemblevar. part i: formulation and simple model experiments. *Tellus A: Dynamic Meteorology and Oceanography* **69**(1): 1271 564, doi:10.1080/16000870.2016.1271564, URL https://doi.org/10.1080/16000870.2016.1271564.

Andersson E, Haseler J, Undén P, Courtier P, Kelly G, Vasiljevic D, Brankovic C, Gaffard C, Hollingsworth A, Jakob C, Janssen P, Klinker E, Lanzinger A, Miller M, Rabier F, Simmons A, Strauss B, Viterbo P, Cardinali C, Thépaut JN. 1998. The ecmwf implementation of three-dimensional variational assimilation (3d-var). iii: Experimental results. *Quarterly Journal of the Royal Meteorological Society* **124**(550): 1831–1860, doi:https://doi.org/10.1002/qj.49712455004, URL https://rmets.onlinelibrary.wiley.com/doi/abs/10.1002/qj.49712455004.

Anil C, Lucas J, Grosse R. 2019. Sorting out lipschitz function approximation. In: *International Conference on Machine Learning*. PMLR, pp. 291–301.

Annan J, Hargreaves J. 2004. Efficient parameter estimation for a highly chaotic system. *Tellus A: Dynamic Meteorology and Oceanography* **56**(5): 520–526, doi:10.3402/tellusa.v56i5.14438, URL https://doi.org/10.3402/tellusa.v56i5.14438.

Antoulas AC. 2005. *Approximation of large-scale dynamical systems*. SIAM.

Arnulf Jentzen BK, von Wurstemberger P. 2023. Mathematical introduction to deep learning: Methods, implementations, and theory. https://arxiv.org/abs/2310.20360.

Asch M, Bocquet M, Nodet M. 2016. *Data assimilation: methods, algorithms, and applications*. Fundamentals of Algorithms, SIAM, URL https://hal.inria.fr/hal-01402885.

Bachlechner T, Majumder BP, Mao H, Cottrell G, McAuley J. 2021. Rezero is all you need: Fast convergence at large depth. In: *Uncertainty in Artificial Intelligence*. PMLR, pp. 1352–1361.

Baldi P, Hornik K. 1989. Neural networks and principal component analysis: Learning from examples without local minima. *Neural Networks* **2**(1): 53–58, doi:https://doi.org/10.1016/0893-6080(89)90014-2, URL https://www.sciencedirect.com/science/article/pii/0893608089900142.

Barreira L, Pesin YB. 2002. *Lyapunov exponents and smooth ergodic theory*, vol. 23. American Mathematical Soc.

Battaglia PW, Hamrick JB, Bapst V, Sanchez-Gonzalez A, Zambaldi V, Malinowski M, Tacchetti A, Raposo D, Santoro A, Faulkner R, *et al.* 2018. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261* .

Bi K, Xie L, Zhang H, Chen X, Gu X, Tian Q. 2022. Pangu-weather: A 3d high-resolution model for fast and accurate global weather forecast.

Bishop CH, Etherton BJ, Majumdar SJ. 2001. Adaptive sampling with the ensemble transform kalman filter. part i: Theoretical aspects. *Monthly Weather Review* **129**(3): 420 – 436, doi:10.1175/1520-0493(2001)129⟨0420:ASWTET⟩2.0.CO;2, URL https://journals.ametsoc.org/view/journals/mwre/129/3/1520-0493_2001_129_0420_aswtet_2.0.co_2.xml.

Björck r, Bowie C. 1971. An iterative algorithm for computing the best estimate of an orthogonal matrix. *SIAM Journal on Numerical Analysis* **8**(2): 358–364, doi:10.1137/0708036, URL https://doi.org/10.1137/0708036.

Bocquet M, Brajard J, Carrassi A, Bertino L. 2020. Bayesian inference of chaotic dynamics by merging data assimilation, machine learning and expectation-maximization. *Foundations of Data Science* **2**(1): 55–80, doi:10.3934/fods.2020004, URL https://www.aimsciences.org/en/article/doi/10.3934/fods.2020004. Publisher: Foundations of Data Science.

Bocquet M, Carrassi A. 2017. Four-dimensional ensemble variational data assimilation and the unstable subspace. *Tellus A: Dynamic Meteorology and Oceanography* **69**(1): 1304 504, doi:10.1080/16000870.2017.1304504, URL https://doi.org/10.1080/16000870.2017.1304504.

Bocquet M, Farchi A. 2014. Introduction to the principles and methods of data assimilation in the geosciences. https://cerea.enpc.fr/HomePages/bocquet/teaching/assim-mb-en.pdf. Lecture notes for the Master M2 MOCIS & WAPE at École des Ponts ParisTech. Last revision: 31st January 2024).

Bocquet M, Gurumoorthy KS, Apte A, Carrassi A, Grudzien C, Jones CKRT. 2017. Degenerate kalman filter error covariances and their convergence onto the unstable subspace. *SIAM/ASA Journal on Uncertainty Quantification* **5**(1): 304–333, doi:10.1137/16M1068712, URL https://doi.org/10.1137/16M1068712.

Bocquet M, Sakov P. 2014. An iterative ensemble kalman smoother. *Quarterly Journal of the Royal Meteorological Society* **140**(682): 1521–1535, doi:https:

//doi.org/10.1002/qj.2236, URL https://rmets.onlinelibrary.wiley.com/doi/abs/10.1002/qj.2236.

Bonavita M. 2023. On some limitations of data-driven weather forecasting models.

Bonavita M, Laloyaux P. 2020. Machine learning for model error inference and correction. *Journal of Advances in Modeling Earth Systems* **12**, doi:10.1029/2020MS002232.

Bottou L. 2010. Large-scale machine learning with stochastic gradient descent. In: *Proceedings of COMPSTAT'2010: 19th International Conference on Computational StatisticsParis France, August 22-27, 2010 Keynote, Invited and Contributed Papers*. Springer, pp. 177–186.

Bouallegue ZB, Alexe M, Chantry M, Clare M, Dramsch J, Lang S, Lessig C, Magnusson L, Nemesio AP, Pinault F, Raoult B, Tietsche) S. 2023. A new ml model in the ecmwf web charts. URL https://www.ecmwf.int/en/about/media-centre/aifs-blog/2023/new-ml-model-ecmwf-web-charts.

Boudier P, Fillion A, Gratton S, Gürol S, Zhang S. 2023. Data assimilation networks. *Journal of Advances in Modeling Earth Systems* **15**(4): e2022MS003 353, doi:https://doi.org/10.1029/2022MS003353, URL https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/2022MS003353. E2022MS003353 2022MS003353.

Bourlard H, Kamp Y. 1988. Auto-association by multilayer perceptrons and singular value decomposition. *Biological cybernetics* **59**: 291–4, doi:10.1007/BF00332918.

Brajard J, Carrassi A, Bocquet M, Bertino L. 2020. Combining data assimilation and machine learning to emulate a dynamical model from sparse and noisy observations: A case study with the lorenz 96 model. *Journal of Computational Science* **44**: 101 171, doi:10.1016/j.jocs.2020.101171, URL https://www.sciencedirect.com/science/article/pii/S1877750320304725.

Burgers G, van Leeuwen PJ, Evensen G. 1998. Analysis scheme in the ensemble kalman filter. *Monthly Weather Review* **126**(6): 1719 – 1724, doi:10.1175/1520-0493(1998)126⟨1719:ASITEK⟩2.0.CO;2, URL https://journals.ametsoc.org/view/journals/mwre/126/6/1520-0493_1998_126_1719_asitek_2.0.co_2.xml.

Calzone O. 2022. An intuitive explanation of lstm. URL https://medium.com/@ottaviocalzone/an-intuitive-explanation-of-lstm-a035eb6ab42c.

Cao Y, Zhu J, Navon IM, Luo Z. 2007. A reduced-order approach to four-dimensional variational data assimilation using proper orthogonal decomposition. *International Journal for Numerical Methods in Fluids* **53**(10): 1571–1583, doi: https://doi.org/10.1002/fld.1365, URL https://onlinelibrary.wiley.com/doi/abs/10.1002/fld.1365.

Carrassi A, Bocquet M, Demaeyer J, Grudzien C, Raanes P, Vannitsem S. 2022. Data assimilation for chaotic dynamics. *Data Assimilation for Atmospheric, Oceanic and Hydrologic Applications (Vol. IV)* : 1–42doi:10.1007/978-3-030-77722-7_1, URL https://doi.org/10.1007/978-3-030-77722-7_1.

Carrassi A, Vannitsem S. 2011. Treatment of the error due to unresolved scales in sequential data assimilation. *I. J. Bifurcation and Chaos* **21**: 3619–3626, doi: 10.1142/S0218127411030775.

Casas CQ, Arcucci R, Wu P, Pain C, Guo YK. 2020. A reduced order deep data assimilation model. *Physica D: Nonlinear Phenomena* **412**: 132 615, doi:https://doi.org/10.1016/j.physd.2020.132615, URL https://www.sciencedirect.com/science/article/pii/S0167278920300488.

Cauchy A, *et al.* 1847. Méthode générale pour la résolution des systemes d'équations simultanées. *Comp. Rend. Sci. Paris* **25**(1847): 536–538.

Chandler GJ, Kerswell RR. 2013. Invariant recurrent solutions embedded in a turbulent two-dimensional kolmogorov flow. *Journal of Fluid Mechanics* **722**: 554–595, doi:10.1017/jfm.2013.122.

Chattopadhyay A, Nabizadeh E, Bach E, Hassanzadeh P. 2023. Deep learning-enhanced ensemble-based data assimilation for high-dimensional nonlinear dynamical systems. *Journal of Computational Physics* **477**: 111 918, doi:10.1016/j.jcp.2023.111918, URL https://www.sciencedirect.com/science/article/pii/S002199912300013X.

Chen K, Han T, Gong J, Bai L, Ling F, Luo JJ, Chen X, Ma L, Zhang T, Su R, Ci Y, Li B, Yang X, Ouyang W. 2023. Fengwu: Pushing the skillful global medium-range weather forecast beyond 10 days lead.

Chen LC, Papandreou G, Kokkinos I, Murphy K, Yuille AL. 2017. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE transactions on pattern analysis and machine intelligence* **40**(4): 834–848.

Cheng S, Chen J, Anastasiou C, Angeli P, Matar OK, Guo YK, Pain CC, Arcucci R. 2022a. Generalised latent assimilation in heterogeneous reduced spaces with machine learning surrogate models. *Journal of Scientific Computing* **94**(1): 11, doi:10.1007/s10915-022-02059-4, URL https://doi.org/10.1007/s10915-022-02059-4.

Cheng S, Liu C, Guo Y, Arcucci R. 2024. Efficient deep data assimilation with sparse observations and time-varying sensors. *Journal of Computational Physics* **496**: 112 581, doi:https://doi.org/10.1016/j.jcp.2023.112581, URL https://www.sciencedirect.com/science/article/pii/S0021999123006769.

Cheng S, Prentice IC, Huang Y, Jin Y, Guo YK, Arcucci R. 2022b. Data-driven surrogate model with latent data assimilation: Application to wildfire forecasting. *Journal of Computational Physics* **464**: 111 302, doi:https://doi.org/10.1016/j.jcp.2022.111302, URL https://www.sciencedirect.com/science/article/pii/S0021999122003643.

Cheng S, Quilodrán-Casas C, Ouala S, Farchi A, Liu C, Tandeo P, Fablet R, Lucor D, Iooss B, Brajard J, Xiao D, Janjic T, Ding W, Guo Y, Carrassi A, Bocquet M, Arcucci R. 2023. Machine learning with data assimilation and uncertainty quantification for dynamical systems: A review. *IEEE/CAA Journal of Automatica Sinica* **10**(6): 1361–1387, doi:10.1109/JAS.2023.123537, URL https://ieeexplore.ieee.org/abstract/document/10141545. Conference Name: IEEE/CAA Journal of Automatica Sinica.

Chernodub A, Nowicki D. 2016. Norm-preserving orthogonal permutation linear unit activation functions (oplu). *ArXiv* **abs/1604.02313**, URL https://api.semanticscholar.org/CorpusID:2600579.

Clevert DA, Unterthiner T, Hochreiter S. 2015. Fast and accurate deep network learning by exponential linear units (elus). *Under Review of ICLR2016 (1997)* .

Courtier P. 2007. Dual formulation of four-dimensional variational assimilation. *Quarterly Journal of the Royal Meteorological Society* **123**: 2449 – 2461, doi:10.1002/qj.49712354414.

Courtier P, Andersson E, Heckley W, Vasiljevic D, Hamrud M, Hollingsworth A, Rabier F, Fisher M, Pailleux J. 1998. The ecmwf implementation of three-dimensional variational assimilation (3d-var). i: Formulation. *Quarterly Journal of the Royal Meteorological Society* **124**(550): 1783–1807, doi:https://doi.org/10.1002/qj.49712455002, URL https://rmets.onlinelibrary.wiley.com/doi/abs/10.1002/qj.49712455002.

Cranmer M, Greydanus S, Hoyer S, Battaglia P, Spergel D, Ho S. 2020. Lagrangian neural networks. *arXiv preprint arXiv:2003.04630* .

Crosley T. 2022. What were the uses of the first computers? URL https://www.quora.com/What-were-the-uses-of-the-first-computers/answer/Tom-Crosley-1.

Cybenko GV. 1989. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems* **2**: 303–314, URL https://api.semanticscholar.org/CorpusID:3958369.

Davies DR, Wilson C, Kramer S. 2011. Fluidity: A fully unstructured anisotropic adaptive mesh computational modeling framework for geodynamics. *Geochemistry Geophysics Geosystems - GEOCHEM GEOPHYS GEOSYST* **12**, doi:10.1029/2011GC003551.

Dee DP. 2005. Bias and data assimilation. *Quarterly Journal of the Royal Meteorological Society* **131**(613): 3323–3343, doi:https://doi.org/10.1256/qj.05.137, URL https://rmets.onlinelibrary.wiley.com/doi/abs/10.1256/qj.05.137.

Dekel O, Gilad-Bachrach R, Shamir O, Xiao L. 2012. Optimal distributed online prediction using mini-batches. *Journal of Machine Learning Research* **13**(1).

Deng Z, He C, Liu Y. 2021. Deep neural network-based strategy for optimal sensor placement in data assimilation of turbulent flow. *Physics of Fluids* **33**(2): 025 119, doi:10.1063/5.0035230, URL https://doi.org/10.1063/5.0035230.

Deng Z, He C, Wen X, Liu Y. 2018. Recovering turbulent flow field from local quantity measurement: turbulence modeling using ensemble-kalman-filter-based data assimilation. *Journal of Visualization* **21**, doi:10.1007/s12650-018-0508-0.

Devlin J, Chang MW, Lee K, Toutanova K. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* .

Doersch C. 2016. Tutorial on variational autoencoders. *arXiv preprint arXiv:1606.05908* .

Duan J, Nadiga B. 2006. Stochastic parameterization for large eddy simulation of geophysical flows. *Proceedings of the American Mathematical Society* **135**, doi: 10.1090/S0002-9939-06-08631-X.

Elman JL. 1990. Finding structure in time. *Cognitive Science* **14**(2): 179–211, doi:https://doi.org/10.1016/0364-0213(90)90002-E, URL https://www.sciencedirect.com/science/article/pii/036402139090002E.

Emerick AA, Reynolds AC. 2013. Ensemble smoother with multiple data assimilation. *Computers & Geosciences* **55**: 3–15, doi:https://doi.org/10.1016/j.cageo.2012.03.011, URL https://www.sciencedirect.com/science/article/pii/S0098300412000994. Ensemble Kalman filter for data assimilation.

Evensen G. 1994. Sequential data assimilation with a nonlinear quasi-geostrophic model using monte carlo methods to forecast error statistics. *Journal of Geophysical Research: Oceans* **99**(C5): 10 143–10 162, doi:https://doi.org/10.1029/94JC00572, URL https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/94JC00572.

Evensen G. 2003. The ensemble kalman filter: Theoretical formulation and practical implementation. *Ocean Dynamics* **53**: 343–367, doi:10.1007/s10236-003-0036-9.

Evensen G. 2009a. *Data assimilation: The ensemble kalman filter, 2nd edition.* Springer-Verlag: Berlin, Heidelberg, ISBN 978-3-642-03710-8.

Evensen G. 2009b. The ensemble kalman filter for combined state and parameter estimation. *IEEE Control Systems Magazine* **29**(3): 83–104, doi:10.1109/MCS.2009.932223.

Fablet R, Chapron B, Drumetz L, Mémin E, Pannekoucke O, Rousseau F. 2021. Learning variational data assimilation models and solvers. *Journal of Advances in Modeling Earth Systems* **13**(10): e2021MS002 572, doi:https://doi.org/10.1029/2021MS002572, URL https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/2021MS002572. E2021MS002572 2021MS002572.

Fablet R, Febvre Q, Chapron B. 2023. Multimodal 4dvarnets for the reconstruction of sea surface dynamics from sst-ssh synergies. *IEEE Transactions on Geoscience and Remote Sensing* **61**: 1–14, doi:10.1109/TGRS.2023.3268006.

Fandry CB, Leslie LM. 1984. A Two-Layer Quasi-Geostrophic Model of Summer Trough Formation in the Australian Subtropical Easterlies. *Journal of the Atmospheric Sciences* **41**(5): 807–818, doi:10.1175/1520-0469(1984)041⟨0807:ATLQGM⟩2.0.CO;2.

Farchi A, Bocquet M, Laloyaux P, Bonavita M, Malartic Q. 2021a. A comparison of combined data assimilation and machine learning methods for offline and online model error correction. *Journal of Computational Science* **55**: 101 468, doi:https://doi.org/10.1016/j.jocs.2021.101468, URL https://www.sciencedirect.com/science/article/pii/S1877750321001435.

Farchi A, Laloyaux P, Bonavita M, Bocquet M. 2021b. Using machine learning to correct model error in data assimilation and forecast applications. *Quarterly Journal of the Royal Meteorological Society* **147**(739): 3067–3084, doi: https://doi.org/10.1002/qj.4116, URL https://rmets.onlinelibrary.wiley.com/doi/abs/10.1002/qj.4116.

Farrell BF, Ioannou PJ. 2001. State estimation using a reduced-order kalman filter. *Journal of the Atmospheric Sciences* **58**(23): 3666 – 3680, doi:10.1175/1520-0469(2001)058⟨3666:SEUARO⟩2.0.CO;2, URL https://journals.ametsoc.org/view/journals/atsc/58/23/1520-0469_2001_058_3666_seuaro_2.0.co_2.xml.

Fillion A, Bocquet M, Gratton S. 2018. Quasi-static ensemble variational data assimilation: a theoretical and numerical study with the iterative ensemble kalman smoother. *Nonlinear Processes in Geophysics* **25**(2): 315–334, doi: 10.5194/npg-25-315-2018, URL https://npg.copernicus.org/articles/25/315/2018/.

Fillion A, Bocquet M, Gratton S, Gürol S, Sakov P. 2020. An iterative ensemble kalman smoother in presence of additive model error. *SIAM/ASA Journal on Uncertainty Quantification* **8**(1): 198–228, doi:10.1137/19M1244147, URL https://doi.org/10.1137/19M1244147.

Fisher M, Gürol S. 2017. Parallelization in the time dimension of four-dimensional variational data assimilation. *Quarterly Journal of the Royal Meteorological Society* **143**(703): 1136–1147, doi:https://doi.org/10.1002/qj.2997, URL https://rmets.onlinelibrary.wiley.com/doi/abs/10.1002/qj.2997.

Frerix T, Kochkov D, Smith JA, Cremers D, Brenner MP, Hoyer S. 2021. Variational data assimilation with a learned inverse observation operator. *CoRR* **abs/2102.11192**, URL https://arxiv.org/abs/2102.11192.

Fukami K, Maulik R, Ramachandra N, Fukagata K, Taira K. 2021. Global field reconstruction from sparse sensors with voronoi tessellation-assisted deep learning. *Nature Machine Intelligence* **3**(11): 945–951, doi:10.1038/s42256-021-00402-2. Funding Information: R.M. and N.R. were supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, under contract DE-AC02-06CH11357. This research was funded in part and used resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under contract DE-AC02-06CH11357. Koji.F. thanks support from the Japan Society for the Promotion of Science (grant nos. 18H03758, 21H05007). K.T. acknowledges support from the US Air Force Office of Scientific Research (grant nos. FA9550-16-1-0650 and FA9550-21-1-0178) and the US Army Research Office (grant no. W911NF-19-1-0032). Publisher Copyright: © 2021, The Author(s), under exclusive licence to Springer Nature Limited.

Gal Y, Ghahramani Z. 2016. A theoretically grounded application of dropout in recurrent neural networks. In: *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16. Curran Associates Inc.: Red Hook, NY, USA, ISBN 9781510838819, p. 1027–1035.

Gauthier P, Pellerin S, Buis S. 2008. Intercomparison of the primal and dual formulations of variational data assimilation. *Quarterly Journal of the Royal Meteorological Society* **134**: 1015 – 1025, doi:10.1002/qj.257.

Geer AJ. 2021. Learning earth system models from observations: machine learning or data assimilation? *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* **379**(2194): 20200 089, doi:10.1098/rsta.2020.0089, URL https://royalsocietypublishing.org/doi/full/10.1098/rsta.2020.0089. Publisher: Royal Society.

Gers F, Schmidhuber J, Cummins F. 1999. Learning to forget: continual prediction with lstm. In: *1999 Ninth International Conference on Artificial Neural Networks ICANN 99. (Conf. Publ. No. 470)*, vol. 2. pp. 850–855 vol.2, doi:10.1049/cp:19991218.

Girshick R, Donahue J, Darrell T, Malik J. 2014. Rich feature hierarchies for accurate object detection and semantic segmentation. In: *2014 IEEE Conference on Computer Vision and Pattern Recognition*. pp. 580–587, doi:10.1109/CVPR.2014.81.

Gladchuk V. 2020. The history of machine learning: How did it all start? URL https://labelyourdata.com/articles/history-of-machine-learning-how-did-it-all-start.

Glorot X, Bordes A, Bengio Y. 2011. Deep sparse rectifier neural networks. In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, *Proceedings of Machine Learning Research*, vol. 15, Gordon G, Dunson D, Dudík M (eds). PMLR: Fort Lauderdale, FL, USA, pp. 315–323, URL https://proceedings.mlr.press/v15/glorot11a.html.

Golub G, Van Loan C. 2013. *Matrix computations*. Johns Hopkins Studies in the Mathematical Sciences, Johns Hopkins University Press, ISBN 9781421407944, URL https://books.google.fr/books?id=X5YfsuCWpxMC.

Goodfellow I, Bengio Y, Courville A. 2016. *Deep learning*. MIT Press. http://www.deeplearningbook.org.

Gottwald GA, Reich S. 2021a. Combining machine learning and data assimilation to forecast dynamical systems from noisy partial observations. *Chaos: An Interdisciplinary Journal of Nonlinear Science* **31**(10): 101 103, doi:10.1063/5.0066080, URL https://doi.org/10.1063/5.0066080.

Gottwald GA, Reich S. 2021b. Supervised learning from noisy observations: Combining machine-learning techniques with data assimilation. *Physica D: Nonlinear Phenomena* **423**: 132 911, doi:https://doi.org/10.1016/j.physd.2021.132911, URL https://www.sciencedirect.com/science/article/pii/S0167278921000695.

Gratton S, Gürol S, Toint P. 2013. Preconditioning and globalizing conjugate gradients in dual space for quadratically penalized nonlinear-least squares problems. *Computational Optimization and Applications* **54**(1): 1–25, doi:10.1007/s10589-012-9478-7. Publication code : FP SB092/2010/10 ; SB04977/2010/10.

Gratton S, Lawless AS, Nichols NK. 2007. Approximate gauss–newton methods for nonlinear least squares problems. *SIAM Journal on Optimization* **18**(1): 106–132.

Gratton S, Tshimanga J. 2009. An observation-space formulation of variational assimilation using a restricted preconditioned conjugate gradient algorithm. *Quarterly Journal of the Royal Meteorological Society* **135**(643): 1573–1585, doi: https://doi.org/10.1002/qj.477, URL https://rmets.onlinelibrary.wiley.com/doi/abs/10.1002/qj.477.

Gregory W, Bushuk M, Adcroft A, Zhang Y, Zanna L. 2023. Deep learning of systematic sea ice model errors from data assimilation increments. *Journal of Advances in Modeling Earth Systems* **15**(10): e2023MS003 757, doi:https://doi.org/10.1029/2023MS003757, URL https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/2023MS003757. E2023MS003757 2023MS003757.

Greydanus S, Dzamba M, Yosinski J. 2019. Hamiltonian neural networks. *Advances in neural information processing systems* **32**.

Griewank A, Walther A. 2008. *Evaluating derivatives: principles and techniques of algorithmic differentiation*. SIAM.

Grudzien C, Carrassi A, Bocquet M. 2018a. Asymptotic forecast uncertainty and the unstable subspace in the presence of additive model error. *SIAM/ASA Journal on Uncertainty Quantification* **6**(4): 1335–1363, doi:10.1137/17M114073X, URL https://doi.org/10.1137/17M114073X.

Grudzien C, Carrassi A, Bocquet M. 2018b. Chaotic dynamics and the role of covariance inflation for reduced rank kalman filters with model error. *Nonlinear Processes in Geophysics* **25**(3): 633–648, doi:10.5194/npg-25-633-2018, URL https://npg.copernicus.org/articles/25/633/2018/.

Gurumoorthy KS, Grudzien C, Apte A, Carrassi A, Jones CKRT. 2017. Rank deficiency of kalman error covariance matrices in linear time-varying system with deterministic evolution. *SIAM Journal on Control and Optimization* **55**(2): 741–759, doi:10.1137/15M1025839, URL https://doi.org/10.1137/15M1025839.

Haber E, Lensink K, Treister E, Ruthotto L. 2019. Imexnet: A forward stable deep neural network.

Haber E, Ruthotto L. 2017. Stable architectures for deep neural networks. *Inverse Problems* **34**(1): 014 004, doi:10.1088/1361-6420/aa9a90, URL https://doi.org/10.1088/1361-6420/aa9a90.

Härter FP, de Campos Velho HF. 2010. Multilayer perceptron neural network in a data assimilation scenario. *Engineering Applications of Computational Fluid Mechanics* **4**(2): 237–245, doi:10.1080/19942060.2010.11015313, URL https://doi.org/10.1080/19942060.2010.11015313.

Härter FP, de Campos Velho HF. 2012. Data assimilation procedure by recurrent neural network. *Engineering Applications of Computational Fluid Mechanics* **6**(2): 224–233, doi:10.1080/19942060.2012.11015417, URL https://doi.org/10.1080/19942060.2012.11015417.

Hatfield S, Chantry M, Dueben P, Lopez P, Geer A, Palmer T. 2021. Building tangent-linear and adjoint models for data assimilation with neural networks. *Journal of Advances in Modeling Earth Systems* **13**(9): e2021MS002 521, doi:https://doi.org/10.1029/2021MS002521, URL https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/2021MS002521. E2021MS002521 2021MS002521.

Hausdorff F. 1918. Dimension und äußeres maß. *Mathematische Annalen* **79**(1): 157–179.

He K, Zhang X, Ren S, Sun J. 2016. Deep residual learning for image recognition. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. pp. 770–778, doi:10.1109/CVPR.2016.90.

He Q, Barajas-Solano D, Tartakovsky G, Tartakovsky AM. 2020. Physics-informed neural networks for multiphysics data assimilation with application to subsurface transport. *Advances in Water Resources* **141**: 103 610, doi:https://doi.org/10.1016/j.advwatres.2020.103610, URL https://www.sciencedirect.com/science/article/pii/S0309170819311649.

Higgins I, Matthey L, Pal A, Burgess C, Glorot X, Botvinick M, Mohamed S, Lerchner A. 2016. beta-vae: Learning basic visual concepts with a constrained variational framework. In: *International conference on learning representations*.

Hinton G, Vinyals O, Dean J. 2015. Distilling the knowledge in a neural network. URL https://arxiv.org/abs/1503.02531.

Hinton GE, Salakhutdinov RR. 2006. Reducing the dimensionality of data with neural networks. *Science* **313**(5786): 504–507, doi:10.1126/science.1127647, URL https://www.science.org/doi/10.1126/science.1127647. Publisher: American Association for the Advancement of Science.

Hinton GE, Zemel R. 1993. Autoencoders, minimum description length and helmholtz free energy. *Advances in neural information processing systems* **6**.

Hochreiter S, Schmidhuber J. 1997. Long short-term memory. *Neural computation* **9**(8): 1735–1780.

Holton JR, Hakim Gregory J. 2004. *An introduction to dynamic meteorology*. Academic Press.

Hotelling H. 1933. Analysis of a complex of statistical variables into principal components. *Journal of Educational Psychology* **24**: 498–520, URL https://api.semanticscholar.org/CorpusID:144828484.

Houtekamer PL, Mitchell HL. 1998. Data assimilation using an ensemble kalman filter technique. *Monthly Weather Review* **126**(3): 796 – 811, doi:10.1175/1520-0493(1998)126⟨0796:DAUAEK⟩2.0.CO;2, URL https://journals.ametsoc.org/view/journals/mwre/126/3/1520-0493_1998_126_0796_dauaek_2.0.co_2.xml.

Hsieh WW, Tang B. 1998. Applying neural network models to prediction and data analysis in meteorology and oceanography. *Bulletin of the American Meteorological Society* **79**(9): 1855 – 1870, doi:10.1175/1520-0477(1998)079⟨1855:ANNMTP⟩2.0.CO;2, URL https://journals.ametsoc.org/view/journals/bams/79/9/1520-0477_1998_079_1855_annmtp_2_0_co_2.xml.

Hunt BR, Kostelich EJ, Szunyogh I. 2007. Efficient data assimilation for spatiotemporal chaos: A local ensemble transform kalman filter. *Physica D: Nonlinear Phenomena* **230**(1): 112–126, doi:https://doi.org/10.1016/j.physd.2006.11.008, URL https://www.sciencedirect.com/science/article/pii/S0167278906004647. Data Assimilation.

Ioffe S, Szegedy C. 2015. Batch normalization: accelerating deep network training by reducing internal covariate shift. In: *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15. JMLR.org, p. 448–456.

Ivakhnenko A, Lapa V. 1967. *Cybernetics and forecasting techniques*. Modern analytic and computational methods in science and mathematics, American Elsevier Publishing Company, ISBN 9780444000200, URL https://books.google.fr/books?id=rGFgAAAAMAAJ.

Ivakhnenko A, Lapa V, ENGINEERING PULISOE. 1965. *Cybernetic predicting devices*. JPRS 37, 803, Joint Publications Research Service [available from the Clearinghouse for Federal Scientific and Technical Information], URL https://books.google.fr/books?id=l38DHQAACAAJ.

Janjić T, McLaughlin D, Cohn SE, Verlaan M. 2014. Conservation of mass and preservation of positivity with ensemble-type kalman filter algorithms. *Monthly Weather Review* **142**(2): 755 – 773, doi:10.1175/MWR-D-13-00056.1, URL https://journals.ametsoc.org/view/journals/mwre/142/2/mwr-d-13-00056.1.xml.

Jazwinski A. 2007. *Stochastic processes and filtering theory*. Dover Books on Electrical Engineering Series, Dover Publications, ISBN 9780486462745, URL https://books.google.fr/books?id=4AqL3vE2J-sC.

Kale A, Altun O. 2023. Face age synthesis: A review on datasets, methods, and open research areas. *Pattern Recognition* **143**: 109 791, doi:https://doi.org/10.1016/j.patcog.2023.109791, URL https://www.sciencedirect.com/science/article/pii/S0031320323004892.

Kalman RE. 1960. A new approach to linear filtering and prediction problems .

Kaplan J, Yorke J. 1979. Functional differential equations and approximation of fixed points. *Lecture notes in mathematics* **730**: 204–227.

Keisler R. 2022. Forecasting global weather with graph neural networks.

Kelley HJ. 1960. Gradient theory of optimal flight paths. *Ars Journal* **30**(10): 947–954.

Kingma DP, Ba J. 2017. Adam: A method for stochastic optimization.

Kingma DP, Welling M. 2013. Auto-encoding variational bayes.

Klinker E, Rabier F, Kelly G, Mahfouf JF. 2000. The ecmwf operational implementation of four-dimensional variational assimilation. iii: Experimental results and diagnostics with operational configuration. *Quarterly Journal of the Royal Meteorological Society* **126**(564): 1191–1215, doi:https://doi.org/10.1002/qj.49712656417, URL https://rmets.onlinelibrary.wiley.com/doi/abs/10.1002/qj.49712656417.

Knuth DE, Moore RW. 1975. An analysis of alpha-beta pruning. *Artificial intelligence* **6**(4): 293–326.

Kochkov D, Yuval J, Langmore I, Norgaard P, Smith J, Mooers G, Klöwer M, Lottes J, Rasp S, Düben P, Hatfield S, Battaglia P, Sanchez-Gonzalez A, Willson M, Brenner MP, Hoyer S. 2024. Neural general circulation models for weather and climate.

Koopman BO. 1931. Hamiltonian systems and transformation in hilbert space. *Proceedings of the National Academy of Sciences* **17**(5): 315–318.

Kramer MA. 1991. Nonlinear principal component analysis using autoassociative neural networks. *AIChE Journal* **37**(2): 233–243, doi:https://doi.org/10.1002/aic.690370209, URL https://aiche.onlinelibrary.wiley.com/doi/abs/10.1002/aic.690370209.

Krizhevsky A, Sutskever I, Hinton GE. 2012. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems* **25**.

Krogh A, Hertz J. 1991. A simple weight decay can improve generalization. In: *Advances in Neural Information Processing Systems*, vol. 4, Moody J, Hanson S, Lippmann R (eds). Morgan-Kaufmann, URL https://proceedings.neurips.cc/paper_files/paper/1991/file/8eefcfdf5990e441f0fb6f3fad709e21-Paper.pdf.

Kuenzer C, Ottinger M, Wegmann M, Guo H, Wang C, Zhang J, Dech S, Wikelski M. 2014. Earth observation satellite sensors for biodiversity monitoring: Potentials and bottlenecks. *International Journal of Remote Sensing* **35**: 6599–6647, doi:10.1080/01431161.2014.964349.

Kullback S, Leibler RA. 1951. On information and sufficiency. *The annals of mathematical statistics* **22**(1): 79–86.

Kuptsov PV, Parlitz U. 2012. Theory and computation of covariant lyapunov vectors. *Journal of Nonlinear Science* **22**: 727 – 762, URL https://api.semanticscholar.org/CorpusID:14052716.

Laloyaux P, Bonavita M, CHRUST M, Gürol S. 2020. Exploring the potential and limitations of weak-constraint 4d-var. *Quarterly Journal of the Royal Meteorological Society* **146**, doi:10.1002/qj.3891.

Laloyaux P, Kurth T, Dueben PD, Hall D. 2022. Deep learning to estimate model biases in an operational nwp assimilation system. *Journal of Advances in Modeling Earth Systems* **14**(6): e2022MS003 016, doi:https://doi.org/10.1029/2022MS003016, URL https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/2022MS003016. E2022MS003016 2022MS003016.

Lam R, Sanchez-Gonzalez A, Willson M, Wirnsberger P, Fortunato M, Alet F, Ravuri S, Ewalds T, Eaton-Rosen Z, Hu W, Merose A, Hoyer S, Holland G, Vinyals O, Stott J, Pritzel A, Mohamed S, Battaglia P. 2023. Graphcast: Learning skillful medium-range global weather forecasting.

Lawless AS, Nichols NK, Boess C, Bunse-Gerstner A. 2008. Using model reduction methods within incremental four-dimensional variational data assimilation. *Monthly Weather Review* **136**(4): 1511 – 1522, doi:10.1175/2007MWR2103.1, URL https://journals.ametsoc.org/view/journals/mwre/136/4/2007mwr2103.1.xml.

Le Dimet FX, Talagrand O. 1986. Variational algorithms for analysis and assimilation of meteorological observations: theoretical aspects. *Tellus A* **38A**(2): 97–110, doi:https://doi.org/10.1111/j.1600-0870.1986.tb00459.x, URL https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1600-0870.1986.tb00459.x.

LeCun Y, Bengio Y, Hinton G. 2015. Deep learning. *nature* **521**(7553): 436.

LeCun Y, Boser B, Denker JS, Henderson D, Howard RE, Hubbard W, Jackel LD. 1989. Backpropagation applied to handwritten zip code recognition. *Neural computation* **1**(4): 541–551.

LeCun Y, Bottou L, Bengio Y, Haffner P. 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE* **86**(11): 2278–2324.

LeCun Y, Cortes C. 2005. The mnist database of handwritten digits. URL https://api.semanticscholar.org/CorpusID:60282629.

Legler S, Janjić T. 2022. Combining data assimilation and machine learning to estimate parameters of a convective-scale model. *Quarterly Journal of the Royal Meteorological Society* **148**(743): 860–874, doi:https://doi.org/10.1002/qj.4235, URL https://rmets.onlinelibrary.wiley.com/doi/abs/10.1002/qj.4235.

Legras B, Vautard R. 1996. A guide to liapunov vectors. In: *Predictability, Seminar Proceedings*, vol. 1. ECMWF, Reading, United-Kingdom, pp. 143–156.

Leith C. 1978. Objective methods for weather prediction. *Annual Review of Fluid Mechanics* **10**(1): 107–128.

Leutbecher M. 2019. Ensemble size: How suboptimal is less than infinity? *Quarterly Journal of the Royal Meteorological Society* **145**(S1): 107–128, doi:https://doi.org/10.1002/qj.3387, URL https://rmets.onlinelibrary.wiley.com/doi/abs/10.1002/qj.3387.

Li M, Zhang T, Chen Y, Smola AJ. 2014. Efficient mini-batch training for stochastic optimization. In: *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. pp. 661–670.

Li X, Xiao C, Cheng A, Lin H. 2022. Joint estimation of parameter and state with hybrid data assimilation and machine learning. doi:10.22541/au.164605938.86704099/v1, URL https://doi.org/10.22541/au.164605938.86704099/v1.

Li Z, Navon IM. 2001. Optimality of variational data assimilation and its relationship with the kalman filter and smoother. *Quarterly Journal of the Royal Meteorological Society* **127**(572): 661–683, doi:https://doi.org/10.1002/qj.49712757220, URL https://rmets.onlinelibrary.wiley.com/doi/abs/10.1002/qj.49712757220.

Liang J, Terasaki K, Miyoshi T. 2023. A machine learning approach to the observation operator for satellite radiance data assimilation. *Journal of the Meteorological Society of Japan. Ser. II* **101**(1): 79–95, doi:10.2151/jmsj.2023-005.

Lin HX, Jin J, van den Herik J. 2019. Air quality forecast through integrated data assimilation and machine learning. In: *International Conference on Agents and Artificial Intelligence*. pp. 787–793, URL https://api.semanticscholar.org/CorpusID:88497517.

Loh K, Omrani PS, van der Linden R. 2018. Deep learning and data assimilation for real-time production prediction in natural gas wells. doi:10.48550/arXiv.1802.05141, URL http://arxiv.org/abs/1802.05141.

Long J, Shelhamer E, Darrell T. 2015. Fully convolutional networks for semantic segmentation. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. pp. 3431–3440.

Lorenc AC. 1986. Analysis methods for numerical weather prediction. *Quarterly Journal of the Royal Meteorological Society* **112**(474): 1177–1194, doi:https://doi.org/10.1002/qj.49711247414, URL https://rmets.onlinelibrary.wiley.com/doi/abs/10.1002/qj.49711247414.

Lorenz EN. 1963. Deterministic nonperiodic flow. *Journal of Atmospheric Sciences* **20**(2): 130 – 141, doi:10.1175/1520-0469(1963)020⟨0130:DNF⟩2.0.CO;2, URL https://journals.ametsoc.org/view/journals/atsc/20/2/1520-0469_1963_020_0130_dnf_2_0_co_2.xml.

Lorenz EN. 1996. Predictability: A problem partly solved. In: *Proc. Seminar on predictability*, vol. 1. Reading, pp. 1–18.

Loshchilov I, Hutter F. 2019. Decoupled weight decay regularization.

Lucor D, Agrawal A, Sergent A. 2021. Physics-aware deep neural networks for surrogate modeling of turbulent natural convection. *arXiv preprint arXiv:2103.03565* .

Mahfouf JF, Rabier F. 2000. The ecmwf operational implementation of four-dimensional variational assimilation. ii: Experimental results with improved physics. *Quarterly Journal of the Royal Meteorological Society* **126**(564): 1171–1190, doi:https://doi.org/10.1002/qj.49712656416, URL https://rmets.onlinelibrary.wiley.com/doi/abs/10.1002/qj.49712656416.

Malardel S. 2022. *Fondamentaux de météorologie*. Cépaduès.

Mandel J, Bergou E, Gürol S, Gratton S, Kasanický I. 2016. Hybrid levenberg–marquardt and weak-constraint ensemble kalman smoother method. *Nonlinear Processes in Geophysics* **23**(2): 59–73, doi:10.5194/npg-23-59-2016, URL https://npg.copernicus.org/articles/23/59/2016/.

McCarthy J. 2007. What is artificial intelligence? https://www-formal.stanford.edu/jmc/whatisai.pdf.

McCulloch WS, Pitts W. 1943. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics* **5**: 115–133.

McNally T, Bonavita M, Thépaut JN. 2014. The role of satellite data in the forecasting of hurricane sandy. *Monthly Weather Review* **142**(2): 634 – 646, doi:https://doi.org/10.1175/MWR-D-13-00170.1, URL https://journals.ametsoc.org/view/journals/mwre/142/2/mwr-d-13-00170.1.xml.

Melinc B, Zaplotnik Z. 2024. 3d-var data assimilation using a variational autoencoder. *Quarterly Journal of the Royal Meteorological Society* **150**(761): 2273–2295, doi:https://doi.org/10.1002/qj.4708, URL https://rmets.onlinelibrary.wiley.com/doi/abs/10.1002/qj.4708.

Mikolov T, Chen K, Corrado G, Dean J. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781* .

Milletari F, Navab N, Ahmadi SA. 2016. V-net: Fully convolutional neural networks for volumetric medical image segmentation. In: *2016 fourth international conference on 3D vision (3DV)*. Ieee, pp. 565–571.

Minsky M, Papert S. 1969. *Perceptrons; an introduction to computational geometry*. MIT Press, ISBN 9780262630221, URL https://books.google.fr/books?id=Ow1OAQAAIAAJ.

Mitchell L, Carrassi A. 2015. Accounting for model error due to unresolved scales within ensemble kalman filtering. *Quarterly Journal of the Royal Meteorological Society* **141**(689): 1417–1428, doi:https://doi.org/10.1002/qj.2451, URL https://rmets.onlinelibrary.wiley.com/doi/abs/10.1002/qj.2451.

Mitchell TM. 1980. The need for biases in learning generalizations. Technical report, Rutgers University, New Brunswick, NJ, URL http://dml.cs.byu.edu/~cgc/docs/mldm_tools/Reading/Need%20for%20Bias.pdf.

Miyoshi T, Kondo K, Imamura T. 2014. The 10,240-member ensemble kalman filtering with an intermediate agcm. *Geophysical Research Letters* **41**(14): 5264–5271, doi:https://doi.org/10.1002/2014GL060863, URL https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1002/2014GL060863.

Mnih V, Kavukcuoglu K, Silver D, Rusu AA, Veness J, Bellemare MG, Graves A, Riedmiller M, Fidjeland AK, Ostrovski G, *et al.* 2015. Human-level control through deep reinforcement learning. *nature* **518**(7540): 529–533.

Moore B. 1981. Principal component analysis in linear systems: Controllability, observability, and model reduction. *IEEE transactions on automatic control* **26**(1): 17–32.

Moore EH. 1920. On the reciprocal of the general algebraic matrix. *Bulletin of the american mathematical society* **26**: 294–295.

Mozer M. 1995. A focused backpropagation algorithm for temporal pattern recognition. *Complex Systems* **3**.

Mücke NT, Boht'e SM, Oosterlee CW. 2024. The deep latent space particle filter for real-time data assimilation with uncertainty quantification. *ArXiv* **abs/2406.02204**, URL https://api.semanticscholar.org/CorpusID:270226381.

Nadler P, Arcucci R, Guo Y. 2020. A neural sir model for global forecasting. In: *Proceedings of the Machine Learning for Health NeurIPS Workshop*, *Proceedings of Machine Learning Research*, vol. 136, Alsentzer E, McDermott MBA, Falck F, Sarkar SK, Roy S, Hyland SL (eds). PMLR, pp. 254–266, URL https://proceedings.mlr.press/v136/nadler20a.html.

Nair V, Hinton GE. 2010. Rectified linear units improve restricted boltzmann machines. In: *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10. Omnipress: Madison, WI, USA, ISBN 9781605589077, p. 807–814.

Nguyen T, Brandstetter J, Kapoor A, Gupta JK, Grover A. 2023. Climax: A foundation model for weather and climate.

Nocedal J, Wright SJ. 2006. *Numerical optimization*. Springer: New York, NY, USA, 2e edn.

Ott E, Hunt BR, Szunyogh I, Zimin AV, Kostelich EJ, Corazza M, Eugenia Kalnay DP, Yorke JA. 2004. A local ensemble kalman filter for atmospheric data assimilation. *Tellus A: Dynamic Meteorology and Oceanography* **56**(5): 415–428, doi: 10.3402/tellusa.v56i5.14462, URL https://doi.org/10.3402/tellusa.v56i5.14462.

Pathak J, Subramanian S, Harrington P, Raja S, Chattopadhyay A, Mardani M, Kurth T, Hall D, Li Z, Azizzadenesheli K, Hassanzadeh P, Kashinath K, Anandkumar A. 2022. Fourcastnet: A global data-driven high-resolution weather model using adaptive fourier neural operators.

Pawar S, San O. 2021. Data assimilation empowered neural network parametrizations for subgrid processes in geophysical flows. *Phys. Rev. Fluids* **6**: 050 501, doi: 10.1103/PhysRevFluids.6.050501, URL https://link.aps.org/doi/10.1103/PhysRevFluids.6.050501.

Pearson K. 1901. Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* **2**(11): 559–572, doi:10.1080/14786440109462720, URL https://doi.org/10.1080/14786440109462720.

Pedlosky J. 1987. *Geophysical fluid dynamics*. Springer.

Penny SG, Smith TA, Chen TC, Platt JA, Lin HY, Goodliff M, Abarbanel HDI. 2022. Integrating recurrent neural networks with data assimilation for scalable data-driven state estimation. *Journal of Advances in Modeling Earth Systems* **14**(3): e2021MS002 843, doi:https://doi.org/10.1029/2021MS002843, URL https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/2021MS002843. E2021MS002843 2021MS002843.

Penrose R. 1955. A generalized inverse for matrices. In: *Mathematical proceedings of the Cambridge philosophical society*, vol. 51. Cambridge University Press, pp. 406–413.

Peyron M, Fillion A, Gürol S, Marchais V, Gratton S, Boudier P, Goret G. 2021. Latent space data assimilation by using deep learning. *Quarterly Journal of the Royal Meteorological Society* **147**(740): 3759–3777, doi:https://doi.org/10.1002/qj.4153, URL https://rmets.onlinelibrary.wiley.com/doi/abs/10.1002/qj.4153.

Plaut E. 2018. From principal subspaces to principal components with linear autoencoders. *CoRR* **abs/1804.10253**, URL http://arxiv.org/abs/1804.10253.

Pottenger WM, Freiberger PA, Swaine MR, Hemmendinger D. 2023. ”computer.” encyclopedia britannica. URL https://www.britannica.com/technology/computer.

Price I, Sanchez-Gonzalez A, Alet F, Ewalds T, El-Kadi A, Stott J, Mohamed S, Battaglia P, Lam R, Willson M. 2023. Gencast: Diffusion-based ensemble forecasting for medium-range weather.

Prince SJ. 2023. *Understanding deep learning*. MIT Press. https://udlbook.github.io/udlbook/.

Raanes PN, Carrassi A, Bertino L. 2015. Extending the square root method to account for additive forecast noise in ensemble methods. *Monthly*

Weather Review **143**(10): 3857 – 3873, doi:10.1175/MWR-D-14-00375.1, URL https://journals.ametsoc.org/view/journals/mwre/143/10/mwr-d-14-00375.1.xml.

Rabier F, McNally A, Andersson E, Courtier P, Undén P, Eyre J, Hollingsworth A, Bouttier F. 1998a. The ecmwf implementation of three-dimensional variational assimilation (3d-var). ii: Structure functions. *Quarterly Journal of the Royal Meteorological Society* **124**(550): 1809–1829, doi:https://doi.org/10.1002/qj.49712455003, URL https://rmets.onlinelibrary.wiley.com/doi/abs/10.1002/qj.49712455003.

Rabier F, Thépaut JN, Courtier P. 1998b. Extended assimilation and forecast experiments with a four-dimensional variational assimilation system. *Quarterly Journal of the Royal Meteorological Society* **124**(550): 1861–1887, doi:https://doi.org/10.1002/qj.49712455005, URL https://rmets.onlinelibrary.wiley.com/doi/abs/10.1002/qj.49712455005.

Raissi M, Perdikaris P, Karniadakis G. 2019. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics* **378**: 686–707, doi:https://doi.org/10.1016/j.jcp.2018.10.045, URL https://www.sciencedirect.com/science/article/pii/S0021999118307125.

Ramachandran P, Zoph B, Le QV. 2017. Searching for activation functions. *arXiv preprint arXiv:1710.05941* .

Randall DA. 1989. Cloud parameterization for climate modeling: Status and prospects. *Atmospheric Research* **23**(3): 345–361, doi:https://doi.org/10.1016/0169-8095(89)90025-2, URL https://www.sciencedirect.com/science/article/pii/0169809589900252.

Rasp S, Hoyer S, Merose A, Langmore I, Battaglia P, Russel T, Sanchez-Gonzalez A, Yang V, Carver R, Agrawal S, Chantry M, Bouallegue ZB, Dueben P, Bromberg C, Sisk J, Barrington L, Bell A, Sha F. 2024. Weatherbench 2: A benchmark for the next generation of data-driven global weather models.

Reddi SJ, Kale S, Kumar S. 2019. On the convergence of adam and beyond.

Redmon J, Divvala S, Girshick R, Farhadi A. 2016. You only look once: Unified, real-time object detection. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. pp. 779–788.

Redmon J, Farhadi A. 2018. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767* .

Reichstein M, Camps-Valls G, Stevens B, Jung M, Denzler J, Carvalhais N, Prabhat M. 2019. Deep learning and process understanding for data-driven earth system science. *Nature* **566**: 195, doi:10.1038/s41586-019-0912-1.

Ren S, He K, Girshick R, Sun J. 2015. Faster r-cnn: Towards real-time object detection with region proposal networks. In: *Advances in Neural Information Processing Systems*, vol. 28, Cortes C, Lawrence N, Lee D, Sugiyama M, Garnett R (eds). Curran Associates, Inc., URL https://proceedings.neurips.cc/paper_files/paper/2015/file/14bfa6bb14875e45bba028a21ed38046-Paper.pdf.

Robbins H, Monro S. 1951. A Stochastic Approximation Method. *The Annals of Mathematical Statistics* **22**(3): 400 – 407, doi:10.1214/aoms/1177729586, URL https://doi.org/10.1214/aoms/1177729586.

Robert C, Durbiano S, Blayo E, Verron J, Blum J, Le Dimet FX. 2005. A reduced-order strategy for 4d-var data assimilation. *Journal of Marine Systems* **57**(1): 70–82, doi:https://doi.org/10.1016/j.jmarsys.2005.04.003, URL https://www.sciencedirect.com/science/article/pii/S0924796305000680.

Robinson AJ, Fallside F. 1987. The utility driven dynamic error propagation network. Technical Report CUED/F-INFENG/TR.1, Engineering Department, Cambridge University, Cambridge, UK.

Ronneberger O, Fischer P, Brox T. 2015. U-net: Convolutional networks for biomedical image segmentation. In: *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, Navab N, Hornegger J, Wells WM, Frangi AF (eds). Springer International Publishing: Cham, ISBN 978-3-319-24574-4, pp. 234–241.

Rosenblatt F. 1958. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review* **65**(6): 386.

Rosenblatt F, *et al.* 1962. *Principles of neurodynamics: Perceptrons and the theory of brain mechanisms*, vol. 55. Spartan books Washington, DC.

Ruckstuhl Y, Janjić T, Rasp S. 2021. Training a convolutional neural network to conserve mass in data assimilation. *Nonlinear Processes in Geophysics* **28**(1):

111–119, doi:10.5194/npg-28-111-2021, URL https://npg.copernicus.org/articles/28/111/2021/.

Ruckstuhl YM, Janjić T. 2018. Parameter and state estimation with ensemble kalman filter based algorithms for convective-scale applications. *Quarterly Journal of the Royal Meteorological Society* **144**(712): 826–841, doi:https://doi.org/10.1002/qj.3257, URL https://rmets.onlinelibrary.wiley.com/doi/abs/10.1002/qj.3257.

Ruder S. 2017. An overview of gradient descent optimization algorithms.

Rumelhart DE, Hinton GE, Williams RJ. 1986. Learning representations by back-propagating errors. *nature* **323**(6088): 533–536.

Rumelhart DE, McClelland JL. 1987. *Learning internal representations by error propagation*, ch. Learning Internal Representations by Error Propagation. MIT Press, pp. 318–362.

Saad Y. 2003. *Iterative methods for sparse linear systems*. Society for Industrial and Applied Mathematics, second edn, doi:10.1137/1.9780898718003, URL https://epubs.siam.org/doi/abs/10.1137/1.9780898718003.

Saha S. 2018. A comprehensive guide to convolutional neural networks - the eli5 way. URL https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.

Sakov P, Bocquet M. 2018. Asynchronous data assimilation with the enkf in presence of additive model error. *Tellus A: Dynamic Meteorology and Oceanography* **70**(1): 1–7, doi:10.1080/16000870.2017.1414545, URL https://doi.org/10.1080/16000870.2017.1414545.

Sakov P, Haussaire JM, Bocquet M. 2018. An iterative ensemble kalman filter in the presence of additive model error. *Quarterly Journal of the Royal Meteorological Society* **144**(713): 1297–1309, doi:https://doi.org/10.1002/qj.3213, URL https://rmets.onlinelibrary.wiley.com/doi/abs/10.1002/qj.3213.

Serrurier M, Mamalet F, González-Sanz A, Boissin T, Loubes JM, del Barrio E. 2020. Achieving robustness in classification using optimal transport with hinge regularization.

Shafkat I. 2018. Intuitively understanding variational autoencoders. URL https://towardsdatascience.com/intuitively-understanding-variational-autoencoders-1bfe67eb5daf.

Shorten C, Khoshgoftaar T. 2019. A survey on image data augmentation for deep learning. *Journal of Big Data* **6**, doi:10.1186/s40537-019-0197-0.

Silva VLS, Heaney CE, Li Y, Pain CC. 2022. Data assimilation predictive GAN (DA-PredGAN) applied to a spatio-temporal compartmental model in epidemiology. *Journal of Scientific Computing* **94**(1): 25, doi:10.1007/s10915-022-02078-1, URL https://doi.org/10.1007/s10915-022-02078-1.

Simonyan K, Zisserman A. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* .

Smith PJ, Thornhill GD, Dance SL, Lawless AS, Mason DC, Nichols NK. 2013. Data assimilation for state and parameter estimation: application to morphodynamic modelling. *Quarterly Journal of the Royal Meteorological Society* **139**(671): 314–327, doi:https://doi.org/10.1002/qj.1944, URL https://rmets.onlinelibrary.wiley.com/doi/abs/10.1002/qj.1944.

Sommer M, Janjić T. 2018. A flexible additive inflation scheme for treating model error in ensemble kalman filters. *Quarterly Journal of the Royal Meteorological Society* **144**(716): 2026–2037, doi:https://doi.org/10.1002/qj.3254, URL https://rmets.onlinelibrary.wiley.com/doi/abs/10.1002/qj.3254.

Srivastava N, Hinton G, Krizhevsky A, Sutskever I, Salakhutdinov R. 2014. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research* **15**(56): 1929–1958, URL http://jmlr.org/papers/v15/srivastava14a.html.

Stensrud DJ. 2007. *Parameterization schemes: Keys to understanding numerical weather prediction models*. Cambridge University Press.

Strogatz SH. 1994. *Nonlinear dynamics and chaos: With applications to physics, biology, chemistry, and engineering*. Westview Press.

Sutton RS, Barto AG. 2018. *Reinforcement learning: An introduction*. MIT press.

Szegedy C, Liu W, Jia Y, Sermanet P, Reed S, Anguelov D, Erhan D, Vanhoucke V, Rabinovich A. 2015. Going deeper with convolutions. In: *2015 IEEE Conference* *on Computer Vision and Pattern Recognition (CVPR)*. IEEE Computer Society: Los Alamitos, CA, USA, pp. 1–9, doi:10.1109/CVPR.2015.7298594, URL https://doi.ieeecomputersociety.org/10.1109/CVPR.2015.7298594.

Tang H, Fu P, Sherman CS, Zhang J, Ju X, Hamon F, Azzolina NA, Burton-Kelly M, Morris JP. 2021a. A deep learning-accelerated data assimilation and forecasting workflow for commercial-scale geologic carbon storage. *International Journal of Greenhouse Gas Control* **112**: 103 488, doi:https://doi.org/10.1016/j.ijggc.2021.103488, URL https://www.sciencedirect.com/science/article/pii/S1750583621002401.

Tang M, Liu Y, Durlofsky LJ. 2020. A deep-learning-based surrogate model for data assimilation in dynamic subsurface flow problems. *Journal of Computational Physics* **413**: 109 456, doi:10.1016/j.jcp.2020.109456, URL https://linkinghub.elsevier.com/retrieve/pii/S0021999120302308.

Tang M, Liu Y, Durlofsky LJ. 2021b. Deep-learning-based surrogate flow modeling and geological parameterization for data assimilation in 3d subsurface flow. *Computer Methods in Applied Mechanics and Engineering* **376**: 113 636, doi:https://doi.org/10.1016/j.cma.2020.113636, URL https://www.sciencedirect.com/science/article/pii/S0045782520308215.

Tibshirani R. 1996. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)* **58**(1): 267–288, URL http://www.jstor.org/stable/2346178.

Trémolet Y. 2006. Accounting for an imperfect model in 4d-var. *Quarterly Journal of the Royal Meteorological Society* **132**(621): 2483–2504, doi:https://doi.org/10.1256/qj.05.224, URL https://rmets.onlinelibrary.wiley.com/doi/abs/10.1256/qj.05.224.

Trémolet Y. 2007. Model-error estimation in 4d-var. *Quarterly Journal of the Royal Meteorological Society* **133**(626): 1267–1280, doi:https://doi.org/10.1002/qj.94, URL https://rmets.onlinelibrary.wiley.com/doi/abs/10.1002/qj.94.

Trémolet Y, Fisher M. 2010. Weak constraint 4d-var. ecmwf feature article - from newsletter number 125 - autumn 2010. URL https://www.ecmwf.int/sites/default/files/elibrary/2010/17456-weak-constraint-4d-var.pdf.

Tuan Pham D, Verron J, Christine Roubaud M. 1998. A singular evolutive extended kalman filter for data assimilation in oceanography. *Journal of*

Marine Systems **16**(3): 323–340, doi:https://doi.org/10.1016/S0924-7963(97)00109-7, URL https://www.sciencedirect.com/science/article/pii/S0924796397001097.

Vallis GK. 2017. *Atmospheric and oceanic fluid dynamics*. Cambridge Univertiy Press.

Van der Maaten L, Hinton G. 2008. Visualizing data using t-sne. *Journal of machine learning research* **9**(11).

Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, Kaiser Ł, Polosukhin I. 2017. Attention is all you need. *Advances in neural information processing systems* **30**.

Verlaan M, Heemink A. 1997. Tidal flow forecasting using reduced rank square root filters. *Stochastic Hydrology and Hydraulics* **11**: 349–368, doi:10.1007/BF02427924.

Vincent P, Larochelle H, Bengio Y, Manzagol PA. 2008. Extracting and composing robust features with denoising autoencoders. In: *Proceedings of the 25th International Conference on Machine Learning*. pp. 1096–1103, doi:10.1145/1390156.1390294.

Vincent P, Larochelle H, Lajoie I, Bengio Y, Manzagol PA, Bottou L. 2010. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of machine learning research* **11**(12).

von Leibniz G, Child J, Gerhardt C. 1920. *The early mathematical manuscripts of leibniz: Translated from the latin texts published by carl immanuel gerhardt with critical and historical notes*. Open court publishing Company, ISBN 9780598818461, URL https://books.google.fr/books?id=bOIGAAAAYAAJ.

Wallace JM, Hobbs PV. 2006. 7 - atmospheric dynamics. In: *Atmospheric Science (Second Edition)*, Wallace JM, Hobbs PV (eds), Academic Press: San Diego, second edition edn, ISBN 978-0-12-732951-2, pp. 271–311, doi:https://doi.org/10.1016/B978-0-12-732951-2.50012-0, URL https://www.sciencedirect.com/science/article/pii/B9780127329512500120.

Wang X, Bishop CH, Julier SJ. 2004. Which is better, an ensemble of positive–negative pairs or a centered spherical simplex ensemble? *Monthly Weather Review* **132**(7): 1590 – 1605, doi:10.1175/1520-0493(2004)132⟨1590:WIBAEO⟩2.

0.CO;2, URL https://journals.ametsoc.org/view/journals/mwre/132/7/1520-0493_2004_132_1590_wibaeo_2.0.co_2.xml.

Watson DF. 1981. Computing the n-dimensional Delaunay tessellation with application to Voronoi polytopes. *The Computer Journal* **24**(2): 167–172, doi:10.1093/comjnl/24.2.167, URL https://doi.org/10.1093/comjnl/24.2.167.

Welch G, Bishop G, *et al.* 1995. An introduction to the kalman filter .

Werbos P. 1974. Beyond regression: New tools for prediction and analysis in the behavioral sciences. *PhD thesis, Committee on Applied Mathematics, Harvard University, Cambridge, MA* .

Werbos PJ. 1988. Generalization of backpropagation with application to a recurrent gas market model. *Neural Networks* **1**(4): 339–356, doi:https://doi.org/10.1016/0893-6080(88)90007-X, URL https://www.sciencedirect.com/science/article/pii/089360808890007X.

Wikner A, Pathak J, Hunt BR, Szunyogh I, Girvan M, Ott E. 2021. Using data assimilation to train a hybrid forecast system that combines machine-learning and knowledge-based components. *Chaos: An Interdisciplinary Journal of Nonlinear Science* **31**(5): 053 114, doi:10.1063/5.0048050, URL https://doi.org/10.1063/5.0048050.

Wu L, Cui P, Pei J, Zhao L, Guo X. 2023. Graph neural networks: Foundation, frontiers and applications. In: *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '23. Association for Computing Machinery: New York, NY, USA, p. 5831–5832, doi:10.1145/3580305.3599560, URL https://doi.org/10.1145/3580305.3599560.

Yoon H, Kadeethum T. 2022. Deep learning-based data assimilation in the latent space for real-time forecasting of geologic carbon storage. *SSRN Electronic Journal* URL https://api.semanticscholar.org/CorpusID:265027708.

Zeng Y, Janjić T. 2016. Study of conservation laws with the local ensemble transform kalman filter. *Quarterly Journal of the Royal Meteorological Society* **142**(699): 2359–2372, doi:https://doi.org/10.1002/qj.2829, URL https://rmets.onlinelibrary.wiley.com/doi/abs/10.1002/qj.2829.

Zeng Y, Janjić T, Ruckstuhl Y, Verlaan M. 2017. Ensemble-type kalman filter algorithm conserving mass, total energy and enstrophy. *Quarterly Journal of the*

Royal Meteorological Society **143**(708): 2902–2914, doi:https://doi.org/10.1002/qj.3142, URL https://rmets.onlinelibrary.wiley.com/doi/abs/10.1002/qj.3142.

Zhang T. 2004. Solving large scale linear prediction problems using stochastic gradient descent algorithms. In: *Proceedings of the twenty-first international conference on Machine learning*. p. 116.

Zou H, Hastie T. 2005. Regularization and Variable Selection Via the Elastic Net. *Journal of the Royal Statistical Society Series B: Statistical Methodology* **67**(2): 301–320, doi:10.1111/j.1467-9868.2005.00503.x, URL https://doi.org/10.1111/j.1467-9868.2005.00503.x.

Titre : Assimilation de données en espace latent par des techniques de deep learning

Mots clés : Assimilation de données, Apprentissage profond, Espace latent, Dynamique de substitution, Filtre de Kalman d'ensemble

Résumé : Cette thèse, située à l'intersection de l'assimilation de données (AD) et de l'apprentissage profond (AP), introduit un concept nouveau : l'assimilation de données en espace latent. Elle permet une réduction considérable des coûts de calcul et des besoins mémoire, tout en offrant le potentiel d'améliorer la précision des résultats.
Il existe de nombreuses façons d'intégrer l'apprentissage profond dans les algorithmes d'assimilation de données, chacune visant des objectifs différents (Loh et al., 2018; Tang et al., 2020; Laloyaux et al., 2020; Bonavita and Laloyaux, 2020; Brajard et al., 2020; Farchi et al., 2021b; Pawar and San, 2021; Leutbecher, 2019; Ruckstuhl et al., 2021; Lin et al., 2019; Deng et al., 2018; Cheng et al., 2024). Nous étendons davantage l'intégration de l'apprentissage profond, en repensant le processus même d'assimilation. Notre approche s'inscrit dans la suite des méthodes à espace réduit (Evensen,1994; Bishop et al., 2001; Hunt et al., 2007; Courtier, 2007; Gratton and Tshimanga, 2009; Gratton et al., 2013; Lawless et al., 2008; Cao et al., 2007), qui résolvent le problème d'assimilation en effectuant des calculs dans un espace de faible dimension. Ces méthodes à espace réduit ont été principalement développées pour une utilisation opérationnelle, car la plupart des algorithmes d'assimilation de données s'avèrent être excessivement coûteux, lorsqu'ils sont implémentés dans leur forme théorique originelle.
Notre méthodologie repose sur l'entraînement conjoint d'un autoencodeur et d'un réseau de neurone surrogate. L'autoencodeur apprend de manière itérative à représenter avec précision la dynamique physique considérée dans un espace de faible dimension, appelé espace latent. Le réseau surrogate est entraîné simultanément à apprendre la propagation temporelle des variables latentes. Une stratégie basée sur une fonction de coût chaînée est également proposée pour garantir la stabilité du réseau surrogate. Cette stabilité peut également être obtenue en implémentant des réseaux surrogate Lipschitz.
L'assimilation de données à espace réduit est fondée sur la théorie de la stabilité de Lyapunov qui démontre mathématiquement que, sous certaines hypothèses, les matrices de covariance d'erreur de prévision et a posteriori se conforment asymptotiquement à l'espace instable-neutre (Carrassi et al., 2022), qui est de dimension beaucoup plus petite que l'espace d'état. Alors que l'assimilation de données en espace physique consiste en des combinaisons linéaires sur un système dynamique non linéaire, de grande dimension et potentiellement multi-échelle, l'assimilation de données latente, qui opère sur les dynamiques internes sous-jacentes, potentiellement simplifiées, est davantage susceptible de produire des corrections significatives.
La méthodologie proposée est éprouvée sur deux cas tests : une dynamique à 400 variables - construite à partir d'un système de Lorenz chaotique de dimension 40 -, ainsi que sur le modèle quasi-géostrophique de la librairie OOPS (Object-Oriented Prediction System). Les résultats obtenus sont prometteurs.

Title: Latent space data assimilation by using deep learning

Key words: Data Assimilation, Deep Learning, Latent space, Surrogate dynamics, Ensemble Kalman Filter

Abstract: This thesis, which sits at the crossroads of data assimilation (DA) and deep learning (DL), introduces latent space data assimilation, a novel data-driven framework that significantly reduces computational costs and memory requirements, while also offering the potential for more accurate data assimilation results.
There are numerous ways to integrate deep learning into data assimilation algorithms, each targeting different objectives (Loh et al., 2018; Tang et al., 2020; Laloyaux et al., 2020; Bonavita and Laloyaux, 2020; Brajard et al., 2020; Farchi et al., 2021b; Pawar and San, 2021; Leutbecher, 2019; Ruckstuhl et al., 2021; Lin et al., 2019; Deng et al., 2018; Cheng et al., 2024). We extend the integration of deep learning further by rethinking the assimilation process itself. Our approach aligns with reduced-space methods (Evensen,1994; Bishop et al., 2001; Hunt et al., 2007; Courtier, 2007; Gratton and Tshimanga, 2009; Gratton et al., 2013; Lawless et al., 2008; Cao et al., 2007), which solve the assimilation problem by performing computations within a lower-dimensional space. These reduced-space methods have been developed primarily for operational use, as most data assimilation algorithms are prohibitively costly, when implemented in their full theoretically form.
Our methodology is based on the joint training of an autoencoder and a surrogate neural network. The autoencoder iteratively learns how to accurately represent the physical dynamics of interest within a low-dimensional space, termed latent space. The surrogate is simultaneously trained to learn the time propagation of the latent variables. A chained loss function strategy is also proposed to ensure the stability of the surrogate network. Stability can also be achieved by implementing Lipschitz surrogate networks.
Reduced-space data assimilation is underpinned by Lyapunov stability theory, which mathematically demonstrates that, under specific hypotheses, the forecast and posterior error covariance matrices asymptotically conform to the unstable-neutral subspace (Carrassi et al., 2022), which is of much smaller dimension than the full state space. While full-space data assimilation involves linear combinations within a high-dimensional, nonlinear, and possibly multi-scale dynamic environment, latent data assimilation, which operates on the core, potentially disentangled and simplified dynamics, is more likely to result in impactful corrections.
We tested our methodology on a 400-dimensional dynamics - built upon a chaotic Lorenz96 system of dimension 40 -, and on the quasi-geostrophic model of the Object-Oriented Prediction System (OOPS) framework. We obtained promising results.