Introduction to the principles and methods of data assimilation in the geosciences

Lecture notes
Master M2 MOCIS & WAPE
École des Ponts ParisTech

Revision 0.52

Marc Bocquet and Alban Farchi
CEREA, École des Ponts and EdF R&D

14 January 2014 - 17 February 2014
based on an update of the 2004-2014 lecture notes in French
Chapter 6 and 7 added in February 2023
Last revision: 20 January 2025

Contents

Synopsis

In the first part of the lecture notes, the statistical tools that represent the foundations of data assimilation are introduced. These tools are closely related to those of ***estimation theory*** and to those of ***optimal control***. The basic concept of ***statistical interpolation*** will be introduced and the pursued goals will be clarified. Under certain conditions, statistical interpolation becomes the so-called ***optimal interpolation***, that entirely relies on the method known as ***BLUE*** (that stands for Best Linear Unbiased Estimator). The BLUE analysis turns out to often be equivalent to a variational problem which consists in minimising a least squares functional.

Time is easily incorporated into the optimal interpolation approach, using cycles of optimal interpolation, which yields the method known as ***3D-Var***. A more sophisticated sequential extension in time is known as the ***Kalman filter***. Under certain conditions, a variational principle equivalent to the Kalman filter can be derived. It is meant to minimise a functional defined over a four-dimensional space (3 for space and 1 for time). This method is known as ***4D-Var***.

In the second part of the lecture notes, key probabilistic and ensemble data assimilation techniques are presented. The hope is that the uncertainty quantification be better achieved with these methods, while, even though the adjoint of the evolution model is not used, non-linearity could be better taken into account. We will explain what the probabilistic Bayesian standpoint to data assimilation is and introduce Bayes' rule. We will sketch the basics of the ***particle filter***, which is essentially a Monte-Carlo implementation of Bayes' rule. Finally we will introduce the ensemble Kalman filter (***EnKF***) which is the practical and efficient adaptation of the Kalman filter for high-dimension applications, such as in the geosciences. To make it viable, one needs a few clever tricks whose explanation together with simple implementations are given.

In the third and last part (added in 2022-2023), we introduce ***machine learning*** and in particular ***deep learning*** tools which can be key in the numerical solution of optimisation problems, and hence of data assimilation problems. Our specific focus here is on the data assimilation challenge consisting in learning the dynamics of a chaotic model through the imperfect observation of the state of the dynamical system.

Textbooks and lecture notes

One of the father of modern data assimilation was Roger Daley, who wrote the first text book on the subject (Daley, 1993). Crystal-clear lecture notes and review articles that have been written by researchers with significant contributions to the subject offer a nice first reading on the topic of data assimilation (Talagrand, 1997; Bouttier, 1997). Furthermore, detailed lecture notes are available on the web (in particular Todling, 1999). A textbook on data assimilation and predictability in meteorology has been written by one of the main researcher in the field, Eugenia Kalnay (Kalnay, 2003), and is filled with nice insights, ideas and explanations. For a clarification on the concept of errors in the field of data assimilation, Cohn (1997) is the recommended review article. For an exposition of conventions and notations in the field, as well as a review of the methods, one is referred to Ide et al. (1999). Other highly recommended but more specific and more recent textbooks are Evensen (2009); Lahoz et al. (2010); Blayo et al. (2015); Fletcher (2017). Others recommended ones but a more mathematical emphasis are Reich and Cotter (2015); Law et al. (2015).

I highly recommend the recent textbook on data assimilation by Asch et al. (2016). I am co-author of the book and I am proud to mention that these notes inspired several chapters of the book. The first part of the book is for beginners (and covers the first three chapters of these notes); the second part covers more advanced topic (including chapter 5 of these notes) while the last part gives many illustrated examples of the use of data assimilation in many fields.

Finally, the review articles Carrassi et al. (2018); Janjić et al. (2018) offer a modern overview on data assimilation, while being shorter than the textbooks.

Acknowledgements

Many thanks to the students of the ENSTA ParisTech B10.2 & École des Ponts ParisTech ADOMO course, and later to the students of the OACOS/WAPE, then MOCIS/WAPE master program, and for their feedback on the lectures notes. A special thank to Jean-Matthieu Haussaire for his detailed feedback on the English version of the lecture notes, and to the PhD students who attended the course, in particular Rémi Gaillard. My gratitude goes to Bruno Sportisse who initiated this course, to Laurent Mortier who supported and promoted the course for more than 10 years through ENSTA or through OACOS/WAPE, to Philippe Courtier, Vladimir Zeitlin, Laurence Picon, and Claude Basdevant who believed that a modern geosciences master program should incorporate not only a modelling course but also a proper introduction to data assimilation.

Part I

Methods of data assimilation

Chapter 1

Statistical interpolation

1.1 Introduction

How does one forecast the state of complex physical systems such as the atmosphere?

Let us take the concrete example of the temperature in Paris. Suppose daily temperature has been measured over the city in the past few days. One wishes to forecast the daily temperature in the forthcoming days. A simple procedure consists in extrapolating past and already monitored temperatures to the coming days, assuming that the historical temperature curve is smooth enough. Or one can rely on a statistical and historical database that gives the average temperature for a given day of the year. This was the practice before the era of numerical computing because the procedure is computationally cheap.

Another approach consists in using the knowledge one has on the dynamics of the system by simulating the evolution of the temperature in the days to come. This is nowadays enabled by considerable computing power. This requires to initialise the numerical ***model*** with the latest recorded temperatures.

However, these methods appear to have serious limitations and soon enough the forecasted temperature curve diverges from the true temperature curve (Figure 1.1), because of the fundamentally poor predictability of the meteorological system.

An optimal procedure would be to combine the most exhaustive theoretical and observational knowledge. In our example, this means initialising the dynamics of the system using all the past and present observations and not only the latest.

Data assimilation is defined as the set of statistical techniques that allows to improve the knowledge of the past, present or future system states, jointly using experimental data and the theoretical (a priori) knowledge on the system.

Statistical interpolation is one of the most simple techniques offering a solution to this problem. Even though the technique is elementary (basically equivalent to a linear regression), its implementation on complex high-dimensional systems is not straightforward.

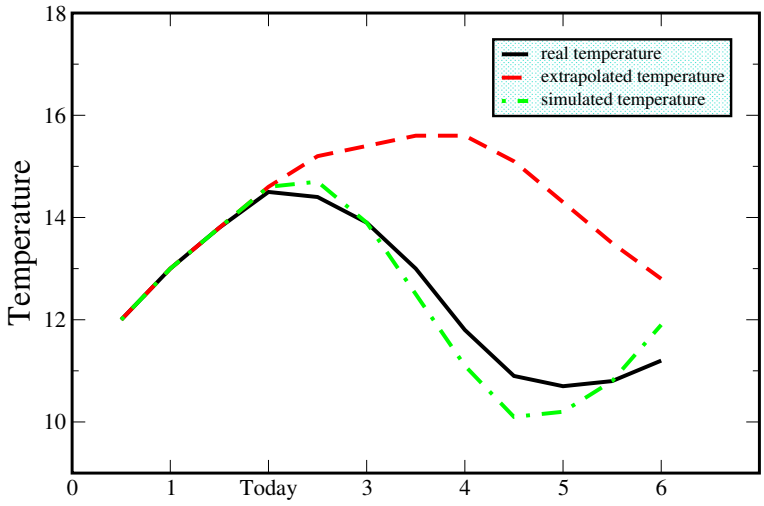


Figure 1.1: True, extrapolated and simulated daily-average temperature curves.

1.1.1 Representation of the physical system

The state of the system is represented by a vector $\mathbf{x}$ composed of scalar entries. For instance, each one of these entries can stand for the value of temperature of an air parcel at precise coordinates (latitude, longitude, altitude) at a given well defined date.

In principle, a comprehensive description of the system state supposes that $\mathbf{x}$ is a continuous vector field of components $x_\alpha(\overrightarrow{OM}, \tau)$ rather than a finite-dimensional vector field. For each location and time $(\overrightarrow{OM}, \tau)$, $\mathbf{x}$ takes value in a vector space, the components of which are indexed by α and can represent temperature, wind speed, moisture, pressure, gas concentrations, etc.

With a view to numerical computations, it is necessary to represent this continuous field in a limited-memory computer and hence to discretise it. Formally, one needs an operator Π that projects the infinite dimensional space of the vector field to the finite-dimensional space of its numerical representation. The image of $\mathbf{x}$ through Π is denoted $\mathbf{x}^{\mathrm{t}}$, with $\mathbf{x}^{\mathrm{t}} \in \mathbb{R}^{N_{\mathrm{x}}}$. The t index refers to ***truth***, the *true* state of the system. It is obviously an abuse of language since $\mathbf{x}^{\mathrm{t}}$ is only a projection of the true continuous state. For instance, $\mathbf{x}^{\mathrm{t}}$ could be the average of field $\mathbf{x}$ within a cell of a grid used for the numerical representation.

However, it is in practice impossible to know the exact true value of the entries of $\mathbf{x}^{\mathrm{t}}$, as much as it is impossible to know those of $\mathbf{x}$. Therefore, our goal is to estimate this vector. The result of the estimation procedure for $\mathbf{x}^{\mathrm{t}}$ is a vector of the same dimension N_{x}, denoted $\mathbf{x}^{\mathrm{a}}$, where a refers to ***analysis***. If the outcome of an earlier or present time partial analysis is used as the starting point of a new analysis, this vector is denoted $\mathbf{x}^{\mathrm{b}} \in \mathbb{R}^{N_{\mathrm{x}}}$ where b stands for ***background***. It represents some prior knowledge meant to be used in the subsequent analysis.

An estimation system needs to feed on observations of the real system in order to reduce the errors committed in the estimation of the system state. These observations can be performed in any location and time. They can even be averages of the state variables

when the instrument resolution is coarse. In realistic systems, these locations are not necessarily collocation points used in the numerical discretisation of the system computer representation. By interpolating, extrapolating and filtering, one should be able to provide a data set of size N_{y} which is related by a map to all or part of the entries of $\mathbf{x}^{\mathrm{t}}$; $\mathbf{y} \in \mathbb{R}^{N_{\mathrm{y}}}$ will denote the vector of observations.

1.1.2 The observational system

Instruments to monitor the atmosphere

An ambitious data assimilation system requires a lot of observational data. Here, we give an overview of the ***global observational system*** used by the numerical weather prediction operational centres.

Fundamentally, there are two types of observations. The first set gathers the conventional observations. Today, they are far less abundant than the remote sensing observations, especially those delivered by space-born instruments (on satellites). However, the conventional observations are less redundant and, in general, more precise. Besides, they offer a direct, often *in situ*, sampling of the physical system as opposed to remote sensing observations, such as radiances. In 2014, 99 % of the data processed by the European Centre for Medium-Range Weather Forecast (ECMWF) originated from satellites but *only* 91.5 % of the observations that are actually used in the analysis were space-born.

These conventional observations include:

- the measurements from synoptical stations (SYNOP network),
- measurements from commercial aircraft on regular flights,
- measurements from the ocean buoys and measurements from the commercial ships,
- measurements from the sounding balloons (radiosounding),
- LIDAR measurements (active sounding by laser ray from the ground),
- radar measurements, that enables to locate water content, precipitations.

The second set of observervations correspond to the satellite measurements that emerged in the 1970's. Since the end of the 1970's until a few years from now, the total number of measurements acquired at each round of observation had increased exponentially. In 2014, the ECMWF processed 70×10^6 scalar data per day, but only 3.6×10^6 scalar data were actually assimilated.

From the data assimilation standpoint, satellite has enabled the use of a considerable amount of data. It has also enabled a much better coverage of the globe, and especially in the southern hemisphere mostly covered by oceans. Significant progress has been recorded when the monitoring of the southern hemisphere by satellite was substantially improved.

1.1.3 Error modelling

Modelling the observation process and the observational error

In practice, the knowledge of $\mathbf{y}$ only gives partial and flawed information about the state vector $\mathbf{x}^{\mathrm{t}}$. There is a chain of transformations between $\mathbf{y}$ and $\mathbf{x}^{\mathrm{t}}$ needed to relate them. Some of these transformations are accurate; others are approximate and erroneous. Let us have a look at this chain.

We have already mentioned the fact that only part of the state vector can be probed by observation. For instance, in oceanography, satellite sounders provide data on the sea surface height, and hence provide information on the sea motion at the surface, but it is much more difficult to get marine data in the depths. The lack of observations or observability is not a source of error *per se*; it is merely the signature of an underdetermination of the system state (roughly $N_{\mathrm{y}} \ll N_{\mathrm{x}}$). For the ECMWF meteorological model (the IFS) and for a data assimilation cycle, vector $\mathbf{x}^{\mathrm{t}}$ is of size $N_{\mathrm{x}} = 2 \times 10^9$[1] whereas the observation vector $\mathbf{y}$ is of dimension $N_{\mathrm{y}} = 2 \times 10^7$!

To complicate the processing of observations, there is not always an immediate relationship between the state variables of $\mathbf{x}^{\mathrm{t}}$ and the observations of $\mathbf{y}$. For instance, radiances measured by satellite instruments depend on the total vertical atmospheric column. The entries of $\mathbf{y}$ are related to the system state only through linear combinations or nonlinear functions of the variables of $\mathbf{x}^{\mathrm{t}}$. Hence, there is a quite involved map that relates $\mathbf{x}^{\mathrm{t}}$ to $\mathbf{y}$.

The underdetermination is not the only source of mismatch between the observations and the system state. These errors can be classified into ***instrumental errors*** and ***representativeness*** errors.

The most obvious usually well identified and characterised, source of error is the ***instrumental error*** induced by the measurement process. It affects the value of $\mathbf{y}$ additively or/and multiplicatively. Knowing the true and continuous state of the system $\mathbf{x}$, one wishes to build the vector of the observations meant to represent $\mathbf{y}$. We shall denote this vector $h\left[\mathbf{x}\right]$, whose outcome is the construction of the map h. In the absence of instrumental error, one has $\mathbf{y} = h\left[\mathbf{x}\right]$. But in the presence of (additive) instrumental error $\mathbf{e}^{\mu} \in \mathbb{R}^{N_{\mathrm{y}}}$, we should write instead

$$\mathbf{y} = h\left[\mathbf{x}\right] + \mathbf{e}^{\mu}. \tag{1.1}$$

However, in practice, one can only use $\mathbf{x}^{\mathrm{t}}$, the numerical counterpart of $\mathbf{x}$. It is therefore useful to build the vector of observations that can be compared to $\mathbf{x}$, using the true discrete state of the system. This somehow different observation map can be written $H[\mathbf{x}^{\mathrm{t}}]$. The map H is known as the ***observation operator***. It includes several transformations such as projections, but also interpolations made necessary by the information lost in the projection by Π. This is formally written as

$$\begin{aligned} \mathbf{y} &= h\left[\mathbf{x}\right] + \mathbf{e}^{\mu} \\ &\triangleq H[\mathbf{x}^{\mathrm{t}}] + \mathbf{e}^{\mathrm{r}} + \mathbf{e}^{\mu}, \end{aligned} \tag{1.2}$$

[1] Essentially T1279L137 since July 2013; curently Cycle 49r1 from November 2024 with an O(1280) Gaussian grid with cell size of about 9 km. See https://confluence.ecmwf.int/display/FCST/Changes+to+the+forecasting+system for the latest.

where

$$\mathbf{e}^{\mathrm{r}} \triangleq h\left[\mathbf{x}\right] - H[\mathbf{x}^{\mathrm{t}}] = h\left[\mathbf{x}\right] - H[\Pi\mathbf{x}] = \left(h - H \circ \Pi\right)\left[\mathbf{x}\right], \tag{1.3}$$

is the ***representativeness, or representation error***. The symbol $\triangleq$ signals a definition. This construct is summarised by the schematic in Fig. 1.2. It can be condensed into

$$\mathbf{y} = H[\mathbf{x}^{\mathrm{t}}] + \mathbf{e}^{\mathrm{o}}, \tag{1.4}$$

with $\mathbf{e}^{\mathrm{o}} \triangleq \mathbf{e}^{\mathrm{r}} + \mathbf{e}^{\mu}$. Here, we have assumed that h and H are well known. But an additional source of error could be an imperfect knowledge of the map H. In that case, the error in the modelling of H is called ***model error***.

Error statistics

We first assume that the observation error has no ***bias***. This means that $\mathbb{E}[\mathbf{e}^{\mathrm{o}}] = \mathbf{0}$, where $\mathbb{E}$ is the expectation operator. If, on the contrary, there is a bias, *i.e.* $\mathbf{b} \triangleq \mathbb{E}[\mathbf{e}^{\mathrm{o}}] \neq \mathbf{0}$, it is often possible to diagnose it and to subtract its value from $\mathbf{e}^{\mathrm{o}}$, so as to make the corrected error unbiased, *i.e.* $\mathbb{E}[\mathbf{e}^{\mathrm{o}} - \mathbf{b}] = \mathbf{0}$. Let us introduce $\mathbf{R}$, the observation error covariance matrix defined by

$$[\mathbf{R}]_{ij} = \mathbb{E}\left[[\mathbf{e}^{\mathrm{o}}]_i \, [\mathbf{e}^{\mathrm{o}}]_j\right]. \tag{1.5}$$

This is a symmetric matrix of dimension $\mathbb{R}^{N_{\mathrm{y}} \times N_{\mathrm{y}}}$. Moreover, we assumed it to be positive definite (*i.e.* $\mathbf{e}^{\top}\mathbf{R}\mathbf{e} > 0$ for all $\mathbf{e} \in \mathbb{R}^{N_{\mathrm{y}}}$ different from $\mathbf{0}$), which implies that it is invertible.

In the following, we shall assume that $\mathbf{R}$ is known. Practically, it is important to either know it or to have a good estimation of it; the quality of the analysis depends on it. Quite often $\mathbf{R}$ is diagonal or assumed so, which means that the observations are statistically independent from one another. In practice, correlations are nevertheless possible: temporal and spatial correlation for the satellite measurements, etc.

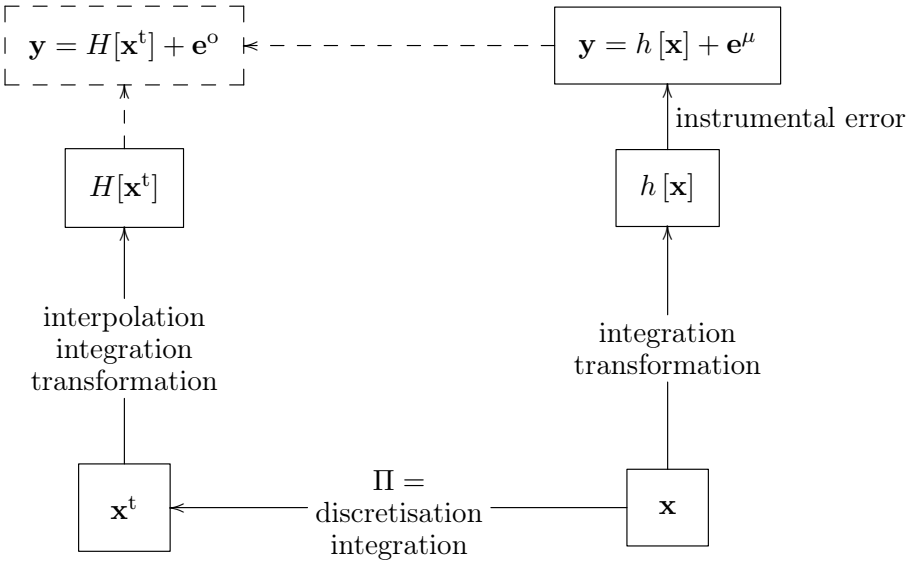


Figure 1.2: Breaking down of the observation process and its attached errors.

Background and analysis errors

The background error is defined by

$$\mathbf{e}^{\mathrm{b}} = \mathbf{x}^{\mathrm{b}} - \mathbf{x}^{\mathrm{t}}. \tag{1.6}$$

This is a measure of the discrepancy between an a priori estimate (before the analysis) and the (unknown) truth. This leads to the definition of the background error covariance matrix $\mathbf{B}$:

$$[\mathbf{B}]_{ij} = \mathbb{E}\left[[\mathbf{e}^{\mathrm{b}}]_i\,[\mathbf{e}^{\mathrm{b}}]_j\right]. \tag{1.7}$$

This is a symmetric matrix of dimension $\mathbb{R}^{N_{\mathrm{x}}\times N_{\mathrm{x}}}$. Moreover, it is assumed to be positive definite, hence invertible. It is assumed that $\mathbf{e}^{\mathrm{b}}$ has no bias, *i.e.* $\mathbb{E}[\mathbf{e}^{\mathrm{b}}] = \mathbf{0}$, or that the bias has been subtracted.

The analysis error is defined by

$$\mathbf{e}^{\mathrm{a}} = \mathbf{x}^{\mathrm{a}} - \mathbf{x}^{\mathrm{t}}. \tag{1.8}$$

It defines the gap between the outcome $\mathbf{x}^{\mathrm{a}}$ of the analysis process and the (unknown) truth $\mathbf{x}^{\mathrm{t}}$. It leads to the definition of the analysis error covariance matrix $\mathbf{P}^{\mathrm{a}}$ as

$$[\mathbf{P}^{\mathrm{a}}]_{ij} = \mathbb{E}\left[[\mathbf{e}^{\mathrm{a}}]_i\,[\mathbf{e}^{\mathrm{a}}]_j\right]. \tag{1.9}$$

which is a symmetric matrix of dimension $\mathbb{R}^{N_{\mathrm{x}}\times N_{\mathrm{x}}}$.

Furthermore, we shall assume that $\mathbf{e}^{\mathrm{o}}$ and $\mathbf{e}^{\mathrm{b}}$ are uncorrelated. There is no fundamental reason why the observation of a system, seen as a stochastic process, be correlated to a prior knowledge of the same system. This seems reasonable even in a realistic context. However, if the background has been obtained from observations of the same type, possibly with the same instruments, this assumption could been breached.

Model error

In the field of geophysical data assimilation, an evolution model Φ relate $\mathbf{x}(\tau+1)$ to $\mathbf{x}(\tau)$, system states at times $\tau+1$ and τ:

$$\mathbf{x}(\tau+1) = \Phi[\mathbf{x}(\tau)]. \tag{1.10}$$

In meteorology, this model could be the integral operator, or ***resolvent***, of the so-called ***primitive*** equations (momentum equations, air mass conservation equation, water mass conservation equation, state equation for the air (perfect gas), conservation of energy (first law of thermodynamics)). In atmospheric chemistry data assimilation, it could correspond to the resolvent of the transport and fate equations of chemical gaseous, particulate and aerosol species.

At first, we assume that this transition operator Φ is known. It is a faithful representation of the dynamics of the real system. But, again, a numerical representation implies a discretisation: $\mathbf{x}^{\mathrm{t}} = \Pi\mathbf{x}$. The propagation Eq. (1.10) has to be projected using Π, which yields

$$\begin{aligned}\mathbf{x}^{\mathrm{t}}(\tau+1) &= \Pi\mathbf{x}(\tau+1) = \Pi\Phi[\mathbf{x}(\tau)]\\ &\triangleq M[\mathbf{x}^{\mathrm{t}}(\tau)] + \mathbf{e}^{\mathrm{m}},\end{aligned} \tag{1.11}$$

where the ***model error*** $\mathbf{e}^{\mathrm{m}}$ is given by

$$\mathbf{e}^{\mathrm{m}} \triangleq \Pi\Phi[\mathbf{x}(\tau)] - M[\mathbf{x}^{\mathrm{t}}(\tau)] = \Pi\Phi[\mathbf{x}(\tau)] - M[\Pi\mathbf{x}(\tau)] = (\Pi \circ \Phi - M \circ \Pi)\,[\mathbf{x}(\tau)]. \quad (1.12)$$

This kind of model error belongs to the class of the ***representativeness (or representation) errors***.

However, because the system that is modelled is complex and because of the necessary approximations in the derivation and implementation of the model, the numerical model Φ has to be imperfect, let alone M. Therefore, the model error $\mathbf{e}^{\mathrm{m}}$ is actually the sum of a representativeness error as already identified, and of an error that characterises the imperfection of the model. This latter error is distinct from the modelling error in the observation operator although they are similar in nature.

1.1.4 The estimation problem

The goal is to study a physical system described by the vector state $\mathbf{x}^{\mathrm{t}}$. One assumes that our best estimation of the system state is $\mathbf{x}^{\mathrm{b}}$. This background is likely to result from an earlier data assimilation analysis or from an earlier statistical analysis. In the absence of any other source of information, it sums up to our best estimation of the system state. Observations $\mathbf{y}$ that are performed on the system brings in new information through the H operator. Moreover, ideally, one assumes to know the statistics of the observation error $\mathbf{e}^{\mathrm{o}}$ up to second-order moments $(\mathbb{E}[\mathbf{e}^{\mathrm{o}}], \mathbf{R})$. We also assume that we know the statistics for the background error $\mathbf{e}^{\mathrm{b}}$ up to second-order moments $(\mathbb{E}[\mathbf{e}^{\mathrm{b}}], \mathbf{B})$.

When new observations are available, we are in a position to improve our estimation and yield $\mathbf{x}^{\mathrm{a}}$, exploiting the information contained in these new observations as well as the background $\mathbf{x}^{\mathrm{b}}$. We are also interested in evaluating the error $\mathbf{e}^{\mathrm{a}}$ committed in the analysis, or its statistics.

There are of course many ways to define an analysis. Yet, we are interested in the best possible analysis, *i.e.* an analysis that would minimise the magnitude of the analysis error, for instance by minimising the trace of the analysis error covariance matrix $\mathrm{Tr}(\mathbf{P}^{\mathrm{a}})$.

1.2 Statistical interpolation

1.2.1 An Ansatz for the estimator

The linear hypothesis

Here, we shall assume that the observation operator H is linear. It can be seen as a matrix in $\mathbb{R}^{N_{\mathrm{y}} \times N_{\mathrm{x}}}$. It will be denoted $\mathbf{H}$, which indicates that the operator is linear, or that it is the ***tangent linear*** operator[2] of the nonlinear operator H : $\mathbf{H} = H'$.

Let us define what a tangent linear operator is as it plays an important role. If the observation operator is the map: $\mathbf{x} \mapsto \mathbf{y} = H[\mathbf{x}]$, the related tangent linear operator is

[2] also called the *Jacobian* matrix.

defined by the expansion of the operator around $\mathbf{x}$; for $i = 1, \ldots, N_\mathrm{y}$

$$\delta y_i = \sum_{j=1}^{N_\mathrm{x}} \frac{\partial H_i}{\partial x_j}_{|\mathbf{x}} \delta x_j. \tag{1.13}$$

The tangent linear operator is the linear operator whose action is defined by the matrix $\mathbf{H}$, of entries

$$[\mathbf{H}]_{ij} = \frac{\partial H_i}{\partial x_j}_{|\mathbf{x}}. \tag{1.14}$$

If H is linear, *i.e.* $H[\mathbf{x}] = \mathbf{H}\mathbf{x}$, then $H' = \mathbf{H}$; the tangent linear identifies with the original observation operator. Consequently, the tangent linear operator $H' = \mathbf{H}$ depends on $\mathbf{x}$ if and only if H is nonlinear.

A simple but nevertheless non-trivial Ansatz for the estimate $\mathbf{x}^\mathrm{a}$ consists in choosing for the analysis a vector of the form

$$\mathbf{x}^\mathrm{a} = \mathbf{L}\mathbf{x}^\mathrm{b} + \mathbf{K}\mathbf{y}, \tag{1.15}$$

where $\mathbf{L}$ is a matrix of dimension $N_\mathrm{x} \times N_\mathrm{x}$ and $\mathbf{K}$ is a matrix of dimension $N_\mathrm{x} \times N_\mathrm{y}$. Hence $\mathbf{x}^\mathrm{a}$ is a linear combination of the available information ($\mathbf{L}$ and $\mathbf{K}$ are linear operators). Given the observation equation $\mathbf{y} = \mathbf{H}\mathbf{x}^\mathrm{t} + \mathbf{e}^\mathrm{o}$, the error attached to this combination can be estimated as follows:

$$\mathbf{x}^\mathrm{a} - \mathbf{x}^\mathrm{t} = \mathbf{L}\left(\mathbf{x}^\mathrm{b} - \mathbf{x}^\mathrm{t} + \mathbf{x}^\mathrm{t}\right) + \mathbf{K}\left(\mathbf{H}\mathbf{x}^\mathrm{t} + \mathbf{e}^\mathrm{o}\right) - \mathbf{x}^\mathrm{t}, \tag{1.16a}$$

$$\mathbf{e}^\mathrm{a} = \mathbf{L}\mathbf{e}^\mathrm{b} + \mathbf{K}\mathbf{e}^\mathrm{o} + (\mathbf{L} + \mathbf{K}\mathbf{H} - \mathbf{I})\,\mathbf{x}^\mathrm{t}. \tag{1.16b}$$

First, one wishes that the errors be unbiased. Recalling that we assumed that the observation and background errors are unbiased ($\mathbb{E}[\mathbf{e}^\mathrm{o}] = \mathbf{0}$ and $\mathbb{E}[\mathbf{e}^\mathrm{b}] = \mathbf{0}$), one can infer from the previous calculation that $\mathbb{E}[\mathbf{e}^\mathrm{a}] = (\mathbf{L} + \mathbf{K}\mathbf{H} - \mathbf{I})\,\mathbb{E}[\mathbf{x}^\mathrm{t}]$. Therefore, we require that

$$\mathbf{L} = \mathbf{I} - \mathbf{K}\mathbf{H}, \tag{1.17}$$

which is a sufficient condition (though not necessary).

As a result, we obtain a more constrained Ansatz which is now ***linear*** and ***unbiased***:

$$\mathbf{x}^\mathrm{a} = (\mathbf{I} - \mathbf{K}\mathbf{H})\mathbf{x}^\mathrm{b} + \mathbf{K}\mathbf{y}, \tag{1.18a}$$

$$\mathbf{x}^\mathrm{a} = \mathbf{x}^\mathrm{b} + \mathbf{K}\underbrace{(\mathbf{y} - \mathbf{H}\mathbf{x}^\mathrm{b})}_{\text{innovation}}. \tag{1.18b}$$

$\mathbf{K}$ is a linear map, *i.e.* a matrix, from $\mathbb{R}^{N_\mathrm{y}}$ to $\mathbb{R}^{N_\mathrm{x}}$. It is called the ***gain***. The vector $\mathbf{y} - \mathbf{H}\mathbf{x}^\mathrm{b}$ in $\mathbb{R}^{N_\mathrm{y}}$ is called the ***innovation*** vector. This innovation is an expression formula for the additional information brought in by the observations compared to the background. The error covariance matrix that can be obtained from the innovation vectors is what is usually called the ***information matrix*** in information theory.

Since $\mathbf{K}$ is linear, the analysis is merely a linear interpolation. Actually, it should more adequately be called a linear regression. This interpolation has an historical naming since the first analyses used in meteorology (Cressman) were genuine mathematical interpolations (*i.e.* $\mathbf{y} = \mathbf{H}[\mathbf{x}^\mathrm{a}]$).

With this linear estimator, the estimation problem now amounts to finding a "satisfying gain" $\mathbf{K}$.

The posterior error

Assume for a moment that we have determined an optimal gain matrix $\mathbf{K}$. Then, what would be the error covariance matrix $\mathbf{P}^{\mathrm{a}}$? In order to compute it, Eq. (1.18a) is written using the error vectors that have already been introduced

$$\mathbf{e}^{\mathrm{a}} = \mathbf{e}^{\mathrm{b}} + \mathbf{K}(\mathbf{e}^{\mathrm{o}} - \mathbf{H}\mathbf{e}^{\mathrm{b}}) \tag{1.19}$$

so that

$$\begin{aligned}\mathbf{P}^{\mathrm{a}} &= \mathbb{E}\left[(\mathbf{e}^{\mathrm{a}})(\mathbf{e}^{\mathrm{a}})^\top\right] = \mathbb{E}\left[\left(\mathbf{e}^{\mathrm{b}} + \mathbf{K}(\mathbf{e}^{\mathrm{o}} - \mathbf{H}\mathbf{e}^{\mathrm{b}})\right)\left(\mathbf{e}^{\mathrm{b}} + \mathbf{K}(\mathbf{e}^{\mathrm{o}} - \mathbf{H}\mathbf{e}^{\mathrm{b}})\right)^\top\right] \\ &= \mathbb{E}\left[\left(\mathbf{L}\mathbf{e}^{\mathrm{b}} + \mathbf{K}\mathbf{e}^{\mathrm{o}}\right)\left(\mathbf{L}\mathbf{e}^{\mathrm{b}} + \mathbf{K}\mathbf{e}^{\mathrm{o}}\right)^\top\right] = \mathbb{E}\left[\mathbf{L}\mathbf{e}^{\mathrm{b}}(\mathbf{e}^{\mathrm{b}})^\top\mathbf{L}^\top)\right] + \mathbb{E}\left[\mathbf{K}\mathbf{e}^{\mathrm{o}}(\mathbf{e}^{\mathrm{o}})^\top\mathbf{K}^\top\right] \\ &= \mathbf{L}\mathbf{B}\mathbf{L}^\top + \mathbf{K}\mathbf{R}\mathbf{K}^\top,\end{aligned} \tag{1.20}$$

where we used the decorrelation of $\mathbf{e}^{\mathrm{o}}$ with $\mathbf{e}^{\mathrm{b}}$ and the linearity of $\mathbf{K}$. To summarise, we have

$$\mathbf{P}^{\mathrm{a}} = (\mathbf{I} - \mathbf{K}\mathbf{H})\mathbf{B}(\mathbf{I} - \mathbf{K}\mathbf{H})^\top + \mathbf{K}\mathbf{R}\mathbf{K}^\top. \tag{1.21}$$

1.2.2 Optimal estimation: the BLUE analysis

Continuing on, we seek to minimise the error committed in the analysis, as measured by the scalar quantity $\mathrm{Tr}(\mathbf{P}^{\mathrm{a}})$. Since we look for an optimal gain $\mathbf{K}$ that will be denoted $\mathbf{K}^\star$, we study the variation of $\mathrm{Tr}(\mathbf{P}^{\mathrm{a}})$ with respect to a variation $\delta\mathbf{K}$ of $\mathbf{K}$ (*i.e.* a generic infinitesimal variation in the entries of $\mathbf{K}$):

$$\begin{aligned}\delta(\mathrm{Tr}(\mathbf{P}^{\mathrm{a}})) &= \mathrm{Tr}\left((-\delta\mathbf{K}\mathbf{H})\mathbf{B}\mathbf{L}^\top + \mathbf{L}\mathbf{B}(-\delta\mathbf{K}\mathbf{H})^\top + \delta\mathbf{K}\mathbf{R}\mathbf{K}^\top + \mathbf{K}\mathbf{R}\delta\mathbf{K}^\top\right) \\ &= \mathrm{Tr}\left((-\mathbf{L}\mathbf{B}^\top\mathbf{H}^\top - \mathbf{L}\mathbf{B}\mathbf{H}^\top + \mathbf{K}\mathbf{R}^\top + \mathbf{K}\mathbf{R})(\delta\mathbf{K})^\top\right) \\ &= 2\,\mathrm{Tr}\left((-\mathbf{L}\mathbf{B}\mathbf{H}^\top + \mathbf{K}\mathbf{R})(\delta\mathbf{K})^\top\right).\end{aligned} \tag{1.22}$$

We have used $\mathrm{Tr}(\mathbf{A}) = \mathrm{Tr}(\mathbf{A}^\top)$ and the fact that $\mathbf{B}$ and $\mathbf{R}$ are both symmetric. At optimality, one infers that $-(\mathbf{I} - \mathbf{K}^\star\mathbf{H})\mathbf{B}\mathbf{H}^\top + \mathbf{K}^\star\mathbf{R} = \mathbf{0}$, from which we obtain

$$\mathbf{K}^\star = \mathbf{B}\mathbf{H}^\top(\mathbf{R} + \mathbf{H}\mathbf{B}\mathbf{H}^\top)^{-1}. \tag{1.23}$$

The estimates of $\mathbf{x}^{\mathrm{a}}$ and $\mathbf{P}^{\mathrm{a}}$, which are the outcomes of this analysis, are called a ***BLUE*** analysis for ***Best Linear Unbiased Estimator***, since it is: linear (primary hypothesis through $\mathbf{L}$ and $\mathbf{K}$), without bias (first step of the derivation: $\mathbf{L} = \mathbf{I} - \mathbf{K}\mathbf{H}$) and optimal (second step of the derivation).

1.2.3 Properties

Sherman-Morrison-Woodbury formula

Formula Eq. (1.23) is the most frequently used form of the optimal gain. However, the same optimal gain can be written differently

$$\mathbf{K}^\star = (\mathbf{B}^{-1} + \mathbf{H}^\top\mathbf{R}^{-1}\mathbf{H})^{-1}\mathbf{H}^\top\mathbf{R}^{-1}. \tag{1.24}$$

The equivalence between Eq. (1.23) and Eq. (1.24) is an immediate consequence of the ***Sherman-Morrison-Woodbury identity***. It is proven through

$$\mathbf{B}\mathbf{H}^\top(\mathbf{H}\mathbf{B}\mathbf{H}^\top+\mathbf{R})^{-1} \tag{1.25a}$$

$$=\left(\mathbf{B}^{-1}+\mathbf{H}^\top\mathbf{R}^{-1}\mathbf{H}\right)^{-1}\left(\mathbf{B}^{-1}+\mathbf{H}^\top\mathbf{R}^{-1}\mathbf{H}\right)\mathbf{B}\mathbf{H}^\top(\mathbf{H}\mathbf{B}\mathbf{H}^\top+\mathbf{R})^{-1} \tag{1.25b}$$

$$=\left(\mathbf{B}^{-1}+\mathbf{H}^\top\mathbf{R}^{-1}\mathbf{H}\right)^{-1}\left(\mathbf{H}^\top+\mathbf{H}^\top\mathbf{R}^{-1}\mathbf{H}\mathbf{B}\mathbf{H}^\top\right)(\mathbf{H}\mathbf{B}\mathbf{H}^\top+\mathbf{R})^{-1} \tag{1.25c}$$

$$=\left(\mathbf{B}^{-1}+\mathbf{H}^\top\mathbf{R}^{-1}\mathbf{H}\right)^{-1}\mathbf{H}^\top\mathbf{R}^{-1}\left(\mathbf{R}+\mathbf{H}\mathbf{B}\mathbf{H}^\top\right)(\mathbf{H}\mathbf{B}\mathbf{H}^\top+\mathbf{R})^{-1} \tag{1.25d}$$

$$=\left(\mathbf{B}^{-1}+\mathbf{H}^\top\mathbf{R}^{-1}\mathbf{H}\right)^{-1}\mathbf{H}^\top\mathbf{R}^{-1}. \tag{1.25e}$$

This equivalence turns out to be useful, both from a theoretical but also a practical standpoint. For instance, the observation space is quite often much smaller than the dimension of the state space, so that the inversion of the matrix $\mathbf{B}^{-1}+\mathbf{H}^\top\mathbf{R}^{-1}\mathbf{H}$ is much more costly than the inversion of the matrix $\mathbf{R}+\mathbf{H}\mathbf{B}\mathbf{H}^\top$. Therefore, it is often useful to resort to (1.23).

Optimal analysis error

Choosing the optimal gain Eq. (1.23) for the error estimation, the posterior error covariance matrix Eq. (1.21) simplifies to

$$\mathbf{P}^{\mathrm{a}}=(\mathbf{I}-\mathbf{K}^\star\mathbf{H})\mathbf{B}. \tag{1.26}$$

Indeed, we have

$$\begin{aligned}\mathbf{P}^{\mathrm{a}}&=(\mathbf{I}-\mathbf{K}\mathbf{H})\mathbf{B}(\mathbf{I}-\mathbf{K}\mathbf{H})^\top+\mathbf{K}\mathbf{R}\mathbf{K}^\top\\&=(\mathbf{I}-\mathbf{K}\mathbf{H})\mathbf{B}+\underbrace{\left[\mathbf{K}\mathbf{R}-(\mathbf{I}-\mathbf{K}\mathbf{H})\mathbf{B}\mathbf{H}^\top\right]}_{=\mathbf{0}\ \text{if}\ \mathbf{K}=\mathbf{K}^\star}\mathbf{K}^\top.\end{aligned} \tag{1.27}$$

The expression in brackets in the right-hand side is zero when the gain is optimal ($\mathbf{K}=\mathbf{K}^\star$) so that we obtain the desired result. This justifies the name ***gain matrix***; the operator $\mathbf{I}-\mathbf{K}^\star\mathbf{H}$ measures the shrinkage of the innovation vector into

$$\mathbf{y}-\mathbf{H}\mathbf{x}^{\mathrm{a}}=(\mathbf{I}-\mathbf{H}\mathbf{K}^\star)(\mathbf{y}-\mathbf{H}\mathbf{x}^{\mathrm{b}}), \tag{1.28}$$

which is called the ***analysis residue***.

Two classical expressions for $\mathbf{P}^{\mathrm{a}}$ can be obtained using Eq. (1.26), together with the two formula for the gain, Eq. (1.23) and Eq. (1.24). Indeed one has in the one hand

$$\mathbf{P}^{\mathrm{a}}=(\mathbf{I}-\mathbf{K}^\star\mathbf{H})\mathbf{B} \tag{1.29a}$$

$$=(\mathbf{I}-\mathbf{B}\mathbf{H}^\top(\mathbf{R}+\mathbf{H}\mathbf{B}\mathbf{H}^\top)^{-1}\mathbf{H})\mathbf{B} \tag{1.29b}$$

$$=\mathbf{B}-\mathbf{B}\mathbf{H}^\top(\mathbf{R}+\mathbf{H}\mathbf{B}\mathbf{H}^\top)^{-1}\mathbf{H}\mathbf{B}, \tag{1.29c}$$

and

$$\mathbf{P}^{\mathrm{a}}=(\mathbf{I}-(\mathbf{B}^{-1}+\mathbf{H}^\top\mathbf{R}^{-1}\mathbf{H})^{-1}\mathbf{H}^\top\mathbf{R}^{-1}\mathbf{H})\mathbf{B} \tag{1.30a}$$

$$=(\mathbf{B}^{-1}+\mathbf{H}^\top\mathbf{R}^{-1}\mathbf{H})^{-1}\left(\mathbf{B}^{-1}+\mathbf{H}^\top\mathbf{R}^{-1}\mathbf{H}-\mathbf{H}^\top\mathbf{R}^{-1}\mathbf{H}\right)\mathbf{B} \tag{1.30b}$$

$$=(\mathbf{B}^{-1}+\mathbf{H}^\top\mathbf{R}^{-1}\mathbf{H})^{-1}, \tag{1.30c}$$

in the other hand. This last formula establishes that $\mathbf{P}^{\mathrm{a}}$ is invertible and hence that it is a symmetric positive definite matrix. It also shows that the inverses of the error covariance matrices (***confidence***, or ***precision*** matrices) are additive because

$$(\mathbf{P}^{\mathrm{a}})^{-1} = \mathbf{B}^{-1} + \mathbf{H}^{\top}\mathbf{R}^{-1}\mathbf{H}. \tag{1.31}$$

The optimal gain $\mathbf{K}^{\star}$ can be related to $\mathbf{P}^{\mathrm{a}}$ through the following useful formula

$$\mathbf{K}^{\star} = \mathbf{P}^{\mathrm{a}}\,\mathbf{H}^{\top}\,\mathbf{R}^{-1}. \tag{1.32}$$

Indeed, one has

$$\mathbf{P}^{\mathrm{a}}\,\mathbf{H}^{\top}\,\mathbf{R}^{-1} = (\mathbf{B}^{-1} + \mathbf{H}^{\top}\mathbf{R}^{-1}\mathbf{H})^{-1}\,\mathbf{H}^{\top}\,\mathbf{R}^{-1} = \mathbf{K}^{\star}. \tag{1.33}$$

The innovation and the analysis residue are unbiased

A consequence of the observation operator linearity and of the absence of bias of the analysis error $\mathbf{e}^{\mathrm{a}}$ is that the analysis residue $\mathbf{y} - \mathbf{H}\mathbf{x}^{\mathrm{a}}$ is also unbiased. Indeed

$$\mathbf{y} - \mathbf{H}\mathbf{x}^{\mathrm{a}} = \mathbf{H}\mathbf{x}^{\mathrm{t}} + \mathbf{e}^{\mathrm{o}} - \mathbf{H}\mathbf{x}^{\mathrm{a}} = \mathbf{e}^{\mathrm{o}} - \mathbf{H}\mathbf{e}^{\mathrm{a}}, \tag{1.34}$$

from which it is immediate to conclude $\mathbb{E}\left[\mathbf{y} - \mathbf{H}\mathbf{x}^{\mathrm{a}}\right] = \mathbf{0}$. Because we have assumed that the background error is unbiased, we have the same for the innovation vector:

$$\mathbf{y} - \mathbf{H}\mathbf{x}^{\mathrm{b}} = \mathbf{H}\mathbf{x}^{\mathrm{t}} + \mathbf{e}^{\mathrm{o}} - \mathbf{H}\mathbf{x}^{\mathrm{b}} = \mathbf{e}^{\mathrm{o}} - \mathbf{H}\mathbf{e}^{\mathrm{b}}, \tag{1.35}$$

from which we obtain $\mathbb{E}\left[\mathbf{y} - \mathbf{H}\mathbf{x}^{\mathrm{b}}\right] = \mathbf{0}$.

Orthogonality of the analysis with the analysis error

Let us calculate the following covariance matrix even if the gain is not optimal

$$\mathbf{C} = \mathbb{E}\left[\mathbf{x}^{\mathrm{a}}\left(\mathbf{e}^{\mathrm{a}}\right)^{\top}\right]. \tag{1.36}$$

We assume that the background satisfies:

$$\mathbb{E}\left[\mathbf{x}^{\mathrm{b}}\left(\mathbf{e}^{\mathrm{b}}\right)^{\top}\right] = \mathbf{0}. \tag{1.37}$$

This means that the background and its related error are uncorrelated.

We have seen that $\mathbf{x}^{\mathrm{a}} = \mathbf{x}^{\mathrm{b}} + \mathbf{K}\left(\mathbf{y} - \mathbf{H}\mathbf{x}^{\mathrm{b}}\right) = \mathbf{x}^{\mathrm{b}} + \mathbf{K}\left(-\mathbf{H}\mathbf{e}^{\mathrm{b}} + \mathbf{e}^{\mathrm{o}}\right)$. As a consequence

$$\mathbf{C} = \mathbb{E}\left[\left(\mathbf{x}^{\mathrm{b}} + \mathbf{K}\left(-\mathbf{H}\mathbf{e}^{\mathrm{b}} + \mathbf{e}^{\mathrm{o}}\right)\right)\left(\left(\mathbf{I} - \mathbf{K}\mathbf{H}\right)\mathbf{e}^{\mathrm{b}} + \mathbf{K}\mathbf{e}^{\mathrm{o}}\right)^{\top}\right] \tag{1.38a}$$

$$= -\mathbf{K}\mathbf{H}\mathbb{E}\left[\mathbf{e}^{\mathrm{b}}(\mathbf{e}^{\mathrm{b}})^{\top}\right]\left(\mathbf{I} - \mathbf{K}\mathbf{H}\right)^{\top} + \mathbf{K}\mathbb{E}\left[\mathbf{e}^{\mathrm{o}}(\mathbf{e}^{\mathrm{o}})^{\top}\right]\mathbf{K}^{\top} \tag{1.38b}$$

$$= \mathbf{K}\left[-\mathbf{H}\mathbf{B}\left(\mathbf{I} - \mathbf{K}\mathbf{H}\right)^{\top} + \mathbf{R}\mathbf{K}^{\top}\right]. \tag{1.38c}$$

Then, if the analysis is optimal, we obtain $\mathbf{K}\mathbf{R} - \left(\mathbf{I} - \mathbf{K}\mathbf{H}\right)\mathbf{B}\mathbf{H}^{\top} = \mathbf{0}$, so that $\mathbf{C} = \mathbf{0}$. Hence, the estimate $\mathbf{x}^{\mathrm{a}}$ and the analysis error $\mathbf{e}^{\mathrm{a}}$ are orthogonal. Actually orthogonality and optimality are equivalent here! That is why the analysis error is uncorrelated from the estimate resulting from the same analysis.

1.3 Variational equivalence

Let us make the same assumptions as for the BLUE derivation, *i.e.* that H is a linear operator and denoted $\mathbf{H}$. All the statistical hypotheses remain the same. We wish to show that the BLUE result can be be obtained using *variation calculus*, *i.e.* through the minimisation of a (multivariate) function. This is a ***variational*** approach to the problem. A sufficiently regular cost function F, *i.e.* a function from $\mathbb{R}^{N_\mathrm{x}}$ to $\mathbb{R}$, have a generic quadratic expansion around $\mathbf{x}_0$ of the form

$$F(\mathbf{x}) = F(\mathbf{x}_0) + (\mathbf{x}-\mathbf{x}_0)^\top \nabla F_{|\mathbf{x}_0} + \frac{1}{2}(\mathbf{x}-\mathbf{x}_0)^\top \mathrm{Hess}_{|\mathbf{x}_0}(\mathbf{x}-\mathbf{x}_0) + o(||\mathbf{x}-\mathbf{x}_0||^2), \quad (1.39)$$

where $\nabla F_{|\mathbf{x}_0}$, a vector that generalises the first derivative, is called the ***gradient***, and Hess, a matrix that generalises the second derivative, is called the ***Hessian***. They are defined by

$$\left[\nabla F_{|\mathbf{x}}\right]_i = \frac{\partial F}{\partial x_i}_{|\mathbf{x}}, \qquad \left[\mathrm{Hess}_{|\mathbf{x}}\right]_{ij} = \frac{\partial^2 F}{\partial x_i \partial x_j}_{|\mathbf{x}}. \quad (1.40)$$

1.3.1 Equivalence with BLUE

We define the following function, from $\mathbb{R}^{N_\mathrm{x}}$ into $\mathbb{R}$

$$J(\mathbf{x}) = \frac{1}{2}\left(\mathbf{x}-\mathbf{x}^\mathrm{b}\right)^\top \mathbf{B}^{-1}\left(\mathbf{x}-\mathbf{x}^\mathrm{b}\right) + \frac{1}{2}\left(\mathbf{y}-\mathbf{H}\mathbf{x}\right)^\top \mathbf{R}^{-1}\left(\mathbf{y}-\mathbf{H}\mathbf{x}\right), \quad (1.41)$$

which is called a ***cost function*** or ***objective function***. Since $\mathbf{H}$ is linear, J is a quadratic functional of $\mathbf{x}$. Since $\mathbf{B}$ is positive definite, this functional is strictly convex and as a consequence has a unique minimum. Where is it?

$$\begin{aligned}
\delta J(\mathbf{x}) &= \frac{1}{2}\left(\delta\mathbf{x}\right)^\top \mathbf{B}^{-1}\left(\mathbf{x}-\mathbf{x}^\mathrm{b}\right) + \frac{1}{2}\left(\mathbf{x}-\mathbf{x}^\mathrm{b}\right)^\top \mathbf{B}^{-1}\delta\mathbf{x} && (1.42a)\\
&+ \frac{1}{2}\left(-\mathbf{H}\delta\mathbf{x}\right)^\top \mathbf{R}^{-1}\left(\mathbf{y}-\mathbf{H}\mathbf{x}\right) + \frac{1}{2}\left(\mathbf{y}-\mathbf{H}\mathbf{x}\right)\mathbf{R}^{-1}\left(-\mathbf{H}\delta\mathbf{x}\right) && (1.42b)\\
&= \left(\delta\mathbf{x}\right)^\top \mathbf{B}^{-1}\left(\mathbf{x}-\mathbf{x}^\mathrm{b}\right) - \left(\delta\mathbf{x}\right)^\top \mathbf{H}^\top\mathbf{R}^{-1}\left(\mathbf{y}-\mathbf{H}\mathbf{x}\right) && (1.42c)\\
&= \left(\delta\mathbf{x}\right)^\top \nabla J. && (1.42d)
\end{aligned}$$

The extremum condition is $\nabla J = \mathbf{B}^{-1}(\mathbf{x}^\star - \mathbf{x}^\mathrm{b}) - \mathbf{H}^\top\mathbf{R}^{-1}(\mathbf{y}-\mathbf{H}\mathbf{x}^\star) = \mathbf{0}$, which can be written

$$\mathbf{x}^\star = \mathbf{x}^\mathrm{b} + \underbrace{(\mathbf{B}^{-1}+\mathbf{H}^\top\mathbf{R}^{-1}\mathbf{H})^{-1}\mathbf{H}^\top\mathbf{R}^{-1}}_{\mathbf{K}^\star}(\mathbf{y}-\mathbf{H}\mathbf{x}^\mathrm{b}). \quad (1.43)$$

Therefore, $\mathbf{x}^\star$ identifies with the BLUE optimal analysis $\mathbf{x}^\mathrm{a}$. Let us remark that the Sherman-Morrison-Woodbury would be needed to prove the equivalence with the initial BLUE result, Eq. (1.23).

1.3.2 Properties of the variational approach

Analysis precision and the Hessian

The functional gradient is obtained from Eq. (1.42a),

$$\nabla J(\mathbf{x}) = \mathbf{B}^{-1}(\mathbf{x}-\mathbf{x}^\mathrm{b}) - \mathbf{H}^\top\mathbf{R}^{-1}(\mathbf{y}-\mathbf{H}\mathbf{x}). \quad (1.44)$$

In this particular case, since the cost function is quadratic, it is simple to compute the Hessian

$$\mathrm{Hess}_{|\mathbf{x}} = \mathbf{B}^{-1} + \mathbf{H}^\top \mathbf{R}^{-1} \mathbf{H}. \tag{1.45}$$

As a matter of fact, this coincides with the expression of the inverse of the analysis error covariance matrix that we have already encountered! Therefore, we have

$$\mathbf{P}^{\mathrm{a}} = \mathrm{Hess}_{|\mathbf{x}}^{-1}. \tag{1.46}$$

This is more than just another expression for $\mathbf{P}^{\mathrm{a}}$. This formula shows that the precision of the analysis is proportional to the curvature of the cost function J. Hence, the narrower the minimum, the better the analysis.

Nonlinear extension

Besides a new light shed on the problem, the variational formalism has two important traits. Firstly, it offers an elegant and straightforward extension of the linear problem to the nonlinear problem when H is not linear anymore. By definition, this extension is impossible within the BLUE formalism without linearising the observation operator, which would be an approximation.

A more general algorithm

Secondly, the variational approach has an algorithmic advantage. Within the BLUE approach, one needs to compute the inverse of matrix $\mathbf{R} + \mathbf{H}\mathbf{B}\mathbf{H}^\top$ that appears in the gain $\mathbf{K}^\star$ (or at least solve a linear system thereof). Within the variational approach, the cost function J is minimised, which requires to compute the product of a vector by the inverses of $\mathbf{B}$ and $\mathbf{R}$ several times. This may show a lower computational burden than the inversion of the full matrix when the number of iterations is limited.

The variational formulation of the statistical interpolation problem is the analysis and the main step of what is usually coined ***3D-Var***. This type of analysis has been used operationally in meteorological weather services in the 1990's replacing the BLUE-like optimal interpolation. In the 2000's it has been replaced in many centres by the ***4D-Var*** for the synoptic scale, a generalisation of the 3D-Var that we shall discuss in chapter 3.

1.3.3 When the observation operator H is non-linear

As we just explained, a significant advantage of the variational method is the possibility to rigorously handle the case where H is a nonlinear operator. Let us look at the modification that this induces in the analysis. In this case, J is defined by

$$J(\mathbf{x}) = \frac{1}{2} \left(\mathbf{x} - \mathbf{x}^{\mathrm{b}} \right)^\top \mathbf{B}^{-1} \left(\mathbf{x} - \mathbf{x}^{\mathrm{b}} \right) + \frac{1}{2} \left(\mathbf{y} - H[\mathbf{x}] \right)^\top \mathbf{R}^{-1} \left(\mathbf{y} - H[\mathbf{x}] \right). \tag{1.47}$$

We introduce the ***tangent linear*** of H at $\mathbf{x}$, denoted $\mathbf{H}$ (see Eq. (1.14)). As opposed to the linear case, $\mathbf{H}$ now depends on where H has been linearised, i.e. $\mathbf{x}$. The gradient of J is similar and reads

$$\nabla J(\mathbf{x}) = \mathbf{B}^{-1}(\mathbf{x} - \mathbf{x}^{\mathrm{b}}) - \mathbf{H}_{|\mathbf{x}}^\top \mathbf{R}^{-1}(\mathbf{y} - H[\mathbf{x}]). \tag{1.48}$$

Again, the precision of the analysis is nicely obtained from $\mathbf{P}^{\mathrm{a}} = \mathrm{Hess}_{|\mathbf{x}}^{-1}$. This time, however, the Hessian really depends on $\mathbf{x}$.

1.3.4 Dual formalism

Here, we assume that the observation operator is linear. The stating point is the following cost function:

$$J(\mathbf{x}) = \frac{1}{2}\left(\mathbf{x}-\mathbf{x}^{\mathrm{b}}\right)^{\top}\mathbf{B}^{-1}\left(\mathbf{x}-\mathbf{x}^{\mathrm{b}}\right) + \frac{1}{2}\left(\mathbf{y}-\mathbf{H}\mathbf{x}\right)^{\top}\mathbf{R}^{-1}\left(\mathbf{y}-\mathbf{H}\mathbf{x}\right). \tag{1.49}$$

A vector of Lagrange multipliers $\mathbf{w}$ is introduced, of dimension that of the observation space, *i.e.* $\mathbf{w} \in \mathbb{R}^{N_{\mathrm{y}}}$. It is used to enforce the observation equation, through the Lagrangian

$$L(\mathbf{x},\boldsymbol{\epsilon},\mathbf{w}) = \frac{1}{2}\left(\mathbf{x}-\mathbf{x}^{\mathrm{b}}\right)^{\top}\mathbf{B}^{-1}\left(\mathbf{x}-\mathbf{x}^{\mathrm{b}}\right) + \frac{1}{2}\boldsymbol{\epsilon}^{\top}\mathbf{R}^{-1}\boldsymbol{\epsilon} + \mathbf{w}^{\top}\left(\mathbf{y}-\mathbf{H}\mathbf{x}-\boldsymbol{\epsilon}\right). \tag{1.50}$$

The optimum with respect to $\mathbf{w}$ of this functional is equivalent to the previous cost function $J(\mathbf{x})$. According to the ***minmax*** theorem, the minimum of the cost function $J(\mathbf{x})$ coincides with the maximum of the functional depending on $\mathbf{w}$ generated by the minimum of L with respect to $\mathbf{x}$ and $\boldsymbol{\epsilon}$, denoted $G(\mathbf{w})$. What is it? The null gradient condition at the minimum implies that

$$\mathbf{x}^{\star} = \mathbf{x}^{\mathrm{b}} + \mathbf{B}\mathbf{H}^{\top}\mathbf{w}, \qquad \boldsymbol{\epsilon}^{\star} = \mathbf{R}\mathbf{w}. \tag{1.51}$$

This leads to the following ***dual*** cost function

$$\begin{aligned} G(\mathbf{w}) &= -L(\mathbf{x}^{\star},\boldsymbol{\epsilon}^{\star},\mathbf{w}) \\ &= \frac{1}{2}\mathbf{w}^{\top}\left(\mathbf{R}+\mathbf{H}\mathbf{B}\mathbf{H}^{\top}\right)\mathbf{w} - \mathbf{w}^{\top}\left(\mathbf{y}-\mathbf{H}\mathbf{x}^{\mathrm{b}}\right). \end{aligned} \tag{1.52}$$

The main advantage of this approach is that the optimisation of this cost function takes place in the observation space $\mathbb{R}^{N_{\mathrm{y}}}$ rather than in state space $\mathbb{R}^{N_{\mathrm{x}}}$. The observation space is usually of dimension much smaller than that of the state space. This formalism is known as ***PSAS*** standing for ***Physical Statistical space Assimilation System***.

1.4 A simple example

Imagine you are shipwrecked at sea, a few kilometres away from the shores. Before boarding a small lifeboat, you just had time to measure your coordinates $(u,v) = (0, v_{\mathrm{b}})$ with high accuracy. The Ox axis is parallel to the shore, while the Oy axis is perpendicular. One hour later, you want to estimate your new coordinates because the lifeboat has drifted in the meantime. To this goal, you roughly guess the distance to the shore, denoted v_0 with a variance σ_0^2. But we want to use our knowledge of the precise coordinate of the wreckage location, one hour ago. In the meantime, and in the absence of major sea stream, the lifeboat has drifted. The probability for it to be at position $(u_{\mathrm{b}}, v_{\mathrm{b}})$ follows a normal distribution of variance σ_{b}^2 which is a linear function of the elapsed time since the wreckage. It is assumed that there is no correlation between the observation process and the drifting process.

1.4.1 Observation equation and error covariance matrix

The state vector is that of the lifeboat coordinates $\mathbf{x} = (u,v)^{\top}$. The observation equation is $y = \mathbf{H}\mathbf{x} + \varepsilon$, where the observation operator is $\mathbf{H} = (0,1)$, the observation vector is

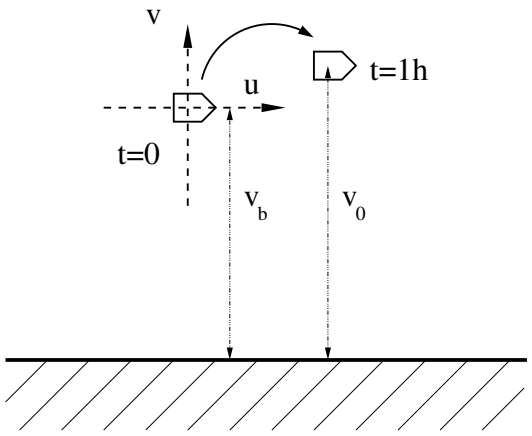


Figure 1.3: The geometry of the lifeboat problem.

(v_0), and the error ε follows a normal distribution of variance σ_0^2. The observation error covariance matrix is $R = (\sigma_0^2)$, while the background error covariance matrix is

$$\mathbf{B} = \begin{pmatrix} \sigma_{\rm b}^2 & 0 \\ 0 & \sigma_{\rm b}^2 \end{pmatrix}. \tag{1.53}$$

1.4.2 Optimal analysis

The linear analysis is of the form $\mathbf{x}^{\rm a} = \mathbf{x}^{\rm b} + \mathbf{K}(\mathbf{y} - \mathbf{H}\mathbf{x}^{\rm b})$ which translates into

$$\begin{pmatrix} u_{\rm a} \\ v_{\rm a} \end{pmatrix} = \begin{pmatrix} 0 \\ v_{\rm b} \end{pmatrix} + \mathbf{K} \left(v_{\rm o} - (0,1) \begin{pmatrix} 0 \\ v_{\rm b} \end{pmatrix} \right). \tag{1.54}$$

Let us compute the optimal gain $\mathbf{K}^\star$. To that end, we use the most efficient expression of $\mathbf{K}^\star$ when the number of observations is smaller than the number of state variables, *i.e.*

$$\begin{aligned} \mathbf{K}^\star &= \sigma_{\rm b}^2 \begin{pmatrix} 0 \\ 1 \end{pmatrix} \left(\sigma_{\rm o}^2 + (0,1) \sigma_{\rm b}^2 \begin{pmatrix} 0 \\ 1 \end{pmatrix} \right)^{-1} \\ &= \frac{\sigma_{\rm b}^2}{\sigma_{\rm o}^2 + \sigma_{\rm b}^2} \begin{pmatrix} 0 \\ 1 \end{pmatrix}. \end{aligned} \tag{1.55}$$

One concludes that

$$\begin{pmatrix} u_{\rm a} \\ v_{\rm a} \end{pmatrix} = \begin{pmatrix} 0 \\ v_{\rm b} + \frac{\sigma_{\rm b}^2}{\sigma_{\rm o}^2 + \sigma_{\rm b}^2} (v_{\rm o} - v_{\rm b}) \end{pmatrix}. \tag{1.56}$$

Since the observation did not bring any new information on coordinate Ox, we stick to the wreckage coordinate $u_{\rm a} = 0$. However, as time goes by, $\sigma_{\rm b}$ increases. Hence, $v_{\rm a}$ converges to $v_{\rm o}$, which remains the most trustworthy information in this limit.

1.4.3 Posterior error

From $\mathbf{P}^{\rm a} = (\mathbf{I} - \mathbf{K}^\star \mathbf{H})\mathbf{B}$, we obtain that

$$\mathbf{P}^{\rm a} = \begin{pmatrix} \sigma_{\rm b}^2 & 0 \\ 0 & \dfrac{\sigma_{\rm o}^2 \sigma_{\rm b}^2}{\sigma_{\rm o}^2 + \sigma_{\rm b}^2} \end{pmatrix}. \tag{1.57}$$

We see that, as expected, the analysis does not improve our knowledge of the coordinate parallel to the shore u. Its uncertainty stems from the drift, hence it is proportional to time and increases with it. Differently, the uncertainty in coordinate v is shrunk by a factor $\sqrt{\frac{\sigma_\mathrm{o}^2}{\sigma_\mathrm{o}^2+\sigma_\mathrm{b}^2}}$. The finer the observation, the smaller the error. Denoting σ_a^2 the error on coordinate v after the analysis, we find

$$\frac{1}{\sigma_\mathrm{a}^2} = \frac{1}{\sigma_\mathrm{o}^2} + \frac{1}{\sigma_\mathrm{b}^2}. \tag{1.58}$$

The interpretation of Eq. (1.30a) is: ***the confidence (or precision) of the analysis is the sum of the confidence of the observation and of the background***.

1.4.4 3D-Var and PSAS

The related cost function of the equivalent variational problem is

$$J(u,v) = \frac{1}{2\sigma_\mathrm{b}^2}\left(u^2 + (v-v_b)^2\right) + \frac{1}{2\sigma_\mathrm{o}^2}(v_\mathrm{o}-v)^2. \tag{1.59}$$

The cost function in the PSAS formalism is

$$G(w) = \frac{1}{2}(\sigma_\mathrm{o}^2+\sigma_\mathrm{b}^2)\,w^2 - w(v_0-v_\mathrm{b}), \tag{1.60}$$

from which it is easy to derive the solution

$$w^\star = \frac{v_\mathrm{o}-v_\mathrm{b}}{\sigma_\mathrm{o}^2+\sigma_\mathrm{b}^2}. \tag{1.61}$$

Chapter 2

Sequential interpolation: The Kalman filter

In the previous chapter, we focused on the analysis part of a data assimilation scheme. We have found an optimal estimator, as well as its uncertainty, given some prior information (the background) and an observation set. A possible temporal dimension of the problem was mentioned but mostly left aside.

In meteorology, we are interested in real time data assimilation. The collection of the observations at the NWP centres, as well as the resulting analyses meant for real time forecasting, need to be cycled sequentially in time, since the objective is to perpetually track the system state. That is why a series of times $t_0, t_1, \ldots, t_k, \ldots, t_K$ is defined. They mark the analyses of the data assimilation scheme. For synoptical scale NWP systems, one typically has $t_{k+1} - t_k = 6$ hours.

Moreover, sequential data assimilation introduces a new ingredient in the problem compared to statistical interpolation: the *evolution* model for the system state typically defined between times t_k and t_{k+1}. In meteorology and oceanography, it is the numerical model that simulates the real system dynamics (numerical implementation of the primitive equations). In atmospheric chemistry, this could be a chemical transport model, which transports the species and have them react.

The typical data assimilation scheme is as follows: at time t_k, we have at our disposal the outcome of a previous forecast, denoted $\mathbf{x}_k^{\mathrm{f}}$, the index f standing for "forecast". Therefore, $\mathbf{x}_k^{\mathrm{f}}$ is the analogue of the background $\mathbf{x}^{\mathrm{b}}$ of statistical interpolation of chapter 1. At time t_k, we collect a set of observations stacked into the vector $\mathbf{y}_k$. Given $\mathbf{x}_k^{\mathrm{f}}$ and the observations $\mathbf{y}_k$, an analysis is performed yielding the state estimate $\mathbf{x}_k^{\mathrm{a}}$. Then, we make use of the forward model to forecast the system state from time t_k to time t_{k+1}, from $\mathbf{x}_k$ to $\mathbf{x}_{k+1}$. Note that we have given up on the index t that referred to the *truth* in chapter 1. The outcome of the forecast is denoted $\mathbf{x}_{k+1}^{\mathrm{f}}$. It will serve as the background in the next cycle (see Fig. 2.1). And so on ....

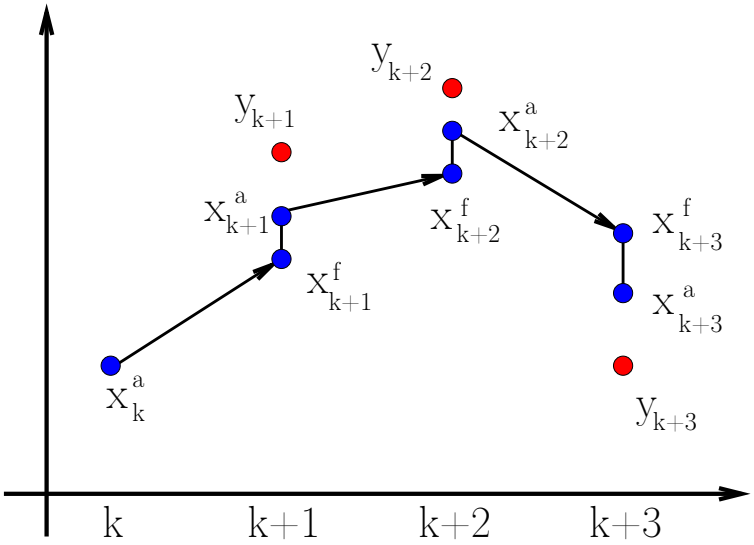


Figure 2.1: The sequential scheme of the Kalman filter.

2.1 Stochastic modelling of the system

The following stochastic equations describe the evolution and observation of the physical system

$$\mathbf{x}_k = M_k(\mathbf{x}_{k-1}) + \mathbf{w}_k, \tag{2.1a}$$
$$\mathbf{y}_k = H_k(\mathbf{x}_k) + \mathbf{v}_k, \tag{2.1b}$$

where $\mathbf{x}_k$ is the true state of the system at time t_k. Let us describe this stochastic system:

- the first equation is the forecast step. M_{k+1} is called the ***resolvent***. It simulates the evolution of the system from t_k to t_{k+1}. It may depend on time (non-autonomous process), hence the k index. $\mathbf{w}_k$ is the model error within M_k compared to the true physical processes. It is assumed that this noise is unbiased, uncorrelated in time (white noise) and of error covariance matrix $\mathbf{Q}_k$ for each time t_k, *i.e.*

$$\mathbb{E}[\mathbf{w}_k] = \mathbf{0} \qquad \text{and} \qquad \mathbb{E}\left[\mathbf{w}_k\,\mathbf{w}_l^\top\right] = \mathbf{Q}_k\delta_{kl}. \tag{2.2}$$

- the second equation corresponds to the observation equation. H_k is the observation operator at time t_k. $\mathbf{v}_k$ is the observation error, which is assumed to be unbiased, uncorrelated in time (white noise) and of error covariance matrix $\mathbf{R}_k$ for each time t_k, *i.e.*

$$\mathbb{E}[\mathbf{v}_k] = \mathbf{0} \qquad \text{and} \qquad \mathbb{E}\left[\mathbf{v}_k\,\mathbf{v}_l^\top\right] = \mathbf{R}_k\delta_{kl}. \tag{2.3}$$

As an additional assumption, though quite a realistic one, there is no correlation between model error and observation error, which translates into

$$\mathbb{E}\left[\mathbf{v}_k\,\mathbf{w}_l^\top\right] = \mathbf{0}. \tag{2.4}$$

From now on, it is assumed that the operators are linear. Hence M_k and H_k are noted $\mathbf{M}_k$ and $\mathbf{H}_k$. This hypothesis will be discussed, possibly lifted, later on.

2.1.1 Analysis step

At time t_k, we have a forecast $\mathbf{x}_k^{\mathrm{f}}$ which is assumed unbiased. It will serve as our background. The error covariance matrix related to $\mathbf{x}_k^{\mathrm{f}}$ is $\mathbf{P}_k^{\mathrm{f}}$. The statistical analysis that stems from these elements, is similar to statistical interpolation. Moreover, if the estimator of this analysis has the minimal possible variance, this analysis is the same as statistical interpolation. Because of the linearity and of the null bias, it must be a BLUE analysis. That is why we now use the results of chapter 1.

The $\mathbf{x}_k^{\mathrm{a}}$ analysis is of the form

$$\mathbf{x}_k^{\mathrm{a}} = \mathbf{x}_k^{\mathrm{f}} + \mathbf{K}_k \left(\mathbf{y}_k - \mathbf{H}_k \mathbf{x}_k^{\mathrm{f}} \right), \tag{2.5}$$

which guarantees that it is unbiased, since the related errors, $\mathbf{y}_k$ and $\mathbf{x}_k^{\mathrm{f}}$, are unbiased. As a consequence, one has $\mathbf{e}_k^{\mathrm{a}} = (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k)\, \mathbf{e}_k^{\mathrm{f}} + \mathbf{K}_k \mathbf{v}_k$, so that the analysis error covariance matrix is

$$\mathbf{P}_k^{\mathrm{a}} = (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k)\, \mathbf{P}_k^{\mathrm{f}}\, (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k)^{\top} + \mathbf{K}_k \mathbf{R}_k \mathbf{K}_k^{\top}. \tag{2.6}$$

If the analysis is optimal, the optimal gain matrix $\mathbf{K}_k^*$ at time t_k satisfies:

$$-\left(\mathbf{I} - \mathbf{K}_k^* \mathbf{H}_k\right) \mathbf{P}_k^{\mathrm{f}} \mathbf{H}_k^{\top} + \mathbf{K}_k^* \mathbf{R} = \mathbf{0}. \tag{2.7}$$

Hence, it can be written

$$\mathbf{K}_k^* = \mathbf{P}_k^{\mathrm{f}} \mathbf{H}_k^{\top} \left(\mathbf{H}_k \mathbf{P}_k^{\mathrm{f}} \mathbf{H}_k^{\top} + \mathbf{R}_k \right)^{-1}. \tag{2.8}$$

Finally, we obtain

$$\mathbf{P}_k^{\mathrm{a}} = (\mathbf{I} - \mathbf{K}_k^* \mathbf{H}_k)\, \mathbf{P}_k^{\mathrm{f}}. \tag{2.9}$$

Within this sequential context, the gain is often called the ***Kalman gain***, in honour of Rudolph Kalman, electrical engineer and mathematician, who invented this filter in the 1960s.

2.1.2 Forecast step

Up to this point, we have used the observations to update our prior knowledge of the system state (the background). Now, we need to forecast the system state from time t_k to time t_{k+1}, using our (often imperfect) knowledge of the dynamics of the system.

We can build the following estimator

$$\mathbf{x}_{k+1}^{\mathrm{f}} = \mathbf{M}_{k+1}\, \mathbf{x}_k^{\mathrm{a}}. \tag{2.10}$$

The linearity of $\mathbf{M}_{k+1}$ ensures that the estimator is unbiased. The associated error is

$$\begin{aligned}
\mathbf{e}_{k+1}^{\mathrm{f}} &= \mathbf{x}_{k+1}^{\mathrm{f}} - \mathbf{x}_{k+1} && \text{(2.11a)}\\
&= \mathbf{M}_{k+1} \left(\mathbf{x}_k^{\mathrm{a}} - \mathbf{x}_k \right) - \left(\mathbf{x}_{k+1} - \mathbf{M}_{k+1} \mathbf{x}_k \right) && \text{(2.11b)}\\
&= \mathbf{M}_{k+1} \mathbf{e}_k^{\mathrm{a}} - \mathbf{w}_{k+1}, && \text{(2.11c)}
\end{aligned}$$

from which is calculated the forecast error covariance matrix:

$$\mathbf{P}^{\mathrm{f}}_{k+1} = \mathbb{E}\left[\mathbf{e}^{\mathrm{f}}_{k+1}\left(\mathbf{e}^{\mathrm{f}}_{k+1}\right)^{\top}\right] \tag{2.12a}$$

$$= \mathbb{E}\left[\left(\mathbf{M}_{k+1}\mathbf{e}^{\mathrm{a}}_{k} - \mathbf{w}_{k+1}\right)\left(\mathbf{M}_{k+1}\mathbf{e}^{\mathrm{a}}_{k} - \mathbf{w}_{k+1}\right)^{\top}\right] \tag{2.12b}$$

$$= \mathbf{M}_{k+1}\mathbb{E}\left[\left(\mathbf{e}^{\mathrm{a}}_{k}\right)\left(\mathbf{e}^{\mathrm{a}}_{k}\right)^{\top}\right]\mathbf{M}^{\top}_{k+1} + \mathbb{E}\left[\mathbf{w}_{k+1}\mathbf{w}^{\top}_{k+1}\right] \tag{2.12c}$$

$$= \mathbf{M}_{k+1}\mathbf{P}^{\mathrm{a}}_{k}\mathbf{M}^{\top}_{k+1} + \mathbf{Q}_{k+1}. \tag{2.12d}$$

Contrary to the analysis step where the *precision* matrices were additive, the error covariance matrices are additive in the forecast step.

2.2 Summary, limiting cases and example

Kalman filter

1. Initialisation
   - System state $\mathbf{x}^{\mathrm{f}}_0$ and error covariance matrix $\mathbf{P}^{\mathrm{f}}_0$.
2. For $t_k = 1, 2, \ldots$
   (a) Analysis
   - Gain computation $\mathbf{K}^{*}_{k} = \mathbf{P}^{\mathrm{f}}_{k}\mathbf{H}^{\top}_{k}\left(\mathbf{H}_k\mathbf{P}^{\mathrm{f}}_{k}\mathbf{H}^{\top}_{k} + \mathbf{R}_k\right)^{-1}$
   - Computation of the analysis
   $$\mathbf{x}^{\mathrm{a}}_{k} = \mathbf{x}^{\mathrm{f}}_{k} + \mathbf{K}^{*}_{k}\left(\mathbf{y}_k - \mathbf{H}_k\mathbf{x}^{\mathrm{f}}_{k}\right)$$
   - Computation of the error covariance matrix
   $$\mathbf{P}^{\mathrm{a}}_{k} = \left(\mathbf{I} - \mathbf{K}^{*}_{k}\mathbf{H}_k\right)\mathbf{P}^{\mathrm{f}}_{k}$$
   (b) Forecast
   - Computation of the forecast $\mathbf{x}^{\mathrm{f}}_{k+1} = \mathbf{M}_{k+1}\,\mathbf{x}^{\mathrm{a}}_{k}$
   - Computation of the error covariance matrix
   $$\mathbf{P}^{\mathrm{f}}_{k+1} = \mathbf{M}_{k+1}\mathbf{P}^{\mathrm{a}}_{k}\mathbf{M}^{\top}_{k+1} + \mathbf{Q}_{k+1}$$

2.2.1 No observation

When there is not any observation to assimilate, the Kalman filter boils down to the forecast step, which reads

$$\mathbf{x}^{\mathrm{f}}_{k+1} = \mathbf{M}_{k+1}\mathbf{x}^{\mathrm{f}}_{k}, \tag{2.13a}$$

$$\mathbf{P}^{\mathrm{f}}_{k+1} = \mathbf{M}_{k+1}\mathbf{P}^{\mathrm{f}}_{k}\mathbf{M}^{\top}_{k+1} + \mathbf{Q}_{k+1}. \tag{2.13b}$$

If the dynamics is unstable (with positive Lyapunov exponents for instance), the error will grow uncontrolled. Only the assimilation of observations can help reduce the global error.

2.2.2 Perfect observations

Imagine we have a great confidence in the observations, so that we set $\mathbf{R}_k = \mathbf{0}$. Also assume that there are as many observations as state variables and that they are independent. That is to say $\mathbf{H}_k$ est invertible. Then:

$$\mathbf{K}_k^* = \mathbf{P}_k^{\mathrm{f}} \mathbf{H}_k^\top \left(\mathbf{H}_k \mathbf{P}_k^{\mathrm{f}} \mathbf{H}_k^\top + \mathbf{R}_k \right)^{-1} \tag{2.14a}$$

$$= \mathbf{P}_k^{\mathrm{f}} \mathbf{H}_k^\top \left(\mathbf{H}_k^\top \right)^{-1} \left(\mathbf{P}_k^{\mathrm{f}} \right)^{-1} \mathbf{H}_k^{-1} \tag{2.14b}$$

$$= \mathbf{H}_k^{-1}. \tag{2.14c}$$

As a result, we have $\mathbf{P}_k^{\mathrm{a}} = \left(\mathbf{I} - \mathbf{K}_k^* \mathbf{H}_k \right) \mathbf{P}_k^{\mathrm{f}} = \mathbf{0}$, so that $\mathbf{x}_{k+1}^{\mathrm{f}} = \mathbf{M}_{k+1} \mathbf{x}_k^{\mathrm{a}} = \mathbf{M}_{k+1} \mathbf{H}_k^{-1} \mathbf{y}_k$. Moreover the forecast step reads $\mathbf{P}_{k+1}^{\mathrm{f}} = \mathbf{M}_{k+1} \mathbf{P}_k^{\mathrm{a}} \mathbf{M}_{k+1}^\top + \mathbf{Q}_{k+1} = \mathbf{Q}_{k+1}$. The forecast errors only depend on model error, and the system can be perfectly known through observation.

2.2.3 A simple example

We are back to the lifeboat problem from chapter 1. On an hourly basis, the castaway assesses the distance between the lifeboat and the shore and proceeds to an analysis. The distance to the shore, assessed at time t_k, k hours after the shipwreck at time t_0, is denoted y_k. This estimator is assumed to be unbiased. As before, the variance is denoted σ_{o}^2 and is supposed to be stationary. The true coordinates of the lifeboat are (u_k, v_k); the coordinates of the analysis are denoted $(u_k^{\mathrm{a}}, v_k^{\mathrm{a}})$; the coordinates of the forecast are denoted $(u_k^{\mathrm{f}}, v_k^{\mathrm{f}})$.

At the beginning (t_0), one has $(u_0^{\mathrm{a}}, v_0^{\mathrm{a}}) = (0, 0)$ by convention. In between times t_k and t_{k+1} we only know that the lifeboat has drifted, but we do not know in which direction. Our model of the lifeboat amounts to a diffusion of the lifeboat position around the origin. In this example, we have $\mathbf{M}_{k+1} = \mathbf{I}$, the identity matrix. Model error is significant and reads

$$\mathbf{Q}_k = \begin{pmatrix} \sigma_{\mathrm{m}}^2 & 0 \\ 0 & \sigma_{\mathrm{m}}^2 \end{pmatrix}, \tag{2.15}$$

where σ_{m} measures the uncertainty of the drift magnitude of the lifeboat between t_k and t_{k+1}.

Finally, we assume that the analysis and forecast error covariance matrix are diagonal, *i.e.*

$$\mathbf{P}_k^{\mathrm{a}} = \begin{pmatrix} \lambda_k & 0 \\ 0 & \mu_k \end{pmatrix}, \qquad \mathbf{P}_k^{\mathrm{f}} = \begin{pmatrix} \nu_k & 0 \\ 0 & \rho_k \end{pmatrix}, \tag{2.16}$$

because no correlation should be induced between the two coordinates of the lifeboat position. This will be verified a posteriori.

Analysis

Let us focus on time t_k. The previous forecast has yielded state $(u_k^{\mathrm{f}}, v_k^{\mathrm{f}})^\top$. The castaway proceeds to measure v_k^{o}. He performs an optimal analysis with these observations. The

Kalman gain is obtained from the calculation

$$\mathbf{K}_k^* = \begin{pmatrix} \nu_k & 0 \\ 0 & \rho_k \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} \left(\sigma_\mathrm{o}^2 + (0,1) \begin{pmatrix} \nu_k & 0 \\ 0 & \rho_k \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} \right)^{-1} \tag{2.17a}$$

$$= \frac{\rho_k}{\sigma_\mathrm{o}^2 + \rho_k} \begin{pmatrix} 0 \\ 1 \end{pmatrix}. \tag{2.17b}$$

The lifeboat coordinates are estimated to be

$$\begin{pmatrix} u_k^\mathrm{a} \\ v_k^\mathrm{a} \end{pmatrix} = \begin{pmatrix} u_k^\mathrm{f} \\ v_k^\mathrm{f} \end{pmatrix} + \mathbf{K}_k^* \left(y_k - (0,1) \begin{pmatrix} u_k^\mathrm{f} \\ v_k^\mathrm{f} \end{pmatrix} \right) \tag{2.18a}$$

$$= \begin{pmatrix} u_k^\mathrm{f} \\ v_k^\mathrm{f} \end{pmatrix} + \mathbf{K}_k^* \left(y_k - v_k^\mathrm{f} \right) \tag{2.18b}$$

$$= \begin{pmatrix} u_k^\mathrm{f} \\ v_k^\mathrm{f} + \frac{\rho_k}{\sigma_\mathrm{o}^2 + \rho_k} \left(y_k - v_k^\mathrm{f} \right) \end{pmatrix}. \tag{2.18c}$$

From $\mathbf{P}_k^\mathrm{a} = (\mathbf{I} - \mathbf{K}_k^* \mathbf{H}_k)\, \mathbf{P}_k^\mathrm{f}$ we infer that

$$\lambda_k = \nu_k \qquad \text{and} \qquad \frac{1}{\mu_k} = \frac{1}{\sigma_\mathrm{o}^2} + \frac{1}{\rho_k}. \tag{2.19}$$

Forecast

Let us now consider the forecast step. The model does not modify the estimate of the position

$$\begin{pmatrix} u_{k+1}^\mathrm{f} \\ v_{k+1}^\mathrm{f} \end{pmatrix} = \begin{pmatrix} u_k^\mathrm{a} \\ v_k^\mathrm{a} \end{pmatrix}. \tag{2.20}$$

Using $\mathbf{P}_{k+1}^\mathrm{f} = \mathbf{P}_k^\mathrm{a} + \sigma_\mathrm{m}^2\, \mathbf{I}$, we can calculate the forecast error covariance matrix

$$\nu_{k+1} = \lambda_k + \sigma_\mathrm{m}^2, \qquad \rho_{k+1} = \mu_k + \sigma_\mathrm{m}^2. \tag{2.21}$$

From the chaining of the analysis step followed by the forecast step, we have

$$u_{k+1}^\mathrm{f} = u_k^\mathrm{f}, \qquad v_{k+1}^\mathrm{f} = v_k^\mathrm{f} + \frac{\rho_k}{\sigma_\mathrm{o}^2 + \rho_k} \left(y_k - v_k^\mathrm{f} \right), \tag{2.22}$$

and

$$\nu_{k+1} = \nu_k + \sigma_\mathrm{m}^2, \qquad \frac{1}{\rho_{k+1} - \sigma_\mathrm{m}^2} = \frac{1}{\sigma_\mathrm{o}^2} + \frac{1}{\rho_k}. \tag{2.23}$$

Since $u_0^\mathrm{f} = 0$, it is clear that $u_k^\mathrm{f} = 0$ for any k: in the absence of observation, nothing is learnt on the coordinate parallel to the shore. However, the uncertainty is a linear increasing function of time since

$$\nu_k = k\, \sigma_\mathrm{m}^2. \tag{2.24}$$

From Eq. (2.23) on the uncertainty of coordinate v, one infers that $\rho_k \geq \sigma_\mathrm{m}^2$. We can look for a fixed point ρ_* of the recurrence equation satisfying this constraint. It is solution of

$$\rho_*^2 - \sigma_\mathrm{m}^2 \rho_* - \sigma_\mathrm{o}^2 \sigma_\mathrm{m}^2 = 0, \tag{2.25}$$

which leads to

$$\rho_* = \frac{\sigma_{\rm m}^2}{2}\left(1+\sqrt{1+4\frac{\sigma_{\rm o}^2}{\sigma_{\rm m}^2}}\right), \tag{2.26}$$

This is the asymptotic result of a compromise between the reduction of uncertainty due to the assimilation of observations, and to the increase of the same uncertainty due to the uncontrolled drift of the lifeboat. A posteriori, we easily check that $\rho_* \geq \sigma_{\rm m}^2$.

2.2.4 Second example: data assimilation with an oscillator

Consider the discrete model

$$x_0 = 0, \quad x_1 = 1 \quad \text{and for } 1 \leq k \leq K: \quad x_{k+1} - 2x_k + x_{k-1} = -\omega^2 x_k. \tag{2.27}$$

Clearly, this is a numerical implementation of the one-dimensional harmonic oscillator

$$\ddot{x} + \Omega^2 x = 0. \tag{2.28}$$

This is a discrete second-order equation with Ω^2 proportional to ω^2. Therefore, a state vector has two entries

$$\mathbf{u}_k = \begin{pmatrix} x_k \\ x_{k-1} \end{pmatrix} \tag{2.29}$$

The resolvent matrix is

$$\mathbf{M}_k = \begin{pmatrix} 2-\omega^2 & -1 \\ 1 & 0 \end{pmatrix}, \tag{2.30}$$

such that $\mathbf{u}_k = \mathbf{M}_k \mathbf{u}_{k-1}$ for $k \geq 1$. The observation operator is $\mathbf{H}_k = \begin{pmatrix} 1 & 0 \end{pmatrix}$. The observation equation is

$$y_k = \mathbf{H}_k\, \mathbf{u}_k + \xi_k, \tag{2.31}$$

with a Gaussian white noise ξ_k of variance g. The value of this variance is supposed to be known. Observations may not be available at each time step.

The Kalman filter is tested on this example. Figure 2.2 is a representation of a data assimilation experiment with a given set of observations.

2.3 The extended Kalman filter

Up to now, we have assumed that the hypothesis of linearity is valid. However, the following conditions are frequently met in geophysical data assimilation:

- the observation operator H_k might be nonlinear. It if often the case when satellite observations are considered. The observation operator that relates optical measurements (radiances) to the state variables may involve a radiative transfer operator. When diffusion is taken into account, this model might become strongly nonlinear. Lidar and radar observations are other important examples,
- the evolution model M_k could be nonlinear. This is the case for the primitive equations (atmosphere and ocean). This is also the case in atmospheric chemistry assuming the chemical reactions are second-order.

This does not mean that we have to abandon the Kalman filter. Instead, we need to adapt it to handle the potential nonlinearity of these operators.

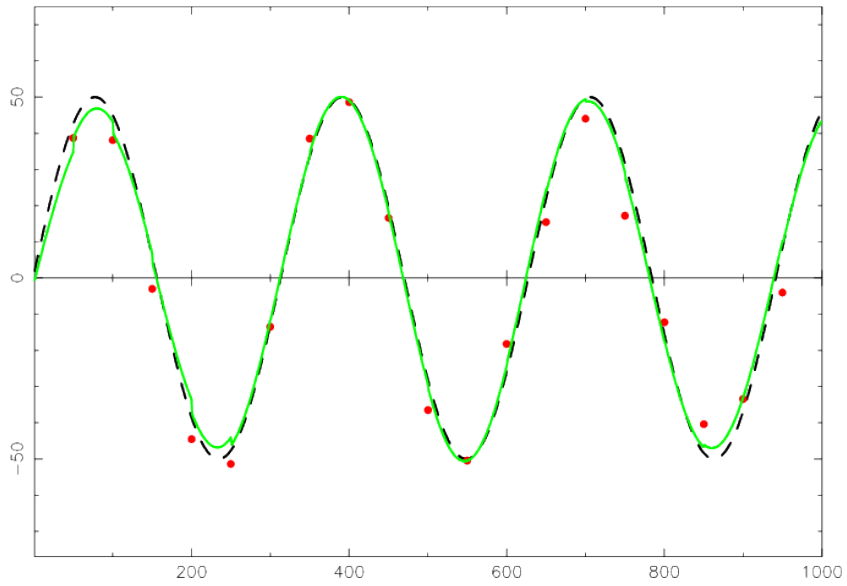


Figure 2.2: The model trajectory is represented by the dashed curve. In addition to the setup parameters given in the text, we have chosen $K = 1000$ time units, $\omega = 0.02\ \mathrm{s}^{-1}$. The dots represent the observations, of variance $g = 7$. The observations are acquired every $\Delta = 50$ time units. The full curve is the forecast of $\mathbf{H}_k\,\mathbf{u}_k^{\mathrm{f}}$ by the data assimilation system.

2.3.1 Linearising the forecast step and the analysis step

Without mystery, the forecast is given by

$$\mathbf{x}_{k+1}^{\mathrm{f}} = M_{k+1}[\mathbf{x}_k^{\mathrm{a}}]. \tag{2.32}$$

It can be shown that this estimator is only accurate to the first order in perturbations and that it can be refined at some (important) numerical cost.

To obtain the forecast error covariance matrix, we shall use the fact that $\mathbf{x}_k^{\mathrm{a}}$ is meant to be close enough to $\mathbf{x}_k$. The tangent linear matrix of M_k at $\mathbf{x}_k$ will be denoted $\mathbf{M}_k$. Let us go again through the forecast step with error

$$\begin{aligned}
\mathbf{e}_{k+1}^{\mathrm{f}} &= \mathbf{x}_{k+1}^{\mathrm{f}} - \mathbf{x}_{k+1} = M_{k+1}[\mathbf{x}_k^{\mathrm{a}}] - \mathbf{x}_{k+1} && \text{(2.33a)}\\
&= M_{k+1}\left[\mathbf{x}_k + (\mathbf{x}_k^{\mathrm{a}} - \mathbf{x}_k)\right] - \mathbf{x}_{k+1} && \text{(2.33b)}\\
&\simeq M_{k+1}[\mathbf{x}_k] + \mathbf{M}_{k+1}(\mathbf{x}_k^{\mathrm{a}} - \mathbf{x}_k) - \mathbf{x}_{k+1} && \text{(2.33c)}\\
&\simeq \mathbf{M}_{k+1}\mathbf{e}_k^{\mathrm{a}} - \mathbf{w}_{k+1}. && \text{(2.33d)}
\end{aligned}$$

As a result, we obtain

$$\mathbf{P}_{k+1}^{\mathrm{f}} = \mathbf{M}_{k+1}\mathbf{P}_k^{\mathrm{a}}\mathbf{M}_{k+1}^{\top} + \mathbf{Q}_{k+1}. \tag{2.34}$$

The estimator for the analysis step remains given by an analysis of the form

$$\mathbf{x}_k^{\mathrm{a}} = \mathbf{x}_k^{\mathrm{f}} + \mathbf{K}_k\left(\mathbf{y}_k - H_k[\mathbf{x}_k^{\mathrm{f}}]\right). \tag{2.35}$$

We infer

$$\mathbf{e}_k^{\mathrm{a}} = \mathbf{x}_k^{\mathrm{a}} - \mathbf{x}_k = \mathbf{x}_k^{\mathrm{f}} - \mathbf{x}_k + \mathbf{K}_k \left(\mathbf{y}_k - H_k[\mathbf{x}_k] + H_k[\mathbf{x}_k] - H_k[\mathbf{x}_k^{\mathrm{f}}] \right) \tag{2.36a}$$

$$= \mathbf{x}_k^{\mathrm{f}} - \mathbf{x}_k + \mathbf{K}_k \left(\mathbf{y}_k - H_k[\mathbf{x}_k] + H_k[\mathbf{x}_k] - H_k[\mathbf{x}_k^{\mathrm{f}} - \mathbf{x}_k + \mathbf{x}_k] \right) \tag{2.36b}$$

$$\simeq \mathbf{e}_k^{\mathrm{f}} + \mathbf{K}_k \left(\mathbf{e}_k^{\mathrm{o}} - \mathbf{H}_k \, \mathbf{e}_k^{\mathrm{f}} \right). \tag{2.36c}$$

and

$$\mathbf{P}_k^{\mathrm{a}} = (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k) \, \mathbf{P}_k^{\mathrm{f}} \, (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k)^\top + \mathbf{K}_k \mathbf{R}_k \mathbf{K}_k^\top. \tag{2.37}$$

The Kalman gain becomes

$$\mathbf{K}_k^* = \mathbf{P}_k^{\mathrm{f}} \mathbf{H}_k^\top \left(\mathbf{H}_k \mathbf{P}_k^{\mathrm{f}} \mathbf{H}_k^\top + \mathbf{R}_k \right)^{-1}. \tag{2.38}$$

and, consequently, one has

$$\mathbf{P}_k^{\mathrm{a}} = (\mathbf{I} - \mathbf{K}_k^* \mathbf{H}_k) \, \mathbf{P}_k^{\mathrm{f}}. \tag{2.39}$$

The optimal estimator is therefore

$$\mathbf{x}_k^{\mathrm{a}} = \mathbf{x}_k^{\mathrm{f}} + \mathbf{K}_k^* \left(\mathbf{y}_k - H_k[\mathbf{x}_k^{\mathrm{f}}] \right). \tag{2.40}$$

Note that the tangent linear of H_k is used in $\mathbf{K}_k^*$, but it is not used in the innovation vector $\mathbf{y}_k - H_k[\mathbf{x}_k^{\mathrm{f}}]$.

2.3.2 Summary

These modifications that extent the Kalman filter to the nonlinear context yields the *extended Kalman filter*.

Extended Kalman filter

1. Initialisation
   - System state $\mathbf{x}_0^{\mathrm{f}}$ and error covariance matrix $\mathbf{P}_0^{\mathrm{f}}$.
2. For $t_k = 1, 2, \ldots$
   (a) Analysis
   - Gain computation

$$\mathbf{K}_k^* = \mathbf{P}_k^{\mathrm{f}} \mathbf{H}_k^\top \left(\mathbf{H}_k \mathbf{P}_k^{\mathrm{f}} \mathbf{H}_k^\top + \mathbf{R}_k \right)^{-1}$$

   - Analysis computation

$$\mathbf{x}_k^{\mathrm{a}} = \mathbf{x}_k^{\mathrm{f}} + \mathbf{K}_k^* \left(\mathbf{y}_k - H_k[\mathbf{x}_k^{\mathrm{f}}] \right)$$

   - Error covariance matrix computation

$$\mathbf{P}_k^{\mathrm{a}} = \left(\mathbf{I} - \mathbf{K}_k^* \mathbf{H}_k \right) \mathbf{P}_k^{\mathrm{f}}$$

   (b) Forecast
   - Forecast state computation $\mathbf{x}_{k+1}^{\mathrm{f}} = M_{k+1}[\mathbf{x}_k^{\mathrm{a}}]$
   - Error covariance matrix computation

$$\mathbf{P}_{k+1}^{\mathrm{f}} = \mathbf{M}_{k+1} \mathbf{P}_k^{\mathrm{a}} \mathbf{M}_{k+1}^\top + \mathbf{Q}_{k+1}$$

2.3.3 Data assimilation with a non-harmonic oscillator

The following discrete model

$$x_0 = 0, \quad x_1 = 1 \quad \text{and for } 1 \leq k \leq K: \quad x_{k+1} - 2x_k + x_{k-1} = \omega^2 \, x_k - \lambda^2 \, x_k^3, \tag{2.41}$$

is a numerical implementation of the anharmonic one-dimensional oscillator

$$\ddot{x} - \Omega^2 \, x + \Lambda^2 \, x^3 = 0, \tag{2.42}$$

where Ω^2 is proportional to ω^2 and Λ^2 is proportional to λ^2. The related potential is

$$V(x) = -\frac{1}{2}\Omega^2 x^2 + \frac{1}{4}\Lambda^2 x^4. \tag{2.43}$$

The second term stabilises the oscillator and plays the role of a spring force, whereas the first term destabilises the point $x = 0$, leading to two potential wells. It is is a second-order discrete equation, with a state vector that can be advantageously written

$$\mathbf{u}_k = \begin{pmatrix} x_k \\ x_{k-1} \end{pmatrix}. \tag{2.44}$$

The state-dependent transition matrix is

$$M_k = \begin{pmatrix} 2 + \omega^2 - \lambda^2 x_{k-1}^2 & -1 \\ 1 & 0 \end{pmatrix}, \tag{2.45}$$

such that $\mathbf{u}_k = \mathbf{M}_k \mathbf{u}_{k-1}$ for $k \geq 1$. The observation operator is $\mathbf{H}_k = (1, 0)$. The observation equation is

$$y_k = \mathbf{H}_k \, \mathbf{u}_k + \xi_k, \tag{2.46}$$

with a Gaussian white noise ξ_k of variance g. The value of this variance is supposed to be known. The extended Kalman filter is tested with this nonlinear system. Figure 2.3 gives an example of a data assimilation experiment with a given set of observations.

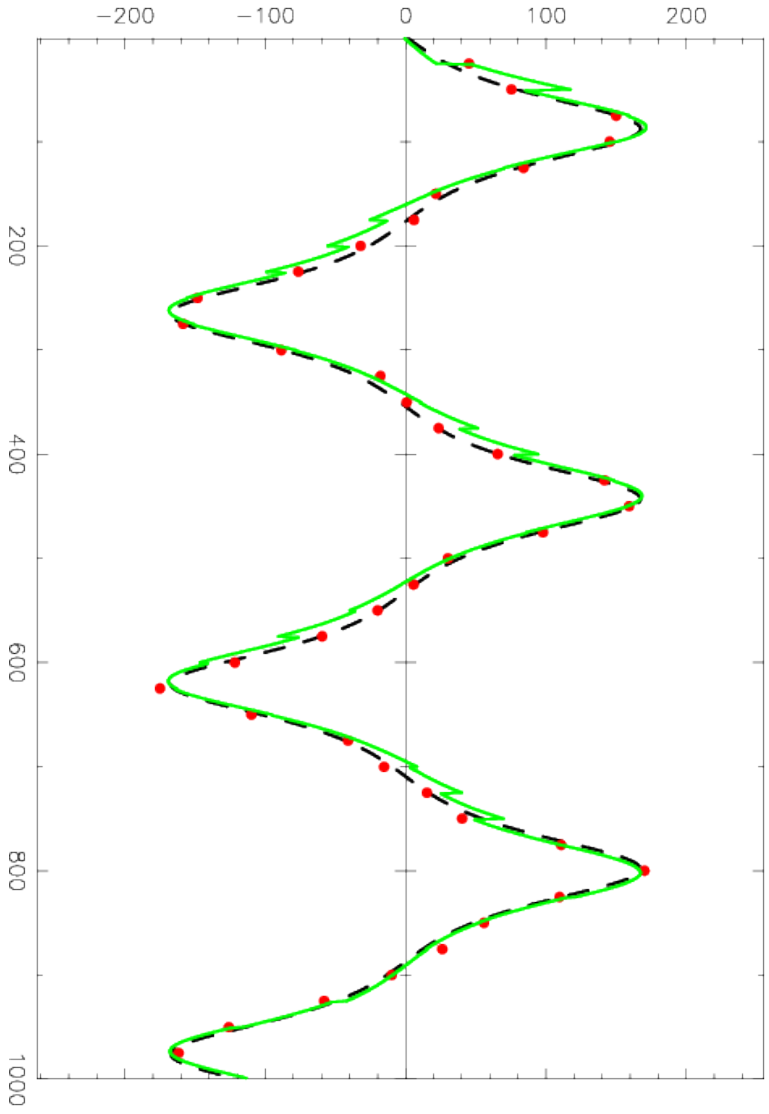


Figure 2.3: The dashed line represents the model trajectory (the truth). In addition to the setup parameters given in the text, we choose $K = 1000$ time steps, $\omega = 0.035$ s^{-1}, and $\lambda = 0.003$ s^{-1}. The dots are the observations, of error variance $g = 7$. The observations are acquired every $\Delta = 25$ time units. The full line represents the forecast of $\mathbf{H}_k \, \mathbf{u}_k^{\mathrm{f}}$ yielded by the extended Kalman filter.

Chapter 3

A generalised variational formalism: 4D-Var

On the one hand, we observed in the previous chapter that optimal interpolation could be generalised to sequential interpolation when

- the system has a temporal dimension,
- its dynamics is described by a model (perfect or not, deterministic or stochastic).

This has led to the Kalman filter formalism.

On the other hand, in chapter 1, we extended the optimal interpolation technique to a variational approach (3D-Var), which offers

- an efficient numerical implementation,
- an (a priori) easy and comprehensive generalisation to nonlinear problems.

Provided some conditions to be discussed later are met, we can contemplate a similar extension for the Kalman filter as seen in chapter 2 by introducing a variational formalism known as ***4D-Var***.

A rigorous equivalence between the analysis step of 4D-Var and that of the Kalman filter at the end of a specific data assimilation time window can only be proven when the observation operator H and the dynamics M are linear. These operators can depend on time (which is recalled by the notation H_k and M_k), and we assume so in this chapter. In addition, we will assume that the evolution model is deterministic and perfect. That is to say, it mirrors the true model of the system and model errors are negligible. The variational problem is said to be implemented with ***strong constraints***.

3.1 Cost function and how to compute its gradient

In this chapter, we consider a partial trajectory of the system state, *i.e.* the $K+1$ vectors $\mathbf{x}_k$ for each of the $K+1$ instants $t_0, t_1, \ldots, t_K$. Each of these state vectors belong to $\mathbb{R}^{N_\mathrm{x}}$.

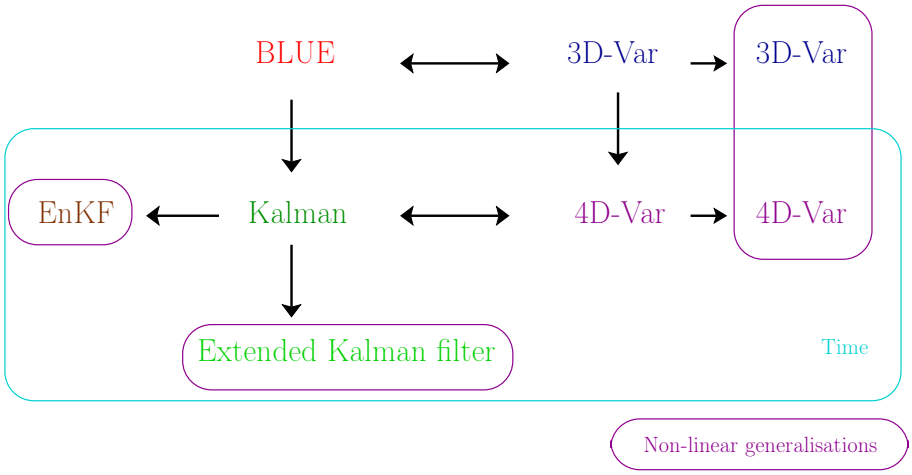


Figure 3.1: The main standard data assimilation schemes and their connections. The EnKF has not been studied yet. 4D-Var is the focus of this chapter.

In addition, we assume that for each of these instants, an observation vector $\mathbf{y}_k \in \mathbb{R}^{N_\mathrm{y}}$ is acquired. Formally, we can generalise the 3D-Var functional (1.41) to

$$J(\mathbf{x}) = \frac{1}{2}\left(\mathbf{x}_0 - \mathbf{x}_0^\mathrm{b}\right)^\top \mathbf{B}_0^{-1}\left(\mathbf{x}_0 - \mathbf{x}_0^\mathrm{b}\right) + \frac{1}{2}\sum_{k=0}^{K}\left(\mathbf{y}_k - \mathbf{H}_k\mathbf{x}_k\right)^\top \mathbf{R}_k^{-1}\left(\mathbf{y}_k - \mathbf{H}_k\mathbf{x}_k\right). \tag{3.1}$$

This functional is defined under the constraint that the state vectors $\{\mathbf{x}_k\}_{k=0,\ldots,K}$ (from which J depends) form an admissible trajectory of the system state, that is to say:

$$\text{for } k = 0, \ldots, K-1: \quad \mathbf{x}_{k+1} = M_{k+1}(\mathbf{x}_k). \tag{3.2}$$

It is momentarily assumed that $M_k \triangleq \mathbf{M}_k$ is linear. Quite often, a background information does not exist for the full trajectory (though the assumption is currently being studied by researchers). However, a background of the initial state of the partial trajectory is very often used. Let us assume that its expectation is $\mathbf{x}_0^\mathrm{b}$ and its error covariance matrix is $\mathbf{B}_0$.

The straightforward way to introduce the model constraint in the cost function is to introduce Lagrange multipliers $\{\mathbf{\Lambda}_k\}_{k=1,\ldots,K}$, where $\mathbf{\Lambda}_k \in \mathbb{R}^{N_\mathrm{x}}$. The associated Lagrangian looks like

$$L(\mathbf{x}, \mathbf{\Lambda}) = J(\mathbf{x}) + \sum_{k=1}^{K} \mathbf{\Lambda}_k^\top \left(\mathbf{x}_k - \mathbf{M}_k\,\mathbf{x}_{k-1}\right). \tag{3.3}$$

This formalism is elegant. Besides, it is quite practical when the variational formalism is in the continuous time limit (time step $t_{k+1} - t_k \longrightarrow 0$), but with no significant added value in the discrete case, where a straightforward reasoning (without the Λ_k) is as efficient as the Lagrangian approach can be. Indeed, in the time discrete framework, the state variable at time t_k, $\mathbf{x}_k$ can be explicitly written in terms of the initial value $\mathbf{x}_0$, as the product of the resolvent matrices applied to the initial state, *i.e.*

$$\mathbf{x}_k = \mathbf{M}_k\mathbf{M}_{k-1}...\mathbf{M}_1\,\mathbf{x}_0. \tag{3.4}$$

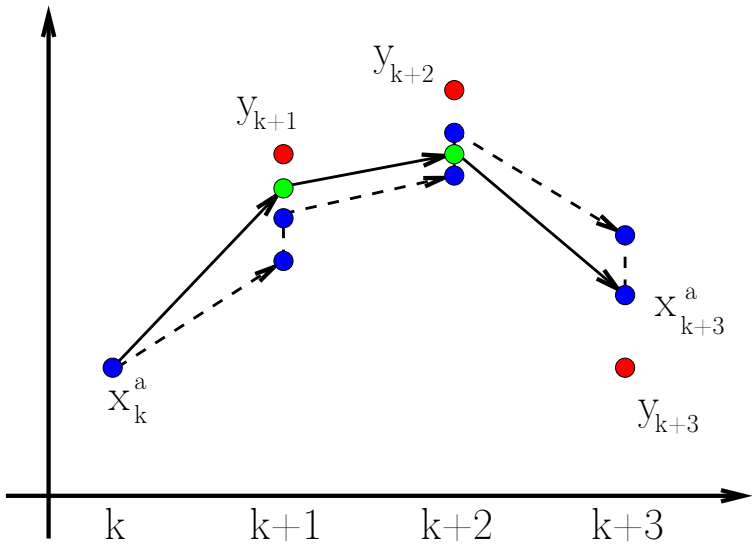


Figure 3.2: 4D-Var data assimilation compared to data assimilation with a Kalman filter.

Note that, with a continuous time model, the state variables are solutions of a system of ordinary differential equations that are less tractable. As a consequence, the functional (3.1) under constraints is an effective functional of $\mathbf{x}_0$ that can now be minimised with respect to $\mathbf{x}_0$. In order to efficiently minimise it, we wish to compute the gradient of J with respect to the initial vector $\mathbf{x}_0$. We introduce:

$$\mathbf{d}_k = \mathbf{y}_k - \mathbf{H}_k \left[\mathbf{M}_k \mathbf{M}_{k-1} ... \mathbf{M}_1\right] \mathbf{x}_0 \qquad \text{and} \quad \Delta_k = \mathbf{R}_k^{-1} \mathbf{d}_k. \tag{3.5}$$

$\mathbf{d}_k$ is an ***innovation vector*** while Δ_k is a ***normalised innovation vector***. For the sake of simplicity, we leave aside the background term in $J(\mathbf{x})$ because such term has already been studied in the optimal interpolation framework, and it is easy to add its contribution afterwards. We have:

$$\delta J(\mathbf{x}_0) = \delta \left\{ \frac{1}{2} \sum_{k=0}^{K} \mathbf{d}_k^\top \mathbf{R}_k^{-1} \mathbf{d}_k \right\} \tag{3.6a}$$

$$= \frac{1}{2} \sum_{k=0}^{K} \delta\mathbf{d}_k^\top \mathbf{R}_k^{-1} \mathbf{d}_k + \frac{1}{2} \sum_{k=0}^{K} \mathbf{d}_k^\top \mathbf{R}_k^{-1} \delta\mathbf{d}_k \tag{3.6b}$$

$$= \sum_{k=0}^{K} \left(\delta\mathbf{d}_k\right)^\top \mathbf{R}_k^{-1} \mathbf{d}_k \tag{3.6c}$$

$$= -\sum_{k=0}^{K} \left(\mathbf{H}_k \left[\mathbf{M}_k \mathbf{M}_{k-1} ... \mathbf{M}_1\right] \delta\mathbf{x}_0\right)^\top \Delta_k \tag{3.6d}$$

$$= -\sum_{k=0}^{K} \delta\mathbf{x}_0^\top \left[\mathbf{M}_1^\top \mathbf{M}_2^\top ... \mathbf{M}_{k-1}^\top \mathbf{M}_k^\top\right] \mathbf{H}_k^\top \Delta_k. \tag{3.6e}$$

We obtain the gradient of the function with respect to $\mathbf{x}_0$

$$\nabla_{\mathbf{x}_0} J = -\sum_{k=0}^{K} \left[\mathbf{M}_1^\top \mathbf{M}_2^\top ... \mathbf{M}_{k-1}^\top \mathbf{M}_k^\top\right] \mathbf{H}_k^\top \Delta_k \tag{3.7a}$$

$$= -\left(\mathbf{H}_0^\top \Delta_0 + \mathbf{M}_1^\top \left[\mathbf{H}_1^\top \Delta_1 + \mathbf{M}_2^\top \left[\mathbf{H}_2^\top \Delta_2 + ... + \left[\mathbf{M}_K^\top \mathbf{H}_K^\top \Delta_K\right] ...\right]\right]\right). \tag{3.7b}$$

This last form of the gradient, which is a ***Horner factorisation***, provides a means to compute it at a lower numerical cost.

1. We first compute the $\mathbf{x}_k$ recursively thanks to the resolvent matrix $\mathbf{M}_k$ and the initial variable $\mathbf{x}_0$.
2. Then one computes the normalised innovation vectors $\Delta_k = \mathbf{R}_k^{-1}(\mathbf{y}_k - \mathbf{H}_k \mathbf{x}_k)$ that we store.
3. We then define an ***adjoint state variable*** $\tilde{\mathbf{x}}_k \in \mathbb{R}^{N_\mathrm{x}}$. It is defined at the final instant by $\tilde{\mathbf{x}}_K = \mathbf{H}_K^\top \Delta_K$. Knowing $\tilde{\mathbf{x}}_{k+1}$, we obtain $\tilde{\mathbf{x}}_k$ using
$$\tilde{\mathbf{x}}_k = \mathbf{H}_k^\top \Delta_k + \mathbf{M}_{k+1}^\top \tilde{\mathbf{x}}_{k+1}. \tag{3.8}$$
One can move back in time recursively, back to $\tilde{\mathbf{x}}_0$.
4. Finally, we obtain $\nabla_{\mathbf{x}_0} J = -\tilde{\mathbf{x}}_0$.

To compute the adjoint state variable backward in time, it is necessary to compute the adjoint operator of the resolvent matrices $\mathbf{M}_k$. This seems trivial when $\mathbf{M}_k$ happens to be a matrix, even a large one since computing its adjoint amounts to taking its transpose. However, in practice, the system's evolution is computed from a numerical code of several dozen of thousand of lines. Hence, we need to compute the (formally well-established) adjoint of this computer programme. This ***adjointisation*** represents a technical and mathematical hardship. Today, it is common to resort to ***automatic differentiation*** programmes (see chapter 15 in Blayo et al. (2015) by L. Hacoët) which, from the code of M_k, yield the code of $\mathbf{M}_k^\top$. Examples of such programmes are: TAPENADE by INRIA (the code to be differentiated can be submitted online and its adjoint code will be returned), TAF (a famous commercial product), but also OpenAD, ADIFOR, ADIC, ADOL-C, etc. More recently, deep learning softwares such as TensorFlow (Google), PyTorch (Facebook) or Theano (University of Montreal) provide adjoint differentiation tools (coined ***retropropagation***) for their neural networks numerical representations. Julia, a recent computer science oriented language, lends itself much more easily to automatic differentiation.

Gradient with background term

Accounting for the background term in the cost function is easy; the gradient becomes:

$$\nabla_{\mathbf{x}_0} J = \mathbf{B}_0^{-1}(\mathbf{x}_0 - \mathbf{x}_0^\mathrm{b}) - \sum_{k=0}^{K} \mathbf{M}_{k,0}^\top \mathbf{H}_k^\top \mathbf{R}_k^{-1}(\mathbf{y}_k - \mathbf{H}_k \mathbf{M}_{k,0} \mathbf{x}_0), \tag{3.9}$$

where for $k > l \geq 0$, we have noted $\mathbf{M}_{k,l} = \mathbf{M}_k \mathbf{M}_{k-1} \cdots \mathbf{M}_{l+1}$. By convention, we choose $\mathbf{M}_{k,k} = \mathbf{I}$. By extension, we impose $\mathbf{M}_{k,l} = \mathbf{M}_{l,k}^{-1} = \mathbf{M}_{k+1}^{-1} \mathbf{M}_{k+2}^{-1} \cdots \mathbf{M}_l^{-1}$ for $l > k \geq 0$. Owing to this convention, we have

$$\forall\, k, l \qquad 0 \leq k,\, l \leq n : \qquad \mathbf{x}_k = \mathbf{M}_{k,l}\, \mathbf{x}_l. \tag{3.10}$$

3.1.1 What happens when the forward model or the observation model are nonlinear?

In that case, we can follow the method of section 1.3.3 used so far within the 3D-Var approach. As before, we denote $\mathbf{M}$ the tangent linear of M with respect to $\mathbf{x}$. One can possibly indicate the point where the tangent linear is computed: $\mathbf{M}(\mathbf{x}_0)$ for instance when evaluated at $\mathbf{x}_0$.

The computation of the gradient is not as simple as in the linear case. Mainly, the calculation of $\delta\mathbf{d}_k$ is modified into (Leibnitz's rule)

$$\delta\mathbf{d}_k = -\mathbf{H}_k(\mathbf{x}_k)\mathbf{M}_k(\mathbf{x}_{k-1})\mathbf{M}_{k-1}(\mathbf{x}_{k-2})\cdots\mathbf{M}_1(\mathbf{x}_0)\,\delta\mathbf{x}_0. \tag{3.11}$$

Then the gradient reads

$$\nabla_{\mathbf{x}_0}J = -\sum_{k=0}^{K}\left[\mathbf{M}_1^\top(\mathbf{x}_0)\mathbf{M}_2^\top(\mathbf{x}_1)\cdots\mathbf{M}_{k-1}^\top(\mathbf{x}_{k-2})\mathbf{M}_k^\top(\mathbf{x}_{k-1})\right]\mathbf{H}_k^\top(\mathbf{x}_k)\Delta_k. \tag{3.12}$$

When considering complex and nonlinear evolution or observation operators, it turns out that we will need not only to know how to derive an adjoint but also the differential (the tangent linear) of the code in the first place. This is again called ***automatic differentiation***.

3.2 Solution of the variational problem

Our first objective is to minimise the cost function $J(\mathbf{x})$. Later, we will show that this variational approach can be equivalent to the Kalman filter under certain conditions.

3.2.1 Solution with respect to $\mathbf{x}_0$

We can obtain a solution to the minimisation problem of J with respect to $\mathbf{x}_0$ by using our expression of the gradient and solving $\nabla_{\mathbf{x}_0}J = 0$. To do so, one can define the Hessian of J over the time interval $[t_0, t_K]$, which will be denoted $\mathcal{H}_{K,0}$. If the cost function is quadratic (which is what we assumed so far), the Hessian matrix does not depend on $\mathbf{x}_0$ and is formally easily obtained from the gradient:

$$\mathcal{H}_{K,0} = \mathbf{B}_0^{-1} + \sum_{k=0}^{K}\mathbf{M}_{k,0}^\top\mathbf{H}_k^\top\mathbf{R}_k^{-1}\mathbf{H}_k\mathbf{M}_{k,0}\,. \tag{3.13}$$

With these notations, the solution $\mathbf{x}_0^{\mathrm{a}}$ is easily obtained from (3.9)

$$\mathbf{x}_0^{\mathrm{a}} = \mathbf{x}_0^{\mathrm{b}} - \mathcal{H}_{K,0}^{-1}\nabla_{\mathbf{x}_0}J(\mathbf{x}_0^{\mathrm{b}}). \tag{3.14}$$

This is the Newton solution to the minimisation problem of a quadratic functional.

If the model is nonlinear and the cost-function is non-quadratic, Eq. (3.13) would turn out to only be an approximation of the exact Hessian since second-order derivatives of the model would appear in the exact Hessian. In that case, the *Newton method* is rather called the *Gauss-Newton method*.

The analysis error of this estimate is given by the following error covariance matrix

$$\mathbf{P}_0^{\mathrm{a}} = \mathrm{E}\left[\left(\mathbf{x}_0^{\mathrm{a}} - \mathbf{x}_0\right)\left(\mathbf{x}_0^{\mathrm{a}} - \mathbf{x}_0\right)^{\top}\right]. \tag{3.15}$$

Hence, we obtain

$$\left(\mathbf{P}_0^{\mathrm{a}}\right)^{-1} = \mathcal{H}_{K\,,0} = \mathbf{B}_0^{-1} + \sum_{k=0}^{K} \mathbf{M}_{k\,,0}^{\top}\,\mathbf{H}_k^{\top}\mathbf{R}_k^{-1}\mathbf{H}_k\mathbf{M}_{k\,,0}\,. \tag{3.16}$$

Solution with respect to $\mathbf{x}_k$

When the evolution model is perfect, the (true) values of the system state $\mathbf{x}_k$ are all connected thanks to the resolvent matrices $\mathbf{M}_{k,l}$. Performing the analysis on the initial condition $\mathbf{x}_0$ is therefore equivalent to performing the analysis on any $\mathbf{x}_j$, $j \in [0, K]$. To make it clearer, one can rewrite the cost function (3.1) with respect to $\mathbf{x}_j$ ($K \geq j \geq 0$) and not $\mathbf{x}_0$ anymore following

$$\begin{aligned} J(\mathbf{x}) =& \frac{1}{2}\left(\mathbf{M}_{0\,,j}\left(\mathbf{x}_j - \mathbf{x}_j^{\mathrm{b}}\right)\right)^{\top}\mathbf{B}_0^{-1}\mathbf{M}_{0\,,j}\left(\mathbf{x}_j - \mathbf{x}_j^{\mathrm{b}}\right) \\ &+ \frac{1}{2}\sum_{k=0}^{K}\left(\mathbf{y}_k - \mathbf{H}_k\mathbf{M}_{k\,,j}\,\mathbf{x}_j\right)\mathbf{R}_k^{-1}\left(\mathbf{y}_k - \mathbf{H}_k\mathbf{M}_{k\,,j}\,\mathbf{x}_j\right), \end{aligned} \tag{3.17}$$

using the convention (3.10). We observe that $\mathbf{M}_{0\,,j}^{\top}\mathbf{B}_0^{-1}\mathbf{M}_{0\,,j} = \left(\mathbf{M}_{j\,,0}\,\mathbf{B}_0\mathbf{M}_{j\,,0}^{\top}\right)^{-1} \triangleq \mathbf{B}_j^{-1}$. The calculation of the gradient with respect to $\mathbf{x}_j$ follows

$$-\nabla_{\mathbf{x}_j} J = \mathbf{B}_j^{-1}\left(\mathbf{x}_j^{\mathrm{b}} - \mathbf{x}_j\right) + \sum_{k=0}^{K}\mathbf{M}_{k\,,j}^{\top}\,\mathbf{H}_k^{\top}\mathbf{R}_k^{-1}\left(\mathbf{y}_k - \mathbf{H}_k\mathbf{M}_{k\,,j}\,\mathbf{x}_j\right). \tag{3.18}$$

This leads to the analysis defined by $\nabla_{\mathbf{x}_j} J(\mathbf{x}) = \mathbf{0}$,

$$\mathbf{x}_j^{\mathrm{a}} = \mathbf{x}_j^{\mathrm{b}} - \mathcal{H}_{K\,,j}^{-1}\nabla_{\mathbf{x}_j} J(\mathbf{x}_j^{\mathrm{b}}), \tag{3.19}$$

where the Hessian is

$$\mathcal{H}_{K\,,j} = \mathbf{B}_k^{-1} + \sum_{k=0}^{K}\mathbf{M}_{k\,,j}^{\top}\,\mathbf{H}_k^{\top}\mathbf{R}_k^{-1}\mathbf{H}_k\mathbf{M}_{k\,,j}\,. \tag{3.20}$$

The related analysis error covariance matrix is

$$\mathbf{P}_k^{\mathrm{a}} = \mathrm{E}\left[\left(\mathbf{x}_j^{\mathrm{a}} - \mathbf{x}_j\right)\left(\mathbf{x}_j^{\mathrm{a}} - \mathbf{x}_j\right)^{\top}\right]. \tag{3.21}$$

By analogy with the result (1.46), we infer

$$\left(\mathbf{P}_j^{\mathrm{a}}\right)^{-1} = \mathcal{H}_{K\,,j} = \mathbf{B}_j^{-1} + \sum_{k=0}^{K}\mathbf{M}_{k\,,j}^{\top}\,\mathbf{H}_k^{\top}\mathbf{R}_k^{-1}\mathbf{H}_k\mathbf{M}_{k\,,j}\,. \tag{3.22}$$

This means that the confidence matrix at $\mathbf{x}_j^{\mathrm{a}}$ is the sum of the confidence of the background and of the confidence attached to the observations propagated by the inverse evolution model back to time t_0.

3.3 Properties

3.3.1 Propagation of the analysis error

It is easy to check that

$$\mathbf{M}_{j\,,l}^{\top}\,\mathcal{H}_{K\,,j}\,\mathbf{M}_{j\,,l}\,=\mathcal{H}_{K\,,l}. \tag{3.23}$$

Because of $\mathcal{H}_{K\,,k}=(\mathbf{P}_k^{\mathrm{a}})^{-1}$, it leads to

$$\mathbf{P}_l^{\mathrm{a}}=\mathbf{M}_{l\,,j}\,\mathbf{P}_j^{\mathrm{a}}\,\mathbf{M}_{l\,,j}^{\top}\,. \tag{3.24}$$

This describes how the analysis error is propagated within the (strong constraint) 4D-Var formalism. Obviously, it is different from the error propagation of the Kalman filter. It originated from the fact that the error analysis in the Kalman filter is causal, meaning that the error depends on the observations before the analysis time, when the 4D-Var analysis error is not causal: the error analysis might depend on future observations.

This relates to one of the weakest point of 4D-Var: it is not simple to estimate the posterior uncertainty.

3.3.2 Transferability of optimality

We now have the main tools to demonstrate one of the most appealing property of the minimisation of the functional. We consider a data assimilation window $[t_0,t_K]$. It is split into two periods $[t_0,t_m]$ and $[t_m,t_K]$ with m an integer between 0 and K. One can minimise the associated cost function following two paths:

1. The full cost function is minimised over the full interval $[t_0,t_K]$.
2. The minimisation is carried out in two steps. We first minimise the cost function on the interval $[t_0,t_m]$. The result is a solution $\mathbf{x}_m^{\mathrm{a}}$ defined at t_m, with an error covariance matrix $\mathbf{P}_m^{\mathrm{a}}$. Then, a second minimisation is carried out on interval $[t_m,t_K]$, **but** using the outcome of the first step, $\mathbf{x}_m^{\mathrm{a}}$ and $\mathbf{P}_m^{\mathrm{a}}$, in a background term in the second cost function.

We would like to show that the two approaches are equivalent.

From what precedes, the result of path (1) is the outcome of the optimisation of functional (3.1) with respect to $\mathbf{x}_0$

$$\begin{aligned}J(\mathbf{x}_0)=&\frac{1}{2}\left(\mathbf{x}_0-\mathbf{x}_0^{\mathrm{b}}\right)^{\top}\mathbf{B}_0^{-1}\left(\mathbf{x}_0-\mathbf{x}_0^{\mathrm{b}}\right)\\&+\frac{1}{2}\sum_{k=0}^{K}\left(\mathbf{x}_k-\mathbf{H}_k\mathbf{M}_{k\,,0}\,\mathbf{x}_0\right)^{\top}\mathbf{R}_k^{-1}\left(\mathbf{y}_k-\mathbf{H}_k\mathbf{M}_{k\,,0}\,\mathbf{x}_0\right).\end{aligned} \tag{3.25}$$

We can decompose the same cost function into

$$\begin{aligned}J_{m\,,0}(\mathbf{x}_0)=&\frac{1}{2}\left(\mathbf{x}_0-\mathbf{x}_0^{\mathrm{b}}\right)^{\top}\mathbf{B}_0^{-1}\left(\mathbf{x}_0-\mathbf{x}_0^{\mathrm{b}}\right)\\&+\frac{1}{2}\sum_{k=0}^{m}\left(\mathbf{y}_k-\mathbf{H}_k\mathbf{M}_{k\,,0}\,\mathbf{x}_0\right)^{\top}\mathbf{R}_k^{-1}\left(\mathbf{y}_k-\mathbf{H}_k\mathbf{M}_{k\,,0}\,\mathbf{x}_0\right),\end{aligned} \tag{3.26}$$

and

$$J_{K\,,\,m}(x_0) = \frac{1}{2}\sum_{k=m+1}^{K} \left(\mathbf{y}_k - \mathbf{H}_k\mathbf{M}_{k\,,\,0}\,\mathbf{x}_0\right)^\top \mathbf{R}_k^{-1}\left(\mathbf{y}_k - \mathbf{H}_k\mathbf{M}_{k\,,\,0}\,\mathbf{x}_0\right), \tag{3.27}$$

so that we have

$$J(\mathbf{x}_0) = J_{m\,,\,0}(\mathbf{x}_0) + J_{K\,,\,m}(\mathbf{x}_0). \tag{3.28}$$

We remark that the functional $J_{m,0}(\mathbf{x}_0)$ is the cost function attached to the first step of path (2). Let $\mathbf{x}_0^{\mathrm{a}}$ and $\mathbf{P}_0^{\mathrm{a}}$ be the result of this analysis. Because of the assumptions of linearity of the operators, $J_{m,0}(\mathbf{x}_0)$ is quadratic and coincides with its second-order Taylor expansion at any point, in particular at $\mathbf{x}_0^{\mathrm{a}}$. We have

$$J_{m\,,\,0}(\mathbf{x}_0) = J_{m\,,\,0}(\mathbf{x}_0^{\mathrm{a}}) + \left(\nabla_{\mathbf{x}_0} J_{m\,,\,0}(\mathbf{x}_0^{\mathrm{a}})\right)^\top (\mathbf{x}_0 - \mathbf{x}^{\mathrm{a}}) + \frac{1}{2}\,(\mathbf{x}_0 - \mathbf{x}^{\mathrm{a}})^\top \mathcal{H}_{m\,,\,0}\,(\mathbf{x}_0 - \mathbf{x}^{\mathrm{a}})\,. \tag{3.29}$$

By definition, $\nabla_{\mathbf{x}_0} J_{m\,,\,0}(\mathbf{x}_0^{\mathrm{a}}) = \mathbf{0}$. As a consequence, minimising $J_{m\,,\,0}$ is equivalent to minimising the following square

$$C(\mathbf{x}_0) = \frac{1}{2}\,(\mathbf{x}_0 - \mathbf{x}_0^{\mathrm{a}})^\top (\mathbf{P}_0^{\mathrm{a}})^{-1}\,(\mathbf{x}_0 - \mathbf{x}_0^{\mathrm{a}})\,, \tag{3.30}$$

with

$$\mathbf{x}_0^{\mathrm{a}} = \mathbf{x}_0^{\mathrm{b}} - \mathcal{H}_{m\,,\,0}^{-1} \nabla_{\mathbf{x}_0} J(\mathbf{x}_0^{\mathrm{b}}) \tag{3.31a}$$

$$(\mathbf{P}_0^{\mathrm{a}})^{-1} = \mathcal{H}_{m\,,\,0} = \mathbf{B}_0^{-1} + \sum_{k=0}^{m} \mathbf{M}_{k\,,\,0}^\top \mathbf{H}_k^\top \mathbf{R}_k^{-1} \mathbf{H}_k \mathbf{M}_{k\,,\,0}\,, \tag{3.31b}$$

since $J_{m\,,\,0}(\mathbf{x}_0) = J_{m\,,\,0}(\mathbf{x}_0^{\mathrm{a}}) + C(\mathbf{x}_0)$. Hence

$$J(\mathbf{x}_0) = J_{m\,,\,0}(\mathbf{x}_0^{\mathrm{a}}) + C(\mathbf{x}_0) + J_{K\,,\,m}(\mathbf{x}_0). \tag{3.32}$$

But $C(\mathbf{x}_0)$ is exactly the background term of the second step of path (2). The result of the global minimisation of $J(\mathbf{x}_0)$ over $[t_0, t_K]$ with respect to $\mathbf{x}_0$ coincides with the result of the minimisation of the two-step procedure with a background term in the second step to inform about the result of the first step. Note that we could have minimised over $\mathbf{x}_K$ as well, $\mathbf{x}_m$, or any other $\mathbf{x}_k$. This property is called ***transferability of optimality***. However, be aware that there is no identity between

- the outcome of the analysis through path (1) when focused on state vector $\mathbf{x}_m^{\mathrm{a}}$ defined at t_m, and
- the outcome of the *intermediary* analysis at time t_m obtained from the first step of path (2).

Indeed, the analysis of an intermediary step through path (1) uses all observations including the ones posterior to the intermediary step, while the analysis of the same intermediary step through path (2), after its first step, ignores future observations.

3.3.3 Equivalence between 4D-Var and the Kalman filter

Here, we justify the use of cost function (3.1) by showing that the final result x_K^{a} at t_K is the same as the sequential analysis of the Kalman filter at t_K. We focus on the data assimilation time window $[t_0, t_K]$. For the two methods to be on equal footing, we assume that they use the same background statistics $\mathbf{x}_0^{\mathrm{b}}$ and $\mathbf{P}_0^{\mathrm{b}}$ at the beginning of the window, *i.e.* at t_0.

We previously showed the transferability of optimality of the variational formalism by splitting the time interval. We can subdivide these two intervals and again invoke the transferability of optimality. To the extreme, we can split $[t_0, t_K]$ into the K subintervals $[t_k, t_{k+1}]$ with $0 \leq k \leq K-1$. But the minimisation of $J_{k\,,\,k+1}$ coincides with the 3D-Var approach, which is equivalent to optimal interpolation in the linear operator context. However, each Kalman filter analysis is based on optimal interpolation! As a result, the variational approach by subdividing interval $[t_0, t_K]$ into K segments $[t_k, t_{k+1}]$ with $0 \leq k \leq K-1$, is equivalent to the Kalman filter analysis. Be aware that this is not equivalent to the optimisation of J over the full $[t_0, t_K]$. Nevertheless, as a result of what precedes, we can claim that the analysis at time t_K by the global optimisation coincides with the optimisation when subdividing into K subintervals and hence to the Kalman filter.

3.4 Minimisation algorithms for the cost functions

Let us now discuss one practical but essential aspect of the variational methods: the minimisation of the cost functions. Suppose we want to minimise the cost function $J(\mathbf{x})$. If J is quadratic and strictly convex, it has a unique global minimum in $\mathbb{R}^{N_{\mathrm{x}}}$. However, more generally, J can exhibit several local minima. Note that if the problem is physically well posed, one expects that a global minimum exists. Determining all of these minima is a very difficult task. Here, we focus on finding the minimum, or one of the minima. Let us denote $\mathbf{g}(\mathbf{x})$ the gradient of $J(\mathbf{x})$, and $\mathbf{H}(\mathbf{x})$ is the related Hessian matrix. A way to test the optimisation algorithms is to apply them to the basic case of a quadratic cost function,

$$J_{\mathbf{Q}}(\mathbf{x}) = \frac{1}{2}\mathbf{x}^\top \mathbf{Q}\mathbf{x} + \mathbf{b}^\top \mathbf{x}, \tag{3.33}$$

where $\mathbf{Q}$ is a positive definite matrix. Hence, $J_{\mathbf{Q}}$ is strictly convex.

3.4.1 Descent algorithms

The minimisation algorithms start at a point $\mathbf{x}_0 \in \mathbb{R}^{N_{\mathrm{x}}}$ and build a sequence of points $\mathbf{x}_k$ which is meant to converge to a local minimum. $\mathbf{x}_0$ typically is in the basin of attraction of the local minimum. At step k of the algorithm, we determine a direction $\mathbf{d}_k \in \mathbb{R}^{N_{\mathrm{x}}}$ which is characteristic of the method. This direction of exploration is used to define the next point of the sequence

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \lambda_k \mathbf{d}_k, \tag{3.34}$$

where λ_k is a positive real number, $\lambda_k \in \mathbb{R}^+$. An optimal λ_k is obtained from a one-dimensional minimisation (***line search***), along the affine direction defined by $\mathbf{x}_k$ and $\mathbf{d}_k$. The optimal λ_k results from the minimisation of

$$\varphi_k(\lambda) \triangleq J\left(\mathbf{x}_k + \lambda \mathbf{d}_k\right). \tag{3.35}$$

Since it is a minimisation problem, we wish to have

$$\varphi_k'(0) \triangleq \partial_\lambda \varphi_k(0) = \mathbf{g}(\mathbf{x}_k)^\top \mathbf{d}_k \; < \; 0. \tag{3.36}$$

Hence, for $\mathbf{d}_k$ to be a descent direction, we must have $\mathbf{g}(\mathbf{x}_k)^\top \mathbf{d}_k < 0$.

Steepest descent, conjugate gradient and Newton-Raphson methods

We can choose for $\mathbf{d}_k$ the direction of steepest descent, yielding the ***steepest descent method***, $\mathbf{d}_k = -\mathbf{g}(\mathbf{x}_k)$, which is pointed at by minus the gradient of the cost function. We can hope for a better algorithm when the $\mathbf{d}_k$ better explores the minimisation space (we have no choice for the $\mathbf{d}_k$ in the steepest descent method).

Before considering the full nonlinear case, let us focus on the quadratic case Eq. (3.33). We can impose the $\{\mathbf{d}_j\}_{0\leq j\leq k}$ to be $\mathbf{Q}$-conjugate, i.e. they are mutually orthogonal with respect to the quadratic form defined by $\mathbf{Q}$, i.e., $\mathbf{d}_j^\top \mathbf{Q} \mathbf{d}_l = 0$ whenever $j \neq l$. This construction is analogous to the Gram-Schmidt orthogonalisation. It ensures that the solution can be obtained in less than $N_\mathrm{x}+1$ steps (recall N_x is the dimension of the system state) if the machine precision is infinite. Moreover, it is clear that in the quadratic case, we can obtain an explicit solution for λ_k (given below). This method is known as the ***conjugate gradient***.

This method also applies to the nonlinear case. In this case, the conjugation is only approximate and the solution usually requires more than $N_\mathrm{x}+1$ steps. The several implementations (Hestenes-Stiefel, Polak-Ribière and Fletcher-Reeves) of this generalisation are distinguished by the explicit formula for λ_k. The crucial point is that none of these methods make use of the Hessian of the cost function.

In spite of the elegance and sophistication of the conjugate gradient and variants, it may be more efficient to use a Newton-Raphson method which may offer a faster convergence rate. Let us explain its main idea. In a neighbourhood of $\mathbf{x}_k$, we wish to approximate the cost function J by a quadratic cost function.

$$J(\mathbf{x}) = \underbrace{J(\mathbf{x}_k) + \mathbf{g}(\mathbf{x}_k)^\top (\mathbf{x}-\mathbf{x}_k) + \frac{1}{2}(\mathbf{x}-\mathbf{x}_k)^\top \mathbf{H}(\mathbf{x}_k)(\mathbf{x}-\mathbf{x}_k)}_{q(\mathbf{x})} + o(\| \mathbf{x}-\mathbf{x}_k \|^2). \tag{3.37}$$

An estimation of the local minimum is given by the minimum of the local quadratic approximation $q(\mathbf{x})$, denoted $\mathbf{x}_{k+1}$, that we obtain by solving $\nabla q(\mathbf{x}) \simeq \mathbf{0}$, *i.e.*

$$\mathbf{0} \simeq \nabla q(\mathbf{x}) = g(\mathbf{x}_k) + \mathbf{H}(\mathbf{x}_k)(\mathbf{x}-\mathbf{x}_k). \tag{3.38}$$

If the Hessian is invertible, a better idea of the minimum point $\overline{\mathbf{x}}$ is given by

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \mathbf{H}(\mathbf{x}_k)^{-1}\mathbf{g}(\mathbf{x}_k). \tag{3.39}$$

That is to say

$$\mathbf{d}_k = -\mathbf{H}(\mathbf{x}_k)^{-1}\mathbf{g}(\mathbf{x}_k). \tag{3.40}$$

This corresponds to making a choice not only for $\mathbf{d}_k \in \mathbb{R}^{N_\mathrm{x}}$, but also for $\lambda_k \in \mathbb{R}^+$ which is taken to be 1. It can be shown that the method is converging provided the starting point

$\mathbf{x}_0$ is close enough to a local minimum. Applied to a quadratic cost function, this method is qualified as ***second-order*** because

$$\mathbf{x}_{k+1} - \overline{\mathbf{x}} = O(\| \mathbf{x}_k - \overline{\mathbf{x}} \|^2), \tag{3.41}$$

assuming $\overline{\mathbf{x}}$ is the solution to the minimisation problem.

This method is meant to be very efficient. It has nevertheless the major drawback that it requires to compute the Hessian and its inverse. For a nonlinear problem, the Hessian must (in principle) be recomputed at each iteration since it is point-dependent. For large dimensional problems, of particular interest to us, this could be a prohibitive computation.

3.4.2 Quasi-Newton algorithm

An alternative minimisation technique consists in using at step k a matrix $\mathbf{H}_k$ that could easily be computed, and behaves similarly to the inverse of the Hessian in the subspace generated by the gradient $\mathbf{g}(\mathbf{x}_k)$, that we shall also denote $\mathbf{g}_k$, so that we could write $\mathbf{d}_k = -\mathbf{H}_k \mathbf{g}_k$. Please bear in mind the abrupt change of notation: $\mathbf{H}_k$ now designates an approximation of the *inverse* of the Hessian (this is customary).

The idea is to build up $\mathbf{H}_k$ along with the iterations. The directions of $\mathbb{R}^{N_\mathrm{x}}$ that are explored when we compute $\mathbf{x}_{k+1}$ knowing $\mathbf{x}_k$ are $\Delta\mathbf{x}_k = \mathbf{x}_{k+1} - \mathbf{x}_k$, and $\Delta\mathbf{g}_k = \mathbf{g}_{k+1} - \mathbf{g}_k$, since we only have access to the $\mathbf{x}_k$ and to the gradients $\Delta\mathbf{g}_k$. But, a simple exact integration shows that $\Delta\mathbf{x}_k$ and $\Delta\mathbf{g}_k$ are related via

$$\left\{ \int_0^1 \mathrm{d}\lambda \, \nabla^2 J(\mathbf{x} + \lambda \mathbf{d}_k) \right\} \Delta\mathbf{x}_k = \Delta\mathbf{g}_k \, . \tag{3.42}$$

The left-hand side $\int_0^1 \mathrm{d}\lambda \, \nabla^2 J(\mathbf{x} + \lambda \mathbf{d}_k)$ is the mean of the Hessian along the trajectory $\Delta\mathbf{x}_k$. The conclusion of this argument is that we can extract from $\Delta\mathbf{g}_k$ and $\Delta\mathbf{x}_k$ some information about the Hessian. Since this information is of low rank, the inverse of the Hessian needs to be built progressively, iteration after iteration.

Building on this heuristic argument, we wish that the sequence $\mathbf{H}_k$ satisfies

$$\forall \, k \geq 0, \quad \mathbf{H}_k \, (\mathbf{g}_{k+1} - \mathbf{g}_k) = \mathbf{x}_{k+1} - \mathbf{x}_k. \tag{3.43}$$

This condition makes the algorithm a ***quasi-Newton*** method. We also wish $\mathbf{H}_k$ to be positive semi-definite. Indeed, we have

$$\varphi_k'(0) = \mathbf{g}_k^\top \mathbf{d}_k = -\mathbf{g}_k^\top \mathbf{H}_k \mathbf{g}_k, \tag{3.44}$$

so that the positivity of $\mathbf{H}_k$ is sufficient to ensure the descent criterion $\varphi_k'(0) \leq 0$.

In spite of their sophistication, the quasi-Newton methods are very popular, because they combine convergence speed while limiting the algorithmic complexity (no explicit computation of the Hessian). Besides, on the shelf quasi-Newton softwares are available and widely distributed. The generic quasi-Newton method is displayed in algorithm 3.4.2.

The sequence $\mathbf{H}_k$ must satisfy the quasi-Newton condition, but this condition does not suffice to fully define it. The complete specification of the sequence distinguishes between variants of the quasi-Newton method.

Generic quasi-Newton algorithm

1. We start at $\mathbf{x}_0$ at time $k = 0$
   and we choose an initial symmetric matrix for $\mathbf{H}_0$.
2. For $k = 0, 1, 2, \ldots$,
   - If $\mathbf{g}_k = 0$, the algorithm ends, otherwise
   $$\mathbf{d}_k = -\mathbf{H}_k \mathbf{g}_k$$
   - We compute (line search)
   $$\lambda_k = \operatorname{argmin} J\left(\mathbf{x}_k + \lambda \mathbf{d}_k\right) \quad \text{for} \quad \lambda \geq 0$$
   then
   $$\mathbf{x}_{k+1} = \mathbf{x}_k + \lambda_k \mathbf{d}_k$$
   - Build $\mathbf{H}_{k+1}$ using $\mathbf{H}_k$ and
   $$\Delta \mathbf{x}_k = \mathbf{x}_{k+1} - \mathbf{x}_k \quad \text{and} \quad \Delta \mathbf{g}_k = \mathbf{g}_{k+1} - \mathbf{g}_k$$

Figure 3.3: The generic quasi-Newton scheme.

Application to a quadratic function

Let us have a look at the way a quasi-Newton method works in the linear case, *i.e.* when the cost function is quadratic $J_{\mathbf{Q}}(\mathbf{x}) = \frac{1}{2}\mathbf{x}^\top \mathbf{Q}\mathbf{x} + \mathbf{b}^\top \mathbf{x}$. The generic algorithm of a quasi-Newton minimisation in the linear case, is given in algorithm 3.4.

The value of λ_k is the outcome of the minimisation of

$$J\left(\mathbf{x}_k + \lambda \mathbf{d}_k\right) = \frac{1}{2}\left(\mathbf{x}_k + \lambda \mathbf{d}_k\right)^\top \mathbf{Q}\left(\mathbf{x}_k + \lambda \mathbf{d}_k\right) + \mathbf{b}^\top \left(\mathbf{x}_k + \lambda \mathbf{d}_k\right), \tag{3.45}$$

of obvious exact solution:

$$\lambda_k = -\frac{\mathbf{g}_k^\top \mathbf{d}_k}{\mathbf{d}_k^\top \mathbf{Q} \mathbf{d}_k}, \tag{3.46}$$

with $\mathbf{g}_k = \mathbf{Q}\mathbf{x}_k + \mathbf{b}$.

Q-Conjugation

A remarkable property of the generic quasi-Newton algorithm *when applied to the quadratic functional* is that the directions $\mathbf{d}_k$ are mutually $\mathbf{Q}$-conjugate.

3.4.3 A quasi-Newton method: the BFGS algorithm

In practice, one of the most efficient quasi-Newton implementation has been proposed in 1970 by Broyden, Fletcher, Goldfarb and Shanno, hence the name of the method: ***BFGS***.

Generic quasi-Newton algorithm (linear case)

1. We start from $\mathbf{x}_0$ at time $k = 0$,
   and a symmetric matrix $\mathbf{H}_0$.
2. For $k = 0, 1, 2, \ldots$,
   - If $\mathbf{g}_k = 0$, the algorithm ends, otherwise

$$\mathbf{d}_k = -\mathbf{H}_k \mathbf{g}_k$$

   - We compute

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \lambda_k \mathbf{d}_k \quad \text{where} \quad \lambda_k = -\frac{\mathbf{g}_k^\top \mathbf{d}_k}{\mathbf{d}_k^\top \mathbf{Q} \mathbf{d}_k}$$

   - Build $\mathbf{H}_{k+1}$ from $\mathbf{H}_k$ and

$$\Delta\mathbf{x}_k = \mathbf{x}_{k+1} - \mathbf{x}_k \quad \text{and} \quad \Delta\mathbf{g}_k = \mathbf{g}_{k+1} - \mathbf{g}_k$$

Figure 3.4: The generic quasi-Newton scheme in the linear case.

The sequence of the matrices $\mathbf{H}_k$ is defined by the following matrix recursive formula

$$\begin{aligned}\mathbf{H}_{k+1} =& \mathbf{H}_k + \left(1 + \frac{(\Delta\mathbf{g}_k, \mathbf{H}_k \Delta\mathbf{g}_k)}{(\Delta\mathbf{g}_k, \Delta\mathbf{x}_k)}\right) \frac{\Delta\mathbf{g}_k (\Delta\mathbf{g}_k)^\top}{(\Delta\mathbf{x}_k, \Delta\mathbf{g}_k)} \\ & - \frac{\mathbf{H}_k \Delta\mathbf{g}_k (\Delta\mathbf{x}_k)^\top + \left(\mathbf{H}_k \Delta\mathbf{g}_k (\Delta\mathbf{x}_k)^\top\right)^\top}{(\Delta\mathbf{g}_k, \Delta\mathbf{x}_k)}, \end{aligned} \tag{3.47a}$$

$$\mathbf{H}_{k+1} = \mathbf{H}_k + \left(1 - \frac{\Delta\mathbf{x}_k \Delta\mathbf{g}_k^\top}{(\Delta\mathbf{g}_k, \Delta\mathbf{x}_k)}\right) \mathbf{H}_k \left(1 - \frac{\Delta\mathbf{g}_k (\Delta\mathbf{x}_k)^\top}{(\Delta\mathbf{g}_k, \Delta\mathbf{x}_k)}\right) + \frac{\Delta\mathbf{x}_k (\Delta\mathbf{x}_k)^\top}{(\Delta\mathbf{g}_k, \Delta\mathbf{x}_k)}. \tag{3.47b}$$

The method is rank-2 because

$$\operatorname{rank}\left(\mathbf{H}_{k+1} - \mathbf{H}_k\right) \leq 2. \tag{3.48}$$

It is easy to check that the method is quasi-Newton, *i.e.* it satisfies Eq. (3.43). Moreover, we check that if $\mathbf{H}_k$ is positive semi-definite, then $\mathbf{H}_{k+1}$ is also positive semi-definite.

Provided that the starting point $\mathbf{x}_0$ is not too far away from the argument $\overline{\mathbf{x}}$ of the minimum of the cost function, the convergence of the BFGS algorithm is superlinear in the nonlinear case, *i.e.*

$$\lim_{k \to \infty} \frac{\| \mathbf{x}_k - \overline{\mathbf{x}} \|}{\| \mathbf{x}_{k+1} - \overline{\mathbf{x}} \|} = 0. \tag{3.49}$$

With stronger requirements (regularity), classes of implementation of the quasi-Newton methods quadratically converge, even in the nonlinear case.

Much more information about numerical optimisation, iterative minimisation schemes including conjugate-gradient and quasi-Newton schemes can be found in Nocedal and Wright (2006).

3.5 Weak-constraint 4D-Var

The variational analogue of the Kalman filter that incorporates model error is given by the ***weak-constraint 4D-Var*** of cost function

$$J(\mathbf{x}) = \frac{1}{2}\left(\mathbf{x}_0 - \mathbf{x}_0^{\mathrm{b}}\right)^{\top} \mathbf{B}_0^{-1}\left(\mathbf{x}_0 - \mathbf{x}_0^{\mathrm{b}}\right) + \frac{1}{2}\sum_{k=0}^{K}\left(\mathbf{y}_k - \mathbf{H}_k\mathbf{x}_k\right)^{\top}\mathbf{R}_k^{-1}\left(\mathbf{y}_k - \mathbf{H}_k\mathbf{x}_k\right) + \frac{1}{2}\sum_{k=1}^{K}\left(\mathbf{x}_k - \mathbf{M}_k\mathbf{x}_{k-1}\right)^{\top}\mathbf{Q}_k^{-1}\left(\mathbf{x}_k - \mathbf{M}_k\mathbf{x}_{k-1}\right). \tag{3.50}$$

Obviously, this method is numerically very costly as one needs to optimise this cost function over all $\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_K$. A proper modelling of the not so-well known $\mathbf{Q}_k$ is also needed. Besides, nothing guarantees that it can properly account for any kind of model error. That is why, in spite of being contemplated very early in the data assimilation field, it is still barely used. However, it has more recently drawn attention because it is amenable to a parallelised implementation as opposed to the strong constraint 4D-Var.

The formalism can also be simplified. For instance, one can consider a forcing weak-constraint 4D-Var where model error is given by a systematic bias which does not vary over the time window. The associated cost function reads

$$J(\mathbf{x}) = \frac{1}{2}\left(\mathbf{x}_0 - \mathbf{x}_0^{\mathrm{b}}\right)^{\top} \mathbf{B}_0^{-1}\left(\mathbf{x}_0 - \mathbf{x}_0^{\mathrm{b}}\right) + \frac{1}{2}\sum_{k=0}^{K}\left(\mathbf{y}_k - \mathbf{H}_k\mathbf{x}_k\right)^{\top}\mathbf{R}_k^{-1}\left(\mathbf{y}_k - \mathbf{H}_k\mathbf{x}_k\right) + \frac{1}{2}\left(\eta - \eta^{\mathrm{b}}\right)^{\top}\mathbf{Q}^{-1}\left(\eta - \eta^{\mathrm{b}}\right), \tag{3.51a}$$

under the constraints:

$$\text{for } k = 0, \ldots, K-1: \quad \mathbf{x}_{k+1} = M_{k+1}(\mathbf{x}_k) + \eta. \tag{3.51b}$$

Hence, compared to the strong-constraint 4D-Var, only the model error vector $\eta \in \mathbb{R}^{N_{\mathrm{x}}}$ has been added to the control variables. $J(\mathbf{x})$ is actually a function of the $\mathbf{x}_0$ and η. This approximation of the weak-constraint 4D-var is in operation at the ECMWF for the stratosphere where the meteorological forecast model (IFS) is known to have systematic biases.

Part II

Advanced methods of data assimilation

Chapter 4

Nonlinear data assimilation

We will illustrate the performance of the algorithms introduced in this chapter with an anharmonic oscillator and some additional model error. That is why we use again the nonlinear oscillator of chapter 2 to which we add some noise corresponding to some model error. This model error, an additive noise, weakens the predictability of the system:

$$x_0 = 0\,, \quad x_1 = 1 \quad \text{and for } 1 \leq k \leq K \quad x_{k+1} - 2x_k + x_{k-1} = \omega^2\, x_k + \lambda^2\, x_k^3 + \xi_k. \quad (4.1)$$

We have chosen the following parameters: $\omega = 0.035$, $\lambda = 3.10^{-5}$. The variance of model error is 0.0025 while the variance of the observation error is 49. The initial condition is $x_0 = 0$ and $x_1 = 1$. The system is iterated 10000 times.

4.1 The limitations of the extended Kalman filter

Let us look at the typical behaviour of an extended Kalman filter applied to a strongly nonlinear system. Let us experiment with the stochastic anharmonic oscillator described above. At first, we choose the observation to be performed every 50 timesteps. This yields the typical run shown in figure 4.1. where the extended Kalman filter succeeds in tracking the system's state.

Secondly, we choose to assimilate an observation every 58 iterations, at a somewhat lower frequency. The behaviour of the extended Kalman filter is illustrated in figure 4.2. We see that the filter ultimately diverges because it does not have enough information on the system state's trajectory and because the error propagation is approximated by the tangent linear model in between two analysis steps.

To reduce the impact of the tangent linear approximation, we could resort to higher order extended Kalman filters, which better handle the nonlinearity of the model. However, these require the use of not only the tangent linear but also an Hessian, which would be very costly to implement (in terms of CPU and memory). This is a stringent limitation of these methods for high dimensional systems.

To understand and improve the performance of the filters with strongly nonlinear dynamical systems, it would be interesting to define a data assimilation method able to account for

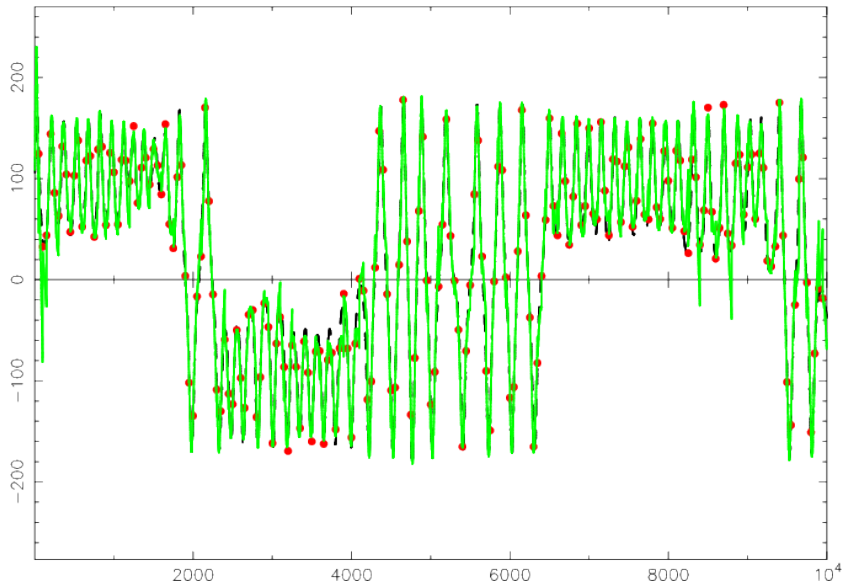


Figure 4.1: Data assimilation with a stochastic anharmonic oscillator, using an extended Kalman filter. The observation frequency is one observation every 50 timesteps.

all statistical moments of the state distribution. Indeed a nonlinear dynamics propagates the statistical moments of the system state in a non-trivial fashion as opposed to linear dynamics.

4.2 The *ultimate* data assimilation method?

So far, we have tried to solve the following problem: given an a priori but partial and imperfect knowledge of the system and a fresh observation set, what is, in the mean square sense, the optimal system state estimate and what is the variance of the analysis error?

Yet, the question could be more general. We could reformulate the problem in terms of probabilities. We do not limit ourselves to the expectation and covariances of the variable distributions but we work on all statistical moments (Tarantola and Valette, 1982).

Provided we know the probability density of the background as well as that of the observations, what is the probability density of a system state? A opposed to the approaches of the previous chapters, there is not any such a priori criterion such as the least squares from which we could define an estimator for the system state and its error. Instead, we will need to choose a criterion that defines optimality, which yields a specific estimator.

4.2.1 Assimilation of an observation

Let us consider a very general observation equation

$$\mathbf{y} = H(\mathbf{x}, \mathbf{v}), \tag{4.2}$$

where the observation operator H could be nonlinear. We assume that we know a priori the probability density function (pdf) of the distribution of $\mathbf{x}$, *i.e.* the pdf of the background,

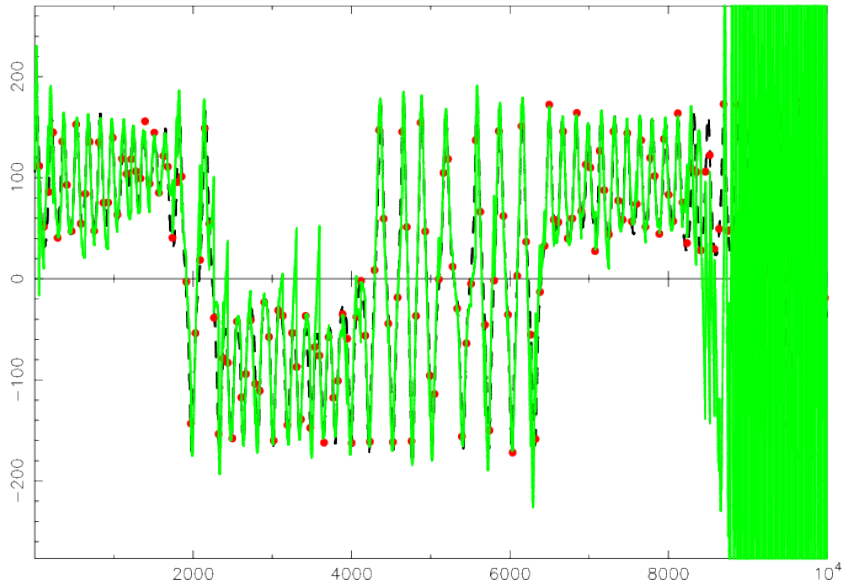


Figure 4.2: Data assimilation with a stochastic anharmonic oscillator, using an extended Kalman filter. The observation frequency is one observation every 58 timesteps which is insufficient and the filter ultimately diverges.

which is denoted $p_X(\mathbf{x})$. Besides, the pdf of the noise, denoted $p_V(\mathbf{v})$ is also assumed to be known. We wish to know the probability density function of $\mathbf{x}$, knowing the value of the observation vector $\mathbf{y}$, *i.e.* the conditional pdf of $\mathbf{x}$, knowing $\mathbf{y}$, denoted $p_{X|Y}(\mathbf{x}|\mathbf{y})$. To that goal, let us consider the joint pdf of the state and the observation vector, $p_{X,Y}(\mathbf{x},\mathbf{y})$. We can easily decompose it in two ways:

$$p_{X,Y}(\mathbf{x},\mathbf{y}) = p_{X|Y}(\mathbf{x}|\mathbf{y})p_Y(\mathbf{y}) = p_{Y|X}(\mathbf{y}|\mathbf{x})p_X(\mathbf{x}). \tag{4.3}$$

This yields the infamous Bayes' rule (that was later rediscovered by Pierre-Simon Laplace in a more general form than inially proposed by Thomas Bayes), we obtain:

$$p_{X|Y}(\mathbf{x}|\mathbf{y}) = p_{Y|X}(\mathbf{y}|\mathbf{x})\frac{p_X(\mathbf{x})}{p_Y(\mathbf{y})}. \tag{4.4}$$

which determines p_Y. Computing this latter term corresponds to a normalisation of the pdf with respect to $\mathbf{x}$:

$$p_Y(\mathbf{y}) = \int \mathrm{d}\mathbf{x}\, p_{Y|X}(\mathbf{y}|\mathbf{x})p_X(\mathbf{x}). \tag{4.5}$$

In practice it is often unnecessary to compute this normalisation since it does not depend on $\mathbf{x}$.

4.2.2 Estimation theory and BLUE analysis

To convince ourselves of the richness of this probabilistic standpoint, let us check that it is consistent with the optimal interpolation (BLUE). To this end, the observation equation is simplified into

$$\mathbf{y} = \mathbf{H}\,\mathbf{x} + \mathbf{v}, \tag{4.6}$$

where the observation operator $\mathbf{H}$ is assumed linear here. We also assume that $\mathbf{x}$ is distributed according to a Gaussian distribution $N(\overline{\mathbf{x}}, \mathbf{P})$, that is to say according to the (normalised) pdf

$$p_X(\mathbf{x}) = \frac{1}{(2\pi)^{N_\mathrm{x}/2}|\mathbf{P}|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x}-\overline{\mathbf{x}})^\top \mathbf{P}^{-1}(\mathbf{x}-\overline{\mathbf{x}})\right), \tag{4.7}$$

with $|\mathbf{P}|$ denoting the determinant of the square matrix $\mathbf{P}$. This constitutes the background pdf of $\mathbf{x}$. Since $\mathbf{v}$ is distributed according to $N(\mathbf{0}, \mathbf{R})$, independent from that of $\mathbf{x}$, the conditional probability of $\mathbf{y}$, knowing $\mathbf{x}$, is

$$p_{Y|X}(\mathbf{y}|\mathbf{x}) = \frac{1}{(2\pi)^{N_\mathrm{y}/2}|\mathbf{R}|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{y}-\mathbf{H}\mathbf{x})^\top \mathbf{R}^{-1}(\mathbf{y}-\mathbf{H}\mathbf{x})\right). \tag{4.8}$$

The calculation of p_Y (normalisation of the product of the two previous densities) consists in convolving two Gaussian pdfs. As a consequence, the resulting density is also Gaussian in $\mathbf{y}$. It is easy to compute its expectation $\overline{\mathbf{x}}_\mathbf{y} = \mathrm{E}[\mathbf{y}]$ and its variance $\mathbf{P}_\mathbf{y}$. Indeed,

$$\overline{\mathbf{x}}_\mathbf{y} = \mathrm{E}[\mathbf{H}\mathbf{x}+\mathbf{v}] = \mathbf{H}\overline{\mathbf{x}}, \tag{4.9a}$$

$$\begin{aligned}\mathbf{P}_\mathbf{y} &= \mathrm{E}\left[(\mathbf{y}-\overline{\mathbf{x}}_\mathbf{y})(\mathbf{y}-\overline{\mathbf{x}}_\mathbf{y})^\top\right] = \mathrm{E}\left[(\mathbf{H}(\mathbf{x}-\overline{\mathbf{x}})+\mathbf{v})(\mathbf{H}(\mathbf{x}-\overline{\mathbf{x}})+\mathbf{v})^\top\right) \\ &= \mathbf{H}\mathrm{E}\left[(\mathbf{x}-\overline{\mathbf{x}})(\mathbf{x}-\overline{\mathbf{x}})^\top\right]\mathbf{H}^\top + \mathrm{E}\left[\mathbf{v}\mathbf{v}^\top\right] = \mathbf{H}\mathbf{P}\mathbf{H}^\top + \mathbf{R}.\end{aligned} \tag{4.9b}$$

This entirely characterises the distribution of $\mathbf{y}$

$$p_Y(\mathbf{y}) = \frac{1}{(2\pi)^{N_\mathrm{y}/2}|\mathbf{H}\mathbf{P}\mathbf{H}^\top + \mathbf{R}|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{y}-\mathbf{H}\overline{\mathbf{x}})^\top(\mathbf{H}\mathbf{P}\mathbf{H}^\top+\mathbf{R})^{-1}(\mathbf{y}-\mathbf{H}\overline{\mathbf{x}})\right). \tag{4.10}$$

Using (4.10), (4.8) and Bayes' rule (4.4), we obtain the targeted conditional density:

$$\begin{aligned}p_{X|Y}(\mathbf{x}|\mathbf{y}) =& \frac{|\mathbf{H}\mathbf{P}\mathbf{H}^\top+\mathbf{R}|^{1/2}}{(2\pi)^{N_\mathrm{x}/2}|\mathbf{P}|^{1/2}|\mathbf{R}|^{1/2}} \times \\ & \exp\left(-\frac{1}{2}(\mathbf{y}-\mathbf{H}\mathbf{x})^\top\mathbf{R}^{-1}(\mathbf{y}-\mathbf{H}\mathbf{x}) - \frac{1}{2}(\mathbf{x}-\overline{\mathbf{x}})^\top\mathbf{P}^{-1}(\mathbf{x}-\overline{\mathbf{x}})\right. \\ & \left.+\frac{1}{2}(\mathbf{y}-\mathbf{H}\overline{\mathbf{x}})^\top(\mathbf{H}\mathbf{P}\mathbf{H}^\top+\mathbf{R})^{-1}(\mathbf{y}-\mathbf{H}\overline{\mathbf{x}})\right).\end{aligned} \tag{4.11}$$

Factorising the argument of the exponential according to $\mathbf{x}$, we obtain

$$p_{X|Y}(\mathbf{x}|\mathbf{y}) = \frac{|\mathbf{H}\mathbf{P}\mathbf{H}^\top+\mathbf{R}|^{1/2}}{(2\pi)^{N_\mathrm{x}/2}|\mathbf{P}|^{1/2}|\mathbf{R}|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x}-\mathbf{x}^\star)^\top\mathbf{P}_\star^{-1}(\mathbf{x}-\mathbf{x}^\star)\right), \tag{4.12}$$

with

$$\mathbf{P}_\star^{-1} = \mathbf{P}^{-1} + \mathbf{H}^\top\mathbf{R}^{-1}\mathbf{H} \quad \text{and} \quad \mathbf{x}^\star = \mathbf{P}_\star(\mathbf{H}^\top\mathbf{R}^{-1}\mathbf{y} + \mathbf{P}^{-1}\overline{\mathbf{x}}). \tag{4.13}$$

4.2.3 Choosing an estimator

Then one needs to choose an ***estimator***. That is, one needs to make rigorous the subjective concept of *most likely* value of an analysis of the system state $\mathbf{x}$, knowing the observations $\mathbf{y}$. This estimator is to be defined from the conditional pdf $p_{X|Y}$. We would like the estimator to be unbiased. In the present case, since the distributions are Gaussian, all reasonable choices lead to the same estimator. However, with more general distributions, we would have to distinguish between the ***minimum variance*** estimator and the ***maximum a posteriori*** which corresponds to the maximum of the pdf $p_{X|Y}$. But other choices are possible.

In the present case, the two coincide with the estimator $\hat{\mathbf{x}} = E[\mathbf{x}|\mathbf{y}]$, which is easily read from the pdf. Now, using the optimal gain $\mathbf{K}^*$, we obtain

$$\begin{aligned}
\hat{\mathbf{x}} &= (\mathbf{P}^{-1} + \mathbf{H}^\top \mathbf{R}^{-1}\mathbf{H})^{-1}(\mathbf{H}^\top \mathbf{R}^{-1}\mathbf{y} + \mathbf{P}^{-1}\overline{\mathbf{x}}) && \text{(4.14a)}\\
&= \mathbf{K}^*\mathbf{y} + (\mathbf{P}^{-1} + \mathbf{H}^\top \mathbf{R}^{-1}\mathbf{H})^{-1}\mathbf{P}^{-1}\overline{\mathbf{x}} && \text{(4.14b)}\\
&= \mathbf{K}^*\mathbf{y} + (\mathbf{I} - \mathbf{K}^*\mathbf{H})\overline{\mathbf{x}} && \text{(4.14c)}\\
&= \overline{\mathbf{x}} + \mathbf{K}^*(\mathbf{y} - \mathbf{H}\overline{\mathbf{x}}), && \text{(4.14d)}
\end{aligned}$$

where we used relationships (1.26) and (1.24). This result should be compared with (1.18a). The result of this probabilistic calculation hence coincides with the result of BLUE, assuming $\mathbf{x}_b = \overline{\mathbf{x}}$.

4.3 Sequential assimilation and probabilistic interpretation

Let us consider a sequential data assimilation scheme. We wish to study the scheme using probability theory following the ideas of the previous section. This is the ***Bayesian filtering*** problem. Let us denote $\mathbf{Z}_k$, the set of all past observation vectors from t_0 to t_k, *i.e.*

$$\mathbf{Z}_k = \{\mathbf{z}_k, \mathbf{z}_{k-1}, \cdots, \mathbf{z}_0\}. \tag{4.15}$$

We wish to compute the conditional pdf of $\mathbf{x}_k$ knowing $\mathbf{Z}_k$, denoted $p(\mathbf{x}_k|\mathbf{Z}_k)$. For the sake of simplicity, we give up the pdf index. Therefore, they will all be denoted p, and only their argument can help discriminate them.

4.3.1 Forecast step

In the forecast step, one wishes to compute the pdf $p(\mathbf{x}_{k+1}|\mathbf{Z}_k)$, knowing the pdf $p(\mathbf{x}_k|\mathbf{Z}_k)$. Without any particular restriction, we have

$$p(\mathbf{x}_{k+1}|\mathbf{Z}_k) = \int \mathrm{d}\mathbf{x}_k \, p(\mathbf{x}_{k+1}|\mathbf{x}_k)\, p(\mathbf{x}_k|\mathbf{Z}_k). \tag{4.16}$$

A very general stochastic modelling of the dynamical system would be

$$\mathbf{x}_{k+1} = F(\mathbf{x}_k, \mathbf{w}_k), \tag{4.17}$$

where $\mathbf{w}_k$ is meant to be a white noise (uncorrelated in time) of density p_W. Thus we have

$$p(\mathbf{x}_{k+1}|\mathbf{x}_k) = \int \mathrm{d}\mathbf{w}_k \, p_W(\mathbf{w}_k)\, \delta\left(\mathbf{x}_{k+1} - F(\mathbf{x}_k, \mathbf{w}_k)\right), \tag{4.18}$$

where δ is the Dirac distribution.

Hence, the densities change between t_k and t_{k-1} according to

$$p(\mathbf{x}_{k+1}|\mathbf{Z}_k) = \int \mathrm{d}\mathbf{x}_k \, \mathrm{d}\mathbf{w}_k \, p_W(\mathbf{w}_k) \, p(\mathbf{x}_k|\mathbf{Z}_k) \, \delta\left(\mathbf{x}_{k+1} - F(\mathbf{x}_k, \mathbf{w}_k)\right). \tag{4.19}$$

When the noise is additive, *i.e.* if (4.17) becomes

$$\mathbf{x}_{k+1} = M(\mathbf{x}_k) + \mathbf{w}_k, \tag{4.20}$$

the convolution of densities simplifies into

$$p(\mathbf{x}_{k+1}|\mathbf{Z}_k) = \int \mathrm{d}\mathbf{x}_k \, p_W(\mathbf{x}_{k+1} - M(\mathbf{x}_k)) \, p(\mathbf{x}_k|\mathbf{Z}_k). \tag{4.21}$$

When model error is zero, we can further simplifies since

$$p_W(\mathbf{x}_{k+1} - M(\mathbf{x}_k)) = \delta(\mathbf{x}_{k+1} - M(\mathbf{x}_k)). \tag{4.22}$$

In that case:

$$p(\mathbf{x}_{k+1}|\mathbf{Z}_k) = \frac{p\left((M_k)^{-1}(\mathbf{x}_{k+1})|\mathbf{Z}_k\right)}{|(M_k') \circ M_k^{-1}(\mathbf{x}_{k+1})|}, \tag{4.23}$$

where $|\mathcal{O}|$ is the determinant of $\mathcal{O}$.

4.3.2 Analysis step

Once a new observation vector is acquired, an analysis is performed. A general observation equation is

$$\mathbf{z}_k = H(\mathbf{x}_k, \mathbf{v}_k), \tag{4.24}$$

where $\mathbf{v}_k$ is a noise of arbitrary distribution. Using Bayes' rule, we obtain:

$$p(\mathbf{x}_k|\mathbf{Z}_k) = p(\mathbf{x}_k|\mathbf{z}_k, \mathbf{Z}_{k-1}) = p(\mathbf{z}_k|\mathbf{x}_k, \mathbf{Z}_{k-1}) \, \frac{p(\mathbf{x}_k|\mathbf{Z}_{k-1})}{p(\mathbf{z}_k|\mathbf{Z}_{k-1})}. \tag{4.25}$$

It is slightly more elaborated than formula (4.4). $p(\mathbf{x}_k|\mathbf{Z}_{k-1})$ is supposed to be known since it is the outcome pdf of the forecast from t_{k-1} to t_k. We remark that we have

$$p(\mathbf{z}_k|\mathbf{Z}_{k-1}) = \int \mathrm{d}\mathbf{x}_k \, p(\mathbf{z}_k|\mathbf{x}_k, \mathbf{Z}_{k-1}) \, p(\mathbf{x}_k|\mathbf{Z}_{k-1}), \tag{4.26}$$

which looks like the normalisation of the targeted pdf. In summary

$$p(\mathbf{x}_k|\mathbf{Z}_k) = \frac{p(\mathbf{z}_k|\mathbf{x}_k, \mathbf{Z}_{k-1}) \, p(\mathbf{x}_k|\mathbf{Z}_{k-1})}{\displaystyle\int \mathrm{d}\mathbf{x}_k \, p(\mathbf{z}_k|\mathbf{x}_k, \mathbf{Z}_{k-1}) \, p(\mathbf{x}_k|\mathbf{Z}_{k-1})}. \tag{4.27}$$

If we assume the noise $\mathbf{v}_k$ to be white in time, $p(\mathbf{z}_k|\mathbf{x}_k, \mathbf{Z}_{k-1}) = p(\mathbf{z}_k|\mathbf{x}_k)$, since the knowledge of $p(\mathbf{z}_k|\mathbf{x}_k)$ is equivalent to that of the white noise at time t_k, which is independent from $\mathbf{Z}_{k-1}$. In that case, the previous formula simplifies into

$$p(\mathbf{x}_k|\mathbf{Z}_k) = \frac{p(\mathbf{z}_k|\mathbf{x}_k) \, p(\mathbf{x}_k|\mathbf{Z}_{k-1})}{\displaystyle\int \mathrm{d}\mathbf{x}_k \, p(\mathbf{z}_k|\mathbf{x}_k) \, p(\mathbf{x}_k|\mathbf{Z}_{k-1})}. \tag{4.28}$$

Provided we can afford the computations, these forecast and analysis steps are sufficient to cycle the system's state pdf and perform sequential data assimilation.

4.3.3 Estimation theory and Kalman filter

Let us again assume the conditions of validity of the Kalman filter, *i.e.* the linearity of operators. We would like to show that, essentially under these conditions, the estimation theory yields the Kalman filter results. Hence, this is a particular implementation of the previous forecast and analysis steps.

Forecast step The model equation (transition from t_k to t_{k+1}) is

$$\mathbf{x}_{k+1} = \mathbf{M}_k\,\mathbf{x}_k + \mathbf{w}_k, \tag{4.29}$$

where $\mathbf{w}_k$ is a Gaussian white noise of covariance matrix $\mathbf{Q}_k$. By construction, we have in one hand

$$p_W(\mathbf{x}_{k+1} - \mathbf{M}_k\mathbf{x}_k) = \frac{1}{(2\pi)^{N_\mathrm{x}/2}|\mathbf{Q}_k|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x}_{k+1} - \mathbf{M}_k\mathbf{x}_k)^\top(\mathbf{Q}_k)^{-1}(\mathbf{x}_{k+1} - \mathbf{M}_k\mathbf{x}_k)\right), \tag{4.30}$$

and

$$p(\mathbf{x}_k|\mathbf{Z}_k) = \frac{1}{(2\pi)^{N_\mathrm{x}/2}|\mathbf{P}_k^a|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x}_k - \mathbf{x}_k^a)^\top\,(\mathbf{P}_k^a)^{-1}\,(\mathbf{x}_k - \mathbf{x}_k^a)\right), \tag{4.31}$$

in the other hand, with $\mathbf{x}_k^a$ and $\mathbf{P}_k^a$ outcome of the analysis at time t_k. Integral (4.21) remains to be calculated. Again, we have to convolve two Gaussian distributions. The calculation is formally equivalent to (4.7), (4.8) and (4.10). We obtain

$$p(\mathbf{x}_{k+1}|\mathbf{Z}_k) = \frac{1}{(2\pi)^{N_\mathrm{x}/2}|\mathbf{P}_{k+1}^f|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x}_{k+1} - \mathbf{x}_{k+1}^f)^\top\,\left(\mathbf{P}_{k+1}^f\right)^{-1}\,(\mathbf{x}_{k+1} - \mathbf{x}_{k+1}^f)\right), \tag{4.32a}$$

with

$$\mathbf{x}_{k+1}^f = \mathbf{M}_k\mathbf{x}_k^a \qquad \text{and} \qquad \mathbf{P}_{k+1}^f = \mathbf{M}_k^\top\mathbf{P}_k^a\mathbf{M}_k + \mathbf{Q}_k. \tag{4.32b}$$

Analysis step: There is nothing to prove for the analysis step, since this part coincides with the derivation of the BLUE analysis by estimation theory, which was described in section 4.2.2.

Hence, we recover that estimation theory yields the Kalman filter in the linear case and with Gaussian a priori statistics. A possible estimator is that of the maximum a posterior (MAP).

4.4 Variational data assimilation and probabilistic interpretation

Thinking in terms of probability, we wish that the nonlinear data assimilation system

$$\mathbf{x}_{k+1} = M_{k+1}(\mathbf{x}_k) + \mathbf{w}_k \tag{4.33a}$$

$$\mathbf{z}_k = H_k(\mathbf{x}_k) + \mathbf{v}_k, \tag{4.33b}$$

has a 4D-Var formulation over the interval $[t_0, t_K]$. The models M_k and H_k could be nonlinear. Model error noise $\mathbf{w}_k$ as well as observation error $\mathbf{v}_k$ are supposed to be Gaussian white noise, unbiased and with error covariance matrices $\mathbf{Q}_k$ and $\mathbf{R}_k$ respectively. Their probability density functions are

$$p_W(\mathbf{w}_k) = N(\mathbf{0}, \mathbf{Q}_k) \qquad \text{and} \qquad p_V(\mathbf{v}_k) = N(\mathbf{0}, \mathbf{R}_k). \tag{4.34}$$

Differently from the targeted pdf of sequential assimilation, we rather wish to compute $p(\mathbf{X}_k|\mathbf{Z}_k)$, a ***Bayesian smoothing*** problem, with

$$\mathbf{X}_k = \{\mathbf{x}_k, \mathbf{x}_{k-1}, \cdots, \mathbf{x}_0\}. \tag{4.35}$$

This requires for us to adapt the probabilistic calculation of the sequential approach. First of all, the Bayes' rule gives

$$p(\mathbf{X}_k|\mathbf{Z}_k) = p(\mathbf{Z}_k|\mathbf{X}_k)\,\frac{p(\mathbf{X}_k)}{p(\mathbf{Z}_k)}. \tag{4.36}$$

Since we are only interested in the dependence in $\mathbf{X}_k$ of $p(\mathbf{X}_k|\mathbf{Z}_k)$, we will target the following pdf $p(\mathbf{X}_k|\mathbf{Z}_k) \propto p(\mathbf{Z}_k|\mathbf{X}_k)\,p(\mathbf{X}_k)$. It is clear that

$$p(\mathbf{X}_k) = p(\mathbf{x}_k, \mathbf{X}_{k-1}) = p(\mathbf{x}_k|\mathbf{X}_{k-1})\,p(\mathbf{X}_{k-1}) = p(\mathbf{x}_k|\mathbf{x}_{k-1})p(\mathbf{X}_{k-1}), \tag{4.37}$$

where we used the fact that the sequence of random variables $\mathbf{X}_k$ has the Markov property. Moreover, we can simplify $p(\mathbf{Z}_k|\mathbf{X}_k)$:

$$\begin{aligned} p(\mathbf{Z}_k|\mathbf{X}_k) &= p(\mathbf{z}_k, \mathbf{Z}_{k-1}|\mathbf{X}_k) = p(\mathbf{z}_k|\mathbf{X}_k)\,p(\mathbf{Z}_{k-1}|\mathbf{X}_k) && \text{(4.38a)}\\ &= p(\mathbf{z}_k|\mathbf{x}_k, \mathbf{X}_{k-1})\,p(\mathbf{Z}_{k-1}|\mathbf{x}_k, \mathbf{X}_{k-1}) && \text{(4.38b)}\\ &= p(\mathbf{z}_k|\mathbf{x}_k)\,p(\mathbf{Z}_{k-1}|\mathbf{X}_{k-1}). && \text{(4.38c)} \end{aligned}$$

We infer that

$$\begin{aligned} p(\mathbf{X}_k|\mathbf{Z}_k) &\propto p(\mathbf{Z}_k|\mathbf{X}_k)\,p(\mathbf{X}_k) = p(\mathbf{z}_k|\mathbf{x}_k)\,p(\mathbf{Z}_{k-1}|\mathbf{X}_{k-1})\,p(\mathbf{x}_k|\mathbf{x}_{k-1})\,p(\mathbf{X}_{k-1}) && \text{(4.39a)}\\ &\propto p(\mathbf{z}_k|\mathbf{x}_k)\,p(\mathbf{x}_k|\mathbf{x}_{k-1})\,p(\mathbf{Z}_{k-1}|\mathbf{X}_{k-1})\,p(\mathbf{X}_{k-1}) && \text{(4.39b)}\\ &\propto p_V(\mathbf{z}_k - H(\mathbf{x}_k))\,p_W(\mathbf{x}_k - M_k(\mathbf{x}_{k-1}))\,p(\mathbf{Z}_{k-1}|\mathbf{X}_{k-1})\,p(\mathbf{X}_{k-1}). && \text{(4.39c)} \end{aligned}$$

This recursive relationship is sufficient to determine the density $p(\mathbf{X}_k|\mathbf{Z}_k)$ taking into account the hypotheses on the noise:

$$\begin{aligned} p(\mathbf{X}_k|\mathbf{Z}_k) &\propto \left[\prod_{j=0}^{k} p_V\left(\mathbf{z}_j - H_j(\mathbf{x}_j)\right) \prod_{j=0}^{k-1} p_W\left(\mathbf{x}_{j+1} - M_{j+1}(\mathbf{x}_j)\right)\right] p(\mathbf{x}_0) && \text{(4.40a)}\\ &\propto \exp\left[-\frac{1}{2}\sum_{j=0}^{k}\left(\mathbf{z}_j - H_j(\mathbf{x}_j)\right)^\top \mathbf{R}_j^{-1}\left(\mathbf{z}_j - H_j(\mathbf{x}_j)\right)\right.\\ &\qquad \left.-\frac{1}{2}\sum_{j=0}^{k-1}\left(\mathbf{x}_{j+1} - M_{j+1}(\mathbf{x}_j)\right)^\top \mathbf{Q}_j^{-1}\left(\mathbf{x}_{j+1} - M_{j+1}(\mathbf{x}_j)\right)\right] p(\mathbf{x}_0). && \text{(4.40b)} \end{aligned}$$

The probability density $p(\mathbf{x}_0)$ defines the system state background at time t_0. If we choose the MAP as the estimator, we need to maximise the log-density (the argument of the exponential). The negative log-density $-\ln p(\mathbf{X}_k|\mathbf{Z}_k)$ is merely the 4D-Var cost function (to be minimised!). This justifies the use of this functional in the 4D-Var formalism even if the model operators are nonlinear. This extends the results of chapter 3. Besides, this is another proof of the equivalence between 4D-Var and the Kalman filter in a linear context.

4.5 Particle filters (Monte Carlo)

Can we actually conceive a numerical algorithm that converges to the solution of the ***Bayesian filtering*** problem? Such numerical approach would likely belong to the Monte Carlo methods. This means that a probability density function is represented by a discrete sample of the targeted pdf. Rather than trying to compute the exact solution of the Bayesian filtering equations, the transformations of such filtering (Bayes' rule for the analysis, model propagation for the forecast) are applied to the members of the sample. The statistical properties of the sample, such as the moments, are meant to be those of the targeted pdf. Of course this sampling strategy can only be exact in the asymptotic limit, that is in the limit where number of members (or particles) is going to infinity.

The most popular and simple algorithm of Monte Carlo type that solves the Bayesian filtering equations, is called the ***bootstrap particle filter*** (Gordon et al., 1993). Its description follows.

Sampling Let us consider a sample of particles $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_M\}$. The related probability density function at time t_k is $p_k(\mathbf{x})$: $p_k(\mathbf{x}) \simeq \sum_{i=1}^{M} w_i^k \delta(\mathbf{x} - \mathbf{x}_k^i)$, which is meant to be an approximation of the exact density that the samples emulates. w_k^i is a scalar number which weights the importance of particle i within the ensemble. At this stage, we assume that the weights w_i^k are uniform $w_i^k = 1/M$.

Forecast At the forecast step, the particles are propagated by the model without approximation, *i.e.* $p_{k+1}(\mathbf{x}) \simeq \sum_{i=1}^{M} w_k^i \delta(\mathbf{x} - \mathbf{x}_{k+1}^i)$, with $\mathbf{x}_{k+1}^i = M_{k+1}(\mathbf{x}_k)$.

Analysis The analysis step of the particle filter is extremely simple and elegant. The rigorous implementation of Bayes' rule makes us attach to each particle a specific statistical weight that corresponds to the likelihood of the particle with respect to the data. The weight of each particle is changed according to

$$w_{k+1,+}^i \propto w_{k+1,-}^i p(\mathbf{y}_{k+1}|\mathbf{x}_{k+1}^i). \tag{4.41}$$

Resampling Unfortunately, these statistical weights have a potentially large amplitude of fluctuation. Worse, as sequential filtering goes on, one particle, *i.e.* one trajectory of the model will stand out among the others. Its weight will largely dominate the others ($w_i \lesssim 1$) while the other weights vanish. Then, the particle filter becomes very inefficient as an estimating tool since it looses any variability. This phenomenon is called ***degeneracy*** of the particle filter. An example of such degeneracy is given in Fig. 4.3, where the statistical properties of the biggest weight is studied on a meteorological toy model of 40 and 80

variables. In a degenerate case, the maximum weight will often reach 1 or close to 1, whereas in a balanced case, values very close to 1 will be less frequent.

One way to mitigate this impeding phenomenon is to resample the particles by redrawing a sample with uniform weights from the degenerate distribution. After resampling, all particles have the same weight, $w_k^i = 1/M$.

The principle of the bootstrap particle filter is illustrated in Fig. 4.4.

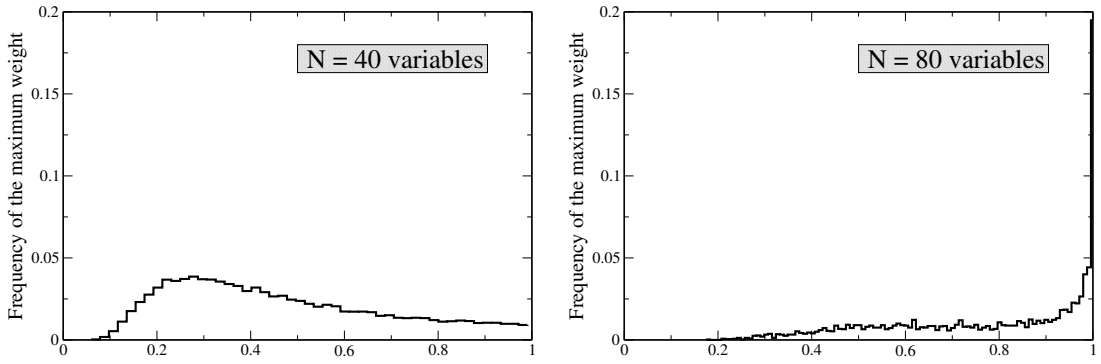


Figure 4.3: On the left: statistical distribution of the weights of the particle bootstrap filter in a balanced case. The physical system is a Lorenz 1995 model with 40 variables (Lorenz and Emanuel, 1998). On the right: the same particle filter is applied to a Lorenz 1995 low-order model, but with 80 variables. The weights clearly degenerate with a peak close to 1.

The particle filter is very efficient on highly nonlinear system but of low dimension. Unfortunately, it is not suited for high-dimensional systems, as soon as the dimension gets over, say about 10. Avoiding degeneracy requires a great many particles. Roughly, this number increases exponentially with the system state space dimension!

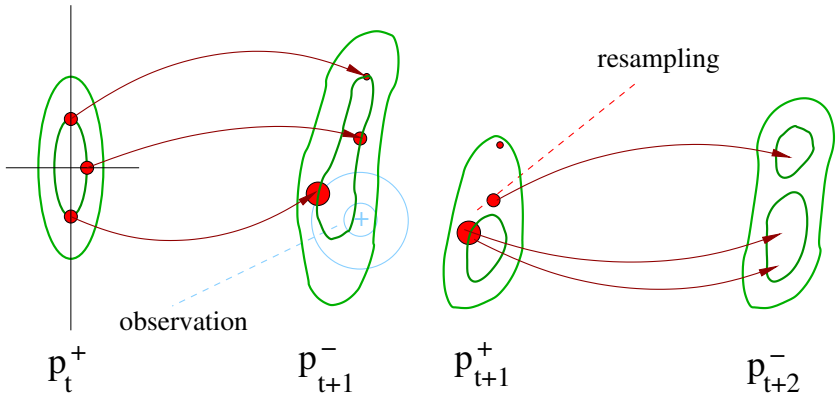


Figure 4.4: Scheme of the bootstrap particle filter.

For the forecast step, it is also crucial to introduce stochastic perturbations of the states. Indeed the ensemble will impoverish with the many resampling to undergo. To enrich the sample, it is necessary to stochastically perturb the states of the system.

Finally, even if it seems inefficient to apply such particle filters, tests have been successfully run in oceanography (Van Leeuwen, 2003) that are promising in the long run.

Chapter 5

The ensemble Kalman filter

In chapter 3, we found that the Kalman filter had two major drawbacks. Firstly, it is numerically hardly affordable in high-dimensional systems. One has to manipulate the error covariance matrix, which requires $N_{\mathrm{x}}(N_{\mathrm{x}}+1)/2$ scalars to be stored. Clearly this is impossible for high-dimensional systems for which the storage of a few state vectors is already challenging. Moreover, during the forecast step from t_k to t_{k+1}, one has to compute a forecast error covariance matrix of the form (see chapter 3)

$$\mathbf{P}^{\mathrm{f}}_{k+1} = \mathbf{M}_{k+1}\mathbf{P}^{\mathrm{a}}_{k}\mathbf{M}^{\top}_{k+1} + \mathbf{Q}_k. \tag{5.1}$$

Such computation would require to use the model $2N_{\mathrm{x}}$ times (a left matrix multiplication by $\mathbf{M}_{k+1}$, followed by a right matrix multiplication by $\mathbf{M}^{\top}_{k+1}$). For a high-dimension system, this is likely to be much too costly, even if leveraging on parallel computing.

Another drawback of the Kalman filter is that its extension to nonlinear models, namely the extended Kalman filter, is an approximation. It makes use of the tangent linear of the model. When the tangent linear approximation is breached, for instance when the timestep between two consecutive updates is long enough, the extended Kalman filter may yield a too coarse approximation of the forecast error covariance matrix, which may induce the divergence of the filter.

5.1 The reduced rank square root filter

The reduced rank square root filter, denoted RRSQRT, is a solution to the main problem of the dimensionality. The issue of the computation and propagation of the error covariance matrices is astutely circumvented. The error covariance matrices are represented by their principal axes (those directions with the largest eigenvalues), that is to say by a limited selection of modes. The update and the forecast step will therefore bear on a limited number of modes, rather than on matrices (*i.e.* a collection of N_{x} modes for a state space of dimension N_{x}).

The initial system state is $\mathbf{x}^{\mathrm{f}}_0$, with an error covariance matrix $\mathbf{P}^{\mathrm{f}}_0$. We assume a decomposition in terms of the principal modes, $\mathbf{P}^{\mathrm{f}}_0 : \mathbf{P}^{\mathrm{f}}_0 \simeq \mathbf{S}^{\mathrm{f}}_0(\mathbf{S}^{\mathrm{f}}_0)^{\top}$, where $\mathbf{S}^{\mathrm{f}}_0$ is a matrix of size $N_{\mathrm{x}} \times N_{\mathrm{m}}$ with N_{m} N_{x}-vector columns that coincide with the N_{m} first dominant modes

of $\mathbf{P}_0^{\mathrm{f}}$. For instance, this could be obtained through the diagonalisation of the symmetric semi-definite positive $\mathbf{P}_0^{\mathrm{f}}$:

$$\mathbf{P}_0^{\mathrm{f}} = \sum_{n=1}^{N_{\mathrm{x}}} \sigma_n \mathbf{s}_n \mathbf{s}_n^\top \approx \sum_{i=1}^{N_{\mathrm{m}}} \sigma_i \mathbf{s}_i \mathbf{s}_i^\top, \tag{5.2}$$

where the eigenvalues are arranged by decreasing order $\sigma_1 \geq \sigma_2 \geq \ldots \sigma_{N_{\mathrm{x}}} \geq 0$. We could choose $\mathbf{S}_0^{\mathrm{f}} = \left[\begin{array}{cccc} \sqrt{\sigma_1}\mathbf{s}_1 & \sqrt{\sigma_2}\mathbf{s}_2 & \cdots & \sqrt{\sigma_{N_{\mathrm{m}}}}\mathbf{s}_{N_{\mathrm{m}}} \end{array} \right]$ so that indeed $\mathbf{P}_0^{\mathrm{f}} \simeq \mathbf{S}_0^{\mathrm{f}} (\mathbf{S}_0^{\mathrm{f}})^\top$.

Next, we assume that such decomposition persists at later time: $\mathbf{P}_k^{\mathrm{f}} \simeq \mathbf{S}_k^{\mathrm{f}} (\mathbf{S}_k^{\mathrm{f}})^\top$, or getting rid of the time index, $\mathbf{P}^{\mathrm{f}} \simeq \mathbf{S}_{\mathrm{f}} \mathbf{S}_{\mathrm{f}}^\top$. The background representation has been simplified. Rather than thinking in terms of $\mathbf{P}^{\mathrm{f}}$, we think about its dominant modes $\mathbf{S}_{\mathrm{f}}$, as well as the transformed matrix of size $N_{\mathrm{y}} \times N_{\mathrm{m}}$ in observation space, $\mathbf{Y}_{\mathrm{f}} = \mathbf{H}\mathbf{S}_{\mathrm{f}}$. $\mathbf{H}$ is either the observation operator when it is linear, or its tangent linear. These matrices appear when computing the Kalman gain in the analysis step,

$$\mathbf{K}^* = \mathbf{P}^{\mathrm{f}} \mathbf{H}^\top \left(\mathbf{H} \mathbf{P}^{\mathrm{f}} \mathbf{H}^\top + \mathbf{R} \right)^{-1} \tag{5.3a}$$

$$= \mathbf{S}_{\mathrm{f}} \mathbf{S}_{\mathrm{f}}^\top \mathbf{H}^\top \left(\mathbf{H} \mathbf{S}_{\mathrm{f}} \mathbf{S}_{\mathrm{f}}^\top \mathbf{H}^\top + \mathbf{R} \right)^{-1} \tag{5.3b}$$

$$= \mathbf{S}_{\mathrm{f}} \left(\mathbf{H}\mathbf{S}_{\mathrm{f}} \right)^\top \left(\left(\mathbf{H}\mathbf{S}_{\mathrm{f}} \right) \left(\mathbf{H}\mathbf{S}_{\mathrm{f}} \right)^\top + \mathbf{R} \right)^{-1}. \tag{5.3c}$$

Hence, the Kalman gain, computed at the analysis step, is simply expressed with the help of the $\mathbf{Y}_{\mathrm{f}}$ matrix:

$$\mathbf{K}^* = \mathbf{S}_{\mathrm{f}} \mathbf{Y}_{\mathrm{f}}^\top \left(\mathbf{Y}_{\mathrm{f}} \mathbf{Y}_{\mathrm{f}}^\top + \mathbf{R} \right)^{-1}. \tag{5.4}$$

The analysis estimate $\mathbf{x}^{\mathrm{a}}$ can be obtained from this gain and the standard update formula.

Then, what happens to the formula that gives the analysis error covariance matrix $\mathbf{P}^{\mathrm{a}}$? We have

$$\mathbf{P}^{\mathrm{a}} = (\mathbf{I}_x - \mathbf{K}^* \mathbf{H}) \mathbf{P}^{\mathrm{f}} \tag{5.5a}$$

$$= \left(\mathbf{I}_x - \mathbf{S}_{\mathrm{f}} \mathbf{Y}_{\mathrm{f}}^\top \left(\mathbf{Y}_{\mathrm{f}} \mathbf{Y}_{\mathrm{f}}^\top + \mathbf{R} \right)^{-1} \mathbf{H} \right) \mathbf{S}_{\mathrm{f}} \mathbf{S}_{\mathrm{f}}^\top \tag{5.5b}$$

$$= \mathbf{S}_{\mathrm{f}} \left(\mathbf{I}_m - \mathbf{Y}_{\mathrm{f}}^\top \left(\mathbf{Y}_{\mathrm{f}} \mathbf{Y}_{\mathrm{f}}^\top + \mathbf{R} \right)^{-1} \mathbf{Y}_{\mathrm{f}} \right) \mathbf{S}_{\mathrm{f}}^\top. \tag{5.5c}$$

We now look for a ***square root*** matrix $\mathbf{S}^{\mathrm{a}}$ such that $\mathbf{S}^{\mathrm{a}} \left(\mathbf{S}^{\mathrm{a}} \right)^\top = \mathbf{P}^{\mathrm{a}}$. One such matrix is

$$\mathbf{S}^{\mathrm{a}} = \mathbf{S}^{\mathrm{f}} \left(\mathbf{I}_m - \mathbf{Y}_{\mathrm{f}}^\top (\mathbf{Y}_{\mathrm{f}} \mathbf{Y}_{\mathrm{f}}^\top + \mathbf{R})^{-1} \mathbf{Y}_{\mathrm{f}} \right)^{1/2}. \tag{5.6}$$

Of size $N_{\mathrm{x}} \times N_{\mathrm{m}}$, $\mathbf{S}^{\mathrm{a}}$ represents a collection of m state vectors, that is to say a posterior ensemble. This avoids a brutal calculation of the error covariance matrices. The computation of the square root matrix might look numerically costly. This is not the case though since $\mathbf{Y}_{\mathrm{f}}$ is of reduced size and $\mathbf{I}_m - \mathbf{Y}_{\mathrm{f}}^\top (\mathbf{Y}_{\mathrm{f}} \mathbf{Y}_{\mathrm{f}}^\top + \mathbf{R})^{-1} \mathbf{Y}_{\mathrm{f}}$ is of dimension $N_{\mathrm{m}} \times N_{\mathrm{m}}$. In addition, the conditioning of the square root matrix is better than that of the initial error covariance matrix (it is the square root of the original conditioning). This ensures an improved numerical precision.

After the analysis step, we seek to reduce the dimension of the system. We wish to shrink the number of modes from N_m to $N_\mathrm{m} - N_\mathrm{q}$. We first diagonalise $\mathbf{S}_\mathrm{a}^\top \mathbf{S}_\mathrm{a} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^\top$. We consider the first $N_\mathrm{m} - N_\mathrm{q}$ eigenmodes with the largest eigenvalues. We keep the first $N_\mathrm{m} - N_\mathrm{q}$ eigenvectors, which correspond to the first $N_\mathrm{m} - N_\mathrm{q}$ columns of $\mathbf{V}$ if the diagonal entries of $\mathbf{\Lambda}$ are stored in decreasing order. These $N_\mathrm{m} - N_\mathrm{q}$ vectors are stored in the $(N_\mathrm{m} - N_\mathrm{q}) \times N_\mathrm{x}$ matrix $\widetilde{\mathbf{V}}$. Then $\mathbf{S}^\mathrm{a}$ is reduced to $\widetilde{\mathbf{S}}^\mathrm{a} \equiv \mathbf{S}^\mathrm{a}\widetilde{\mathbf{V}}$.

At the forecast step, the analysis is forecasted through $\mathbf{x}_{k+1}^\mathrm{f} = M_{k+1}(\mathbf{x}_k^\mathrm{a})$. The square root matrix $\mathbf{S}_k^\mathrm{a}$ is propagated with the tangent model. Then, the matrix is enlarged by adding N_q modes that are meant to introduce model error variability. The matrix of these augmented modes is of the form

$$\mathbf{S}_{k+1}^\mathrm{f} = [\mathbf{M}_{k+1}\widetilde{\mathbf{S}}_k^\mathrm{a}, \mathbf{T}_k], \tag{5.7}$$

where $\mathbf{M}$ is the tangent linear model of $\mathcal{M}$. It has N_m modes, so that the assimilation procedure can be cycled.

This type of filter has been put forward in air quality where the number of chemical species considerably increases the state space dimension (Segers, 2002). A similar filter, known as SEEK, has also been used in oceanography for the same reasons (Pham et al., 1998).

These filters clearly overcome the main drawback of the Kalman filter, assuming the dynamics of the system can indeed be represented with a limited $N_\mathrm{m} \ll N_\mathrm{x}$ number of modes. However, by still making use of the tangent linear model, they only approximate the nonlinear propagation of uncertainty. Moreover, it requires to develop the tangent linear of the model. In that respect, one can propose a better reduced Kalman filter, known as the ***ensemble Kalman filter***.

5.2 The stochastic ensemble Kalman filter

The ensemble Kalman filter was proposed by Geir Evensen in 1994, and later amended in 1998 (Evensen, 1994; Burgers et al., 1998; Houtekamer and Mitchell, 1998; Evensen, 2009). Its semi-empirical justification could be disconcerting and not as obvious as that of the RRSQRT. Nevertheless, the ensemble Kalman filter has proven very efficient on a large number of academic and operational data assimilation problems.

The ensemble Kalman filter (EnKF in the following) is a reduced-order Kalman filter, just like the RRSQRT filter. It only handles the error statistics up to second order. Therefore, the EnKF is not a particle filter of the family that we discussed in chapter 4. It is a Gaussian filter. It has been shown that in the limit of a large number of particles, the EnKF does not solve the Bayesian filtering problem as exposed in chapter 4, as opposed to the particle filter, except when the models are linear and when the initial error distribution is Gaussian.

Yet, just as the particle filter and the RRSQRT filter, the EnKF is based on the concept of particles, a collection of state vectors, the members of the ensemble. Rather than propagating huge covariance matrices, the errors are emulated by scattered particles, a collection of state vectors whose dispersion (variability) is meant to be representative of the uncertainty of the system's state. Just like the particle filter, but unlike the RRSQRT, the members are to be propagated by the model without any linearisation.

Reduced rank square root (RRSQRT) filter

1. Initialisation
   - System state $\mathbf{x}_0^{\mathrm{f}}$ and error covariance matrix $\mathbf{P}_0^{\mathrm{f}}$.
   - Decomposition: $\mathbf{Q}_k = \mathbf{T}_k \mathbf{T}_k^\top$ and $\mathbf{P}_0^{\mathrm{f}} = \mathbf{S}_0^{\mathrm{f}} {\mathbf{S}_0^{\mathrm{f}}}^\top$.
2. For $t_k = 1, 2, \cdots$
   (a) Analysis
   - Gain computation:

$$\mathbf{Y}_k = \mathbf{H}_k \mathbf{S}_k^{\mathrm{f}}, \qquad \mathbf{K}_k^* = \mathbf{S}_k^{\mathrm{f}} \mathbf{Y}_k^\top \left(\mathbf{Y}_k \mathbf{Y}_k^\top + \mathbf{R}_k \right)^{-1}$$

   - Computation of the analysis

$$\mathbf{x}_k^{\mathrm{a}} = \mathbf{x}_k^{\mathrm{f}} + \mathbf{K}_k^* \left(\mathbf{y}_k - H_k(\mathbf{x}_k^{\mathrm{f}}) \right)$$

   - Computing the (square root) matrix of the modes

$$\mathbf{S}_k^{\mathrm{a}} = \mathbf{S}_k^{\mathrm{f}} \left(\mathbf{I}_m - \mathbf{Y}_k^\top (\mathbf{Y}_k \mathbf{Y}_k^\top + \mathbf{R}_k)^{-1} \mathbf{Y}_k \right)^{1/2}$$

   - $\mathbf{S}_k^{\mathrm{a}}$ has m modes.

   (b) Reduction
   - Diagonalisation $\mathbf{V} \mathbf{\Lambda} \mathbf{V}^\top = {\mathbf{S}_k^{\mathrm{a}}}^\top \mathbf{S}_k^{\mathrm{a}}$
   - Reduction of the ensemble $\widetilde{\mathbf{S}}_k^{\mathrm{a}} = \mathbf{S}_k^{\mathrm{a}} \widetilde{\mathbf{V}}$
   - $\widetilde{\mathbf{S}}_k^{\mathrm{a}}$ has $m - q$ modes.

   (c) Forecast
   - Computation of the forecast estimate $\mathbf{x}_{k+1}^{\mathrm{f}} = M_{k+1}(\mathbf{x}_k^{\mathrm{a}})$
   - Computing the matrix of the modes

$$\mathbf{S}_{k+1}^{\mathrm{f}} = [\mathbf{M}_{k+1} \widetilde{\mathbf{S}}_k^{\mathrm{a}}, \mathbf{T}_k]$$

   - $\mathbf{S}_{k+1}^{\mathrm{f}}$ has m modes.

5.2.1 The analysis step

The EnKF seeks to mimic the analysis step of the Kalman filter but with an ensemble of limited size rather than with error covariance matrices. The goal is to perform for each member of the ensemble an analysis of the form

$$\mathbf{x}_i^{\mathrm{a}} = \mathbf{x}_i^{\mathrm{f}} + \mathbf{K}^* \left(\mathbf{y}_i - H(\mathbf{x}_i^{\mathrm{f}}) \right). \tag{5.8}$$

where $i = 1, \ldots, N_{\mathrm{e}}$ is the member index in the ensemble, $\mathbf{x}_i^{\mathrm{f}}$ is state vector i forecasted at the analysis time. For the moment, we shall assume that $\mathbf{y}_i \triangleq \mathbf{y}$, the observation vector, does not depend on i. $\mathbf{K}^*$ should be the Kalman gain, that we would like to compute from the ensemble statistics

$$\mathbf{K}^* = \mathbf{P}^{\mathrm{f}} \mathbf{H}^\top \left(\mathbf{H} \mathbf{P}^{\mathrm{f}} \mathbf{H}^\top + \mathbf{R} \right)^{-1}, \tag{5.9}$$

where $\mathbf{R}$ is the given observational error covariance matrix. The forecast error covariance matrix is estimated from the ensemble

$$\overline{\mathbf{x}}^{\mathrm{f}} = \frac{1}{N_{\mathrm{e}}} \sum_{i=1}^{N_{\mathrm{e}}} \mathbf{x}_i^{\mathrm{f}}, \qquad \mathbf{P}^{\mathrm{f}} = \frac{1}{N_{\mathrm{e}}-1} \sum_{i=1}^{N_{\mathrm{e}}} \left(\mathbf{x}_i^{\mathrm{f}} - \overline{\mathbf{x}}^{\mathrm{f}}\right) \left(\mathbf{x}_i^{\mathrm{f}} - \overline{\mathbf{x}}^{\mathrm{f}}\right)^{\top}. \tag{5.10}$$

Thanks to Eq. (5.8), we obtain a posterior ensemble $\{\mathbf{x}_i^{\mathrm{a}}\}_{i=1,\dots,N_{\mathrm{e}}}$ from which we can compute the posterior statistics. Firstly, the analysis is computed as the mean of the posterior ensemble

$$\overline{\mathbf{x}}^{\mathrm{a}} = \frac{1}{N_{\mathrm{e}}} \sum_{i=1}^{N_{\mathrm{e}}} \mathbf{x}_i^{\mathrm{a}}. \tag{5.11}$$

Secondly, the analysis error covariance matrix is computed from the posterior ensemble

$$\mathbf{P}^{\mathrm{a}} = \frac{1}{N_{\mathrm{e}}-1} \sum_{i=1}^{N_{\mathrm{e}}} \left(\mathbf{x}_i^{\mathrm{a}} - \overline{\mathbf{x}}^{\mathrm{a}}\right) \left(\mathbf{x}_i^{\mathrm{a}} - \overline{\mathbf{x}}^{\mathrm{a}}\right)^{\top}. \tag{5.12}$$

Note, however, that the computation of $\mathbf{P}^{\mathrm{a}}$ is optional since it is not required by the sequential algorithm. If $\mathbf{y}_i \triangleq \mathbf{y}$, the ensemble anomalies, $\mathbf{e}_i^{\mathrm{a}} = \mathbf{x}_i^{\mathrm{a}} - \overline{\mathbf{x}}^{\mathrm{a}}$, *i.e.* the deviations of the ensemble members from the mean are

$$\mathbf{e}_i^{\mathrm{a}} = \mathbf{e}_i^{\mathrm{f}} + \mathbf{K}^* \left(\mathbf{0} - \mathbf{H}\mathbf{e}_i^{\mathrm{f}}\right) = \left(\mathbf{I}_x - \mathbf{K}^*\mathbf{H}\right) \mathbf{e}_i^{\mathrm{f}}, \tag{5.13}$$

which yields

$$\mathbf{P}^{\mathrm{a}} = \frac{1}{N_{\mathrm{e}}-1} \sum_{i=1}^{N_{\mathrm{e}}} \left(\mathbf{x}_i^{\mathrm{a}} - \overline{\mathbf{x}}^{\mathrm{a}}\right) \left(\mathbf{x}_i^{\mathrm{a}} - \overline{\mathbf{x}}^{\mathrm{a}}\right)^{\top} = \left(\mathbf{I}_x - \mathbf{K}^*\mathbf{H}\right) \mathbf{P}^{\mathrm{f}} \left(\mathbf{I}_x - \mathbf{K}^*\mathbf{H}\right)^{\top}. \tag{5.14}$$

However, to mimic the BLUE analysis of the Kalman filter, we should have obtained instead

$$\mathbf{P}^{\mathrm{a}} = \left(\mathbf{I}_x - \mathbf{K}^*\mathbf{H}\right) \mathbf{P}^{\mathrm{f}} \left(\mathbf{I}_x - \mathbf{K}^*\mathbf{H}\right)^{\top} + \mathbf{K}^* \mathbf{R} \mathbf{K}^{*\top}. \tag{5.15}$$

Therefore, the error is systematically underestimated since the second positive term is ignored, which is likely to lead to the divergence of the EnKF.

A solution around this problem is to perturb the observation vector for each member, $\mathbf{y}_i = \mathbf{y} + \mathbf{u}_i$, where $\mathbf{u}_i$ is drawn from the Gaussian distribution $\mathbf{u}_i \sim N(\underline{0}, \mathbf{R})$[1]. Firstly, we would like to ensure that $\sum_{i=1}^{N_{\mathrm{e}}} \mathbf{u}_i = \underline{0}$ to avoid biases. Besides, we define the empirical error covariance matrix

$$\mathbf{R}_u = \frac{1}{N_{\mathrm{e}}-1} \sum_{i=1}^{N_{\mathrm{e}}} \mathbf{u}_i \mathbf{u}_i^{\top}, \tag{5.16}$$

which should coincide with $\mathbf{R}$ in the asymptotic limit $N_{\mathrm{e}} \longrightarrow \infty$.

The anomalies are modified accordingly,

$$\mathbf{e}_i^{\mathrm{a}} = \mathbf{e}_i^{\mathrm{f}} + \mathbf{K}_u^* \left(\mathbf{e}_i^o - \mathbf{H}\mathbf{e}_i^{\mathrm{f}}\right), \tag{5.17}$$

where the gain $\mathbf{K}_u^*$ is the same as the Kalman gain $\mathbf{K}^*$ but with $\mathbf{R}$ replaced by the empirical $\mathbf{R}_u$. This error yields the correct analysis error covariance matrix:

$$\mathbf{P}^{\mathrm{a}} = \left(\mathbf{I}_x - \mathbf{K}_u^*\mathbf{H}\right) \mathbf{P}^{\mathrm{f}} \left(\mathbf{I}_x - \mathbf{K}_u^*\mathbf{H}\right)^{\top} + \mathbf{K}_u^* \mathbf{R} \mathbf{K}_u^* = \left(\mathbf{I}_x - \mathbf{K}_u^*\mathbf{H}\right) \mathbf{P}^{\mathrm{f}}. \tag{5.18}$$

[1] More to the point, it is $H(\mathbf{x}_i^{\mathrm{f}})$ that needs to be perturbed rather than $\mathbf{y}$. Since both are formally equivalent, *perturbing the observations* is an expression that remains very common.

5.2.2 The forecast step

In the forecast step, the updated ensemble obtained in the analysis is propagated by the model to the next timestep,

$$\text{for } i = 1, \dots, N_\mathrm{e} \quad \mathbf{x}_i^\mathrm{f} = M(\mathbf{x}_i^\mathrm{a}). \tag{5.19}$$

The forecast estimate is the mean of the forecast ensemble, *i.e.*

$$\overline{\mathbf{x}}^\mathrm{f} = \frac{1}{N_\mathrm{e}} \sum_{i=1}^{N_\mathrm{e}} \mathbf{x}_i^\mathrm{f}, \tag{5.20}$$

while the forecast error covariance matrix is estimated via

$$\mathbf{P}^\mathrm{f} = \frac{1}{N_\mathrm{e} - 1} \sum_{i=1}^{N_\mathrm{e}} \left(\mathbf{x}_i^\mathrm{f} - \overline{\mathbf{x}}^\mathrm{f}\right) \left(\mathbf{x}_i^\mathrm{f} - \overline{\mathbf{x}}^\mathrm{f}\right)^\top. \tag{5.21}$$

It is important to observe that we have avoided the use of the tangent linear operator, or of any linearisation. This makes a significant difference with the RRSQRT filter. This difference should particularly matter in stronger nonlinear regimes.

5.2.3 Assets of the EnKF

Like the RRSQRT filter, the particle filter or the 3D-Var, the EnKF does not require the adjoint model. More interestingly it can help avoid the computation of the adjoint of the observation operator. Let us consider the formula of the Kalman gain Eq. (5.9) that clearly makes use of the adjoint of the observation operator. Instead of computing the matrix products $\mathbf{P}^\mathrm{f}\mathbf{H}^\top$ and $\mathbf{H}\mathbf{P}^\mathrm{f}\mathbf{H}^\top$, we compute

$$\overline{\mathbf{y}}^\mathrm{f} = \frac{1}{N_\mathrm{e}} \sum_{i=1}^{N_\mathrm{e}} H\left(\mathbf{x}_i^\mathrm{f}\right), \tag{5.22a}$$

$$\begin{aligned} \mathbf{P}^\mathrm{f}\mathbf{H}^\top &= \frac{1}{N_\mathrm{e} - 1} \sum_{i=1}^{N_\mathrm{e}} \left(\mathbf{x}_i^\mathrm{f} - \overline{\mathbf{x}}^\mathrm{f}\right) \left[\mathbf{H}\left(\mathbf{x}_i^\mathrm{f} - \overline{\mathbf{x}}^\mathrm{f}\right)\right]^\top \\ &\simeq \frac{1}{N_\mathrm{e} - 1} \sum_{i=1}^{N_\mathrm{e}} \left(\mathbf{x}_i^\mathrm{f} - \overline{\mathbf{x}}^\mathrm{f}\right) \left[H\left(\mathbf{x}_i^\mathrm{f}\right) - \overline{\mathbf{y}}^\mathrm{f}\right]^\top, \end{aligned} \tag{5.22b}$$

$$\begin{aligned} \mathbf{H}\mathbf{P}^\mathrm{f}\mathbf{H}^\top &= \frac{1}{N_\mathrm{e} - 1} \sum_{i=1}^{N_\mathrm{e}} \left[\mathbf{H}\left(\mathbf{x}_i^\mathrm{f} - \overline{\mathbf{x}}^\mathrm{f}\right)\right] \left[\mathbf{H}\left(\mathbf{x}_i^\mathrm{f} - \overline{\mathbf{x}}^\mathrm{f}\right)\right]^\top \\ &\simeq \frac{1}{N_\mathrm{e} - 1} \sum_{i=1}^{N_\mathrm{e}} \left[H\left(\mathbf{x}_i^\mathrm{f}\right) - \overline{\mathbf{y}}^\mathrm{f}\right] \left[H\left(\mathbf{x}_i^\mathrm{f}\right) - \overline{\mathbf{y}}^\mathrm{f}\right]^\top. \end{aligned} \tag{5.22c}$$

This avoids the use of not only the adjoint model, but also the tangent linear of the observation operator.

Except with the computation of the Kalman gain, all the operations on the ensemble members are independent. This implies that their parallelisation can be trivially carried out. This is one of the main reasons for the success and popularity of the EnKF.

Note that there are other variants of the EnKF. The habit of the author of these notes is to call this variant with perturbation of the observations ***the stochastic EnKF***.

The stochastic ensemble Kalman filter

1. Initialisation
   - Initial system state $\mathbf{x}_0^{\mathrm{f}}$ and initial error covariance matrix $\mathbf{P}_0^{\mathrm{f}}$.
2. For $t_k = 1, 2, \cdots$
   (a) Observation
   - Draw a statistically consistent observation set: for $i = 1, .., N_{\mathrm{e}}$:

$$\mathbf{y}_i = \mathbf{y} + \mathbf{u}_i \quad \sum_{i=1}^{N_{\mathrm{e}}} \mathbf{u}_i = 0$$

   - Compute the empirical error covariance matrix

$$\mathbf{R}_u = \frac{1}{N_{\mathrm{e}} - 1} \sum_{i=1}^{N_{\mathrm{e}}} \mathbf{u}_i \mathbf{u}_i^{\top}$$

   (b) Analysis
   - Compute the gain $\mathbf{K}_u^* = \mathbf{P}^{\mathrm{f}} \mathbf{H}^{\top} \left(\mathbf{H} \mathbf{P}^{\mathrm{f}} \mathbf{H}^{\top} + \mathbf{R}_u \right)^{-1}$
   - Ensemble of analyses for $i = 1, .., N_{\mathrm{e}}$

$$\mathbf{x}_i^{\mathrm{a}} = \mathbf{x}_i^{\mathrm{f}} + \mathbf{K}_u^* \left(\mathbf{y}_i - H(\mathbf{x}_i^{\mathrm{f}}) \right)$$

     and their mean

$$\overline{\mathbf{x}}^{\mathrm{a}} = \frac{1}{N_{\mathrm{e}}} \sum_{i=1}^{N_{\mathrm{e}}} \mathbf{x}_i^{\mathrm{a}}$$

   - Compute the analysis error covariance matrix

$$\mathbf{P}^{\mathrm{a}} = \frac{1}{N_{\mathrm{e}} - 1} \sum_{i=1}^{N_{\mathrm{e}}} \left(\mathbf{x}_i^{\mathrm{a}} - \overline{\mathbf{x}}^{\mathrm{a}} \right) \left(\mathbf{x}_i^{\mathrm{a}} - \overline{\mathbf{x}}^{\mathrm{a}} \right)^{\top}$$

   (c) Forecast
   - Compute the ensemble forecast for $i = 1, .., N_{\mathrm{e}}$ $\mathbf{x}_i^{\mathrm{f}} = M(\mathbf{x}_i^{\mathrm{a}})$ and their mean

$$\overline{\mathbf{x}}^{\mathrm{f}} = \frac{1}{N_{\mathrm{e}}} \sum_{i=1}^{N_{\mathrm{e}}} \mathbf{x}_i^{\mathrm{f}}$$

   - Compute the forecast error covariance matrix

$$\mathbf{P}^{\mathrm{f}} = \frac{1}{N_{\mathrm{e}} - 1} \sum_{i=1}^{N_{\mathrm{e}}} \left(\mathbf{x}_i^{\mathrm{f}} - \overline{\mathbf{x}}^{\mathrm{f}} \right) \left(\mathbf{x}_i^{\mathrm{f}} - \overline{\mathbf{x}}^{\mathrm{f}} \right)^{\top}$$

5.2.4 Examples

Let us again consider the nonlinear oscillator of chapter 2. The performance of the stochastic EnKF on this test case is shown in Fig. 5.1.

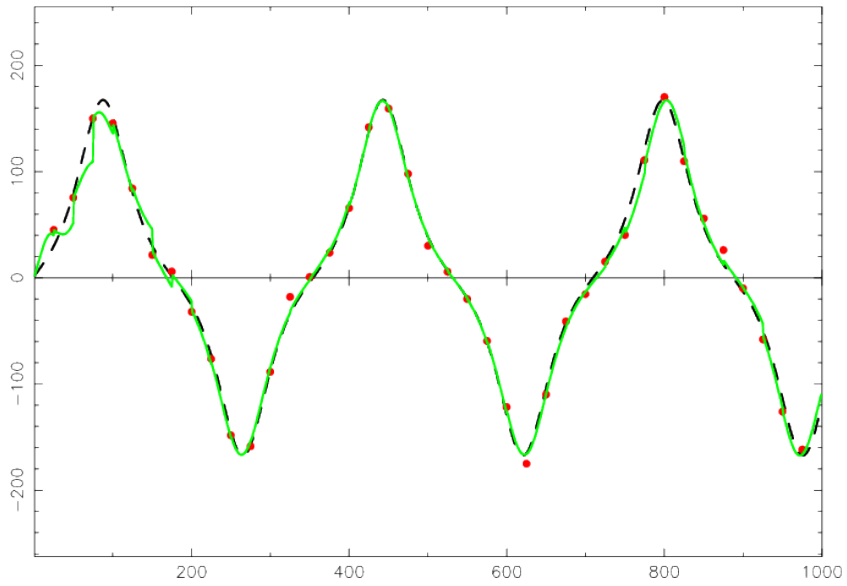


Figure 5.1: A stochastic ensemble Kalman filter applied to the anharmonic oscillator with one observations every 25 timesteps.

Let us now apply the stochastic EnKF on the somewhat more difficult perturbed anharmonic oscillator introduced in chapter 4. It shows the divergence of the extended Kalman filter in the absence of frequent observations (less than one observation every 58 timesteps). Under the same setting, we implement the EnKF with a frequency of observation of one observation every 50 timesteps, then one observation every 100 timesteps. The results are shown in Fig. 5.2 and Fig. 5.3 respectively. No divergence is observed.

Hence, the EnKF accommodates a lower observation frequency than the extended Kalman filter.

5.3 The deterministic ensemble Kalman filter

The stochastic EnKF enables to individually track each member of the ensemble, with the analysis step for them to interact. This nice property comes with a price: we need to independently perturb the observation vector of each member. Although this seems elegant, it also introduces numerical noise when drawing the perturbation $\mathbf{u}_i$. This can affect the performance of the stochastic EnKF, especially when the number of observations for a single analysis is limited.

An alternative idea to perform a statistically consistent EnKF analysis would be to follow the square root approach of the RRSQRT filter. This yields the so-called ***ensemble square root Kalman filter (EnSRF)***. Because with this scheme the observations are not perturbed, it qualifies as a ***deterministic EnKF***.

There are several variants of the EnSRF. Be aware that they are, to a large extent, theoretically equivalent. In this course, we shall focus on the so-called ensemble-transform variant of the deterministic EnKF, usually abbreviated ***ETKF*** (Bishop et al., 2001; Hunt et al., 2007). Rather than performing the linear algebra in state or observation space, the

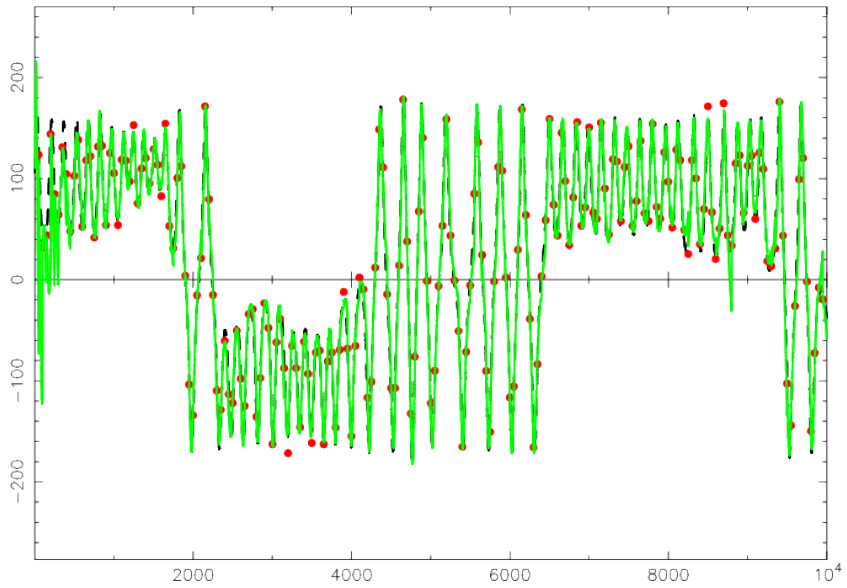


Figure 5.2: Implementation of the stochastic EnKF on the perturbed anharmonic oscillator. The system is observed once every 50 integration timesteps. This graph can be compared to that of Fig. 4.1.

algebra is mostly performed in the ***ensemble space***.

5.3.1 Algebra in the ensemble space

Let us define this ensemble space. We can write the forecast error covariance matrix as

$$\mathbf{P}^{\mathrm{f}} = \frac{1}{N_{\mathrm{e}}-1} \sum_{i=1}^{N_{\mathrm{e}}} \left(\mathbf{x}_i^{\mathrm{f}} - \overline{\mathbf{x}}^{\mathrm{f}}\right) \left(\mathbf{x}_i^{\mathrm{f}} - \overline{\mathbf{x}}^{\mathrm{f}}\right)^{\top} = \mathbf{X}_{\mathrm{f}} \mathbf{X}_{\mathrm{f}}^{\top}. \tag{5.23}$$

where $\mathbf{X}_{\mathrm{f}}$ is a $N_{\mathrm{x}} \times N_{\mathrm{m}}$ matrix whose columns are the *normalised* anomalies

$$[\mathbf{X}_{\mathrm{f}}]_i = \frac{\mathbf{x}_i^{\mathrm{f}} - \overline{\mathbf{x}}^{\mathrm{f}}}{\sqrt{N_{\mathrm{e}}-1}}. \tag{5.24}$$

The matrix of the anomalies, $\mathbf{X}_{\mathrm{f}}$, plays the role of the matrix of the reduced modes $\mathbf{S}^{\mathrm{f}}$ of the RRSQRT filter.

We shall assume that the analysis belongs to the affine space generated by

$$\mathbf{x} = \overline{\mathbf{x}}^{\mathrm{f}} + \mathbf{X}_{\mathrm{f}} \mathbf{w} \tag{5.25}$$

where $\mathbf{w}$ is a vector of coefficients in the vector space $\mathbb{R}^{N_{\mathrm{e}}}$. We call it the ***ensemble space***. Note that the decomposition of $\mathbf{x}$ is not unique, because the vector of coefficients $\mathbf{w} + \lambda \mathbf{1}$ with $\mathbf{1} = (1, \ldots, 1)^{\top} \in \mathbb{R}^{N_{\mathrm{e}}}$ yields the same state vector $\mathbf{x}$ since $\mathbf{X}_{\mathrm{f}} \mathbf{1} = \underline{0}$.

Again we shall use the notation $\mathbf{Y}_{\mathrm{f}}$ to represent $\mathbf{H}\mathbf{X}_{\mathrm{f}}$ if the observation operator is linear. If it is nonlinear, we consider $\mathbf{Y}_{\mathrm{f}}$ to be the matrix of the ***observation anomalies***

$$[\mathbf{Y}_{\mathrm{f}}]_i = \frac{H(\mathbf{x}_i^{\mathrm{f}}) - \overline{\mathbf{y}}^{\mathrm{f}}}{\sqrt{N_{\mathrm{e}}-1}} \quad \text{with} \quad \overline{\mathbf{y}}^{\mathrm{f}} = \frac{1}{N_{\mathrm{e}}} \sum_{i=1}^{N_{\mathrm{e}}} H(\mathbf{x}_i^{\mathrm{f}}), \tag{5.26}$$

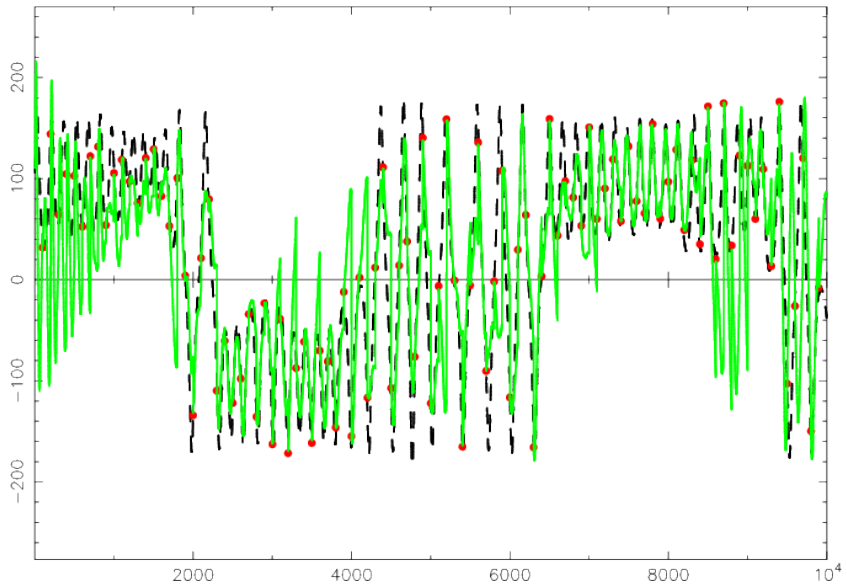


Figure 5.3: Implementation of the stochastic EnKF on the perturbed anharmonic oscillator. The system is observed once every 100 integration timesteps. The filter remains stable and manages to track the truth. This graph can be compared to that of Fig. 4.2.

which stands as a fine approximation following the ideas of section 5.2.3.

5.3.2 Analysis in ensemble space

As opposed to the stochastic EnKF, we wish to perform a single analysis rather than performing an analysis for each member of the ensemble. For the analysis estimate, we can adopt the mean analysis of the stochastic EnKF, *i.e.*

$$\overline{\mathbf{x}}^{\mathrm{a}} = \overline{\mathbf{x}}^{\mathrm{f}} + \mathbf{K}^{*}\left(\mathbf{y} - H(\overline{\mathbf{x}}^{\mathrm{f}})\right) \tag{5.27}$$

where we used $\mathbf{K}^{*} = \mathbf{P}^{\mathrm{f}}\mathbf{H}^{\top}\left(\mathbf{H}\mathbf{P}^{\mathrm{f}}\mathbf{H}^{\top} + \mathbf{R}\right)^{-1}$ rather than $\mathbf{K}_u$, as defined by Eq. (5.17), since the observations are not perturbed in a deterministic approach.

We shall reformulate this analysis but working in the ensemble space. We look for the optimal coefficient vector $\mathbf{w}^{\mathrm{a}}$ that stands for $\mathbf{x}^{\mathrm{a}} \equiv \overline{\mathbf{x}}^{\mathrm{a}}$ defined above. To do so we write

$$\mathbf{x}^{\mathrm{a}} = \overline{\mathbf{x}}^{\mathrm{f}} + \mathbf{X}_{\mathrm{f}}\mathbf{w}^{\mathrm{a}}. \tag{5.28}$$

Inserting this decomposition into Eq. (5.27), and denoting $\boldsymbol{\delta} = \mathbf{y} - H(\overline{\mathbf{x}}^{\mathrm{f}})$, we obtain

$$\overline{\mathbf{x}}^{\mathrm{f}} + \mathbf{X}_{\mathrm{f}}\mathbf{w}^{\mathrm{a}} = \overline{\mathbf{x}}^{\mathrm{f}} + \mathbf{X}_{\mathrm{f}}\mathbf{X}_{\mathrm{f}}^{\top}\mathbf{H}^{\top}\left(\mathbf{H}\mathbf{X}_{\mathrm{f}}\mathbf{X}_{\mathrm{f}}^{\top}\mathbf{H}^{\top} + \mathbf{R}\right)^{-1}\boldsymbol{\delta}, \tag{5.29}$$

which suggests

$$\begin{aligned}\mathbf{w}^{\mathrm{a}} &= \mathbf{X}_{\mathrm{f}}^{\top}\mathbf{H}^{\top}\left(\mathbf{H}\mathbf{X}_{\mathrm{f}}\mathbf{X}_{\mathrm{f}}^{\top}\mathbf{H}^{\top} + \mathbf{R}\right)^{-1}\boldsymbol{\delta} \\ &= \mathbf{Y}_{\mathrm{f}}^{\top}\left(\mathbf{Y}_{\mathrm{f}}\mathbf{Y}_{\mathrm{f}}^{\top} + \mathbf{R}\right)^{-1}\boldsymbol{\delta}.\end{aligned} \tag{5.30}$$

The gain $\mathbf{K}^*$ is computed in the observation space. Remember that using the Sherman-Morrison-Woodbury lemma, it was possible to compute the gain in state space. We showed in chapter 1 that $\mathbf{P}^{\mathrm{b}}\mathbf{H}^\top\left(\mathbf{H}\mathbf{P}^{\mathrm{b}}\mathbf{H}^\top+\mathbf{R}\right)^{-1}=\left(\mathbf{P}_{\mathrm{b}}^{-1}+\mathbf{H}^\top\mathbf{R}^{-1}\mathbf{H}\right)^{-1}\mathbf{H}^\top\mathbf{R}^{-1}$. Now, with the identifications $\mathbf{P}^{\mathrm{b}}=\mathbf{I}_e$ and $\mathbf{H}=\mathbf{Y}_{\mathrm{f}}$, we obtain

$$\mathbf{w}^{\mathrm{a}}=\left(\mathbf{I}_e+\mathbf{Y}_{\mathrm{f}}^\top\mathbf{R}^{-1}\mathbf{Y}_{\mathrm{f}}\right)^{-1}\mathbf{Y}_{\mathrm{f}}^\top\mathbf{R}^{-1}\boldsymbol{\delta}. \tag{5.31}$$

The gain matrix is now computed in the ensemble space rather than the observation space or the state space! This makes a difference as far numerical complexity is concerned.

5.3.3 Generating the posterior ensemble

Writing the analysis in ensemble space is merely a reformulation of the stochastic EnKF, hardly more. The genuine difference between the deterministic EnKF and the stochastic EnKF comes in the generation of the posterior ensemble.

To generate a posterior ensemble that would be representative of the posterior uncertainty, we would like to factorise $\mathbf{P}^{\mathrm{a}}=\mathbf{X}_{\mathrm{a}}\mathbf{X}_{\mathrm{a}}^\top$. We can repeat the derivation of the RRSQRT filter, Eq. (5.6),

$$\begin{aligned}
\mathbf{P}^{\mathrm{a}} &= (\mathbf{I}_x-\mathbf{K}_k^*\mathbf{H})\mathbf{P}^{\mathrm{f}} && \text{(5.32a)}\\
&\simeq \left(\mathbf{I}_x-\mathbf{X}_{\mathrm{f}}\mathbf{Y}_{\mathrm{f}}^\top\left(\mathbf{Y}_{\mathrm{f}}\mathbf{Y}_{\mathrm{f}}^\top+\mathbf{R}\right)^{-1}\mathbf{H}\right)\mathbf{X}_{\mathrm{f}}\mathbf{X}_{\mathrm{f}}^\top && \text{(5.32b)}\\
&\simeq \mathbf{X}_{\mathrm{f}}\left(\mathbf{I}_e-\mathbf{Y}_{\mathrm{f}}^\top\left(\mathbf{Y}_{\mathrm{f}}\mathbf{Y}_{\mathrm{f}}^\top+\mathbf{R}\right)^{-1}\mathbf{Y}_{\mathrm{f}}\right)\mathbf{X}_{\mathrm{f}}^\top. && \text{(5.32c)}
\end{aligned}$$

That is why we choose

$$\mathbf{X}_{\mathrm{a}}=\mathbf{X}_{\mathrm{f}}\left(\mathbf{I}_e-\mathbf{Y}_{\mathrm{f}}^\top(\mathbf{Y}_{\mathrm{f}}\mathbf{Y}_{\mathrm{f}}^\top+\mathbf{R})^{-1}\mathbf{Y}_{\mathrm{f}}\right)^{1/2}. \tag{5.33}$$

Remarquably, this expression can be simplified into

$$\begin{aligned}
\mathbf{X}_{\mathrm{a}} &= \mathbf{X}_{\mathrm{f}}\left(\mathbf{I}_e-\mathbf{Y}_{\mathrm{f}}^\top(\mathbf{Y}_{\mathrm{f}}\mathbf{Y}_{\mathrm{f}}^\top+\mathbf{R})^{-1}\mathbf{Y}_{\mathrm{f}}\right)^{1/2} && \text{(5.34a)}\\
&= \mathbf{X}_{\mathrm{f}}\left(\mathbf{I}_e-\left(\mathbf{I}_e+\mathbf{Y}_{\mathrm{f}}^\top\mathbf{R}^{-1}\mathbf{Y}_{\mathrm{f}}\right)^{-1}\mathbf{Y}_{\mathrm{f}}^\top\mathbf{R}^{-1}\mathbf{Y}_{\mathrm{f}}\right)^{1/2} && \text{(5.34b)}\\
&= \mathbf{X}_{\mathrm{f}}\left[\left(\mathbf{I}_e+\mathbf{Y}_{\mathrm{f}}^\top\mathbf{R}^{-1}\mathbf{Y}_{\mathrm{f}}\right)^{-1}\left(\mathbf{I}_e+\mathbf{Y}_{\mathrm{f}}^\top\mathbf{R}^{-1}\mathbf{Y}_{\mathrm{f}}-\mathbf{Y}_{\mathrm{f}}^\top\mathbf{R}^{-1}\mathbf{Y}_{\mathrm{f}}\right)\right]^{1/2} && \text{(5.34c)}\\
&= \mathbf{X}_{\mathrm{f}}\left(\mathbf{I}_e+\mathbf{Y}_{\mathrm{f}}^\top\mathbf{R}^{-1}\mathbf{Y}_{\mathrm{f}}\right)^{-1/2}, && \text{(5.34d)}
\end{aligned}$$

where we have used the Sherman-Morrison-Woodbury lemma again between the first line and the second line.

This posterior ensemble of anomalies is all that we need to cycle the deterministic EnKF. Defining $\mathbf{T}=\left(\mathbf{I}_e+\mathbf{Y}_{\mathrm{f}}^\top\mathbf{R}^{-1}\mathbf{Y}_{\mathrm{f}}\right)^{-1/2}$, we can build the posterior ensemble as

$$\text{for } i=1,\ldots,N_{\mathrm{e}} \qquad \mathbf{x}_i^{\mathrm{a}}=\overline{\mathbf{x}}^{\mathrm{a}}+\sqrt{N_{\mathrm{e}}-1}\mathbf{X}_{\mathrm{f}}\left[\mathbf{T}\right]_i=\overline{\mathbf{x}}^{\mathrm{f}}+\mathbf{X}_{\mathrm{f}}\left(\mathbf{w}^{\mathrm{a}}+\sqrt{N_{\mathrm{e}}-1}\left[\mathbf{T}\right]_i\right). \tag{5.35}$$

The deterministic ensemble Kalman filter
(Ensemble transform Kalman variant)

1. Initialisation
   - Ensemble of state vectors $\mathbf{E}_0^{\mathrm{f}} = \{\mathbf{x}_0, \ldots, \mathbf{x}_{N_\mathrm{e}}\}$.
2. For $t_k = 1, 2, \cdots$
   (a) Analysis
   - Compute forecast mean, the ensemble anomalies and the observation anomalies:

$$\begin{aligned}\overline{\mathbf{x}}_k^{\mathrm{f}} &= \mathbf{E}_k^{\mathrm{f}}\mathbf{1}/N_\mathrm{e}\\ \mathbf{X}_k^{\mathrm{f}} &= \left(\mathbf{E}_k^{\mathrm{f}} - \overline{\mathbf{x}}_k^{\mathrm{f}}\mathbf{1}^\top\right)/\sqrt{N_\mathrm{e}-1}\\ \mathbf{Y}_k^{\mathrm{f}} &= \left(H_k(\mathbf{E}_k^{\mathrm{f}}) - H_k(\overline{\mathbf{x}}_k^{\mathrm{f}})\mathbf{1}^\top\right)/\sqrt{N_\mathrm{e}-1}\end{aligned}$$

   - Computation of ensemble transform matrix

$$\mathbf{\Omega} = \left(\mathbf{I}_e + \mathbf{Y}_k^\top \mathbf{R}_k^{-1}\mathbf{Y}_k\right)^{-1}$$

   - Analysis estimate in ensemble space

$$\mathbf{w}_k^{\mathrm{a}} = \mathbf{\Omega}\left(\mathbf{Y}_k^{\mathrm{f}}\right)^\top \mathbf{R}_k^{-1}\left(\mathbf{y}_k - H_k(\overline{\mathbf{x}}_k^{\mathrm{f}})\right)$$

   - Generating the posterior ensemble

$$\mathbf{E}_k^{\mathrm{a}} = \overline{\mathbf{x}}_k^{\mathrm{f}}\mathbf{1}^\top + \mathbf{X}_k^{\mathrm{f}}\left(\mathbf{w}_k^{\mathrm{a}}\mathbf{1}^\top + \sqrt{N_\mathrm{e}-1}\mathbf{\Omega}^{1/2}\right)$$

   (b) Forecast
   - Forecast ensemble

$$\mathbf{E}_{k+1}^{\mathrm{f}} = M_{k+1}\left(\mathbf{E}_k^{\mathrm{a}}\right)$$

Note that the updated ensemble is centred on $\overline{\mathbf{x}}^{\mathrm{a}}$, since

$$\frac{1}{N_\mathrm{e}}\sum_{i=1}^{N_\mathrm{e}}\mathbf{x}_i^{\mathrm{a}} = \overline{\mathbf{x}}^{\mathrm{a}} + \frac{\sqrt{N_\mathrm{e}-1}}{N_\mathrm{e}}\mathbf{X}_{\mathrm{f}}\mathbf{T}\mathbf{1} = \overline{\mathbf{x}}^{\mathrm{a}} + \frac{\sqrt{N_\mathrm{e}-1}}{N_\mathrm{e}}\mathbf{X}_{\mathrm{f}}\mathbf{1} = \overline{\mathbf{x}}^{\mathrm{a}}, \tag{5.36}$$

where $\mathbf{1} = [1, \ldots, 1]^\top$. Not all EnSRF have a centred ensemble. It has been shown that a centred posterior ensemble often (if not always) yields better performance.

5.4 Localisation and inflation

The EnKF approach seems magical. We traded the extended Kalman filter for a considerably computationally cheaper filter meant to achieve similar performances if not better. But there is no free lunch and this comes with significant drawbacks to cope with. Fundamentally, one cannot hope to represent the full error covariance matrix of a complex

high-dimensional system with a few modes $N_e \ll N_x$, usually from a few dozen to a few hundred. This implies large ***sampling errors***, meaning that the error covariance matrix is only sampled by a limited number of modes. Even though the unstable degrees of freedom of dynamical systems that we wish to control with a filter are usually far fewer than the dimension of the system, they often still represent thousands, possibly millions of independent degrees of freedom. Nonetheless, forecasting an ensemble of such size is usually not affordable.

The consequence of this issue always is the divergence of the filter which often makes the EnKF a useless tool without the implementation of efficient fixes. In other words, in order to make the EnKF a viable algorithm, one needs to cope with the sampling errors.

Fortunately, there are clever tricks to overcome this major issue. They arte known as the ***localisation*** and the ***inflation***.

5.4.1 Localisation

For most geophysical systems distant observables are barely correlated. In other words, two distant parts of the system are almost independent at least for short time scales. We could exploit this and spatially ***localise*** the analysis, *i.e.* restrict its spatial scope. There are two main ways to achieve this goal.

The first one is called ***domain localisation***, or ***local analysis***. Instead of performing a global analysis valid at any location in the domain, we perform a local analysis to update the local state variables using local observations. Typically one would update a state variable at a location M by assimilating only the observations within a fixed range of M. If this range, known as the ***localisation length***, is too long, then the analysis becomes global as before, and may fail. On the other hand, if the localisation length is too small, some short to medium-range correlation might be neglected resulting in a possibly viable but a poor data assimilation system. One would then update all the state variables performing those local analyses. That may seem a formidable computational task but all these analyses can be carried out in parallel. Besides, since the number of local observations is a fraction of the total, the computation of the local gain is much faster than that of the global gain. All in all, such an approach is viable even on very large systems. Figure 5.4 is a schematic representation of a local update around M in the domain localisation approach where only the surrounding observations of a grid-cell to update are assimilated. One significant drawback of domain localisation is its inability to process non-local observations such as space-borne radiance measurements, since they involve state vector components across several local domains. On has to resort to drastic approximation to assimilate them.

The second approach called ***covariance localisation*** focuses on the forecast error covariance matrix. It is based on the remark that the forecast error covariance matrix $\mathbf{P}^f$ is of low rank, at most $N_e - 1$. If the empirical $\mathbf{P}^f$ is probably a good approximation of the true error covariance matrix at short distances, the insufficient rank will induce long-range correlations. These correlations are ***spurious***: they are likely to be unphysical and are not present in the true forecast error covariance matrix. A brilliant idea is to regularise $\mathbf{P}^f$ by smoothing out these long-range spurious correlations and increasing the rank of $\mathbf{P}^f$. To do so, one can regularise $\mathbf{P}^f$ by multiplying it point-wise with a short-range predefined correlation matrix $\boldsymbol{\rho} \in \mathbb{R}^{N_x}$. The point-wise multiplication is called a Schur product and

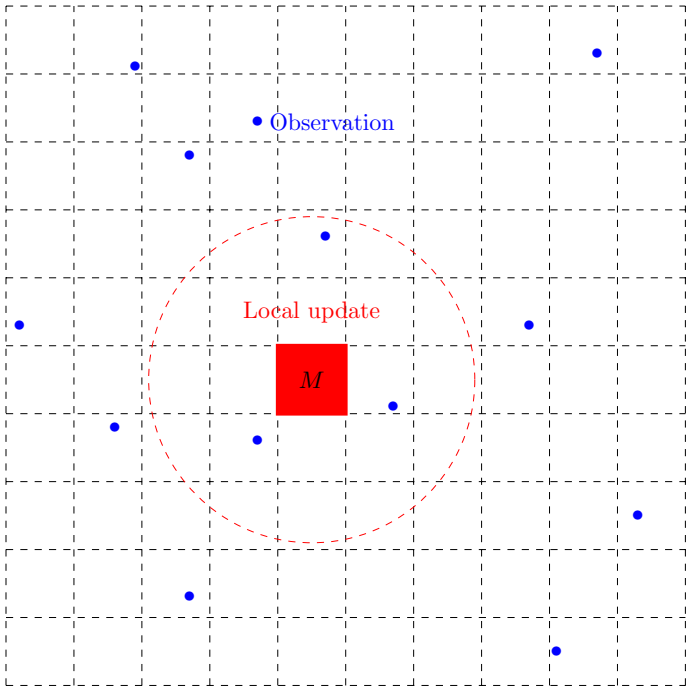


Figure 5.4: Schematic representation of the local update of the variable defined over the red grid-cell with 3 surrounding observations (marked by blue dots).

denoted with a $\circ$. It is defined by

$$\left[\mathbf{P}^{\mathrm{f}} \circ \boldsymbol{\rho}\right]_{ij} = \left[\mathbf{P}^{\mathrm{f}}\right]_{ij} [\boldsymbol{\rho}]_{ij} \,. \tag{5.37}$$

With reasonable conditions on ρ, one can mathematically ensure that $\mathbf{P}^{\mathrm{f}} \circ \boldsymbol{\rho}$ is full-rank definite-positive. As can be read from Eq. (5.37), the spurious correlations have been levelled off by $\boldsymbol{\rho}$, if $\boldsymbol{\rho}$ is short-ranged. This regularised $\mathbf{P}^{\mathrm{f}} \circ \boldsymbol{\rho}$ will be used in the EnKF analysis as well as in the generation of the posterior ensemble.

Of course, as a side-effect, it might also remove some physical and true long-distance correlations, thus inducing some physical imbalance in the analysis.

Covariance localisation and domain localisation seem quite different. Indeed they are mathematically distinct, though they are both based on two sides of the same paradigm. However, it can be shown that they should yield very similar results if the innovation at each time step is not too strong (Sakov and Bertino, 2011).

Figure 5.5 illustrates covariance localisation. We consider a one-dimensional state space of $N_{\mathrm{x}} = 200$ variables, and an error covariance matrix defined by

$$[\mathbf{P}]_{n,m} = e^{-\frac{|n-m|}{L}}, \tag{5.38}$$

where $n, m = 1, \ldots, N_{\mathrm{x}}$ and $L = 10$. An ensemble of $N_{\mathrm{e}} = 20$ members is generated from the exact Gaussian distribution of error covariance matrix $\mathbf{P}$. The empirical error covariance matrix is computed from this ensemble, and a Schur localisation is applied to

the empirical covariance matrix to form a regularised covariance matrix. For the sake of the demonstration, we take the localisation matrix to be the correlation matrix of the true covariance matrix.

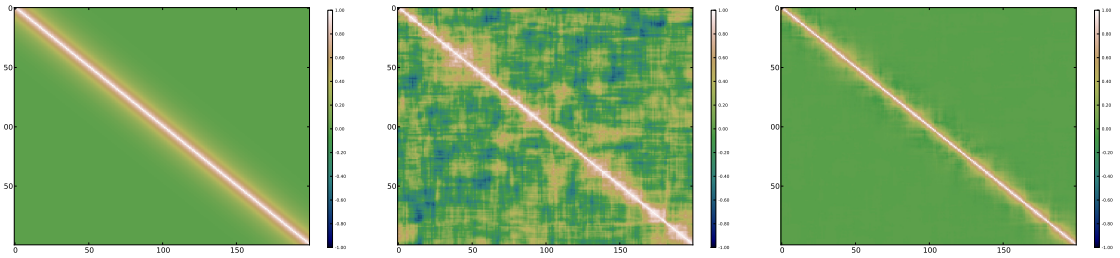


Figure 5.5: Covariance localisation. Left: the true covariance matrix. Middle: the empirical covariance matrix. Right: the regularised empirical covariance matrix using Schur localisation.

5.4.2 Inflation

Even when the analysis is made local, the error covariance matrices are still evaluated with a finite-size ensemble. This often leads to sampling errors. With a proper localisation scheme, they might be small. However small they may be, they will accumulate and they will carry over to the next cycles of the sequential EnKF scheme. As a consequence, there is always a risk that the filter may ultimately diverge. One way around is to inflate the error covariance matrix by an empirical factor λ^2 slightly greater than one. It can be applied before or after the analysis. For instance, after the analysis, inflating means

$$\mathbf{P}^{\mathrm{a}} \longrightarrow \lambda^2 \mathbf{P}^{\mathrm{a}}. \tag{5.39}$$

Another way to achieve this is to inflate the ensemble

$$\mathbf{x}_i^{\mathrm{a}} \longrightarrow \overline{\mathbf{x}}^{\mathrm{a}} + \lambda \left(\mathbf{x}_i^{\mathrm{a}} - \overline{\mathbf{x}}^{\mathrm{a}} \right). \tag{5.40}$$

This type of inflation is called ***multiplicative inflation***.

In a perfect model context, the multiplicative inflation is meant to compensate for sampling errors and the approximate assumption of Gaussianity (Bocquet, 2011). In that case, it helps cure an intrinsic source of error of the EnKF scheme.

Yet, inflation can also compensate for the underestimation of the errors due to the presence of model error, a source of error external to the EnKF scheme. In this context, additive type of inflation can also be used, either by

$$\mathbf{P}^{\mathrm{f}} \longrightarrow \mathbf{P}^{\mathrm{f}} + \mathbf{Q}, \tag{5.41}$$

of by adding noise to the ensemble members

$$\mathbf{x}_i^{\mathrm{f}} \longrightarrow \mathbf{x}_i^{\mathrm{f}} + \boldsymbol{\epsilon}_i \quad \text{with} \quad \mathrm{E}\left[\boldsymbol{\epsilon}_i \boldsymbol{\epsilon}_i^{\top} \right] = \mathbf{Q}. \tag{5.42}$$

5.5 Numerical tests on the Lorenz-96 model

The Lorenz-96 low-order model is a one-dimensional toy-model introduced by the famous meteorologist Edward Lorenz in 1996 (Lorenz and Emanuel, 1998). It represents a mid-latitude zonal circle of the global atmosphere. It has $N_\mathrm{x} = 40$ variables $\{x_n\}_{n=1,\ldots,N_\mathrm{x}}$. Its dynamics are given by the following set of ordinary differential equations,

$$\frac{\mathrm{d}x_n}{\mathrm{d}t} = (x_{n+1} - x_{n-2})x_{n-1} - x_n + F, \tag{5.43}$$

for $n = 1, \ldots, N_\mathrm{x}$. The domain on which is defined the 40 variables is circle-like, so that $x_{-1} \equiv x_{39}$, $x_0 \equiv x_{40}$, and $x_{41} \equiv x_1$. Hence it is assumed periodic. F is chosen to be 8. The term $(x_{n+1} - x_{n-2})x_{n-1}$ is a nonlinear convective term. It preserves the energy $\sum_{n=1}^{N_\mathrm{x}} x_n^2$. $-x_n$ is a dampening term while F is an energy injection/extraction term (depending on the sign of the variables). This makes the dynamics of this model chaotic. The model has 13 positive Lyapunov exponents and one is equal to zero. This implies that, out of the 40 degrees of freedom of this model, 14 (asymptotic) directions correspond to growing or neutral modes. In practice and for short-term forecast, around 13 directions (that change with time) in the 40 model state space make a small perturbation grows under the action of the model. The model is usually integrated using the 4th-order Runge-Kutta scheme using an integration time-step of $\delta t = 0.05$.

We shall apply the deterministic EnKF to this model. The time interval between observational update will be $\Delta t = 0.05$, meant to be representative of a data assimilation cycle of global meteorological models (about 6 hours). With such value for Δt, the data assimilation system is considered to be weakly nonlinear, leading to statistics of the errors weakly diverging from Gaussianity.

Figure 5.6 displays a trajectory of the model state. The model is characterised by about eight nonlinear waves that interact. As can be seen, it seems difficult to predict the behaviour of these waves except that they have the tendency to drift westward. These waves are meant to represent large-scale Rossby waves.

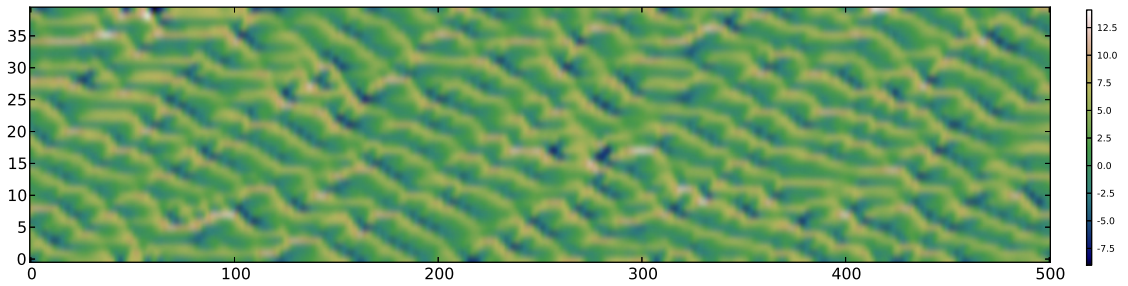


Figure 5.6: Trajectory of a state of the Lorenz-96 model. The Ox coordinate represents time in units of $\Delta t = 0.05$. The Oy coordinate represents the 40 state variables.

A ***twin experiment*** is conducted. The truth is represented by a free model run, meant to be tracked by the data assimilation system. The system is assumed to be fully observed ($N_\mathrm{y} = 40$) every Δt, so that $\mathbf{H}_k = \mathbf{I}_{40}$, with the observation error covariance matrix $\mathbf{R}_k = \mathbf{I}_{40}$. The related synthetic observations are generated from the truth, and perturbed

according to the same observation error prior. The performance of a scheme is measured by the temporal mean of a root mean square difference between a state estimate ($\mathbf{x}^{\mathrm{a}}$) and the truth ($\mathbf{x}^{\mathrm{t}}$). Typically, one averages over time the analysis root mean square error

$$\sqrt{\frac{1}{N_{\mathrm{x}}}\sum_{n=1}^{N_{\mathrm{e}}}(x_n^{\mathrm{a}}-x_n^{\mathrm{t}})^2}. \tag{5.44}$$

All data assimilation runs will extend over 10^5 cycles, after a burn-in period of 5×10^3 cycles, which guarantees satisfying convergence of the error statistics (due to ergodicity of the dynamics).

We vary the ensemble size from $N_{\mathrm{e}}=5$ to $N_{\mathrm{e}}=50$ and compare the performance in terms of the root mean square error of

- the deterministic EnKF without inflation and without localisation,
- the deterministic EnKF with optimally-tuned inflation and without localisation,
- the deterministic EnKF without inflation and with optimally-tuned localisation,
- and the deterministic EnKF with optimally-tuned inflation and with optimally-tuned localisation.

The average root mean square errors of the analyses over the long run are displayed in Fig. 5.7.

By optimally-tuned, it is meant that the parameters in the localisation and inflation are chosen in order to minimise the root mean square error. If one agrees that the application of the EnKF to this low-order model captures several of the difficulties of realistic data assimilation, it becomes clear from these numerical results that localisation and inflation are indeed mandatory ingredients of a satisfying implementation of the EnKF.

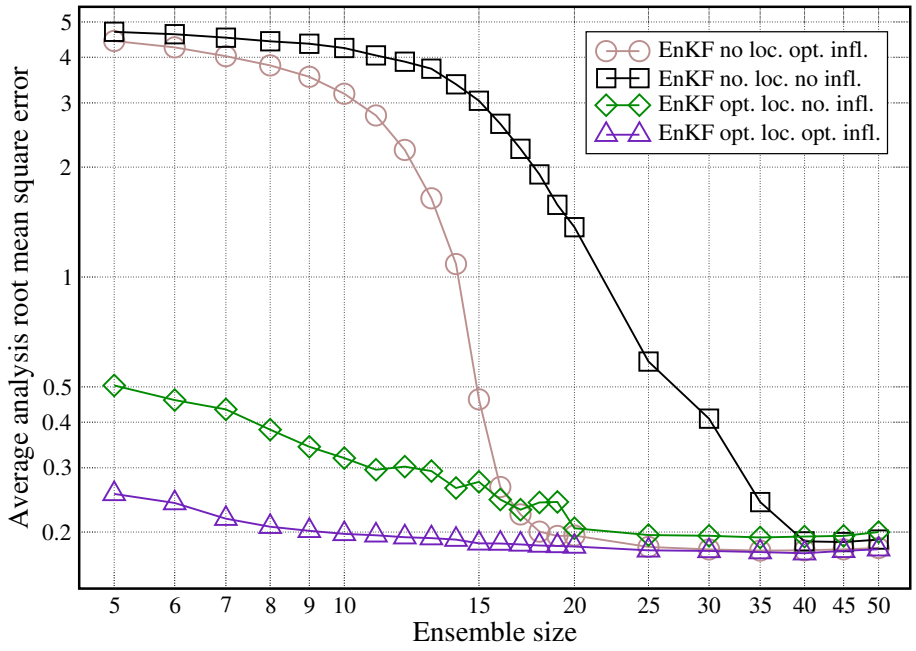


Figure 5.7: Average analysis root mean square error for a deterministic EnKF (ETKF) without localisation and without inflation (□); without localisation and with optimally-tuned inflation (◯); with optimally-tuned localisation and no inflation (◊); with optimally-tuned localisation and with optimally-tuned inflation (△).

Part III

Data assimilation, machine learning and dynamical systems

Preambule: Data assimilation, machine learning and dynamical systems

The goal of this chapter is to give a brief and very limited introduction to the realm of **machine** and **deep leaning**. This introduction, however, remains within the scope of recent developments of **data assimilation** techiques. Machine learning has found many new convincing applications over the past recent years, besides vision or natural language. The geosciences are among them. Even within the geosciences, there is a considerable range of potential applications of machine learning and deep learning; some of them have been evidenced recently (Reichstein et al., 2019). Here we will only learn some of the key ideas when using **artificial neural networks** (NNs) to extend the playground of data assimilation techniques.

Chapter 6

Links between data assimilation and deep learning

In this chapter, we will discover some connections between data assimilation and deep learning when applied to dynamical systems.

Our specific goal will be to not only learn the state of a physical system through its observation and a prior of this state but also its dynamics. This contrasts with traditional data assimilation where the model is usually assumed to be known. Hence, the method is qualifed as **data-driven** since *both the state and the model*, are meant to be discovered through the observations *only*.

Note that machine learning usually seeks to find a good statistical model to explain a set of data and potentially extrapolate to new data. This goal is very close to that of data assimilation. However, data assimilation, at least in the geosciences, exploits large and computationally challenging though known models. On the contrary, machine learning aims at building fast models. Note that both disciplines make use of a large amount of data.

6.1 Links between data assimilation and machine learning

6.1.1 The Bayesian approach to data assimilation (reminder)

Recall that the analysis performed in data assimilation can most of the time be seen as an inference using an approximation of Bayes' rule which has been discussed in Chapter 4 of the lecture notes of this course. Bayes' rule reads

$$p_{X|Y}(\mathbf{x}|\mathbf{y}) = p_{Y|X}(\mathbf{y}|\mathbf{x})\frac{p_X(\mathbf{x})}{p_Y(\mathbf{y})}, \tag{6.1}$$

where $p_{X|Y}$ is the conditional probability density function (pdf), i.e. the *ultimate pdf* in data assimilation and more generally inference; $p_{Y|X}$ is the **likelihood**, p_X is the **prior** and p_Y is the **evidence**, a constant as far as X is concerned.

This is a remarkably elegant and powerful formula. However, since it is also too general, it is often very difficult to implement and almost systematically calls for approximations.

In Chapter 4 of these lecture notes, we assumed Gaussian statistics for the observation likelihood

$$p_{Y|X}(\mathbf{y}|\mathbf{x}) = \frac{1}{(2\pi)^{N_y/2}|\mathbf{R}|^{1/2}} \exp\left(-\frac{1}{2}\|\mathbf{y} - H(\mathbf{x})\|^2_{\mathbf{R}^{-1}}\right), \tag{6.2}$$

where the Mahalanobis norm notation was used:

$$\|\mathbf{x}\|^2_{\mathbf{A}} = \mathbf{x}^\top \mathbf{A}\mathbf{x}. \tag{6.3}$$

We also assumed Gaussian statistics for the prior

$$p_X(\mathbf{x}) = \frac{1}{(2\pi)^{N_x/2}|\mathbf{B}|^{1/2}} \exp\left(-\frac{1}{2}\|\mathbf{x} - \mathbf{x}^{\mathrm{b}}\|^2_{\mathbf{B}^{-1}}\right). \tag{6.4}$$

Remarkably, as shown in Chapter 4, the conditional pdf can then be computed using Bayes' rule, which rigorously yields a Gaussian pdf. The mean and maximum of this pdf (called the maximum a posteriori) is the BLUE solution (see Chapter 1), and its associated error covariance matrix is that of BLUE. This result is actually obvious if one focuses on the state vector $\mathbf{x}$ in Bayes' equation, and substitutes the lilelihood and state prior with their Gaussian conterparts. The associated cost function is obtained as

$$J(\mathbf{x}) = -\ln\big(p_{X|Y}(\mathbf{x}|\mathbf{y})\big) = \frac{1}{2}\|\mathbf{y} - H(\mathbf{x})\|^2_{\mathbf{R}^{-1}} + \frac{1}{2}\|\mathbf{x} - \mathbf{x}^{\mathrm{b}}\|^2_{\mathbf{B}^{-1}} + \text{constants}. \tag{6.5}$$

This is none other than the 3D-Var cost function studied in Chapter 1, which is another route to BLUE.

6.1.2 Bayesian justification of the weak-constraint 4D-Var

Let us consider a data assimilation problem with time. We wish to apply Bayes' rule over a time window $[t_0, t_K]$, with batches of observations $\mathbf{y}_k$ at each time step t_k. Let us denote $\mathbf{x}_{0:K}$ the collection of all vector states $\mathbf{x}_0, \ldots, \mathbf{x}_K$ within this time window. Similarly $\mathbf{y}_{0:K} = \mathbf{y}_0, \ldots, \mathbf{y}_K$ for the collection of all observation vectors. Then the most general conditional pdf of interest if $p(\mathbf{x}_{0:K}|\mathbf{y}_{0:K})$, where we omitted the subscript of the pdf as it is unambiguous from the pdf argument. Applying Bayes' rule, we obtain:

$$p(\mathbf{x}_{0:K}|\mathbf{y}_{0:K}) \propto p(\mathbf{y}_{0:K}|\mathbf{x}_{0:K})p(\mathbf{x}_{0:K}), \tag{6.6}$$

where the evidence $p(\mathbf{y}_{0:K})$ can be (and has been) omitted since it does not depend on $\mathbf{x}_{0:K}$.

Now, we assume that the observation errors are Gaussian and uncorrelated in time, with error covariance matrices $\mathbf{R}_0, \ldots, \mathbf{R}_K$, so that:

$$p(\mathbf{y}_{0:K}|\mathbf{x}_{0:K}) = \prod_{k=0}^{K} p(\mathbf{y}_k|\mathbf{x}_k) \propto \exp\left(-\frac{1}{2}\sum_{k=0}^{K}\|\mathbf{y}_k - H_k(\mathbf{x}_k)\|^2_{\mathbf{R}_k^{-1}}\right). \tag{6.7}$$

Next, we assume that the prior pdf $p(\mathbf{x}_{0:K})$ is Markovian, i.e. the state $\mathbf{x}_k$ conditional on the previous state $\mathbf{x}_{k-1}$ does not depend on all other previous past states. This yields:

$$p(\mathbf{x}_{0:K}) = p(\mathbf{x}_0) \prod_{k=1}^{K} p(\mathbf{x}_k|\mathbf{x}_{0:k-1}) = p(\mathbf{x}_0) \prod_{k=1}^{K} p(\mathbf{x}_k|\mathbf{x}_{k-1}). \tag{6.8}$$

Furthermore, we assume Gaussian statistics for the model error which are uncorrelated in time, with zero bias and error covariance matrices $\mathbf{Q}_1, \ldots, \mathbf{Q}_K$ so that:

$$p(\mathbf{x}_{0:K}) \propto p(\mathbf{x}_0) \exp\left(-\frac{1}{2} \sum_{k=1}^{K} \| \mathbf{x}_k - M_k\left(\mathbf{x}_{k-1}\right) \|^2_{\mathbf{Q}_k^{-1}} \right). \tag{6.9}$$

Like in the static case, we can assemble the likelihood and prior pieces to obtain the cost function associated to to the conditional pdf $p(\mathbf{x}_{0:K}|\mathbf{y}_{0:K})$:

$$J(\mathbf{x}_{0:K}) = -\ln p(\mathbf{x}_{0:K}|\mathbf{y}_{0:K}) \tag{6.10}$$

$$= -\ln p(\mathbf{x}_0) + \frac{1}{2} \sum_{k=0}^{K} \|\mathbf{y}_k - H_k(\mathbf{x}_k)\|^2_{\mathbf{R}_k^{-1}} \tag{6.11}$$

$$+ \frac{1}{2} \sum_{k=1}^{K} \| \mathbf{x}_k - M_k\left(\mathbf{x}_{k-1}\right) \|^2_{\mathbf{Q}_k^{-1}} + \text{constants}. \tag{6.12}$$

Unsurprisingly, this is the cost function of weak-constraint 4D-Var (see Chapter 4). Indeed, the associated statistical assumptions explicitly assume that the model could be flawed.

With this type of weak-constraint 4D, one believes that the model can be corrected with some stochastic noise to be added to the state vector. Instead of considering a known model $\mathbf{x}_k = M_k(\mathbf{x}_{k-1})$, one could alternatively assume a parametric form of the model

$$\mathbf{x}_k = M_k(\boldsymbol{\omega}, \mathbf{x}_{k-1}), \tag{6.13}$$

that depends on unknow time-independent parameters $\boldsymbol{\omega} \in \mathbb{R}^{N_p}$. This implies that the model dynamics is autonomous, i.e. it does not depend on time.

Many choices can be contemplated for $\boldsymbol{\omega}$. They can be a few important physical parameters of a well understood geophysical model. But, $\boldsymbol{\omega}$ could also stand for a large set of parameters that has a deep structural impact on $M_k(\boldsymbol{\omega}, \mathbf{x}_{k-1})$. For instance, M_k could be a linear combination of a set of various functions, with $\boldsymbol{\omega}$ the coefficients attached to each function of the basis. This latter option is in the spirit of **machine learning**. This correspondance between data assimilation and machine learning has been put forward in e.g., Hsieh and Tang (1998); Abarbanel et al. (2018); Bocquet et al. (2019).

6.1.3 Bayesian analysis with model parameters

We can piggyback on the previous Bayesian analysis, but now adding the model parameter vector $\boldsymbol{\omega}$ to the Bayesian analysis:

$$p(\mathbf{x}_{0:K}, \boldsymbol{\omega}|\mathbf{y}_{0:K}) \propto p(\mathbf{y}_{0:K}|\mathbf{x}_{0:K}, \boldsymbol{\omega}) p(\mathbf{x}_{0:K}, \boldsymbol{\omega}) \propto p(\mathbf{y}_{0:K}|\mathbf{x}_{0:K}, \boldsymbol{\omega}) p(\mathbf{x}_{0:K}|\boldsymbol{\omega}) p(\boldsymbol{\omega}), \tag{6.14}$$

which requires to introduce a prior pdf $p(\boldsymbol{\omega})$ on the parameters. In the language of Bayesian statistics, this is called a **hierarchical** decomposition of the **conditional pdf**.

As a consequence, the cost function for the state and model parameters problem is

$$J(\mathbf{x}_{0:K}, \boldsymbol{\omega}) = -\ln\, p(\mathbf{x}_{0:K}, \boldsymbol{\omega}|\mathbf{y}_{0:K}) \tag{6.15}$$

$$= +\frac{1}{2}\sum_{k=0}^{K} \|\mathbf{y}_k - \mathbf{H}_k\mathbf{x}_k\|^2_{\mathbf{R}_k^{-1}} + \frac{1}{2}\sum_{k=1}^{K} \|\mathbf{x}_k - M_k\left(\boldsymbol{\omega}, \mathbf{x}_{k-1}\right)\|^2_{\mathbf{Q}_k^{-1}} \tag{6.16}$$

$$-\ln\, p(\mathbf{x}_0) - \ln\, p(\boldsymbol{\omega}) + \text{constants}. \tag{6.17}$$

This cost function is again similar to the weak-constraint 4D-var, but (i) $\boldsymbol{\omega}$ is now part of the control variables, and (ii) there is a prior/background term on $\boldsymbol{\omega}$ that may or may not play a role depending on the relative importance of the dataset.

We note that, to be effective, a data assimialtion analysis based on this cost function would require not only the gradient of the cost function with respect to the whole state trajectory, i.e. $\nabla_{\mathbf{x_{0:K}}} J$, but also the gradient of the cost function with respect to the model parameters, i.e. $\nabla_{\boldsymbol{\omega}} J$. This entails the construction of a tangent linear and adjoint of the model with respect to the model parameters, which could represent a significant hardship and will be discussed later on.

6.1.4 Machine learning limit

This (Bayesian) data assimilation standpoint on the problem of estimating the model (together with the state trajectory) is remarkable as it allows for noisy and partial observations of the physical system, as in traditional data assimilation. Classical and simple machine learning approaches of the problem would rather use a dataset which relies on a complete observation of the physical system with minimal noise, using a simple least-square **loss** function.

It turns out that this can be seen as a limiting case of the data assimilation problem previously discussed. Let us again consider the cost function $J(\mathbf{x}_{0:K}, \boldsymbol{\omega})$. Let us assume that the physical system is fully and directly observed, i.e. $\mathbf{H}_k \equiv \mathbf{I}$, and that the observation errors tend to zero, i.e. $\mathbf{R}_k \to \mathbf{0}$. Then the observation term in the cost function is completely frozen and imposes that $\mathbf{x}_k \simeq \mathbf{y}_k$, so that, in this limit, the cost function becomes

$$J(\boldsymbol{\omega}) = \frac{1}{2}\sum_{k=0}^{K} \|\mathbf{y}_k - M_k(\boldsymbol{\omega}, \mathbf{y}_{k-1})\|^2_{\mathbf{Q}_k^{-1}} - \ln\, p(\boldsymbol{\omega}) + \text{constants}. \tag{6.18}$$

This coincides with the tyical machine learning **loss function**, where we would try to fit the best model parametrised by $\boldsymbol{\omega}$ to the observation sequence over the time window. Again, solving the numerical problem would require the gradient of this loss function with respect to the parameters, i.e. $\nabla_{\boldsymbol{\omega}} J$, which is certainly computationally challenging with complex parametrised models, as developed for the goesiences.

6.2 Neural network surrogate model

Hence, we would like to build a representation of a parametrised model which would be:

- complex enough so that it can represent various (nonlinear) behaviours depending on the choice of the parameters (which could be plenty),
- simple enough so that computing the exact tangent linear and adjoint can be enforced in an efficient and automatic manner.

Artifical neural networks are the ideal mathematical tools that satisfy both properties.

6.2.1 What is a neural network?

An artificial neural netwrok (NN) is a function from a Cartesian space to another. Hence there is an input and an output space. A simple (feed-forward or sequential) NN is built as a succession of **layers** of **neurons**. Each layer is a stack of neurons. Each neuron in a layer holds a scalar value, and has connections to both the neurons of the previous layer and the neurons of the subsequent layer, which shows dependence between the neuron values. The value in a neuron is determined from the values of the neurons in the previous layer following the typical formula:

$$x_{k,i} = \sigma\left(\sum_{j=1}^{N} w_{i,j} x_{k-1,j} + b_i\right), \tag{6.19}$$

where: 1. $w_{i,j}$ is the scalar that weights the input of neuron j of the previous layer to neuron i of the current layer, 2. $x_{k,i}$ is the value hold by neuron i in the k-th layer, 3. b_i is a bias attached to this neuron, i.e. just a scalar, so that the transformation within the argument of σ is affine, 4. and, finally, σ is the **activation function**, usually the identity or a nonlinear function (*sigmoid*, *tanh*, a rectifier such as *relu*, etc.)

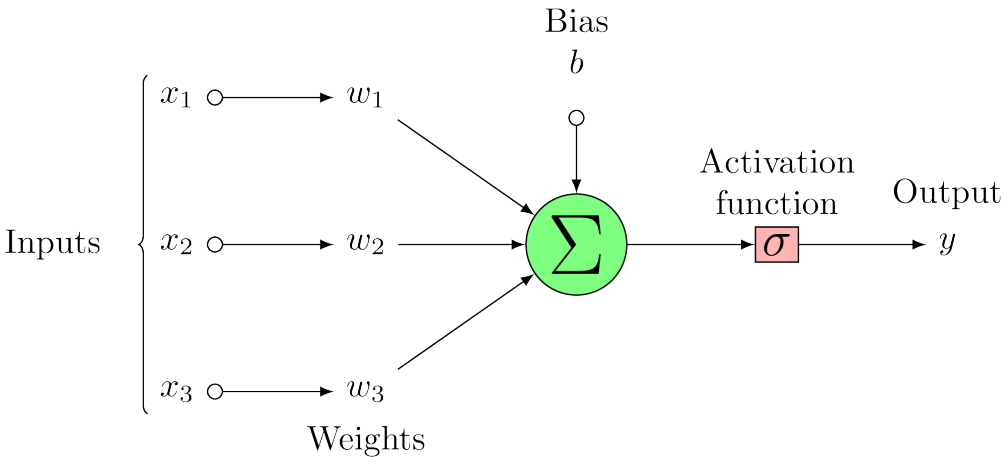


Figure 6.1: schematic of neuron and its computation.

Figure 6.1 shows a schematic of the operation of such neuron. Note that if σ is the identity, the layer acts on the values of the neurons as a linear operator. Hence the nonlinearity is essentially encoded in the activitation function, and its successive applications, layer after layer. Relying on a linear neural network is tantamount to *linear regression*, i.e. the estimation technique BLUE is based on.

A key strength of NNs lies in these activation functions that make NNs excellent tools for **nonlinear regression**. It was found very early that essential properties of the NNs do not depend on the specification of the activation function as long as they are not polynomial. However, quantitative results obviously depend on their choices.

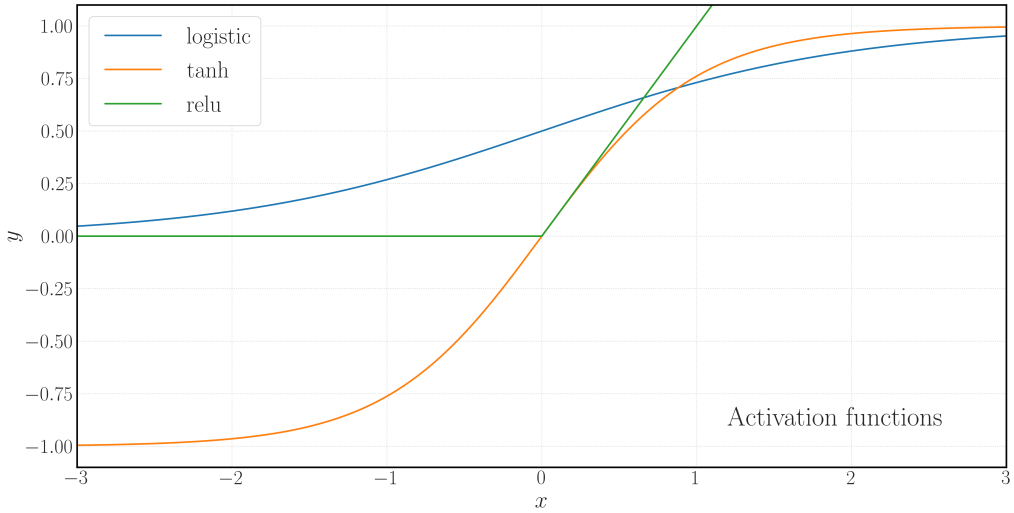


Figure 6.2: Classical activation functions.

Figure 6.2 shows examples of classical activation functions. The neurons are stacked into layers and the layers are ordered sequentially. The arrangment of the layers is the **architecture** of the neural network.

6.2.2 Universal approximation theorems

From a mathematical standpoint, there is a fundamental reason why NNs are valuable approximation tools. Indeed it can be shown that any sufficently regular function $\mathbf{y} = \mathcal{F}(\mathbf{x})$ can be approximated (to arbitrary precision) by a neural network with sufficiently large hidden layers. This is called the **universality approximation theorem** (Cybenko, 1989; Hornik et al., 1989; Barron, 1993), i.e. the NNs constitute a general enough set of functions in between two Cartesian spaces. Yet, the number of required neurons is exponential with the complexity of the function to be approximated. However, this can be significantly mitigated by increasing the number of layers, rather than the number of neurons per layer. This helps breaking the *curse of dimensionality* of the functional representation. This led to the so-called **deep learning**, i.e. using NN architectures with many layers. This type of results does not however tells us how, and how efficienlty the set of weights of such NN can be learned.

6.2.3 Dense neural networks

For instance, a NN could be of the basic form shown in Fig. 6.3

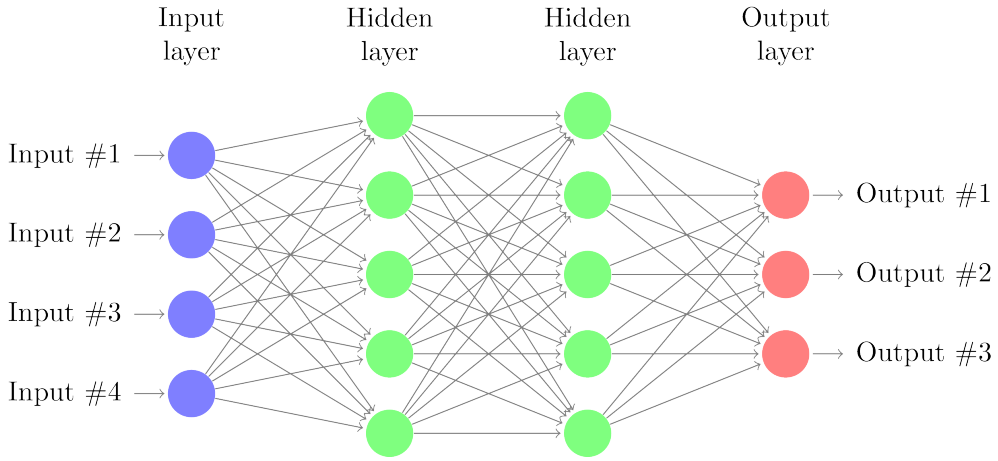


Figure 6.3: A typical dense neural network.

A layer with neurons connected to all neurons of the previous and subsequent layer are called **dense** layers. When the NN is composed of dense layers, it is unsurprisingly called a dense NN.

Obviously an architecture built with many dense layers may be numerically very demanding as a single pass through the network would require a lot of matrix-vector computations and would introduce a great number of weights.

6.2.4 Convolutional neural networks

For input and output spaces of high dimension, the systematic use of dense layers becomes prohibitive. One can exploit the fact that similar local patterns in the input space would produce similar outputs. For instance, an concrete object, say a hammer, in a picture should yield the "hammer" tag, irrespective of its position in the picture.

A convolutional layer exploits this *invariance* idea. It sweeps the input space and applies a fixed number of filters specified by weights whose span is a local patch of variables in the input space. This has the potential to represent coherent and regular structures defined in the input space.

Figures 6.4 and 6.5 show a schematic of the computation of two different neuron values in the output of the same convolutional layer with one filter. Mathematically speaking, these operations are called **convolutions**.

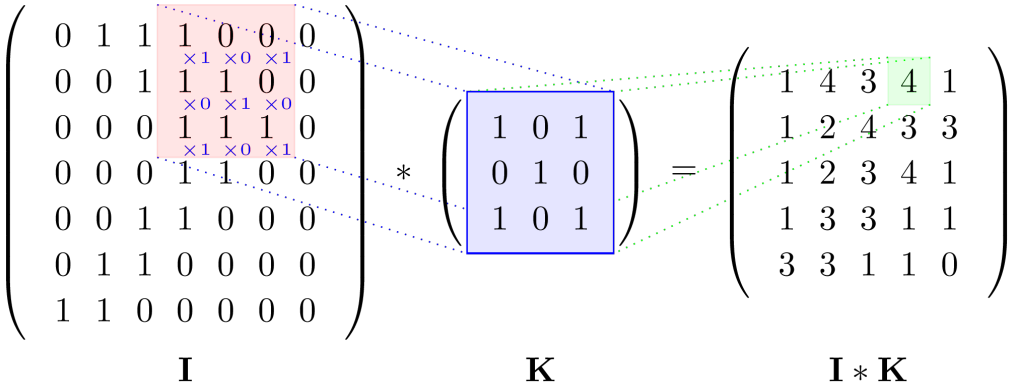


Figure 6.4: Convolution operating on a tensor.

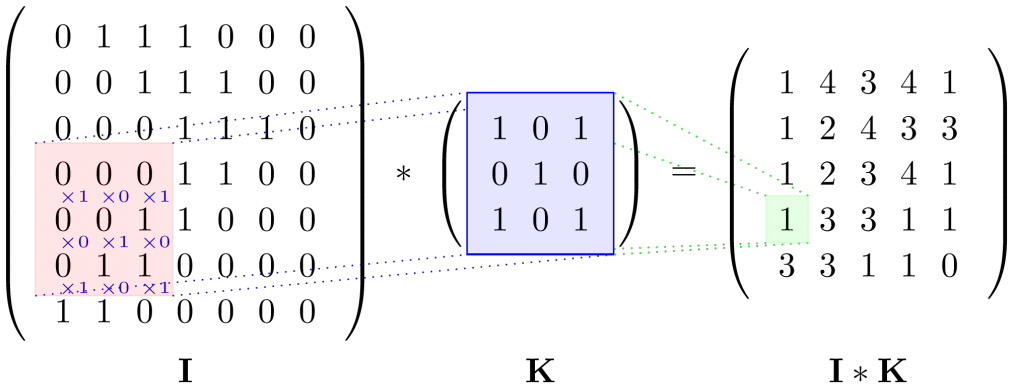


Figure 6.5: Convolution operating on a tensor.

Then one introduces as many kernels as desired **features**, where a ferature is a trait, or property, that one wants to extract from the input. This leads to NNs of the form explamplified by Fig. 6.6:

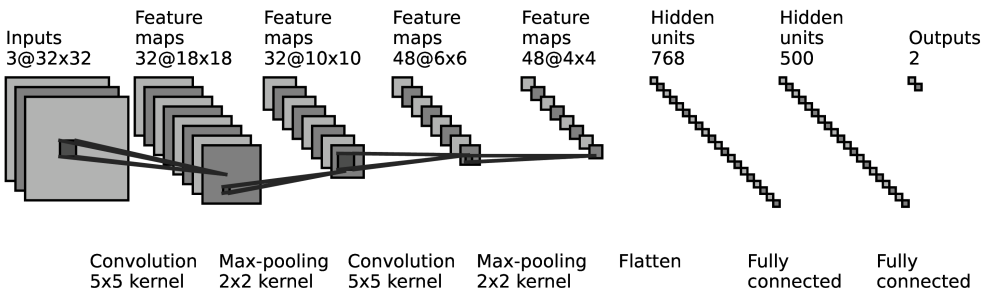


Figure 6.6: A typical deep convolutional neural network, or convnet.

Such NNs are called **convolutional neural networks**, or CNNs.

6.2.5 Loss function, training and backpropagation

Imagine that there is a relationship between states from the input space ($\mathbb{R}^{N_x}$) to the output space ($\mathbb{R}^{N_y}$), for instance $\mathbf{y} = \mathcal{F}(\mathbf{x})$, but where $\mathcal{F}$ is not known to us. However, we can sample this relationship through observations and obtain many couples of the form $(\mathbf{x}_i, \mathbf{y}_i)$, say N_s samples. Our objective is to learn this relationship using a neural network that would approximate $\mathcal{F}$.

Assuming $\mathcal{F}$ is a linear function, i.e. a matrix from the input space to the output space, then the solution is known if there are enough independent sample points: one would use the least square technique and minimise the quadratic cost function

$$\mathcal{L}(\mathbf{F}) = \sum_{i=1}^{N_s} \|\mathbf{y}_i - \mathbf{F}\mathbf{x}_i\|^2. \tag{6.20}$$

Then the solution (linear regression estimator) is

$$\mathbf{F}^\star = \left(\sum_{i=1}^{N_s} \mathbf{y}_i \mathbf{x}_i^\top\right) \left(\sum_{i=1}^{N_s} \mathbf{x}_i \mathbf{x}_i^\top\right)^{-1}, \tag{6.21}$$

where the matrix inverse should be replaced by the Moore–Penrose pseudo-inverse if the matrix is not invertible (e.g., because the number of samples N_s is insufficient).

The generalisation to much more complex, nonlinear approximations, consists in looking for a neural network $\mathbf{x} \mapsto \mathcal{N}(\mathbf{x})$ which minimises

$$\mathcal{L}(\mathcal{N}) = \sum_{i=1}^{N_s} \|\mathbf{y}_i - \mathcal{N}(\mathbf{x}_i)\|^2. \tag{6.22}$$

This is the **loss function** of machine learning, which is the counterpart to the cost function of data assimilation. Learning $\mathcal{N}$ this way is called **supervised learning** because samples with *both* inputs and outputs are provided. Also note that minimising the loss function has been called **training** in machine learning (analysis in data assimilation) since one wishes to train the NN, i.e. learn its weights and biases.

However, as we learned on several occasions in this introduction to data assimilation, the only efficient way to minimise this loss function is through numerical nonlinear optimisation, using an iterative solver that efficiently makes use of the gradient of $\mathcal{L}$ with respect to $\mathcal{N}$, specifically the weights and biases of the NN. We also learned that this entails to compute the tangent linear and its adjoint of the operator $\mathcal{N}$. Making use of the adjoint of the tangent linear to compute the gradient with respect to the weights and biases has been called **backpropagation** in machine learning, a renaming of **adjoint modelling**.

This is where a hidden technical revolution took place. Building on the efficacy and force of Google, FaceBook, Apache, etc., tools were developed by these companies to efficiently compute the tangent linear and adjoint of NNs, such as TensorFlow, PyTorch, Jax or the Julia language. Moreover these tools were developed to efficiently leverage GPUs, TPUs and parallel computing, while the user would still use Python commands. These tools make automatic differentiation for deep learning problems accessible to the many.

6.2.6 Regularisation and validation

With deep learning, NNs may have millions of weigths and biases. This may require a huge dataset, and still may not guarantee the absence of over-fitting to the data. Hence, as we learned in this course, a regularisation is needed, typically in the form of a background on the weights and biases. This is actually one of the technique used in deep learning (simple Tikhonov regularisation). Another, which is equivalent to **cross validation**, consists in sparing a fraction of the dataset for, at each iteration of the descent, checking the progess on independent data. This is the **validation dataset**. When the validation score (computed on the validation dataset) saturates or rebounces, and even though the training score continues to decrease, it is worthy to stop the iterative minimisation in order to avoid overfitting. This will be discussed a bit later one, and much more in part 2 of this lecture. Dataset splitting is illustrated in Fig. 6.7:

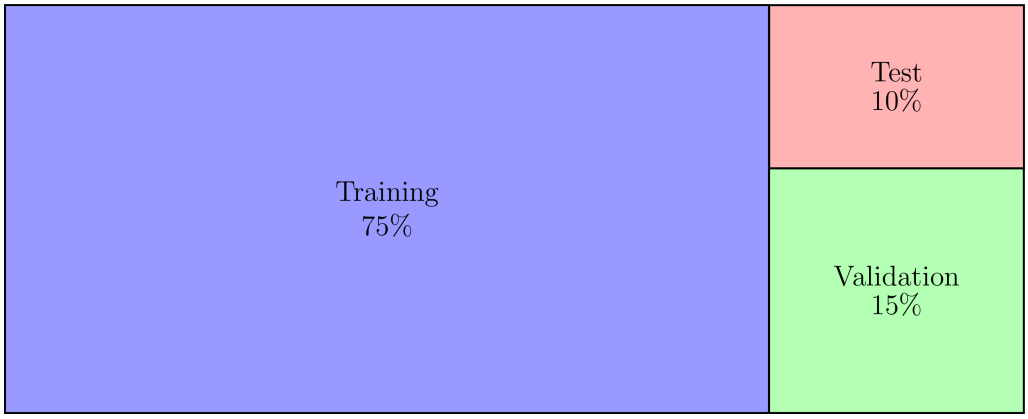


Figure 6.7: Dataset splitting.

Figure 6.7 gives an example of a dataset splitting into a training, validation and testing sub-datasets. Practitionners usually incude most of the data in the training subset, but the ratio 75%/15%/10% is not a golden rule.

6.3 Coding a neural network and its training with TensorFlow

In the following we give a minimal example on how to create and use a simple NN using the google suite TensorFlow 2.x with the Python API. We also rely on Keras, which is now part of TensorFlow, and considerably facilitates the creation and use of NNs (Chollet, 2021).

There are three ways to code a NN model in TensorFlow 2.x, the **sequential** model, the **functional model**, and the **subclassing** model. Here, we focus on the sequential approach which is the simplest one and is designed for simple architectures with plain staked layers.

In the following, we closely follow the TensorFlow documentation `https://www.tensorflow.org/guide/keras/sequential_model`.

6.3.1 Importing all modules

[3]:
```
import numpy as np
import tensorflow as tf
from tqdm.auto import tqdm

import utils

utils.set_style()
seeds = [3, 31,
314, 3141, 31415]
```

6.3.2 Defining the neural network model

Let us define a sequential model with 3 layers. This NN is defined incrementally using the `add()` method, layer by layer. The four (Keras) layers are chained down, in sequential order. Layer1 is the first layer, layer2 and layer3 are the hidden layers and layer4 is the ouput layer. Keep in mind that, as we explained earlier, this NN model is nothing more than a function from the input space to the output space. One needs to specify the size of the input vector and the size of the rest follows.

[4]:
```
Nx = Ny = 3
initialiser = tf.keras.initializers.GlorotNormal(seed=seeds.pop(0))
model = tf.keras.Sequential()
model.add(tf.keras.layers.InputLayer(input_shape=(Nx,)))
model.add(tf.keras.layers.Dense(5, activation='relu',␣
 ↪kernel_initializer=initialiser, name='layer1'))
model.add(tf.keras.layers.Dense(10, activation='relu',␣
 ↪kernel_initializer=initialiser, name='layer2'))
model.add(tf.keras.layers.Dense(5, activation='relu',␣
 ↪kernel_initializer=initialiser, name='layer3'))
model.add(tf.keras.layers.Dense(Ny, kernel_initializer=initialiser,␣
 ↪name='layer4'))
```

We can have access to a summary of the model, i.e. a brief look at its architecture:

[5]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 layer1 (Dense)              (None, 5)                 20

 layer2 (Dense)              (None, 10)                60

 layer3 (Dense)              (None, 5)                 55
```

```
 layer4 (Dense)              (None, 3)                 18

=================================================================
Total params: 153
Trainable params: 153
Non-trainable params: 0
_________________________________________________________________
```

which tells us about the layers, their format, and the number of weights and biases in the full network. We can have access to the internal parameters of the NN. For instance, let us display the weights of the second layer:

[6]:
```
print('weights of hidden layer', model.layers[1].weights)
```

```
weights of hidden layer [<tf.Variable 'layer2/kernel:0' shape=(5, 10)
dtype=float32, numpy=
array([[-0.27878687,  0.6957586 , -0.39197323,  0.25236636, -0.1739648 ,
        -0.20580482,  0.17030655,  0.26538625, -0.02842008, -0.01333474],
       [-0.47075042, -0.01118908, -0.64544606, -0.22949693,  0.2183459 ,
         0.14189577,  0.039797  , -0.09955318,  0.05919757, -0.53678584],
       [ 0.00384476, -0.5295028 , -0.57023287,  0.49235034,  0.5225375 ,
         0.01377615,  0.75166893,  0.54394966, -0.01680202,  0.3406608 ],
       [ 0.39751792,  0.2899968 ,  0.43232402, -0.33761302, -0.01084888,
        -0.08306244, -0.6389947 , -0.02747615, -0.69207895, -0.58695346],
       [ 0.44832727,  0.1439877 , -0.01968908,  0.1745272 , -0.19021218,
         0.38763624, -0.18479653, -0.4940335 ,  0.22560808,  0.45554948]],
      dtype=float32)>, <tf.Variable 'layer2/bias:0' shape=(10,)␣
↪dtype=float32,
numpy=array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0.], dtype=float32)>]
```

They are implicitely initialised when creating the NN. Finally, let us use the NN as a function and apply it to a couple of random N_x-vectors:

[7]:
```
rng = np.random.default_rng(seed=seeds.pop(0))
x = rng.normal(loc=0, scale=1, size=(2, Nx))
print('input', x)
y = model(x)
print('output', y)
```

```
input [[-0.39530129  0.26391489  0.60712827]
 [-0.9721598   0.76766425  0.25505812]]
output tf.Tensor(
[[ 0.0080108  -0.05308522 -0.05787446]
 [-0.11240995  0.27459753 -0.16657892]], shape=(2, 3), dtype=float32)

2023-02-13 17:09:18.878082: I
tensorflow/compiler/xla/stream_executor/cuda/cuda_blas.cc:630]␣
↪TensorFloat-32
```

```
will be used for the matrix multiplication. This will only be logged once.
```

6.3.3 The true model dynamics: 3-variable Lorenz model

Now that we have defined our NN, let us define the true function. We choose it to be the *tendency* of the chaotic Lorenz 1963 model, i.e. the right-hand side of its ordinary differential system, or from a more physical standpoint, its velocity

$$\begin{cases} \frac{\mathrm{d}x}{\mathrm{d}t} &= \sigma(y-x) \\ \frac{\mathrm{d}y}{\mathrm{d}t} &= \rho x - y - xz \\ \frac{\mathrm{d}z}{\mathrm{d}t} &= xy - \beta z, \end{cases} \tag{6.23}$$

where $\sigma = 10$, $\beta = 8/3$, and $\rho = 28$. The map which, given the position, returns the velocity is defined in python as

[8]:
```
sigma = 10
beta = 8/3
rho = 28
def F(x):
    y = np.empty(x.shape)
    y[..., 0] = sigma*(x[..., 1]-x[..., 0])
    y[..., 1] = rho*x[..., 0]-x[..., 1]-x[..., 0]*x[..., 2]
    y[..., 2] = x[..., 0]*x[..., 1]-beta*x[..., 2]
    return y
```

The **Lorenz 1963** model has been introduced by the acclaimed meteorologist **Edward Lorenz** as the examplar of the emergence of chaos in meteorology (Lorenz, 1963) Its attractor has the famous nice butterfly shape as shown in Fig. 6.8

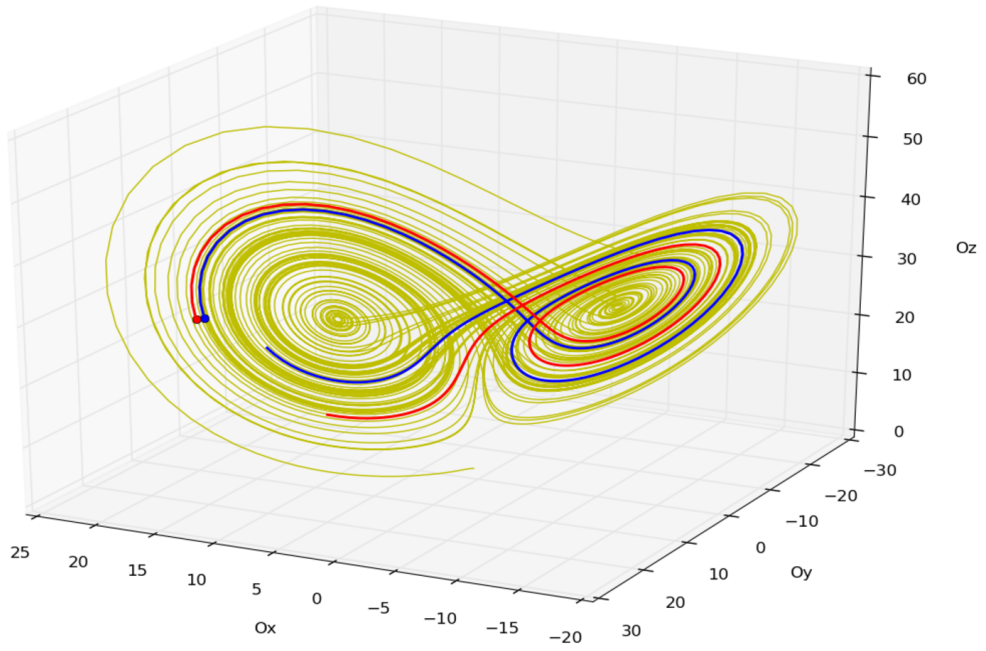


Figure 6.8: Two departing trajectories of the Lorenz 1963 model that shapes the attractor of the mnodel and its two wings.

6.3.4 Generating the training dataset

We want to learn the optimal set of weights and biases for the NN to fit the tendencies and hence learn the dynamics of the model from observing one of his long, possibly noisy trajectory. $N_s = 2000$ samples (or snapshots) are collected from a model trajectory. Random errors are added to these observations. These samples are input/output couples $(x_i, y_i)_{1 \leq i \leq N_s}$, with a time separation of $\Delta_t = 0.1$ model time unit (MTU).

Generating such trajectory of the L63 model, then observing regularly spaced snapshots of this trajectory, with a normal i.i.d. observational noise $\boldsymbol{\varepsilon} \sim n(\mathbf{0}, r^2\mathbf{I}_3)$, can be achieved through the following basic code:

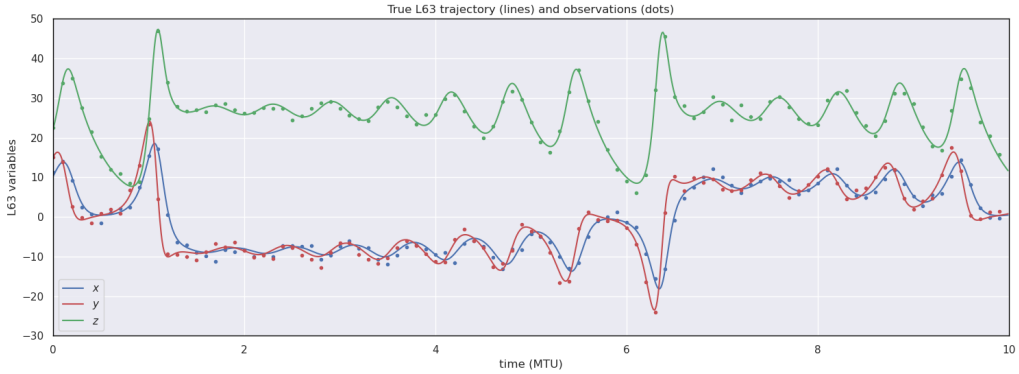
[9]:
```
# Set random seed
rng = np.random.default_rng(seed=seeds.pop(0))

# Key parameters
Ns = 2000      # Number of samples
Nt_shift = 10 # Number of integration time steps between samples
dt = 0.01      # Integration time step for the Euler explicit method
Dt = 10        # Forecast lead time in units of dt
r = 1          # Observation stddev

# Spinup run length and data generating run length
Nt_spinup = 5000
```

```
Nt_truth = Ns*Nt_shift+Dt

# Spinup L63 model run using the simplest explicit Euler integration␣
 ↪method
x = rng.uniform(low=-10, high=10, size=Nx)
with tqdm(total=Nt_spinup, desc='spinup') as progress:
    for i in range(Nt_spinup):
        x += dt*F(x)
        progress.set_postfix_str(x, refresh=False)
        progress.update()

# Generate a noiseless trajectory of the L63 model
# using the explicit first-order Euler integration method
x_truth = np.empty((Nt_truth, Nx))
with tqdm(total=Nt_truth, desc='L63 trajectory') as progress:
    for i in range(Nt_truth):
        x += dt*F(x)
        x_truth[i] = x
        progress.set_postfix_str(x, refresh=False)
        progress.update()

# Generate the data couples spaced by Dt (x_raw[i], y_raw[i])_{i=1,..,Ns}
x_perturb = x_truth + rng.normal(loc=0, scale=r, size=(Nt_truth, Nx))
x_raw = x_perturb[:Nt_truth-Dt:Nt_shift]
y_raw = x_perturb[Dt:Nt_truth:Nt_shift]

# Split into training and validation datasets,
# with 10% of the data being reserved for validation
index_train = np.array([i for i in range(Ns) if i%10])
index_valid = np.array([i for i in range(Ns) if not i%10])
x_train = x_raw[index_train]
y_train = y_raw[index_train]
x_valid = x_raw[index_valid]
y_valid = y_raw[index_valid]
```

```
spinup:   0%|          | 0/5000 [00:00<?, ?it/s]
```

```
L63 trajectory:   0%|          | 0/20010 [00:00<?, ?it/s]
```

[10]:
```
utils.plot_l63_traj_truth_obs(
    x_truth=x_truth,
    x_raw=x_raw,
    t_plot=10,
    dt=dt,
    Nt_shift=Nt_shift,
    linewidth=18,
)
```



6.3.5 The loss function

In order to learn some optimal weights and biases of the NN, $\mathcal{N}(\boldsymbol{\omega}, \cdot)$, we would like to define then minimise the loss function (mean square error or **mse**)

$$\mathcal{L}(\boldsymbol{\omega}) = \frac{1}{N_s} \sum_{i=1}^{N_s} \|\mathbf{y}_i - \mathcal{N}(\boldsymbol{\omega}, \mathbf{x}_i)\|^2 . \tag{6.24}$$

To implement this optimisation program, we need to set the parameters of the model training, such as the choice of the minimisation method and the related hyperparameters. To do so, the model must be **compiled**:

[11]:
```
opt = tf.keras.optimizers.Adam(learning_rate=5e-3)
model.compile(optimizer=opt, loss="mse", metrics=["mae"])
```

where we choose the **Adam** minimisation method (a **stochastic gradient** method), the mean square error (**mse**) loss, and choose for the validation a different metric, the mean absolute error (**mae**):

$$\mathcal{V}(\mathcal{N}) = \frac{1}{N_s} \sum_{i=1}^{N_s} \sum_{n=1}^{N_x} |[\mathbf{y}_i]_n - [\mathcal{N}(\mathbf{x}_i)]_n| . \tag{6.25}$$

Now, we can run the training using the Keras `fit()` method. Besides the input and output vectors, we can specifiy the number of iterations of the minimisation, `num_epochs`, which is the number of **epochs**. To do so, we can also specify the `batch_size` which tells how many samples are samples are used at once in the miminisation, and we also provide the validation dataset for cross-validation.

6.3.6 The training of the NN surrogate model

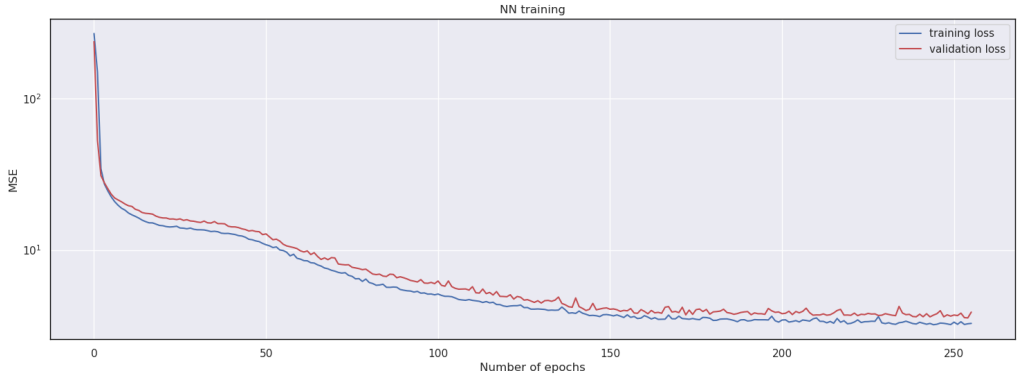
[12]:
```
tf.keras.utils.set_random_seed(seeds.pop(0))
num_epochs = 256
callback = [ utils.tqdm_callback(num_epochs, 'NN training') ]
history = model.fit(x_train, y_train,
```

```
          epochs=num_epochs,
          batch_size=64,
          validation_data=(x_valid, y_valid),
          verbose=0,
          callbacks=callback)
```

```
NN training:   0%|          | 0/256 [00:00<?, ?it/s]
```

6.3.7 Plotting the training and validation loss as a function of the epoch

[13]:
```
utils.plot_learning_curve(
    history.history['loss'],
    history.history['val_loss'],
    title='NN training',
    linewidth=18,
)
```



Testing the surrogate model $\mathcal{N}(\omega, \cdot)$ will be described in the second part of this lecture. However, for now, we can observe that after the initial drops of the loss function, say before 50 epochs, the validation loss remains close to the training loss. This suggests that the surrogate model is not prone to overfitting, and that it may generalise well.

Chapter 7

Application to the dynamics of a low-order chaotic model

In this chapter, we will apply standard machine learning methods to learn the dynamics of the Lorenz 1996 model.

7.1 The Lorenz 1996 model

The Lorenz 1996 (L96, Lorenz and Emanuel 1998) is a low-order chaotic model commonly used in data assimilation to asses the performance of new algorithms. It represents the evolution of some unspecified scalar meteorological quantity (perhaps vorticity or temperature) over a latitude circle.

The model **dynamics** is driven by the following set of ordinary differential equations (ODEs):

$$\forall n \in [1, N_x], \quad \frac{\mathrm{d}x_n}{\mathrm{d}t} = (x_{n+1} - x_{n-2})x_{n-1} - x_n + F, \tag{7.1}$$

where the indices are periodic: $x_{-1} = x_{N_x-1}$, $x_0 = x_{N_x}$, and $x_1 = x_{N_x+1}$, and where the system size N_x can take arbitrary values.

In the standard configuration, $N_x = 40$ and the forcing coefficient is $F = 8$. The ODEs are integrated using a fourth-order Runge-Kutta scheme with a time step of 0.05 model time unit (MTU). The resulting dynamics is **chaotic** with a doubling time of errors around 0.42 MTU (the corresponding Lyapunov is hence 0.61 MTU). For comparison, 0.05 MTU represent six hours of real time and correspond to an average autocorrelation around 0.967. Finally, the model variability (spatial average of the time standard deviation per variable) is 3.64.

7.2 The true model dynamics

In this series of experiments, we will try to emulate the dynamics of the L96 model using artificial neural networks (NN).

1. We start by running the **true model** to build a training dataset,
2. We build and **train a NN** using this dataset,
3. We evaluate the **forecast skill** of the surrogate model (the NN).

7.2.1 Importing all modules

[2]:
```
import numpy as np
import tensorflow as tf
from tqdm.auto import tqdm, trange

import utils

utils.set_style()
seeds = [3, 31, 314, 3141, 31415, 314159, 3141592, 31415926]
```

7.2.2 Defining the neural network model

In the following cell, we define the true Lorenz 1996 model using standard values: - the number of variables N_x is set to `Nx=40`; - the forcing coefficient F is set to `F=8`; - the integration time step is set to `dt=0.05`.

[3]:
```
# create model
true_model = utils.Lorenz1996Model(Nx=40, dt=0.05, F=8)

# save some statistics about the model
true_model.model_var = 3.64
true_model.doubling_time = 0.42
true_model.lyap_time = 0.61
```

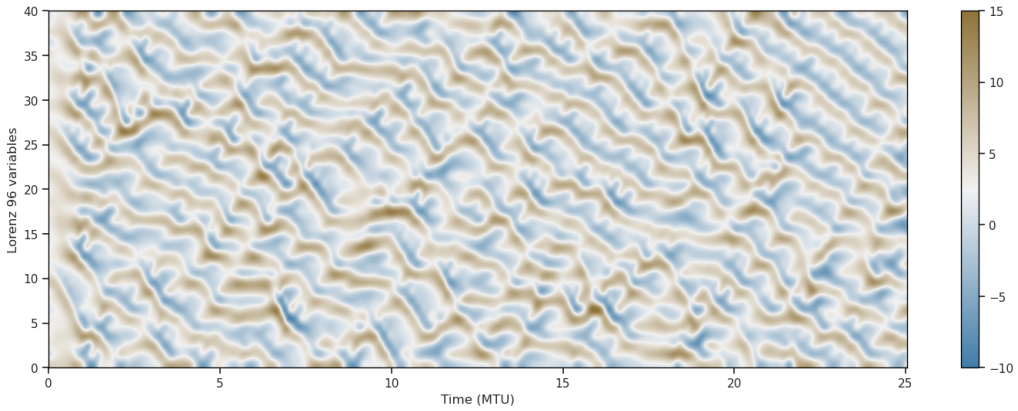
7.2.3 Short model integration

In the following cells, we perform a rather short model integration, in order to illustrate the model dynamics. The initial condition is a random field.

[4]:
```
# define rng
rng = np.random.default_rng(seed=seeds.pop(0))

# allocate memory
Nt_plot = 500
xt_plot = np.zeros((Nt_plot+1, true_model.Nx))

# initialisation and integrate
xt_plot[0] = rng.normal(loc=3, scale=1, size=true_model.Nx)
for t in trange(Nt_plot, desc='model integration'):
    xt_plot[t+1] = true_model.forward(xt_plot[t])
```

```
model integration:   0%|          | 0/500 [00:00<?, ?it/s]
```

[5]:
```
utils.plot_l96_traj(
    xt_plot,
    true_model,
    linewidth=18,
)
```



We see first a spin-up period of about 1 MTU, where the initial condition is progressively forgotten and the trajectory progressively gets back to the model attractor. After this spin-up period, the dynamics is characterised by waves moving slowly towards the east (i.e. decreasing variable index).

7.3 Prepare the dataset

7.3.1 A long model integration for the training data

We now use a true model trajectory to make the **training dataset**. This trajectory starts from a random field (different than for the plotting trajectory) and we discard the first 100 time steps to get rid of the spin-up process.

[6]:
```
# define rng
rng = np.random.default_rng(seed=seeds.pop(0))

# allocate memory
Nt_train = 10000
Nt_spinup = 100
xt_train = np.zeros((Nt_train+1, true_model.Nx))

# initialisation and spin-up
xt_train[0] = rng.normal(loc=3, scale=1, size=true_model.Nx)
for t in trange(Nt_spinup, desc='spin-up integration'):
```

```
    xt_train[0] = true_model.forward(xt_train[0])

# model integration
for t in trange(Nt_train, desc='model integration (train)'):
    xt_train[t+1] = true_model.forward(xt_train[t])
```

```
spin-up integration:   0%|          | 0/100 [00:00<?, ?it/s]

model integration (train):   0%|          | 0/10000 [00:00<?, ?it/s]
```

7.3.2 Preprocess the training data

The training dataset is made of input/output pairs, where the input is the state at a given time, and the output is the state at the following time.

[7]:
```
# make input/output pairs: input = xt[t], output = xt[t+1]
x_train_raw = xt_train[:-1]
y_train_raw = xt_train[1:]
```

The training dataset is then **normalised** and **split** into training and validation data. For our experiments, we keep one tenth of the data for validation.

[8]:
```
# normalise the training data using numpy's broadcasting rules
x_mean = x_train_raw.mean(axis=0)
y_mean = y_train_raw.mean(axis=0)
x_std = x_train_raw.std(axis=0)
y_std = y_train_raw.std(axis=0)
def normalise_x(x):
    return (x - x_mean)/x_std
def normalise_y(y):
    return (y - y_mean)/y_std
def denormalise_x(x_norm):
    return x_norm*x_std + x_mean
def denormalise_y(y_norm):
    return y_norm*y_std + y_mean
x_train_raw_norm = normalise_x(x_train_raw)
y_train_raw_norm = normalise_y(y_train_raw)

# split into training / validation
index_train = np.array([i for i in range(Nt_train) if i%10])
index_valid = np.array([i for i in range(Nt_train) if not i%10])
x_train_norm = x_train_raw_norm[index_train]
y_train_norm = y_train_raw_norm[index_train]
x_valid_norm = x_train_raw_norm[index_valid]
y_valid_norm = y_train_raw_norm[index_valid]
```

7.3.3 A shorter model integration for the test data

We repeat the same process to make the **test dataset**. In this case, the trajectory starts from another random field (and we still get rid of the spin-up process) and can be somewhat shorter, but the normalisation must be the same as for the training dataset.

[9]:
```
# define rng
rng = np.random.default_rng(seed=seeds.pop(0))

# allocate memory
Nt_test = 1000
Nt_spinup = 100
xt_test = np.zeros((Nt_test+1, true_model.Nx))

# initialisation and spin-up
xt_test[0] = rng.normal(loc=3, scale=1, size=true_model.Nx)
for t in trange(Nt_spinup, desc='spin-up integration'):
    xt_test[0] = true_model.forward(xt_test[0])

# model integration
for t in trange(Nt_test, desc='model integration (test)'):
    xt_test[t+1] = true_model.forward(xt_test[t])

# make input/output pairs: input = xt[t], output = xt[t+1]
x_test = xt_test[:-1]
y_test = xt_test[1:]

# normalise the test data using numpy's broadcasting rules
x_test_norm = normalise_x(x_test)
y_test_norm = normalise_y(y_test)
```

```
spin-up integration:   0%|          | 0/100 [00:00<?, ?it/s]
```

```
model integration (test):   0%|          | 0/1000 [00:00<?, ?it/s]
```

7.3.4 An ensemble model integration for the forecast skill data

In order to assess the forecast skill of the surrogate model, we will use a different test dataset, in which we record an ensemble of **trajectories** (instead of an ensemble of input/output pairs). This will allow us to measure the accuracy of the forecast for longer integration times.

[10]:
```
# define rng
rng = np.random.default_rng(seed=seeds.pop(0))

# allocate memory
Nt_fs = 400
Nt_spinup = 100
```

```
Ne_fs = 512
xt_fs = np.zeros((Nt_fs+1, Ne_fs, true_model.Nx))

# initialisation and spin-up
xt_fs[0] = rng.normal(loc=3, scale=1, size=(Ne_fs, true_model.Nx))
for t in trange(Nt_spinup, desc='spin-up integration'):
    xt_fs[0] = true_model.forward(xt_fs[0])

# model integration
for t in trange(Nt_fs, desc='model integration (ensemble)'):
    xt_fs[t+1] = true_model.forward(xt_fs[t])
```

```
spin-up integration:   0%|          | 0/100 [00:00<?, ?it/s]
```

```
model integration (ensemble):   0%|          | 0/400 [00:00<?, ?it/s]
```

7.4 The baseline model: persistence

In this first test series, we use **persistence** as surrogate model. This will provide baselines for our NN results. Persistence is defined as the model for which there is no time evolution.

7.4.1 Evaluate the model

The mean square error (MSE) is the loss function that we will use to train our NNs later. Therefore, the test MSE is a measure of the efficiency of the learning/training process.

The test MSE of persistence is a number whose absolute value is not that important per se (because the input and output data have been normalised) but it will be useful to normalise the test MSE of our trained NNs.

[11]:
```
# compute test MSE
test_mse_baseline = np.mean(np.square(y_test_norm - x_test_norm))

# compute forecast skill
fs_baseline = np.sqrt(np.mean(np.square(xt_fs-xt_fs[0]), axis=2))

# show test MSE
print(f'test mse of persistence = {test_mse_baseline}')
```

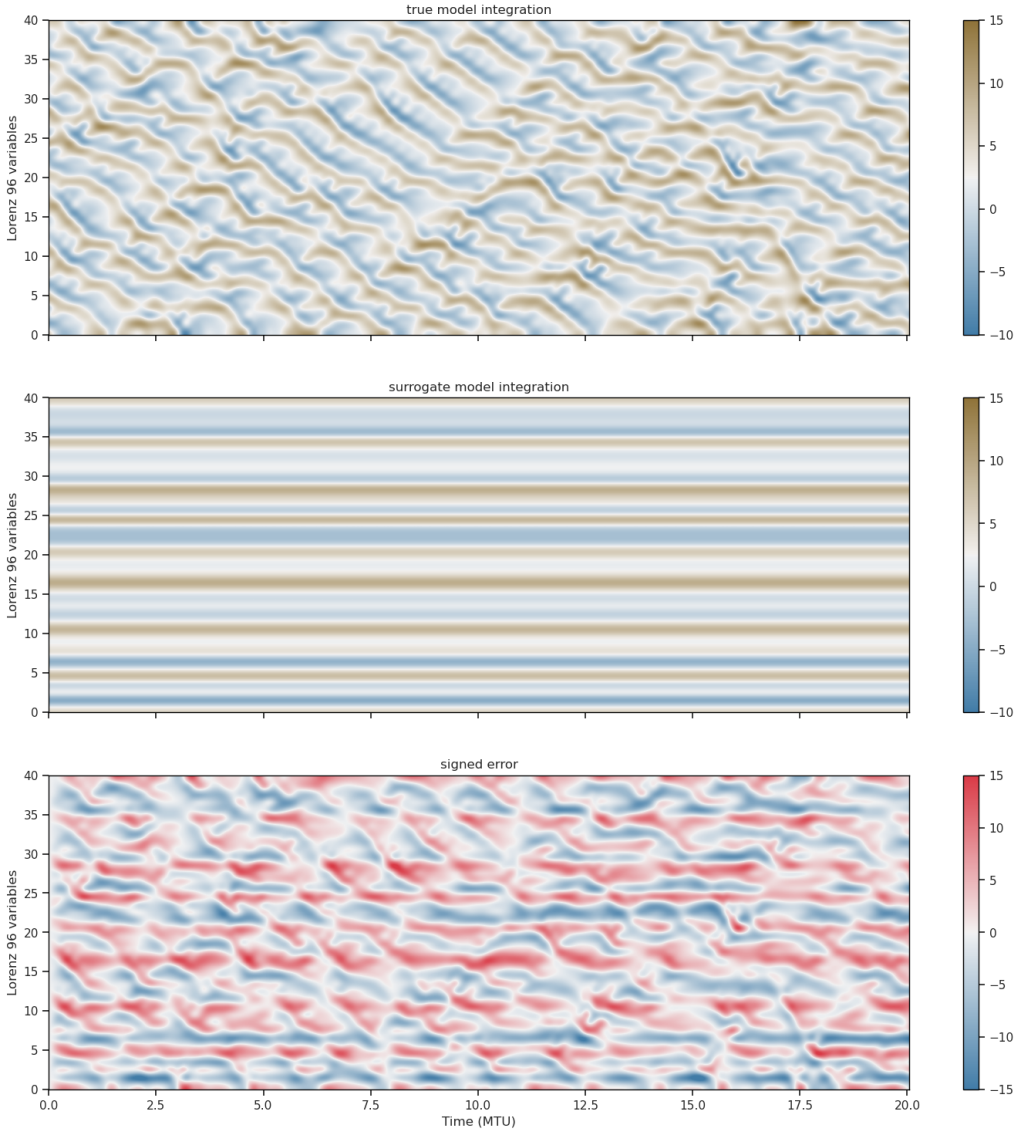
```
test mse of persistence = 0.06183308040943624
```

7.4.2 Example of surrogate model integration

In the following cell, we show one example of model integration.

[12]:
```
utils.plot_l96_compare_traj(
    xt_fs[:, 0],
    np.broadcast_to(xt_fs[0, 0], shape=xt_fs[:, 0].shape),
```

```
    true_model,
    linewidth=18,
)
```

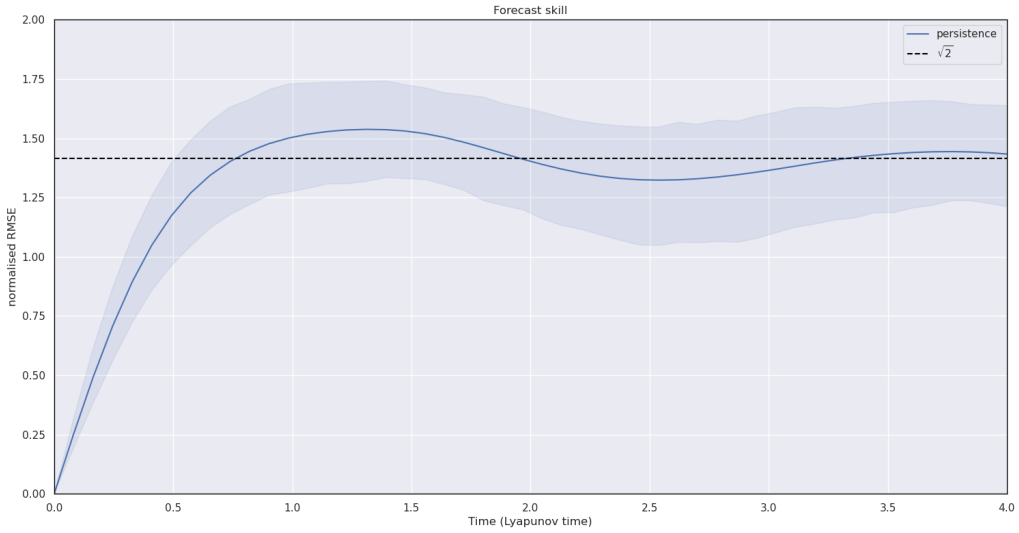


7.4.3 Forecast skill

In the following cell, we plot the average forecast skill, normalised by the model variability. The shadow delimits the 90% confidence interval (percentiles 5 and 95).

[13]:
```
utils.plot_l96_forecast_skill(
    dict(
```

```
        persistence=fs_baseline,
    ),
    true_model,
    p1=5,
    p2=95,
    xmax=4,
    linewidth=18,
)
```



The error rapidly grows as time evolves. After about 1 Lyapunov time, the error oscillates around $\sqrt{2}$, which is the theoretical asymptotic value due to the normalisation and which is consistent with the wave behaviour of the dynamics.

7.5 A naive ML model

7.5.1 Construct and train the model

In this second test series, we train and evaluate a dense NN (sequential NN with only dense layers). In order to create this model, we use the sequential API of tensorflow as follows.

[14]:
```
def make_sequential_network(num_layers, num_nodes, activation):
    # create a sequential network
    network = tf.keras.models.Sequential()
    # add the input layers
    network.add(tf.keras.Input(shape=(true_model.Nx,)))
    # add the internal layers
    for i in range(num_layers):
```

```
        network.add(tf.keras.layers.Dense(num_nodes,␣
↪activation=activation))
    # add the output layer without activation
    network.add(tf.keras.layers.Dense(true_model.Nx))
    return network
```

In the following cell, we actually build a dense NN with 4 internal layers and 128 nodes per layer. The total number of parameters of this model is 59944. This is actually quite large for a 40-variable system. This is because the dense architecture is rather "inefficient" in terms of parameters.

[15]:
```
# set seed
tf.keras.utils.set_random_seed(seeds.pop(0))

# define the NN
num_layers = 4
num_nodes = 128
activation = 'relu'
naive_network = make_sequential_network(num_layers, num_nodes, activation)

# compilation
naive_network.compile(loss='mse', optimizer='adam')

# print short summary
naive_network.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 128)               5248

 dense_1 (Dense)             (None, 128)               16512

 dense_2 (Dense)             (None, 128)               16512

 dense_3 (Dense)             (None, 128)               16512

 dense_4 (Dense)             (None, 40)                5160

=================================================================
Total params: 59,944
Trainable params: 59,944
Non-trainable params: 0
_________________________________________________________________
```

In the following cell, we actually train the model for 256 epochs. Going beyond part I, we

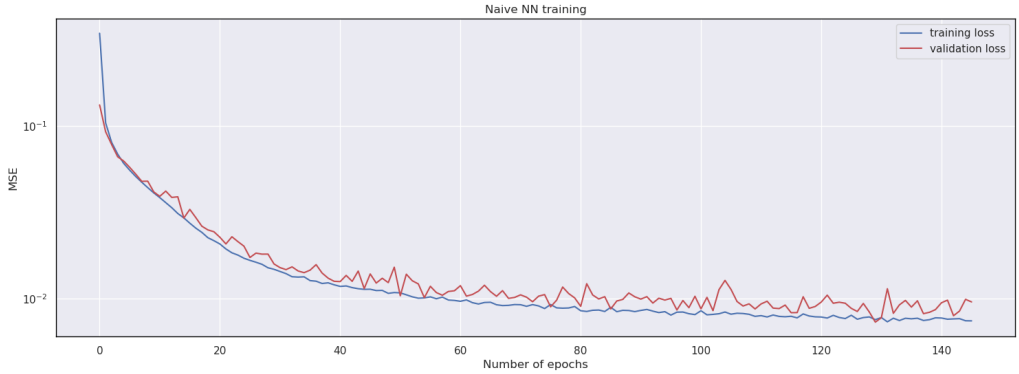
use an EarlyStopping callback to end the training when the validation loss stops improving. This should avoid overfitting.

[16]:
```
# train the ML model
tf.keras.utils.set_random_seed(seeds.pop(0))
num_epochs = 256
tqdm_callback = utils.tqdm_callback(num_epochs, 'naive NN training')
early_stopping_callback = tf.keras.callbacks.EarlyStopping(
    monitor='val_loss',
    patience=16,
    verbose=0,
    restore_best_weights=True,
)
fit_naive = naive_network.fit(
    x_train_norm,
    y_train_norm,
    verbose=0,
    epochs=num_epochs,
    validation_data=(x_valid_norm, y_valid_norm),
    callbacks=[tqdm_callback, early_stopping_callback],
)
```

```
naive NN training:   0%|          | 0/256 [00:00<?, ?it/s]
```

In the following cell we plot the training history, that is, the evolution of the training MSE (the `loss`) and the validation MSE (the `val_loss`) as a function of the number of epochs.

[17]:
```
utils.plot_learning_curve(
    fit_naive.history['loss'],
    fit_naive.history['val_loss'],
    title='Naive NN training',
    linewidth=18,
)
```



Both values are visually closely related. The validation MSE is more noisy than the training MSE, which is expected because the training data is nine times as large as the validation data. After several epochs, the validation MSE gets a bit higher than the training MSE. This is explained by the fact that this data is not used in the gradient descent algorithm. Finally, at the end the validation MSE stops improving. This is the sign that the model is starting to overfit the training data and that we should stop the training.

7.5.2 Evaluate the model

We now compute the test MSE to evaluate our surrogate model.

[18]:
```
# compute test MSE
test_mse_naive = naive_network.evaluate(x_test_norm, y_test_norm,␣
 ↪verbose=0, batch_size=Nt_test)

# compute forecast skill
xt_naive = np.zeros(xt_fs.shape)
xt_naive[0] = xt_fs[0]
for t in trange(xt_naive.shape[0]-1, desc='naive surrogate model␣
 ↪integration'):
    x_norm = normalise_x(xt_naive[t])
    y_norm = naive_network.predict(x_norm, batch_size=Ne_fs, verbose=0)
    xt_naive[t+1] = denormalise_y(y_norm)
fs_naive = np.sqrt(np.mean(np.square(xt_fs-xt_naive), axis=2))

# show test MSE
print(f'test mse of persistence = {test_mse_baseline}')
print(f'test mse of naive model = {test_mse_naive}')
print()
print(f'relative test mse of naive model = {test_mse_naive/
 ↪test_mse_baseline}')
```

```
naive surrogate model integration:   0%|          | 0/400 [00:00<?, ?it/s]

test mse of persistence = 0.06183308040943624
test mse of naive model = 0.011359243653714657

relative test mse of naive model = 0.18370819597694088
```

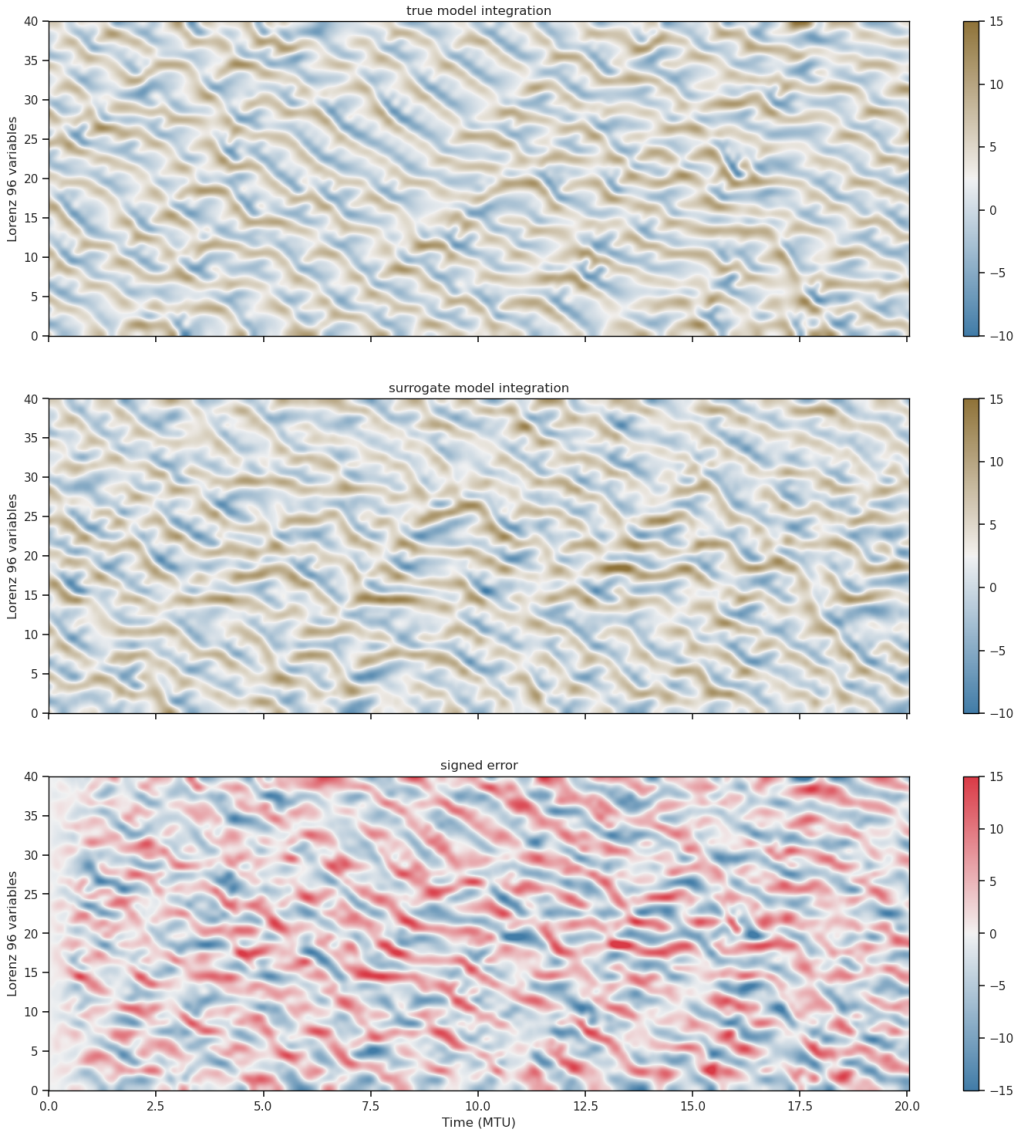
We obtain a reduction of about 80%, which is already quite good, but we will see later that it is possible to do much better.

7.5.3 Example of surrogate model integration

In the following cell, we show once again one example of model integration.

[19]:
```
utils.plot_l96_compare_traj(
    xt_fs[:, 0],
```

```
    xt_naive[:, 0],
    true_model,
    linewidth=18,
)
```

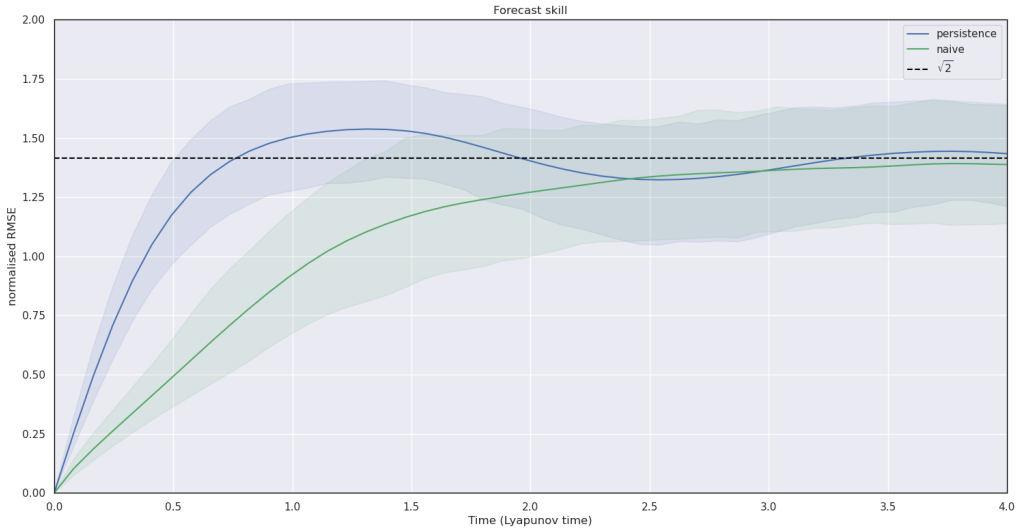


The error is lower than in the first test series, but only during the first few integration steps.

7.5.4 Forecast skill

In the following cell, we plot once again the average forecast skill, normalised by the model variability.

[20]:
```
utils.plot_l96_forecast_skill(
    dict(
        persistence=fs_baseline,
        naive=fs_naive,
    ),
    true_model,
    p1=5,
    p2=95,
    xmax=4,
    linewidth=18,
)
```



This curve confirms that the naive surrogate model is more accurate than persistence for one integration step, and that it remains more accurate until about 2 Lyapunov times.

7.6 A smart ML model

7.6.1 Build and train the model

In this third and last test series, we train and evaluate a smart NN. This NN uses a sparse architecture with convolutional NN and controlled nonlinearity to reproduce the **model tendencies**, as well as a Runge-Kutta integration scheme to **emulate the dynamics**. In order to implement this NN, we use both the functional API (for the model tendency) and the subclassing API (for the integration scheme) of tensorflow.

[21]:
```
class SmartNetwork(tf.keras.Model):

    def __init__(self, num_filters, kernel_size, dt=0.05, **kwargs):
```

```
        super(SmartNetwork, self).__init__(**kwargs)
        self.dt = dt

        # reshape layers
        reshape_input = tf.keras.layers.Reshape((true_model.Nx, 1))
        reshape_output = tf.keras.layers.Reshape((true_model.Nx,))

        # padding layer
        border = kernel_size//2
        def apply_padding(x):
            x_left = x[..., -border:, :]
            x_right = x[..., :border, :]
            return tf.concat([x_left, x, x_right], axis=-2)
        padding_layer = tf.keras.layers.Lambda(apply_padding)

        # convolutional layers
        conv_layer_1 = tf.keras.layers.Conv1D(num_filters, kernel_size)
        conv_layer_2 = tf.keras.layers.Conv1D(1, 1)

        # network for the model tendencies
        x_in = tf.keras.Input(shape=(true_model.Nx,))
        x = reshape_input(x_in)
        x = padding_layer(x)
        x1 = conv_layer_1(x)
        x2 = x1 * x1
        x3 = tf.concat([x1, x2], axis=-1)
        x_out = conv_layer_2(x3)
        x_out = reshape_output(x_out)
        self.tendency = tf.keras.Model(inputs=x_in, outputs=x_out)

    @tf.function
    def call(self, x):
        dx_dt_0 = self.tendency(x)
        dx_dt_1 = self.tendency(x+0.5*self.dt*dx_dt_0)
        dx_dt_2 = self.tendency(x+0.5*self.dt*dx_dt_1)
        dx_dt_3 = self.tendency(x+self.dt*dx_dt_2)
        dx_dt =  (dx_dt_0 + 2*dx_dt_1 + 2*dx_dt_2 + dx_dt_3)/6
        return x + self.dt*dx_dt
```

[22]:
```
# set seed
tf.keras.utils.set_random_seed(seeds.pop(0))

# define the NN
num_filters = 6
kernel_size = 5
smart_network = SmartNetwork(num_filters, kernel_size, dt=true_model.dt)
```

```
# compilation
smart_network.compile(loss='mse', optimizer='adam')

# print short summary
smart_network.tendency.summary()
```

```
Model: "model"
__________________________________________________________________________________
__________________
 Layer (type)                   Output Shape         Param #     Connected␣
 ↪to
==================================================================================
==================
 input_2 (InputLayer)           [(None, 40)]         0           []

 reshape (Reshape)              (None, 40, 1)        0
['input_2[0][0]']

 lambda (Lambda)                (None, 44, 1)        0
['reshape[0][0]']

 conv1d (Conv1D)                (None, 40, 6)        36
['lambda[0][0]']

 tf.math.multiply (TFOpLambda)  (None, 40, 6)        0
['conv1d[0][0]',
'conv1d[0][0]']

 tf.concat (TFOpLambda)         (None, 40, 12)       0
['conv1d[0][0]',
'tf.math.multiply[0][0]']

 conv1d_1 (Conv1D)              (None, 40, 1)        13
['tf.concat[0][0]']

 reshape_1 (Reshape)            (None, 40)           0
['conv1d_1[0][0]']

==================================================================================
==================
Total params: 49
Trainable params: 49
Non-trainable params: 0
__________________________________________________________________________________
__________________
```

The total number of parameters is only 49. Furthermore in this case, with well-chosen parameters it is possible to reproduce the true dynamics up to machine precision: the model is said to be **identifiable**. Also note that this network is built in such a way that we don't need the input and output data to be normalised.

In the following cell, we actually train the model for up to 128 epochs. Once again, we use an EarlyStopping callback to end the training when the validation loss stops improving in order to avoid overfitting.

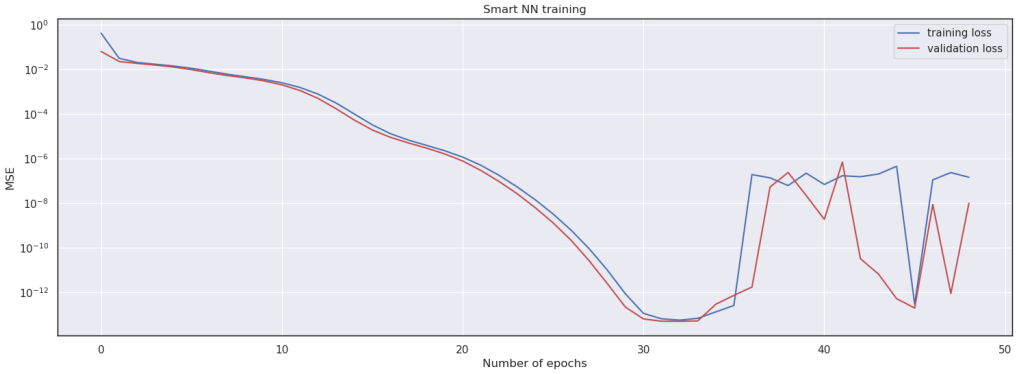
[23]:
```
# train the ML model
tf.keras.utils.set_random_seed(seeds.pop(0))
num_epochs = 128
tqdm_callback = utils.tqdm_callback(num_epochs, 'smart NN training')
early_stopping_callback = tf.keras.callbacks.EarlyStopping(
    monitor='val_loss',
    patience=16,
    verbose=0,
    restore_best_weights=True)
fit_smart = smart_network.fit(
    denormalise_x(x_train_norm),
    denormalise_y(y_train_norm),
    verbose=0,
    epochs=num_epochs,
    validation_data=(denormalise_x(x_valid_norm),␣
 ↪denormalise_y(y_valid_norm)),
    callbacks=[tqdm_callback, early_stopping_callback])
```

```
smart NN training:   0%|          | 0/128 [00:00<?, ?it/s]
```

```
2023-02-13 17:29:53.164812: I
tensorflow/compiler/xla/stream_executor/cuda/cuda_dnn.cc:428] Loaded cuDNN
version 8401
```

In the following cell we plot the training history.

[24]:
```
utils.plot_learning_curve(
    fit_smart.history['loss'],
    fit_smart.history['val_loss'],
    title='Smart NN training',
    linewidth=18,
)
```



Once again, the training and validation MSE are visually closely related. However, by contrast with the previous test series, after about 30 epochs, the MSEs have decreased to 10^{-12}, which should be very close to the numerical precision zero (tensorflow is working on simple precision for real numbers). Passed 30 epochs, the MSEs oscillate at very low values. This behaviour can be considered as numerical noise.

7.6.2 Evaluate the model

We now compute the test MSE to evaluate our surrogate model.

[25]:
```
# compute test MSE
y_test_smart = smart_network.predict(denormalise_x(x_test_norm),␣
 ↪batch_size=Nt_test, verbose=0)
test_mse_smart = np.mean(np.square(normalise_y(y_test_smart)-y_test_norm))

# compute forecast skill
xt_smart = np.zeros(xt_fs.shape)
xt_smart[0] = xt_fs[0]
for t in trange(xt_smart.shape[0]-1, desc='smart surrogate model␣
 ↪integration'):
    xt_smart[t+1] = smart_network.predict(xt_smart[t], batch_size=Ne_fs,␣
 ↪verbose=0)
fs_smart = np.sqrt(np.mean(np.square(xt_fs-xt_smart), axis=2))

# show test MSE
print(f'test mse of persistence = {test_mse_baseline}')
print(f'test mse of naive model = {test_mse_naive}')
print(f'test mse of smart model = {test_mse_smart}')
print()
print(f'relative test mse of naive model = {test_mse_naive/
 ↪test_mse_baseline}')
print(f'relative test mse of smart model = {test_mse_smart/
 ↪test_mse_baseline}')
```

```
smart surrogate model integration:   0%|          | 0/400 [00:00<?, ?it/s]
```

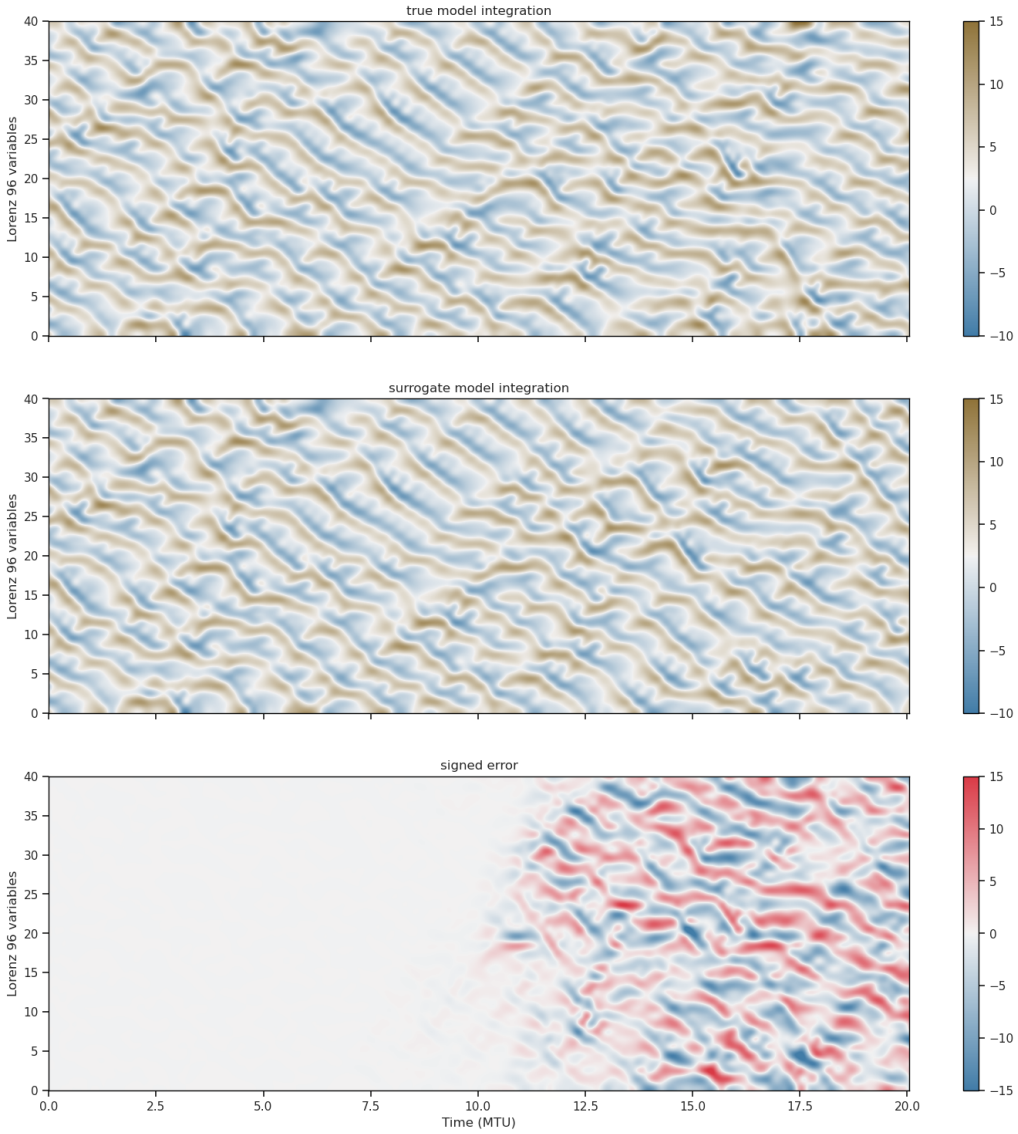
```
test mse of persistence = 0.06183308040943624
test mse of naive model = 0.011359243653714657
test mse of smart model = 2.8139790552492487e-15

relative test mse of naive model = 0.18370819597694088
relative test mse of smart model = 4.550928138491726e-14
```

The test MSE is sufficiently close to zero so that we can consider that our surrogate model reproduces the true model dynamics up to numerical precision.

7.6.3 Example of surrogate model integration

In the following cell, we show once again one example of model integration.

[26]:
```
utils.plot_l96_compare_traj(
    xt_fs[:, 0],
    xt_smart[:, 0],
    true_model,
    linewidth=18,
)
```

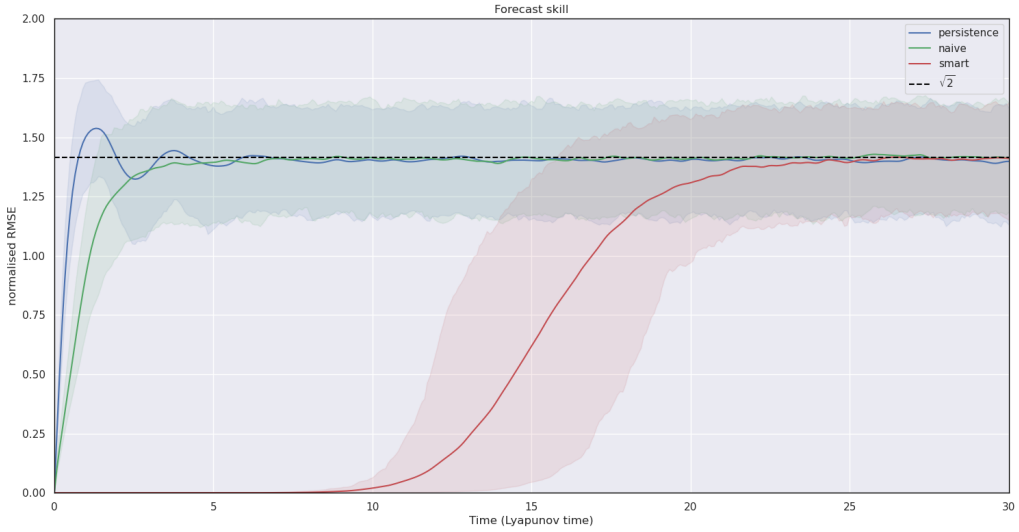


This time, the error is so low that it is not visible until about 6 MTU. At that time, the true model trajectory and the surrogate model trajectory diverge from each other. Indeed, the two models are equivalent up to numerical precision, but they are not bit-wise equivalent, which means that this divergence is unavoidable because of the chaotic nature of the dynamics.

7.6.4 Forecast skill

In the following cell, we plot once again the average forecast skill, normalised by the model variability.

[27]:
```
utils.plot_l96_forecast_skill(
    dict(
        persistence=fs_baseline,
        naive=fs_naive,
        smart=fs_smart,
    ),
    true_model,
    p1=5,
    p2=95,
    xmax=30,
    linewidth=18,
)
```



This curve confirms that the the smart surrogate model is equivalent to the true model up to numerical precision. The numerical divergence between the true and surrogate model happens on average after about 10 Lyapunov times.

Bibliography

Abarbanel, H. D. I., Rozdeba, P. J., Shirman, S., 2018. Machine learning: Deepest learning as statistical data assimilation problems. Neural Computation 30, 2025–2055.

Asch, M., Bocquet, M., Nodet, M., 2016. Data Assimilation: Methods, Algorithms, and Applications. SIAM.

Barron, A. R., 1993. Universal approximation bounds for superpositions of a sigmoidal function. IEEE Transactions on Information Theory 39, 930–945.

Bishop, C. H., Etherton, B. J., Majumdar, S. J., 2001. Adaptive sampling with the ensemble transform Kalman filter. Part I: Theoretical aspects. Mon. Wea. Rev. 129, 420–436.

Blayo, E., Bocquet, M., Cosme, E., Cugliandolo, L. F. (Eds.), 2015. Advanced Data Assimilation for Geosciences. Oxford University Press, lecture Notes of the Les Houches School of Physics: Special Issue, June 2012.

Bocquet, M., 2011. Ensemble Kalman filtering without the intrinsic need for inflation. Nonlin. Processes Geophys. 18, 735–750.

Bocquet, M., Brajard, J., Carrassi, A., Bertino, L., 2019. Data assimilation as a learning tool to infer ordinary differential equation representations of dynamical models. Nonlin. Processes Geophys. 26, 143–162.

Bouttier, F. et Courtier, P., 1997. Data assimilation, concepts and methods. Training course notes of ECMWF, European Centre for Medium-range Weather Forecasts, Reading, UK. 53.

Burgers, G., van Leeuwen, P., Evensen, G., 1998. Analysis scheme in the ensemble kalman filter. Mon. Wea. Rev. 126, 1719–1724.

Carrassi, A., Bocquet, M., Bertino, L., Evensen, G., 2018. Data assimilation in the geosciences: An overview on methods, issues, and perspectives. WIREs Climate Change 9, e535.
URL https://onlinelibrary.wiley.com/doi/abs/10.1002/wcc.535

Chollet, F., 2021. Deep Learning with Python, 2nd Edition. Manning Publications Co., Shelter Island.

Cohn, S., 1997. An introduction to estimation theory. Journal of the Meteorological Society of Japan, 257–288.

Cybenko, G., 1989. Approximation by superpositions of a sigmoidal function. Math. Control Signal Systems 2, 303–314.

Daley, R., 1993. Atmospheric Data Analysis. Cambridge University Press.

Evensen, G., 1994. Sequential data assimilation with a nonlinear quasi-geostrophic model using Monte Carlo methods to forecast error statistics. J. Geophys. Res. 99, 10,143–10,162.

Evensen, G., 2009. Data Assimilation: The Ensemble Kalman Filter, 2nd Edition. Springer-Verlag Berlin Heildelberg.

Fletcher, S. J., 2017. Data assimilation for the geosciences: From theory to application. Elsevier.

Gordon, N. J., Salmond, D. J., Smith, A. F. M., 1993. Novel approach to nonlinear/non-gaussian bayesian state estimation. IEE-F 140, 107–113.

Hornik, K., Stinchcombe, M., White, H., 1989. Multilayer feedforward networks are universal approximators. Neural Networks 2, 359–366.

Houtekamer, P. L., Mitchell, H. L., 1998. Data assimilation using an ensemble Kalman filter technique. Mon. Wea. Rev. 126, 796–811.

Hsieh, W. W., Tang, B., 1998. Applying neural network models to prediction and data analysis in meteorology and oceanography. Bull. Amer. Meteor. Soc. 79, 1855–1870.

Hunt, B. R., Kostelich, E. J., Szunyogh, I., 2007. Efficient data assimilation for spatiotemporal chaos: A local ensemble transform Kalman filter. Physica D 230, 112–126.

Ide, K., Courtier, P., Ghil, M., Lorenc, A., 1999. Unified notation for data assimilation: Operational, sequential and variational. J. Meteor. Soc. Japan 75, 181–189.

Janjić, T., Bormann, N., Bocquet, M., Carton, J. A., Cohn, S. E., Dance, S. L., Losa, S. N., Nichols, N. K., Potthast, R., Waller, J. A., Weston, P., 2018. On the representation error in data assimilation. Q. J. R. Meteorol. Soc. 144, 1257–1278.

Kalnay, E., 2003. Atmospheric Modeling, Data Assimilation and Predictability. Cambridge University Press.

Lahoz, W., Khattatov, B., Ménard, R. (Eds.), 2010. Data Assimilation: Making Sense of Observations. Springer-Verlag.

Law, K. J. H., Stuart, A. M., Zygalakis, K. C., 2015. Data Assimilation: A Mathematical Introduction. Text in Applied Mathematics 62. Springer International Publishing Switzerland.

Lorenz, E. N., 1963. Deterministic nonperiodic flow. J. Atmos. Sci. 20, 130–141.

Lorenz, E. N., Emanuel, K. A., 1998. Optimal sites for supplementary weather observations: simulation with a small model. J. Atmos. Sci. 55, 399–414.

Nocedal, J., Wright, S. J., 2006. Numerical Optimization. Springer Series in Operations Research. Springer.

Pham, D. T., Verron, J., Roubaud, M. C., 1998. A singular evolutive extended kalman filter for data assimilation in oceanography. J. Marine Systems, 323–340.

Reich, S., Cotter, C., 2015. Probabilistic Forecasting and Bayesian Data Assimilation. Cambridge University Press.

Reichstein, M., Camps-Valls, G., Stevens, B., Denzler, J., Carvalhais, N., Prabhat, 2019. Deep learning and process understanding for data-driven Earth system science. Nature 566, 195–204.

Sakov, P., Bertino, L., 2011. Relation between two common localisation methods for the EnKF. Comput. Geosci. 15, 225–237.

Segers, A. J., 2002. Data assimilation in atmospheric chemistry models using kalman filtering. Ph.D. thesis, Delft University, Netherlands.

Talagrand, O., 1997. Assimilation of observations, an introduction. Journal of the Meteorological Society of Japan 75, 191–209.

Tarantola, A., Valette, B., 1982. Generalized nonlinear inverse problems solved using the least square criterion. Reviews of Geophysics and Space Physics 20, 219–232.

Todling, R., 1999. Estimation theory and foundations of atmospheric data assimilation. Data Assimilation Office Note, Goddard Laboratory for Atmospheres.

Van Leeuwen, P. J., 2003. A variance-minimizing filter for large-scale applications. Mon. Wea. Rev. 131, 2071–2084.