

# Reduced dimension models for weather forecasting

NICOLAS BRODU

Master of Science by Research  
in  
Pattern Analysis and Neural Networks



ASTON UNIVERSITY

September 2000

This copy of the thesis has been supplied on condition that anyone who consults it is understood to recognise that its copyright rests with its author and that no quotation from the thesis and no information derived from it may be published without proper acknowledgement.

ASTON UNIVERSITY

# Reduced dimension models for weather forecasting

NICOLAS BRODU

Master of Science by Research

in

Pattern Analysis and Neural Networks, 2000

## Thesis Summary

Weather forecasting is one of the most computationally intensive activities that is routinely undertaken. This project studies the possibility of reducing the dimensions of the models and data sets considered, while maintaining reasonably good predictions.

Techniques dealing with the two problems separately, dimension reduction and forecasting, are applied on real data provided by the European Center for Medium-Range Weather Forecasting, and on an artificial data set generated using the Lorenz equations.

A new algorithm is presented as an extension of the principal interaction patterns framework to neural networks, allowing a simultaneous optimization of the subspace basis for the data projection and the model considered to make predictions.

Advantages and drawbacks of those methods are discussed, and conclusions are drawn from this study regarding the feasibility of reducing the dimensions in the forecasting problem.

**Keywords:** meteorology, time series, prediction, reduced dimension models

# Acknowledgements

I would like to thank Dr Dan Cornford for all his help, his most valuable comments, and his open-mindedness which allowed a great flexibility throughout this project.

Among the other MSC students I would also like to thank Michael Doubez for his feedback.

Thanks to the British Atmospheric Data Center for supplying the data, <http://www.badc.rl.ac.uk/data/ecmwf-era>.

# Contents

<b>1</b>	<b>Introduction</b>	<b>7</b>
1.1	Presentation of the problem . . . . .	7
1.2	Presentation of the data set . . . . .	8
1.3	Thesis overview . . . . .	8
<b>2</b>	<b>Standard techniques in previous work</b>	<b>10</b>
2.1	Dimension reduction . . . . .	10
2.1.1	The problem . . . . .	10
2.1.2	Empirical Orthogonal Functions . . . . .	10
2.1.3	Principal Interaction Patterns . . . . .	11
2.2	Forecasting . . . . .	14
2.2.1	Time series issues . . . . .	14
2.2.2	Auto-Regressive models . . . . .	14
2.2.3	Neural networks . . . . .	15
2.3	State vector estimation . . . . .	17
2.3.1	Extended Kalman Filter . . . . .	17
2.3.2	Variational methods . . . . .	18
<b>3</b>	<b>Prediction in a fixed subspace</b>	<b>19</b>
3.1	EOFs in practice . . . . .	19
3.1.1	Reconstruction error . . . . .	19
3.1.2	Interpretation . . . . .	23
3.2	Prediction . . . . .	27
3.2.1	AR models . . . . .	27
3.2.2	Multi-Layer Perceptron . . . . .	27
<b>4</b>	<b>Joint optimization</b>	<b>30</b>
4.1	Principal Interaction Patterns in a neural network context . . . . .	30
4.2	A two-stage algorithm . . . . .	32
4.2.1	Optimization in the reduced dimension space . . . . .	32
4.2.2	Optimization in the data space . . . . .	34

## CONTENTS

<b>5</b>	<b>An artificial data set</b>	<b>37</b>
5.1	The Lorenz equations . . . . .	37
5.2	Results . . . . .	38
5.2.1	Empirical Orthogonal Functions . . . . .	38
5.2.2	2-stage PIP algorithm . . . . .	39
5.2.3	Prediction comparison . . . . .	40
<b>6</b>	<b>Real data</b>	<b>43</b>
6.1	Selected data . . . . .	43
6.2	Sub-sampled real data . . . . .	45
<b>7</b>	<b>Conclusion</b>	<b>48</b>
<b>A</b>	<b>Data plots</b>	<b>50</b>
<b>B</b>	<b>Coding issues</b>	<b>52</b>
B.1	Presentation . . . . .	52
B.2	Algorithm implementation example . . . . .	53
B.3	Notions revisited in practice . . . . .	55

# List of Figures

2.1	Feed-forward network structure with a time delay input vector. . . . .	16
2.2	Detail of a neuron computation. . . . .	16
3.1	Sub-sampling and mixing variables. . . . .	21
3.2	Training and test sets reconstruction errors. . . . .	22
3.3	Amplitude of the seasonal sine fitted to the temperature at 850mb. . . . .	23
3.4	Phase of the seasonal sine fitted to the geopotential height at 500mb. . . . .	23
3.5	EOF 1, geopotential height 500mb. . . . .	24
3.6	EOF 2, geopotential height 500mb. . . . .	25
3.7	EOF 5, geopotential height 500mb. . . . .	25
3.8	EOF 15, geopotential height 500mb. . . . .	25
3.9	EOF 150, geopotential height 500mb. . . . .	26
3.10	Series for EOF 1. . . . .	26
3.11	Series for EOF 2. . . . .	26
3.12	Series for EOF 5. . . . .	27
3.13	Series for EOF 15. . . . .	27
3.14	A simple 2-layer MLP structure. . . . .	28
3.15	2-layer MLP with an augmented input vector of 2 time steps. . . . .	29
4.1	PIPs in a neural network context. . . . .	31
4.2	Network for minimizing the error in the subspace, with a lag vector. . . . .	33
4.3	Complementary model, to optimize computations in data space. . . . .	35
5.1	3D representation of the Lorenz attractor. . . . .	38
5.2	EOF reconstruction error on the artificial data. . . . .	39
5.3	Artificial data structure as captured by the EOFs. . . . .	40
5.4	One-step ahead prediction structure. . . . .	41
5.5	Comparison of prediction methods on a Lorenzian system. . . . .	42
6.1	Comparison of different forecasting models on real data. . . . .	44
6.2	Comparison of different forecasting models on subsampled data. . . . .	46
A.1	Temperature at 850mb on November 26 <sup>th</sup> , 1993. . . . .	50
A.2	Geopotential height at 500mb on November 26 <sup>th</sup> , 1993. . . . .	50
A.3	Relative humidity at 850mb on November 26 <sup>th</sup> , 1993. . . . .	51
A.4	Wind vector field at 850mb on April 4 <sup>th</sup> , 1992. . . . .	51

# Chapter 1

## Introduction

### 1.1 Presentation of the problem

Today's biggest computers in the world are used in military applications and to make weather predictions. This isn't a scientists whim, and neither it is a waste of resources. In fact, weather forecasting is one of the most arduous challenges, partly because of the vast amount of computational power required, and partly because of the techniques involved, which range from statistical analysis to dynamical systems theory.

The atmosphere is intrinsically tridimensional, and thus must not be treated only at the Earth surface. Data is retrieved on different levels defined by surfaces of constant pressure.

It is possible to infer more or less accurately the sea level pressure, the temperature of the atmosphere and the oceans, the relative humidity, the geopotential height<sup>1</sup>, the wind velocities,... at the levels considered thanks to satellites and direct observations. But it may not be possible to use all the variables because of practical limitations, and a choice has to be done to select the most relevant ones.

The atmosphere can be modeled, non-uniquely, with a subset of those variables. For example temperature, geopotential height, wind velocities, and relative humidity in the ECMWF reanalysis reduced-resolution archive<sup>2</sup>. In this project, longitude-latitude gridded data was retained, but it is also possible to work in a frequency space [8].

Primary differential equations on those variables, drawn from physical principles, can be used to model the evolution of the data [6]. This is currently the approach favored in computation centers, as much for prognostic as for diagnostic purposes<sup>3</sup>. Due to the excessive amount of computational power and data storage space required to carry on such methods, this project rather uses a data-based approach in its ex-

---

<sup>1</sup>Geopotential height  $Z$  is a measure of height defined with the gravity acceleration. It relates to the altitude  $z$  by the relation  $Z = \int_0^z \frac{g(h)}{g_0} dh$ , with  $g(h)$  the gravity acceleration at height  $h$ ,  $g_0$  the mean acceleration at the Earth surface. It is useful in meteorological applications because  $g$  can be linked to the density at that point, which in turn can be written in terms of pressure and temperature. It is roughly equal to  $z$  at low altitudes where  $g(h) \simeq g_0$ . More details can be found in [10].

<sup>2</sup>European Center for Medium-Range Weather Forecasting (ECMWF 1996). See next Section 1.2.

<sup>3</sup>Diagnostic concerns the correction of the model considered in the light of observed data, prognostic the making of predictions.

periments. Models allowing a certain flexibility, like neural networks, are fitted to the data.

This project neither deals with the data assimilation, nor the physical interpretation of the variables. Rather, only the prediction aspect will be addressed. The goal isn't so much to make accurate predictions. The main concern is to study the possibility of reducing the dimension of the data while keeping the predictions within reasonably good error bounds.

## 1.2 Presentation of the data set

The data used in this project was provided by the European Center for Medium-Range Weather Forecasting (ECMWF 1996). It consists of the north Atlantic<sup>4</sup> subset of the 2.5° longitude-latitude gridded, initialized, re-analysis global atmospheric data archive, over the period 1979-1994. It contains 6 variables: temperature, geopotential height, relative humidity, and the wind velocities. Some of those variables are plotted in Appendix A. In practice for this project, only temperature and geopotential height were selected, on the atmospheric levels of 850mb, 500mb, and 200mb. 850mb roughly corresponds to the top of the cloud-free lowest part of the atmosphere directly above the surface. The 500mb level is a standard benchmark in meteorology, and lies at a medium altitude where the atmosphere dynamics are 'smoother' than at lower levels. 200mb variations exhibit another time scale, and to some extent reflect the longer range evolution of the atmosphere.

The internal dynamics of this north Atlantic region are relatively smooth, and mainly driven by a West to East propagation. The 850mb level is more turbulent than the higher levels, and also partly forced by them. The 200mb level is the smoothest, and the slowest to vary. Those simple observations helped to interpret the results in the course of the experiments, justifying the choice for this region.

## 1.3 Thesis overview

Chapter 2 introduces some techniques for dimension reduction and forecasting that are already in use. This presentation also serves as a quick reference throughout the rest of the thesis. The reader may skip known parts, and come back later when reaching such a reference.

Chapter 3 deals with the approach where dimension reduction and forecasting are treated separately. Practical results are shown, as is a justification for the next chapter. It is recommended to read both, in order.

Chapter 4 presents an algorithm able to optimize jointly the dimension reduction and the forecasting model. Theoretical aspects and practical limits are discussed, together with possible uses for the algorithm.

---

<sup>4</sup>The region selected spans 140°W to 57.5°E in longitude, and 67.5°N down to 30°N in latitude.

## CHAPTER 1. INTRODUCTION

In Chapter 5, an artificial data set is constructed from the Lorenz equations, with a much reduced size compared to the real data set. Performances and results of the methods presented in the preceding chapters are then compared. The reader is invited to use this chapter as a benchmark test for the algorithms introduced in this project.

The description and results of experiments carried out on the real world data are presented in Chapter 6. Differences and similarities with results from the artificial set are highlighted. This chapter is helpful when trying to understand the difficulties arising when applying the algorithms to real data.

Appendix A gives a graphical idea of the data set used in this project.

Appendix B mentions the coding difficulties encountered in this project, and the solutions adopted. While not strictly necessary on a theoretical point of view, it could prove to be of great value if the reader has to implement similar techniques practically.

# Chapter 2

## Standard techniques in previous work

### 2.1 Dimension reduction

#### 2.1.1 The problem

In a high dimensional dynamic data set, the trajectory in time may span only a limited subspace, or it may be driven by a few hidden variables. The ultimate goal of dimension reduction in forecasting problems would be to restrict, project, or transform the data to retain only the information necessary for making predictions. Because dynamics have to be taken into account, dimension reduction doesn't consist only in classifying observations into fixed groups. This is the main difficulty, retaining not only information about the data itself, but also on its evolution. Gathering observations in a restricted number of classes, as is often done to interpret high dimensional data, doesn't tell much about the internal evolution of the individuals, assuming there is a time dimension.

Nevertheless, similarities exist and can be exploited. For example, Empirical Orthogonal Functions (EOFs) act like a Principal Component Analysis, using a variance maximization. On the other hand, the model used for the prediction is equally important, and EOFs do not take it in account. For this, Principal Interaction Patterns (PIPs) are a possible solution.

#### 2.1.2 Empirical Orthogonal Functions

EOFs are the basis vectors of an orthogonal decomposition of the data. Statistics are used to define the projection, in such a way as to maximize the variance of the data in each subspace in turn.

More precisely, if  $e_1$  is the first basis vector, it is computed by minimizing:

$$\left\langle \left\| X_t - (X_t \cdot e_1) e_1 \right\|^2 \right\rangle_T, \quad (2.1)$$

where  $X_t$  is a data vector at date  $t$ . The notation  $\langle \rangle_t$  is used to denote averaging over

time. The second EOF  $e_2$  can be constructed with:

$$\left\langle \left\| X_t - (X_t \cdot e_1)e_1 - (X_t \cdot e_2)e_2 \right\|^2 \right\rangle_T, \quad (2.2)$$

once  $e_1$  has been obtained. The process can be repeated sequentially to compute all the EOFs.

As detailed in [9], in practice a simpler singular value decomposition on the data matrix,

$$\begin{bmatrix} x_{1,t=1} & x_{1,t=2} & \dots & x_{1,t=T} \\ x_{2,t=1} & x_{2,t=2} & \dots & x_{2,t=T} \\ \dots & \dots & \dots & \dots \\ x_{n,t=1} & x_{n,t=2} & \dots & x_{n,t=T} \end{bmatrix},$$

is used. The columns  $x_1, \dots, x_n$  are the components of the data vector  $X$ , for each date  $t$ .  $X$  can be composed of different variables, at different atmospheric levels, in the hope of improving the dimension reduction by using the correlation between the variables. Section 3.1, illustrates this on practical examples. Also, for this reason the prediction obtained by running models<sup>1</sup> on each EOF component series independently is not equivalent to running similar models in the initial data space.

Since the variables aren't homogeneous, it is necessary to mean correct and normalize them before reshaping and appending in one single column vector. As is customary in meteorology, it is also possible to remove the seasonality before computing the EOFs. In fact, since seasonality is a strong signal, if not removed it will appear mostly in the first EOF component.

The main advantage of EOFs resides in their ability to capture most of the variance in the original data series, using only a few components. A quantitative example using radiosonde data is presented in [9, p293]. Unfortunately, maximizing the variance may not be relevant to the model used for prediction. Methods like the Principal Interaction Patterns, presented in the next section, have been designed to overcome this drawback. In practice, though, EOFs are easy to compute and still give acceptable results.

### 2.1.3 Principal Interaction Patterns

Principal Interaction Patterns are constructed so as to best choose simultaneously a subspace to project the data on, and a model within a given model class. The projection is linear, but the model is generic. In the particular case where the model is also linear, the basis vectors are called Principal Oscillation Patterns.

In the following presentation, the notation  $M : (a, b)$  stands for a vector or matrix  $M$  of  $a$  rows and  $b$  columns.

**PIPs** The data vector  $X_t$  is projected on a (time-independent) reduced dimension subspace, by minimizing the residual reconstruction error. Within this subspace,

---

<sup>1</sup>Autoregressive models are used in Section 3.2.1.

a class of model is chosen. An objective function for the error minimization process is built to take in account both the model parameters and the subspace basis vectors. In the continuous case, this error is based on the difference between the rate of change of the true data vector, and its modeling using the subspace dynamics. In the discrete case, the function is a measure of the error committed after prediction and reconstruction. Averaging over the whole time series ensures the global best fit. In fact, more constraints and additional weighting terms can be added in the objective functions, and the metric chosen for the error measure can take in account statistical properties (the inverse of the covariance matrix is a common choice). The point is to always parameterize the model and the basis vectors, so as to optimize them together during the minimization process. The PIPs are the low-dimensional space basis vectors. A more detailed introduction to the PIPs scheme can be found in [5].

**POPs** Principal Oscillation Patterns are the linear case of PIPs. A data vector  $X_t$  is expanded as  $X_t = \sum_j z_{j,t} e_j + \xi_t$  with  $e_j$  the POPs,  $\xi_t$  residual noise. The model class is in the form of an update matrix  $A$ ,  $Z_{t+1} = AZ_t + \nu_t$ , with  $\nu_t$  a stochastic forcing term (see Equation 2.11 for comparison). The minimization process leads to  $e_j$  being eigenvectors of the averaged matrix  $\langle X_{t+1} X_t^T \rangle \langle X_t X_t^T \rangle^{-1}$ . A derivation of POPs from statistical principles can be found in [9, p335].

Let  $X : (n, 1)$  be the data vector in the high  $n$ -dimensional space. The trajectory to first order of  $X$  is governed by:

$$\frac{dX}{dt} = \dot{X} = F(X),$$

and in the discrete case:

$$X_{t+1} = F(X_t). \tag{2.3}$$

$X$  differs from the reconstructed variable  $Y : (n, 1)$  by the error  $\xi$ :

$$X = Y + \xi, \tag{2.4}$$

The reconstruction from  $Z : (m, 1)$  in a  $m$ -dimensional subspace is achieved using the PIPs  $P : (n, m)$ :

$$Y = PZ. \tag{2.5}$$

Whereas  $X$  may range the original data space,  $Y$  can only lie in the region corresponding to the projection of the reduced dimension space using  $P$ . Thus, the assumption is that  $X$  varies mostly along a few dimensions for  $\xi$  to be small.

Adjoint patterns  $D : (n, m)$  are defined as a set of vectors orthogonal to  $P$ , with respect to a metric  $M : (n, n)$ . Setting:

$$D = P(P^T M P)^{-1}, \tag{2.6}$$

ensures that  $P^TMD = I_m$ , since:

$$P^TMD = P^TM(P(P^TMP)^{-1}) \quad (2.7)$$

$$= (P^TMP)(P^TMP)^{-1} \quad (2.8)$$

$$= I_m \quad (2.9)$$

The reduced dimension vector  $Z$  is defined from the original data vector  $X$  using the adjoint patterns:

$$Z = D^TMX. \quad (2.10)$$

A parameterized class of model  $G$  is then chosen to drive the evolution of the reduced dimension data  $Z$ , with a stochastic forcing term  $\nu$ :

$$\dot{Z} = G(\alpha, Z) + \nu, \quad \text{with } \alpha = \alpha_1, \dots, \alpha_p \text{ the model parameters,}$$

and in the discrete case:

$$Z_{t+1} = G(\alpha, Z_t) + \nu_t. \quad (2.11)$$

To build the objective error function  $\varepsilon(\alpha, P, G)$ , one may introduce another metric  $\bar{M}$ . The model  $G$  is fixed in the PIPs framework. The optimization is a least squares minimization of the residual error of the rates of change (continuous case) or the residual error after prediction and reconstruction (discrete case), averaged over the whole time period  $T$  considered:

$$\begin{aligned} \varepsilon &= \int_T \|\dot{X} - \dot{Y}\|_{\bar{M}}^2 dt \\ &= \int_T \|\dot{X} - PG(\alpha, D^TMX)\|_{\bar{M}}^2 dt, \quad \text{using Equations (2.5), (2.11) and (2.10).} \end{aligned}$$

and in the discrete case:

$$\begin{aligned} \varepsilon &= \left\langle \left\| X_{t+1} - Y_{t+1} \right\|_{\bar{M}}^2 \right\rangle_T \\ &= \left\langle \left\| X_{t+1} - PG(\alpha, D^TMX_t) \right\|_{\bar{M}}^2 \right\rangle_T. \end{aligned} \quad (2.12)$$

The second form of the equations shows that the optimization variables are  $\alpha$  and  $P$  (since  $D$  depends solely on  $P$  according to (2.6)).

Solving the partial derivatives equations with respect to a parameter  $\alpha_i$ ,  $\frac{\partial \varepsilon}{\partial \alpha_i} = 0$ , and with respect to a basis vector  $p_j$ ,  $\frac{\partial \varepsilon}{\partial p_j} \delta p_j = 0$ , in the hope of finding a local minimum, isn't easy. In the linear case, though, the search for basis vectors becomes an eigenvalue problem. Iterative methods can also be applied directly to minimize the objective function  $\varepsilon$ .

Another strategy could be to include statistical properties, defined using another metric  $M'$  for the projected subspace, in the error measure. Denoting the mean corrected state vector as  $\hat{X}$ , one can use the error function:

$$\varepsilon = \frac{1}{\left\langle \|D^T M \hat{X}\|_{M'}^2 \right\rangle_T} \left\langle \|\hat{X}_{t+1} - PG(\alpha, D^T M \hat{X}_t)\|_M^2 \right\rangle_T. \quad (2.13)$$

This latter form is found in [2] and has the advantage of taking into account the variance of the data in the reduced subspace, since  $\|D^T M \hat{X}\|_{M'}^2$  is maximized at the same time.

For this project, PIPs were used in a neural network context. See Section 4.1, for a detailed presentation of the model chosen.

## 2.2 Forecasting

### 2.2.1 Time series issues

Weather forecasting in this project isn't a problem of predicting physical observations, like rain. Rather, it consists in estimating the values of the fundamental physical variables that define the actual state of the atmosphere. For this reason, the forecasting problem draws on time series techniques, which sometimes necessitate adaptations to account for the high dimensionality of the data sets.

As is often the case in time series analysis, existence of a low-dimensional manifold for the dynamics is presumed, the measured variables being driven by the trajectories of the state vector there, and noise. Lag vectors can be defined from the observations  $x_t$ , with a delay  $\tau$  and a dimension  $d$ , by  $\vec{x} = (x_t, x_{t-\tau}, \dots, x_{t-(d-1)\tau})$ . The time-delay embedding theory ensures, under certain conditions, the existence of a diffeomorphism between the manifold on which the lag vector trajectories lie, and the presumed manifold for the data state vector dynamics. One of those conditions requires the dimension  $d$  to be sufficiently large. A presentation can be found in [7, p 407], and the models used in this project have time delayed input vectors to take this property in account. Unfortunately, the lag dimensions used could not be very large, due to computational limitations.

Another problem is the stationarity of the series. All through this project, it is assumed that effects like global warming and the solar cycles are negligible, and the series obey fixed laws on the period considered. The mean and variance are computed on the training set, and used as an estimate for the validation set, even if the actual value may differ for some cases. In Section 4.2.2, an algorithm is presented that can be applied on-line, and is thus able to deal with those approximations.

### 2.2.2 Auto-Regressive models

A complete description of Auto-Regressive, Moving Average models, their various combinations and other stochastic processes, can be found in [4]. The following presentation

only introduces the simple Auto-Regressive model of order  $p$  usually denoted  $AR(p)$ , which is used in practice for this project.

For a single variable  $x$ , an  $AR(p)$  model can be written as:

$$x_t = \alpha_1 x_{t-1} + \dots + \alpha_p x_{t-p} + z_t, \quad (2.14)$$

where  $z$  is a random process with mean zero and variance  $\sigma_z^2$ . The assumption is that each value can be described in terms of a fixed linear combination of the  $p$  previous values, plus noise.

Let  $X : (m, 1)$  be the series vector in a  $m$ -dimensional space. This is, for example, the data projected on an EOF space. Two possibilities can be considered to extend the  $AR(p)$  model to vectors. The first is to use one coefficient per vector:

$$X_t = \alpha_1 X_{t-1} + \dots + \alpha_p X_{t-p} + Z_t. \quad (2.15)$$

This ensures a global fit on all the variables, but is a very limited model as it assumes the  $m$  components evolve in a similar way, and contains only  $p$  parameters.

The other solution, chosen in this project, is to run  $m$   $AR(p)$  models in parallel, one for each component:

$$x_{i,t} = \alpha_{i,1} x_{i,t-1} + \dots + \alpha_{i,p} x_{i,t-p} + z_{i,t}, \quad \text{along each dimension } i. \quad (2.16)$$

This has the advantage of being much more flexible. The drawback is that each series is now trained independently from the others, and correlations between them aren't taken in account. Still, in practice it gives better results than the first solution. Section 3.2.1 details the AR models used in this project.

### 2.2.3 Neural networks

Neural networks can be used in numerous areas, and in particular for time series prediction. The interested reader can refer to [7] for an analysis of neural networks and their applications in specific contexts. The following description presents the use of neural networks for making predictions.

The method consists in using the ability of neural networks to model an unknown function, by training a network to produce a time series values from the previous values over a past period. Then, assuming the evolution rules driving the data don't change, the neural network is used to produce future values forecasts from the current values.

Time delay samples are used as inputs for the neural network, ultimately large enough for the embedding theory to be verified. They are equivalent to lag vectors with delay  $\tau = 1$  and dimension  $d$ . Figure 2.1 illustrates this structure for a fully connected feed-forward network with 2 hidden layers.

Training a neural network typically consists in finding a set of parameters (weights, biases,...) minimizing an error function. This can be done quite efficiently when the back-propagation algorithm can be applied. In the case of a simple feed-forward network, this consists in computing the gradient of the error function considered on the

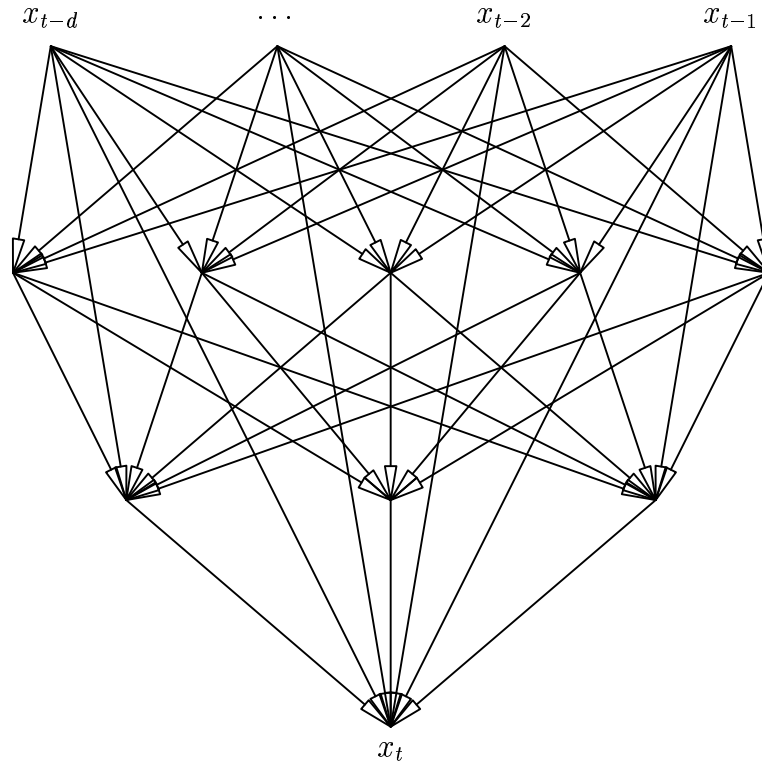


Figure 2.1: Feed-forward network structure with a time delay input vector.

outputs with respect to the weights and biases, starting from the output layer and deducing the hidden layers in order using the chain rule. More exactly, for 2 layers  $x$  and  $y$ , denoting  $\omega_{ij}$  the weight from  $x_i$  to  $y_j$ ,  $a_j$  the activation for the neuron  $y_j$ , and  $f$  the transfer function, as sketched in Figure 2.2, the gradient of the error function  $\varepsilon$  with respect to  $\omega_{ij}$  is:

$$\begin{aligned} \frac{\partial \varepsilon}{\partial \omega_{ij}} &= \sum_k \frac{\partial \varepsilon}{\partial a_k} \frac{\partial a_k}{\partial \omega_{ij}}, \quad \text{for all activations } k, \\ &= \frac{\partial \varepsilon}{\partial a_j} x_i. \end{aligned} \tag{2.17}$$

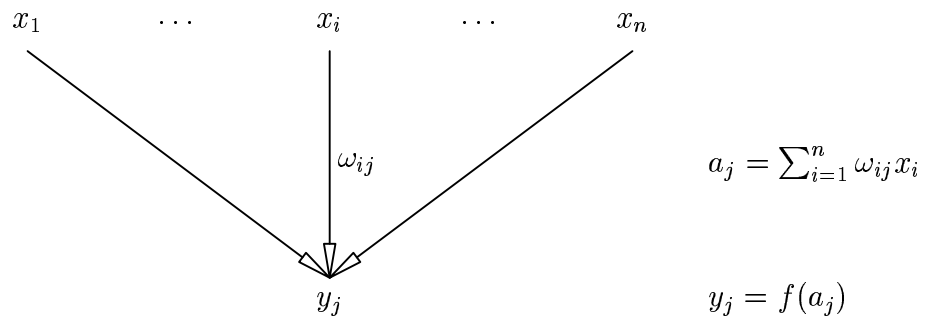


Figure 2.2: Detail of a neuron computation.

Starting from the output units,  $\frac{\partial \varepsilon}{\partial a_j}$  can be computed directly. Then, in a feed-forward network,

$$\begin{aligned}
 \frac{\partial \varepsilon}{\partial a_i} &= \sum_j \frac{\partial \varepsilon}{\partial a_j} \frac{\partial a_j}{\partial a_i} \\
 &= \sum_j \frac{\partial \varepsilon}{\partial a_j} \frac{\partial (\sum_{k=1}^n \omega_{kj} y_k)}{\partial a_i} \\
 &= \sum_j \frac{\partial \varepsilon}{\partial a_j} \frac{\partial (\omega_{ij} f(a_i))}{\partial a_i}, \quad \text{the network is feed-forward, only } k = i \text{ remains} \\
 &= f'(a_i) \sum_j \frac{\partial \varepsilon}{\partial a_j} \omega_{ij}.
 \end{aligned} \tag{2.18}$$

Thus, it is possible to propagate the gradient from layer  $y$  to layer  $x$ , and so on up to the input layer, with an algorithm which complexity is the order of the number of weights. Back-propagation is used in this project when it is possible.

## 2.3 State vector estimation

### 2.3.1 Extended Kalman Filter

This technique is presented for completeness and for its popularity, but isn't used in practice in this project. Some similarity can be observed with the algorithm presented in 4.2.2, though.

The idea is to try to evolve the state of the underlying system and update it to the observations. It is separated into 2 steps, the forecast and the analysis. Let  $X_a$  be the current guess (analysis) of the underlying system, and  $X_f$  the forecast. The forecast step is:

$$X_f = M(X_a), \tag{2.19}$$

where  $M()$  represents the model evolution function, usually differential equations based on physical laws.

In the Extended Kalman Filter, this is in fact a matrix  $M$  that corresponds to the first order linearisation, so  $X_f = M * X_a$ .

The analysis step is:

$$X_a = X_f + K * (Y_o - H * X_f). \tag{2.20}$$

Here,  $X_a$  is the next analysis state,  $Y_o$  the observed data,  $K$  the filter (matrix) and  $H$  a first order linearisation of the model to observation transfer function.

Methods for evaluating  $K$  vary from updating it each step according to the observations, to methods like optimal interpolation found in [3] where the variance of the error made is minimized.

This process can be done in real time if the filter function isn't too expensive to evaluate, and takes the observations in account. If no data is available or if it's incomplete, the analysis step can be omitted and a new forecasting can nevertheless be evaluated. If for any reason the results are not acceptable, this method allows the state  $X_a$  to be reset to a supposedly better new initial condition.

On the other hand, one needs the model equations to apply this technique, and also the model to observation transfer function. The Kalman filter is used to update  $X_a$  (and  $X_f$ ) but with a fixed known model. This is the reason why it wasn't used in this project.

The key point of this method is the evaluation of a good filter, which isn't necessarily an easy task. It may also be computationally intensive.

### 2.3.2 Variational methods

Variational methods are currently in use in many of the meteorological centers across the world. They are applied on all available data, and require a great amount of computational power. These methods are mainly used for diagnostic purposes.

Observations are taken over a time interval, and are used to minimize an objective function, of the type:

$$J = \frac{1}{2} \sum_t (H(X_t) - Y_t)^T * R_t^{-1} * (H(X_t) - Y_t), \quad (2.21)$$

where  $H$  is the model to observation transfer function,  $X$  the state of the underlying model,  $Y$  the observed data and  $R$  the estimated covariance matrix of the noise. The model is fixed, and a time window moved along the available data. The target variable for the optimization is  $X$ ,  $Y$  being known. This is called a 3D variational analysis.

More elaborate objective functions can also include an *a priori* weight matrix and a term for evolving the state  $X_t$  itself, in which case the method is called a 4D variational analysis because a time evolution rule is also applied.

The results depend on the objective function, which can be tuned arbitrarily. This method also takes in account all the data on a given time period to ensure a global best fit. Unfortunately, depending on the choice for  $H$ , the model can have a very large number of free parameters. Thus, if the time period is too short, there will not be enough data to account for all the free parameters. If it's too long, the relation between the data at the edges of the time interval might be null or too complex for the model.

When the predictions become too bad, the state has to be re-estimated.

# Chapter 3

## Prediction in a fixed subspace

In the chapter, we consider separately the dimension reduction and the forecasting problem. The projection is done independently from the prediction model considered.

### 3.1 EOFs in practice

#### 3.1.1 Reconstruction error

One way to measure the ability of EOFs to faithfully represent the data is to compute the error committed when reconstructing the series from a few selected components. The data is projected on the set of EOFs retained, and reconstructed. The EOF vectors being orthogonal with norm 1, this transformation can be written:

$$R = U(U^T \mathcal{D}), \tag{3.1}$$

where  $R$  is the reconstruction,  $U$  the matrix which columns are the EOFs vectors retained, and  $\mathcal{D}$  the 0-mean, variance corrected data.

Experiments have been undertaken to investigate the influence of mixing variables together in the EOF computations. Specifically, for two variables  $X$  and  $Y$ , the computations are run with the matrix structures:

$$\begin{bmatrix} x_{1,t=1} & x_{1,t=2} & \dots & x_{1,t=T} \\ x_{2,t=1} & x_{2,t=2} & \dots & x_{2,t=T} \\ \dots & \dots & \dots & \dots \\ x_{n,t=1} & x_{n,t=2} & \dots & x_{n,t=T} \\ y_{1,t=1} & y_{1,t=2} & \dots & y_{1,t=T} \\ y_{2,t=1} & y_{2,t=2} & \dots & y_{2,t=T} \\ \dots & \dots & \dots & \dots \\ y_{n,t=1} & y_{n,t=2} & \dots & y_{n,t=T} \end{bmatrix}, \tag{3.2}$$

and:

$$\begin{bmatrix} x_{1,t=1} & x_{1,t=2} & \cdots & x_{1,t=T} \\ x_{2,t=1} & x_{2,t=2} & \cdots & x_{2,t=T} \\ \dots & \dots & \dots & \dots \\ x_{n,t=1} & x_{n,t=2} & \cdots & x_{n,t=T} \end{bmatrix}, \begin{bmatrix} y_{1,t=1} & y_{1,t=2} & \cdots & y_{1,t=T} \\ y_{2,t=1} & y_{2,t=2} & \cdots & y_{2,t=T} \\ \dots & \dots & \dots & \dots \\ y_{n,t=1} & y_{n,t=2} & \cdots & y_{n,t=T} \end{bmatrix}, \text{ separately} \quad (3.3)$$

$X$  and  $Y$  themselves can be a union of other variables. For example,  $X$  can be the temperature at all atmospheric levels considered, and  $Y$  the geopotential height at the same levels. In each case, the reconstruction Root Mean Square Error (RMSE) was computed for each separate variable  $Z$  (for example the geopotential height at 200mb):

$$\varepsilon = \sqrt{\frac{1}{n} \|Z_r - Z\|^2}, \quad (3.4)$$

with  $Z_r$  the corresponding part of the reconstructed data  $R$  in Equation (3.1), according to the matrix structure chosen, and  $n$  its dimension.

Since the only factor modified is the grouping of variables, it is responsible for the difference observed in the reconstruction errors. This operation also serves as a good benchmark and helps the interpretation of the prediction errors in later stages.

The influence of sub-sampling the series in time was also assessed, but the main factor for the reconstruction error remains the number of EOF components retained.

Figure 3.1 is a comparison of reconstruction errors obtained when sub-sampling in time 1 every 3 or 5 days, together with the effect of mixing variables. In each case, EOFs were computed using the first 10 years as a training set. The remaining 5 years, used as the out of period test set, were projected onto the selected number of components, and reconstructed. The hypothesis tested here is of the series being stationary enough, in the sense that EOFs defined on the training set also span the test set.

6 variables were considered together; temperature and geopotential height, on 3 levels: 850mb, 500mb, and 200mb. These variables were processed as described above, and once reconstructed the RMSE was computed for each of them. Figure 3.1 shows the results only for the geopotential height at 500mb. Other variables behave similarly. The 2 lower curves correspond to the same computations using the geopotential height at 500mb alone.

The general shape of the curve is the same whatever the number of variables, levels, and the sub-sampling used. The reconstruction error decreases quickly at the beginning, then tails off. A possible explanation is because the first EOFs introduced capture most of the variance, thus define an orthogonal subspace that best contains the data. Hence a quick decrease, so long as another dimension is necessary to describe the data, the amount of information introduced by doing so is quite large. Then, when the subspace dimension is nearly large enough to describe the data, the reconstruction error doesn't benefit as much from an increase in the dimensionality. At last, the error tails off to a plateau, which can probably be attributed to noise. It finally drops to 0 at the very end, the reconstruction is exact when the EOFs form a basis spanning the whole data space (see also Figure 5.2).

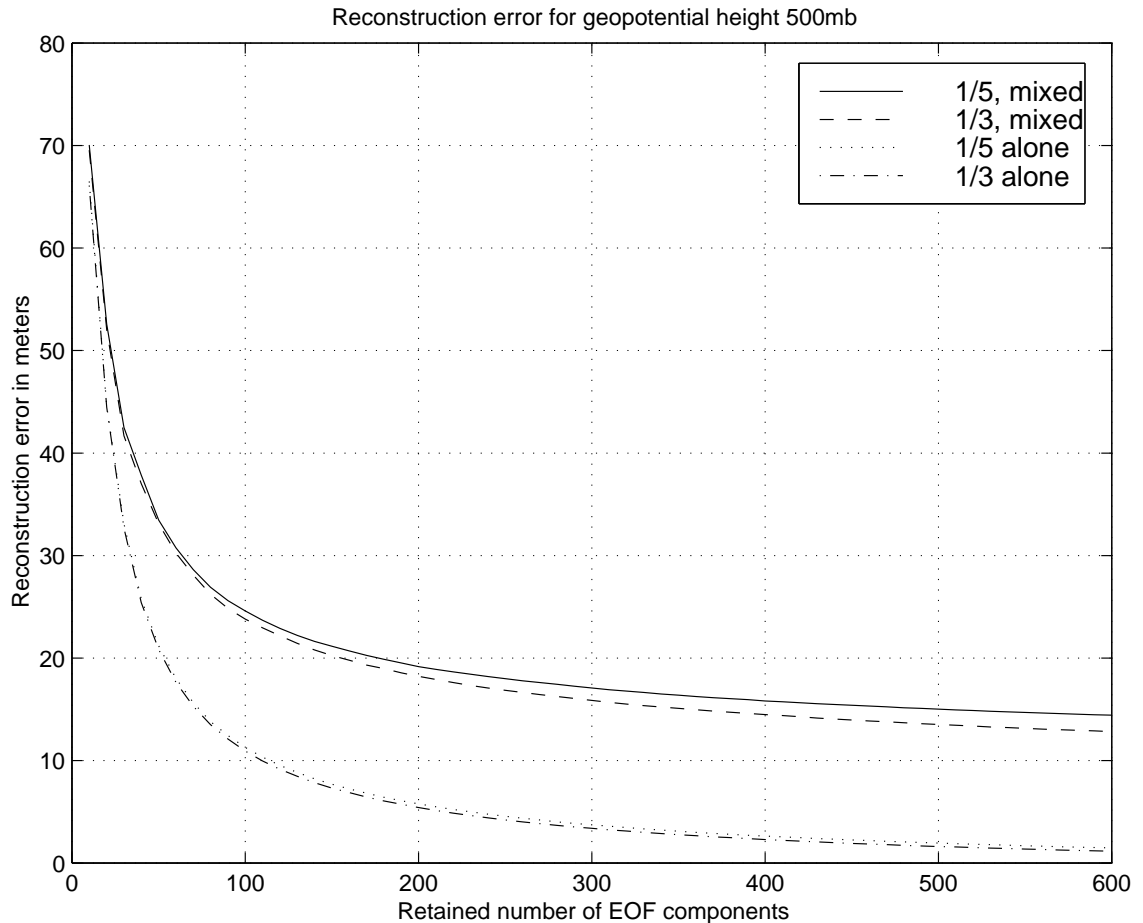


Figure 3.1: Sub-sampling and mixing variables.

When the reconstruction computation is done on the same set used to define the EOFs, the error committed is lower than what is obtained on the test set otherwise. Figure 3.2 is the result of an experiment using only the temperature, but at the 3 levels considered, and sub-sampling in time 1 every 3 days. The general shape of the curve is the same as in Figure 3.1. However, the reconstruction error on the training set is clearly lower than on the test set. This indicates a change in the series, the data does not always span the same subspace through time. Either shorter time windows could be used to overcome this problem, or a longer training period if we assume the variations are due to slow oscillations only partially captured.

This, together with the fact there was no real sharp break in the real data reconstruction error curve, suggests there isn't any clear simple attractor in the real set on the period considered. It could also be that the region selected is too narrow and the corresponding system is sensitive to external factors not taken in account. More experiments than an EOF reconstruction error would be necessary to confirm those hypothesis.

Concerning the dimension reduction problem, the choice for the number of retained components has to be made on the systematic error one is ready to accept before the

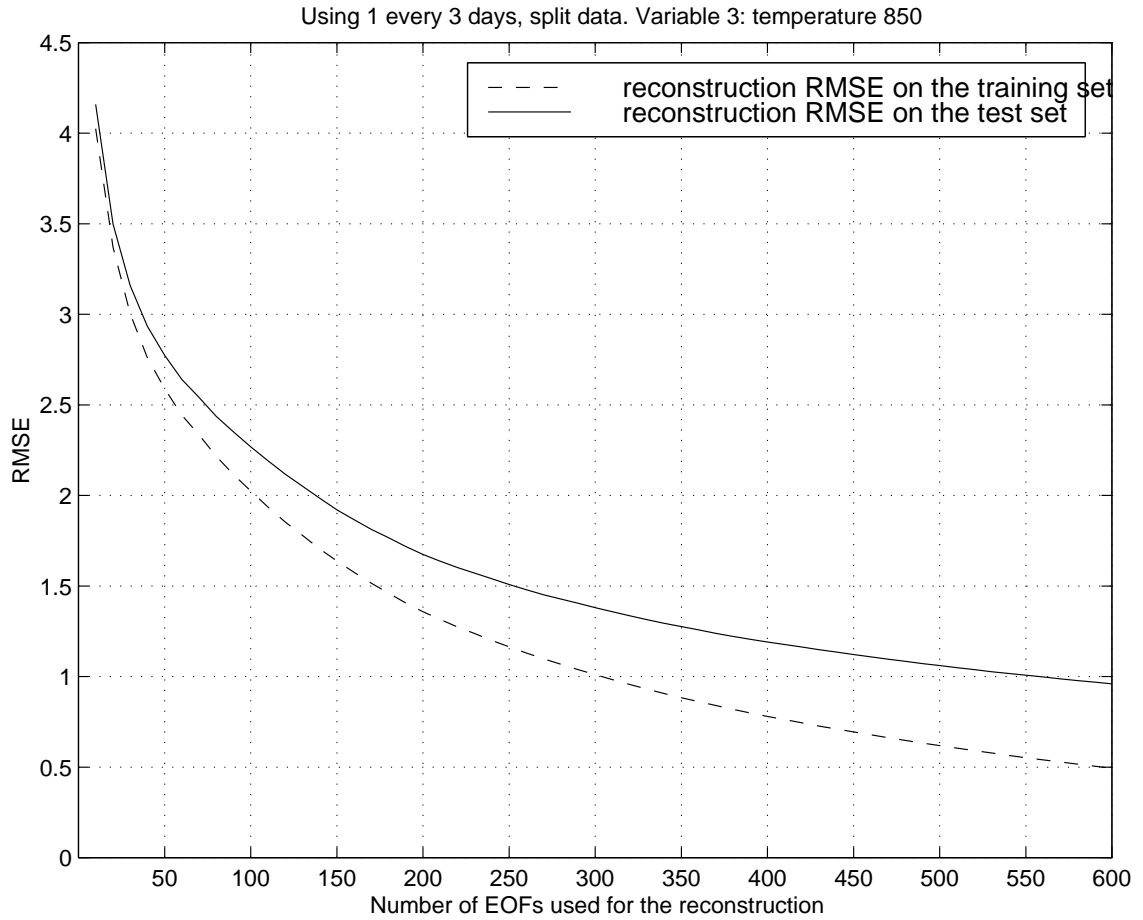


Figure 3.2: Training and test sets reconstruction errors.

prediction computations. There is no sharp break in the curve to help this decision, as can be observed on the artificial data set on Figure 5.2. This suggests there is no clearly defined attractor as well in the real data. A good criterion is a comparison with prediction errors previously published; ECMWF results in [1] are a  $20m$  root mean square difference between 1-day ahead forecasts from a 4D-variational method and observations, for geopotential height at  $500mb$ . This corresponds roughly to a restriction to 150 EOFs according to Figure 3.1, when considering temperature and geopotential height together, and 50 EOFs when using geopotential height alone.

Another choice is whether to keep the variables together or split them. Once again, it depends on the error one is ready to accept. If it is more important to have a very good reconstruction, then splitting each variable for the computation is best. On the other hand, the total dimension becomes the number of EOFs retained times the number of variables, which might be unacceptable. In the previous example, this would be  $6 \times 50 = 300$  EOFs instead of 150, for the same reconstruction error. Mixing the variables leads to a higher plateau in the curves, but with EOFs common to all it reduces the dimension by as great a factor as there are variables. An explanation could be the correlation between the variables, with the definition of common EOFs to

exploit it.

Sub-sampling increases the error as the frequency decreases. This might be expected, since the lower the frequency is, the more information is lost by sub-sampling. Compared to the error values, though, this is a minor factor. Thus it is better to keep all the observations, but if this isn't possible for practical computational purposes, sub-sampling can be considered so as to reduce the time and memory consumption.

### 3.1.2 Interpretation

#### Seasonal components

The seasonal signal was removed before the computations, by fitting a sine curve with a fixed period of one year to each mean corrected data grid point. The corresponding code is presented in Appendix B. This preprocessing is quite fast.

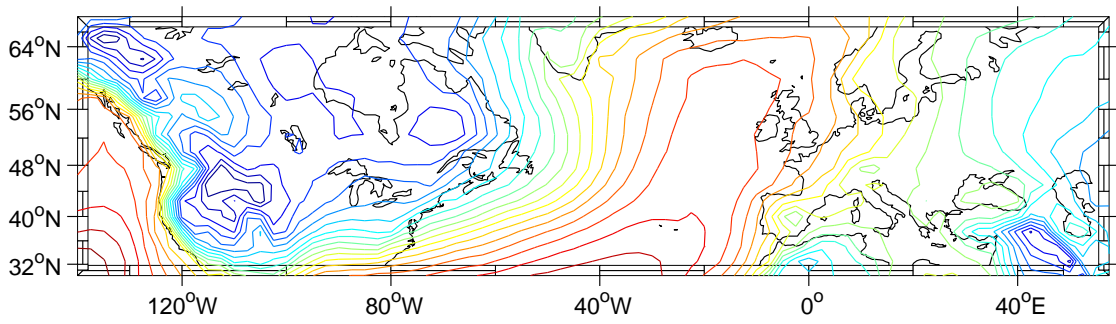


Figure 3.3: Amplitude of the seasonal sine fitted to the temperature at 850mb.

Figure 3.3 shows contours of same amplitude for the seasonal sine fitted to the temperature at 850mb. It is interesting how the contours follow the shorelines at this low level. Continental climates are typically much sensitive to the seasons, with large extrema marking Summer and Winter. In comparison, oceanic climates have relatively damped oscillations.

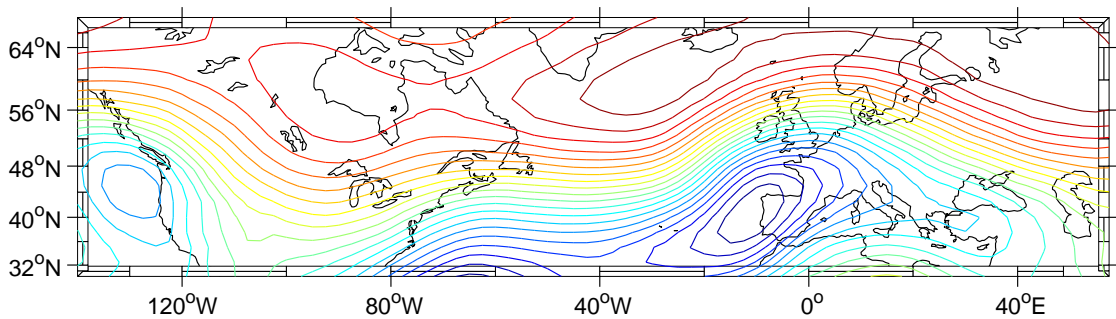


Figure 3.4: Phase of the seasonal sine fitted to the geopotential height at 500mb.

The phase information for the temperature is also mainly driven by the coast lines. On the other hand, for geopotential height at 500mb, Figure 3.4 exhibits a strong dependency on latitude. This is coherent with points on the same contour having their seasons happening at the same time. But longitude also has an influence, which may also be attributed to continental masses, especially on the west sides where the winds blowing from the ocean hit the continents. Nevertheless, the relation is not as clear as for the amplitude signal.

In both case, the dependency on the geographic position justifies the procedure of fitting one sine signal per grid point.

### Empirical Orthogonal Functions

The following graphs were obtained by mixing the 6 variables as described above. Only the geopotential height at 500mb part of the EOFs computed are presented here, reshaped in the original grid. These selections of the full EOFs can be compared to real EOFs obtained using only geopotential height at 500mb, but are not necessarily orthogonal.

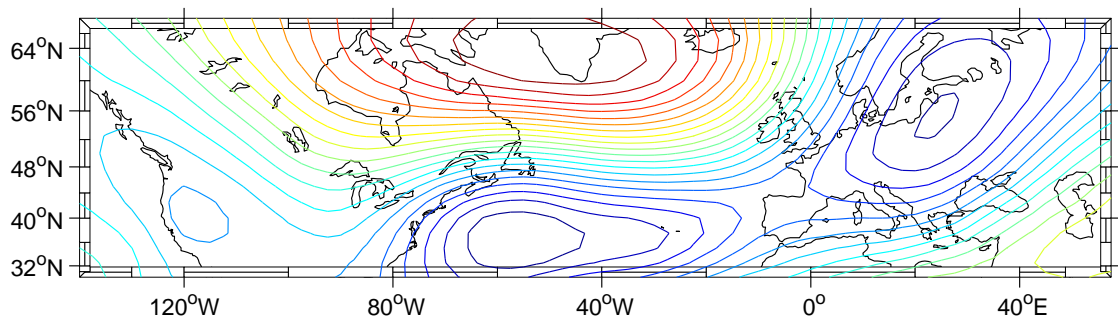


Figure 3.5: EOF 1, geopotential height 500mb.

Figure 3.5 shows mainly a North-South pattern. There is also a center at the north of eastern Europe, and another on the western north America. This first EOF corresponds to the observed main trends in the north Atlantic climate, the winds in between the 2 main centers blowing eastward<sup>1</sup>.

The second EOF corresponds to a West-East decomposition. Though not a formal result at all, such a behavior might be suspected. If the first EOF captures most of the variance in the North-South direction, then the second would have to settle for the complementary pattern, that captures most of the remaining variance. Yet, taking care in this interpretation is necessary. The full EOFs are constrained to be orthogonal, and defined from statistics in a high-dimensional space, which doesn't necessarily lead to intuitive results.

<sup>1</sup>The sign of the EOFs components isn't relevant, so the direction cannot be guessed from EOF 1. See the wind field in Figure A.4 for an actual observation.

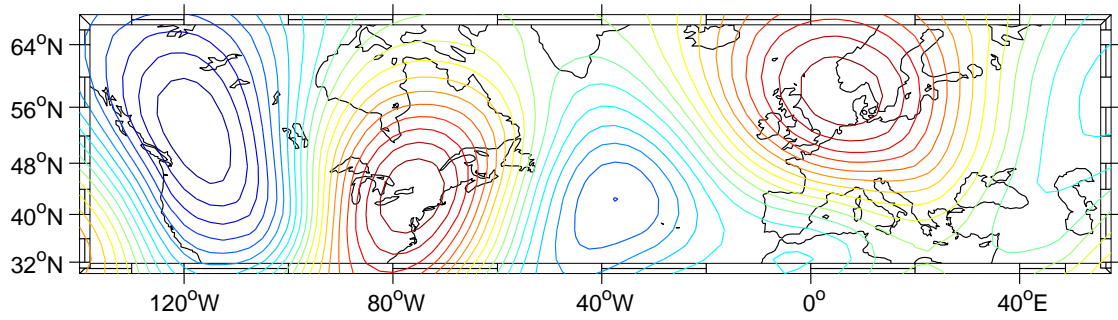


Figure 3.6: EOF 2, geopotential height 500mb.

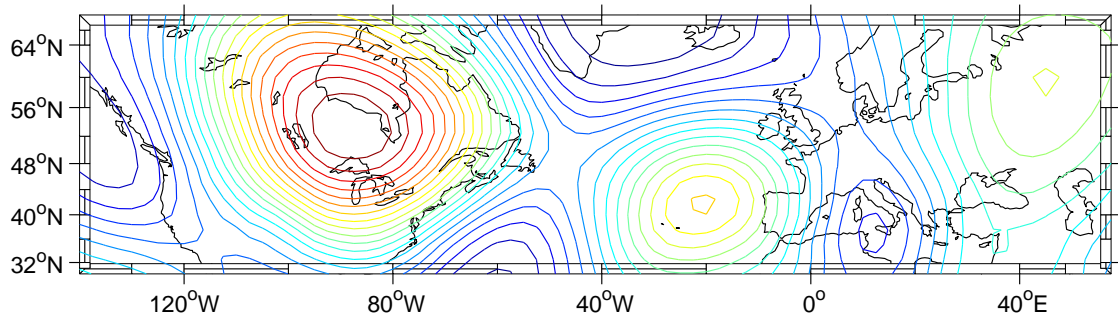


Figure 3.7: EOF 5, geopotential height 500mb.

The fifth EOF reverts to larger scale structures. An interpretation can still be given, considering the centers to represent the Oceanic, Mediterranean, and Continental climates.

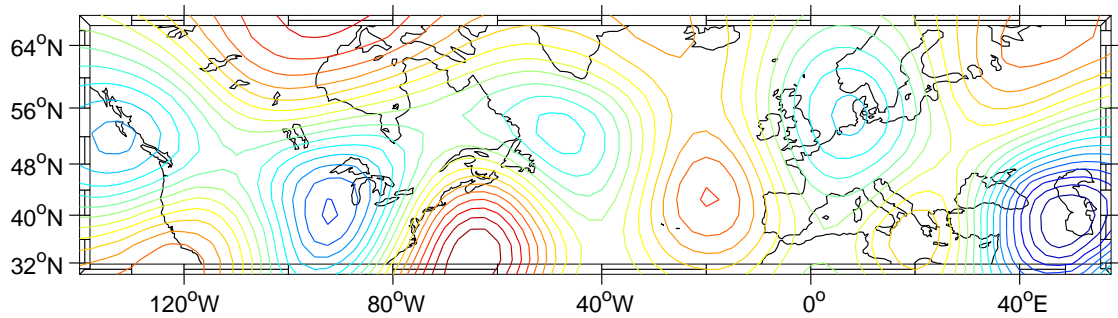


Figure 3.8: EOF 15, geopotential height 500mb.

With EOF 15, more centers are appearing and the interpretation becomes difficult. In fact, Figure 3.8 is here to show this effect. As the number of EOFs increases, their interpretability in terms of the observed weather gets poorer, and the number of minima or maxima increases. The associated time series confirm this suspicion, as does EOF 150 shown in Figure 3.9.

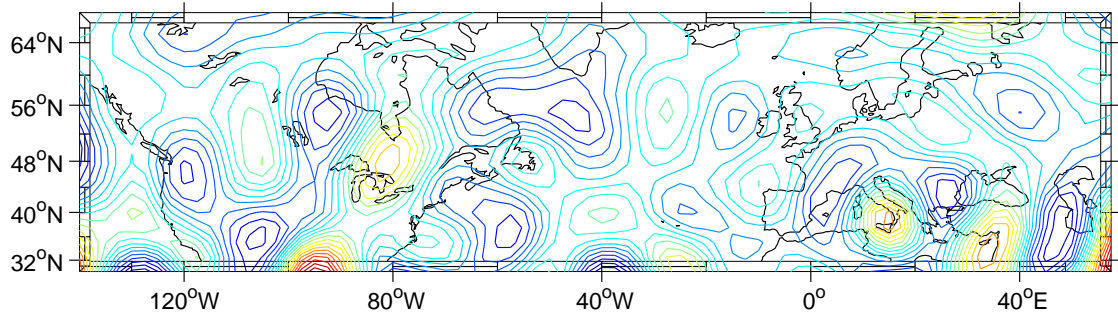


Figure 3.9: EOF 150, geopotential height 500mb.

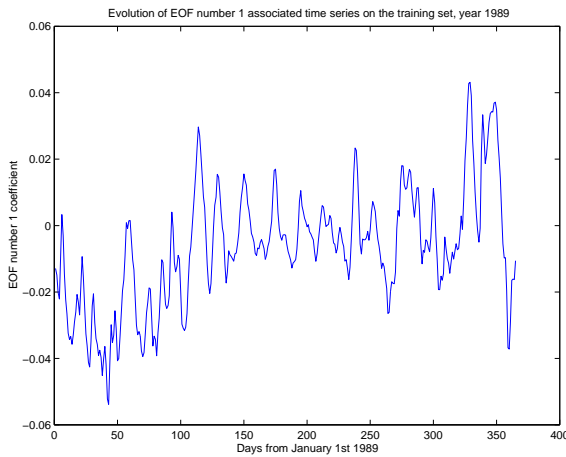


Figure 3.10: Series for EOF 1.

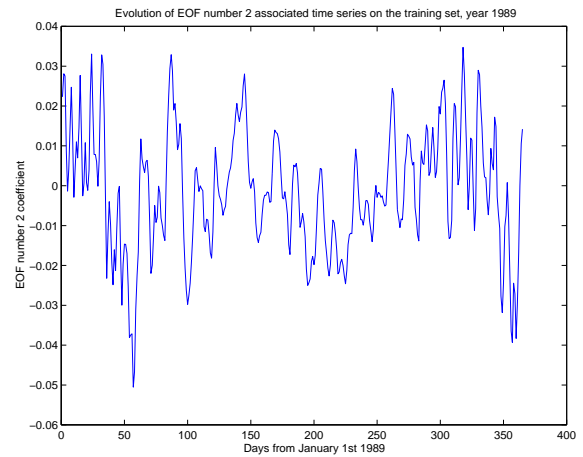


Figure 3.11: Series for EOF 2.

The series associated with EOFs 1 and 2 are shown in Figure 3.10 and 3.11, and seem very noisy, but a signal may still be detected there. The seasonal component was removed prior to the EOF computations. On results where it was not, the time series for EOF 1 was strongly dominated by a sine curve of period one year.

While the series associated with EOF 5 in Figure 3.12 may still be considered to be a strongly noisy signal, it is arguable whether the series for EOF 15 shown in Figure 3.13 is not just noise. On the other hand, in Section 3.1.1, the reconstruction error obtained using just 15 components is very high, and more EOFs are required to represent the full signal.

The conclusion is that while EOFs can offer a good dimension reduction, they may not be significant in terms of the internal dynamics. Since they are derived from a statistical principle, the interpretation for the first EOFs is possible, and this remains a great advantage. In fact, when using a scheme as the Principal Interaction Patterns presented in Section 2.1.3, the interpretation becomes secondary, since the subspaces are built to be more relevant with respect to a prediction model. This is a tradeoff, which can be partially overcome when using the EOFs as starting point for other algorithms, as done in Section 4.1.

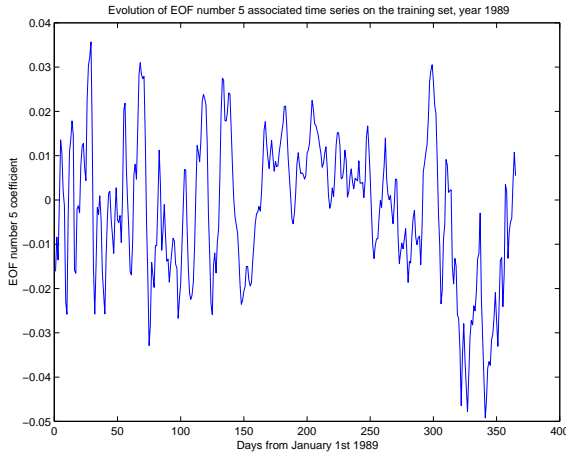


Figure 3.12: Series for EOF 5.

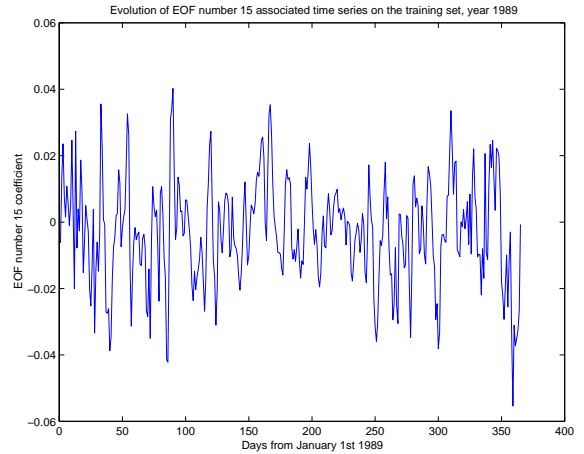


Figure 3.13: Series for EOF 15.

## 3.2 Prediction

### 3.2.1 AR models

In this section we present a practical way to fit an AR model. Refer to Section 2.2.2 for an introduction.

Consider the series  $x_t$ . The problem is to find the set of coefficients  $\alpha_i$  that minimizes the distance between the sum  $\alpha_1 x_{t-1} + \dots + \alpha_p x_{t-p}$  and  $x_t$ , for all  $t > p$  the lag dimension.

This problem can be solved easily using a least square minimization:

$$\begin{bmatrix} \alpha_1 \\ \dots \\ \alpha_p \end{bmatrix} \text{ minimizes } \min_a \|Xa - y\|^2, \quad (3.5)$$

with:

$$X = \begin{bmatrix} x_{t-1} & \dots & x_{t-p} \\ \dots & \dots & \dots \\ x_{t+T-1} & \dots & x_{t+T-p} \end{bmatrix}$$

$$y = \begin{bmatrix} x_t \\ \dots \\ x_T \end{bmatrix}.$$

The actual computations were performed by the public domain LAPACK least square minimization routine. See Appendix B, for more coding information.

### 3.2.2 Multi-Layer Perceptron

A Multi-Layer Perceptron is a feed-forward neural network, with fully connected layers of neurons. A simple 2-layer MLP is described in Figure 3.14, with the same input and output dimension.

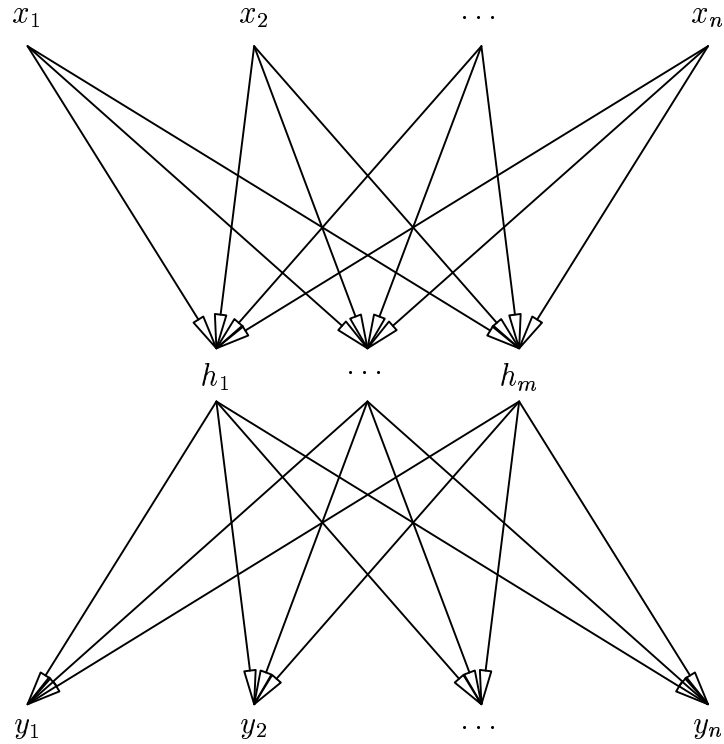


Figure 3.14: A simple 2-layer MLP structure.

The neuron transfer function considered in this project is the hyperbolic tangent, and the output activation function is linear:

$$h_j = \tanh\left(\sum_i \omega_{ij}x_i + c_j\right) \quad (3.6)$$

$$y_i = \sum_j \nu_{ji}h_j, \quad (3.7)$$

with  $\omega_{ij}$  and  $\nu_{ji}$  the weights for the first and second layers. A bias is introduced for the hidden layer, but not on the output because the data considered is mean-corrected.

Prediction can be achieved by training the network to produce the next data, from a lag vector. The corresponding structure is shown in Figure 3.15. Unlike the AR models used in the previous section, all the components can interact with each other. This means, for  $i, j$  two dimension index, the predicted values  $y_i$  and  $y_j$  both depend on  $x_i$  and  $x_j$ . The aforementioned AR models considered each component as an independant time series.

The training was done using a batch back-propagation with a scaled conjugate gradient descent algorithm. Regularization using weight decay was also applied.

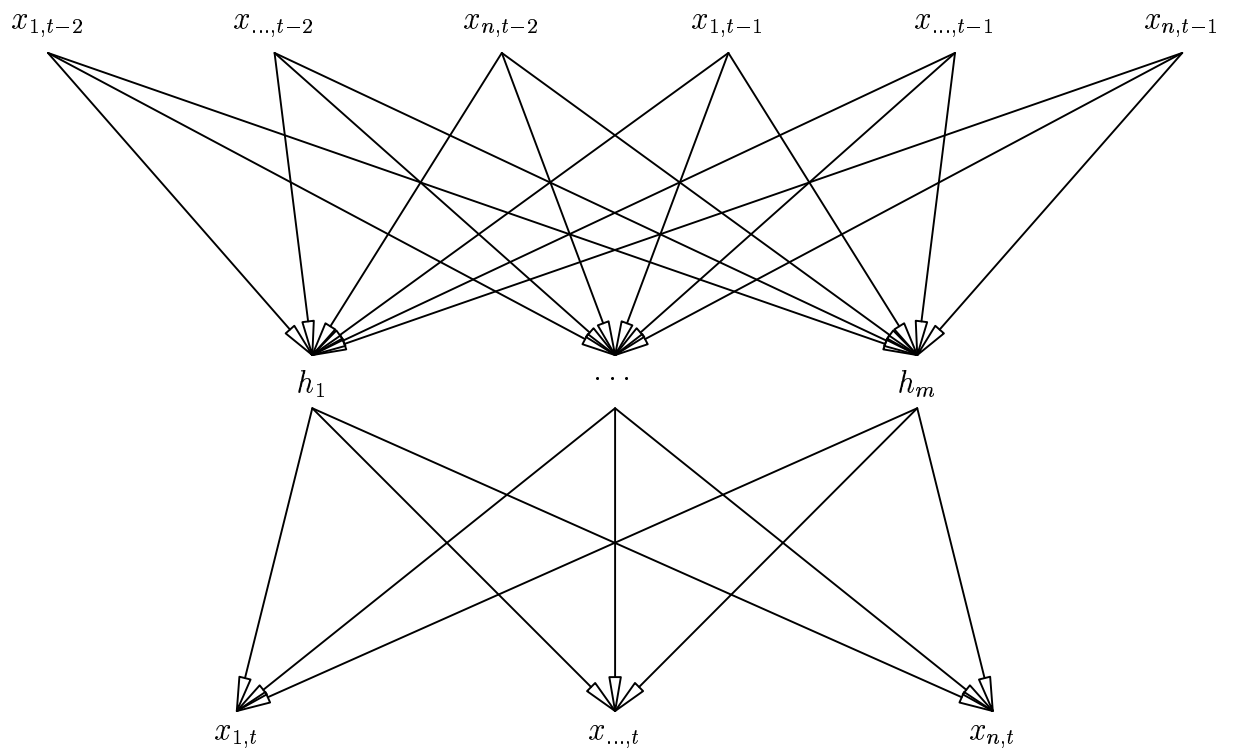


Figure 3.15: 2-layer MLP with an augmented input vector of 2 time steps.

# Chapter 4

## Joint optimization

In the chapter, we consider the dimension reduction and the forecasting problem jointly. The projection is done with respect to the prediction model considered.

### 4.1 Principal Interaction Patterns in a neural network context

For a description of the general PIP framework, please refer to section 2.1.3. The following discussion presents the case where the predictive model is chosen to be a multi-layer perceptron with one layer of hidden units. Because of the nature of the data set, a series of observations at regular time intervals, the discrete version of the equations is used.

Denoting the coefficients of the adjoint patterns  $D^T M$  as  $(\beta_{ji})$ , the projected data can be obtained by:

$$z_j = \sum_i \beta_{ji} x_i. \quad (4.1)$$

This can be interpreted as the first layer of a neural network, with input neurons the data  $x_i$ , no bias, and a linear transfer function.

Similarly, if  $\rho_{ij}$  denotes the PIP components, the reconstruction equation (2.5) is now the last layer of the network, with the same characteristics. The full network is represented in Figure 4.1.

Training proceeds by comparison of the network outputs  $y'_t$  to the data vectors  $x_{t+1}$ . With a sum of square error function, this scheme corresponds exactly to the framework presented in Section 2.1.3:

$$\varepsilon = \left\langle \left\| X_{t+1} - PG(\alpha, D^T M X_t) \right\|^2 \right\rangle_T. \quad (4.2)$$

The norm is Euclidean,  $G$  is the 2-layer MLP, and the  $\alpha$  parameters are the network weights and biases.

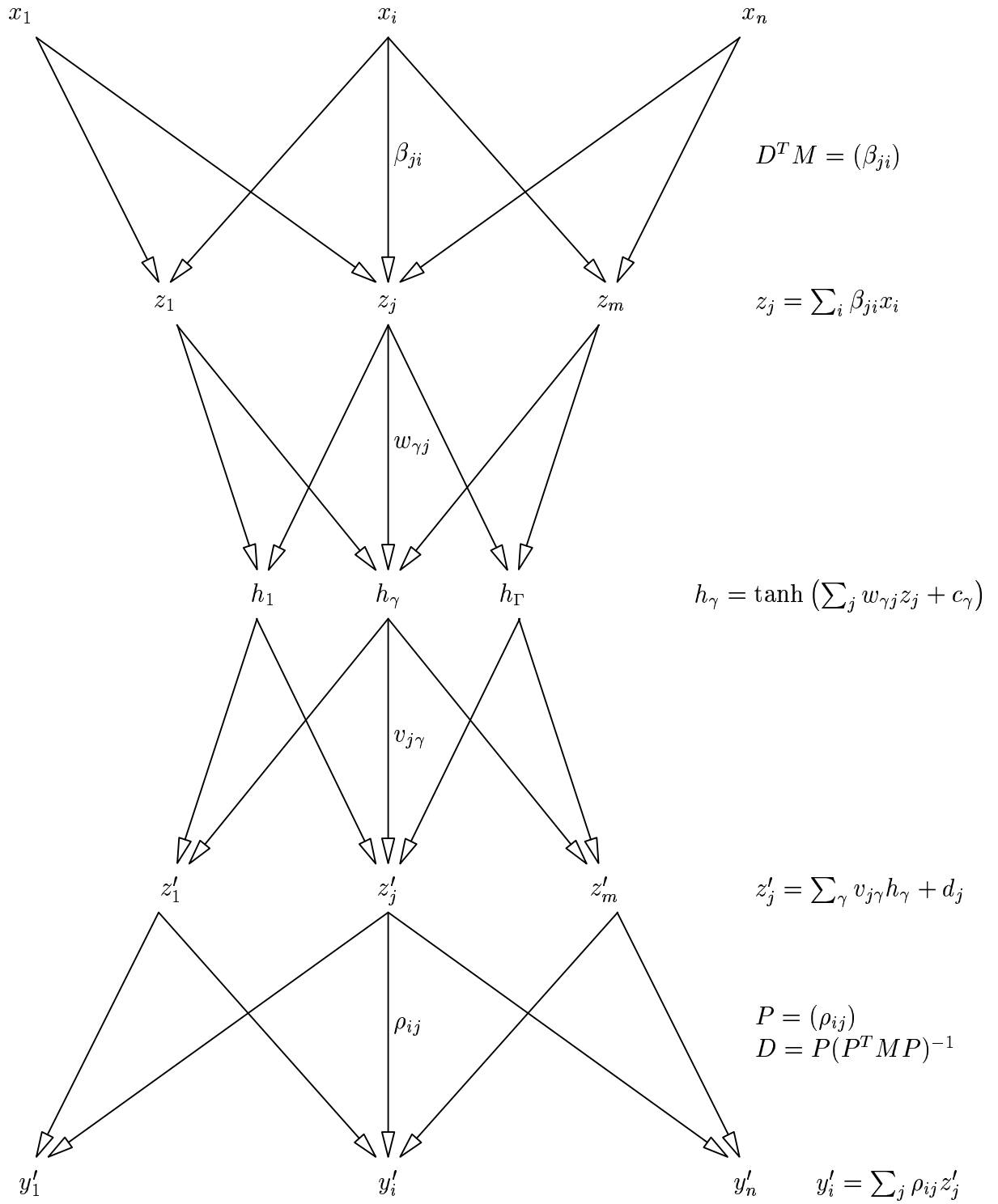


Figure 4.1: PIPs in a neural network context.

Unfortunately, the relation between the first and last layer weights,  $D^T M P = I$ , doesn't allow to write an individual  $\beta_{ji}$  in terms of the corresponding  $\rho_{ij}$ . Thus, even though the network is feed-forward, it is impossible to use back-propagation.

An additional complication is that the network is really huge. For example, when using the temperature and the geopotential height on the 3 levels, as done in Section 3.1.1, the input dimension is 7680. Supposing that optimizing the subspace basis together with the prediction model does indeed improve the data representation, we assume only 100 patterns are needed instead of the 150 EOFs. This requires a network with over 1.5 million weights. Since back-propagation is unusable, the training is prohibitively slow in practice.

## 4.2 A two-stage algorithm

To make up for the drawbacks of the full neural network implementation presented above, the idea is to decouple the algorithm. In a first stage, the projection and model are optimized so as to minimize the error committed in the reduced dimension space. Then, an adjoint model can be created, and the minimization process can be completed in the data space.

This new algorithm can no longer be described as a PIPs implementation, but similarities exist, especially in terms of the error functions considered.

### 4.2.1 Optimization in the reduced dimension space

Instead of optimizing directly in the data space, the idea is in the first stage to minimize the error committed in the projected space. The mean corrected, normalized data,  $\hat{X}$ , is used instead of the raw values. The corresponding error function is:

$$\varepsilon = \frac{1}{2} \left\langle \left\| D^T M \hat{X}_{t+1} - G(\alpha, D^T M \hat{X}_t) \right\|^2 \right\rangle_T, \quad (4.3)$$

multiplying each term inside equation (4.2) by  $D^T M$ , and the result by  $\frac{1}{2}$  for practical derivation purpose.

The assumption that the next state depends only on the previous one may not be true in practice. For this reason, the error function can be modified to include a lag vector (see Figure 4.2):

$$\varepsilon = \frac{1}{2} \left\langle \left\| D^T M \hat{X}_{t+1} - G(\alpha, D^T M \hat{X}_t, D^T M \hat{X}_{t-1}, \dots, D^T M \hat{X}_{t-lag}) \right\|^2 \right\rangle_T. \quad (4.4)$$

But, using this formula, there is a trivial possible minimum for the prediction. From the network point of view, setting  $\beta_{ji} = 0$  solves the prediction problem. But we're not looking for a null solution, and the goal is to solve the problem with a norm different than 0. A penalty term forcing the  $\beta_j$  vectors to have a fixed norm (of 1) has been introduced in the error measure. This doesn't affect the problem, since only the direction of the vectors is relevant as any scale factor is absorbed by the MLP weights.

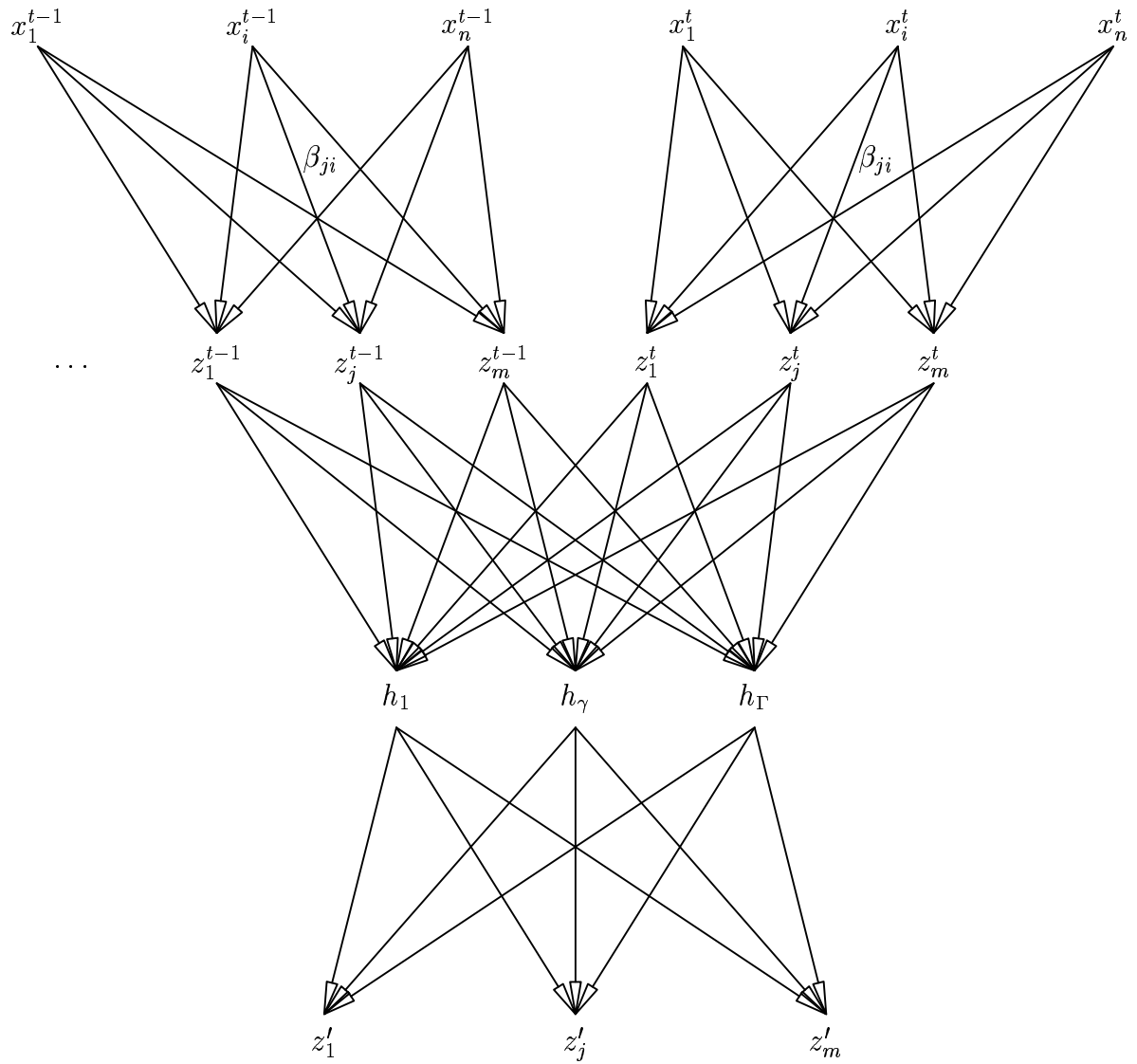


Figure 4.2: Network for minimizing the error in the subspace, with a lag vector.

Given the number of parameters, regularizing the network seems also necessary. The standard weight decay term, with a prior  $\frac{1}{\sigma^2}$ , is applied to the MLP weights. The final error function is:

$$\begin{aligned} \varepsilon = & \frac{1}{2} \left( \left\langle \left\| D^T M \hat{X}_{t+1} - G(\alpha, D^T M \hat{X}_t, \dots, D^T M \hat{X}_{t-lag}) \right\|^2 \right\rangle_T + \sum_j \left( \sum_i \beta_{ji}^2 - 1 \right)^2 \right. \\ & \left. + \frac{1}{\sigma^2} \sum_{\gamma j} (w_{\gamma j}^2 + v_{j\gamma}^2) \right). \end{aligned} \quad (4.5)$$

It is possible to use back-propagation for this network, with some care. Indeed,  $\beta_{ji}$  appears multiple times in the first layer, and the output comparison  $z^{t+1} - z^t$  also depends on  $\beta_{ji}$ . The back-propagation can be done as usual, except for the terms:

$$\frac{\partial \varepsilon}{\partial \beta_{ji}} = \sum_{l=1}^{lag} \delta_{z_j}^{t+1-l} x_i^{t+1-l} - \delta_j^{out} x_i^t + 2\beta_{ji} \left( \sum_i \beta_{ji}^2 - 1 \right). \quad (4.6)$$

In practice, a term similar to the variance maximization in equation (2.13) is also included:

$$\begin{aligned} \varepsilon = & \frac{1}{2} \left( \frac{1}{\left\langle \left\| D^T M \hat{X} \right\|^2 \right\rangle_T} \left\langle \left\| D^T M \hat{X}_{t+1} - G(\alpha, D^T M \hat{X}_t, \dots, D^T M \hat{X}_{t-lag}) \right\|^2 \right\rangle_T \right. \\ & \left. + \sum_j \left( \sum_i \beta_{ji}^2 - 1 \right)^2 + \frac{1}{\sigma^2} \sum_{\gamma j} (w_{\gamma j}^2 + v_{j\gamma}^2) \right), \end{aligned} \quad (4.7)$$

and the corresponding correction in the back-propagation formula.

The training phase is still very slow, but can be done. This method works well in batch mode, averaging computations over all training patterns, and with the scaled conjugate gradient algorithm. The patterns  $\beta_{ji}$  can be initialized using the EOFs, so that the starting point is relevant and not chosen at random. They are updated at each optimization step as any other network weight.

### 4.2.2 Optimization in the data space

The second stage of this algorithm is to complete the minimization process in the data space. The same MLP is kept untouched, but the direct patterns are computed from the complementary ones. The corresponding network structure is shown in Figure 4.3.

The error function for this model is a simple sum of squares function, with the same weight decay term as in the first stage:

$$\varepsilon = \frac{1}{2} \left( \left\langle \left\| X_{t+1} - Y' \right\|^2 \right\rangle_T + \frac{1}{\sigma^2} \sum_{\gamma j} (w_{\gamma j}^2 + v_{j\gamma}^2) \right) \quad (4.8)$$

Back-propagation can be used for this network without great difficulty. The main problem is after one iteration, the projection in the reduced space has to be recomputed, and thus the training set changes at each iteration. Adjoint patterns must be defined

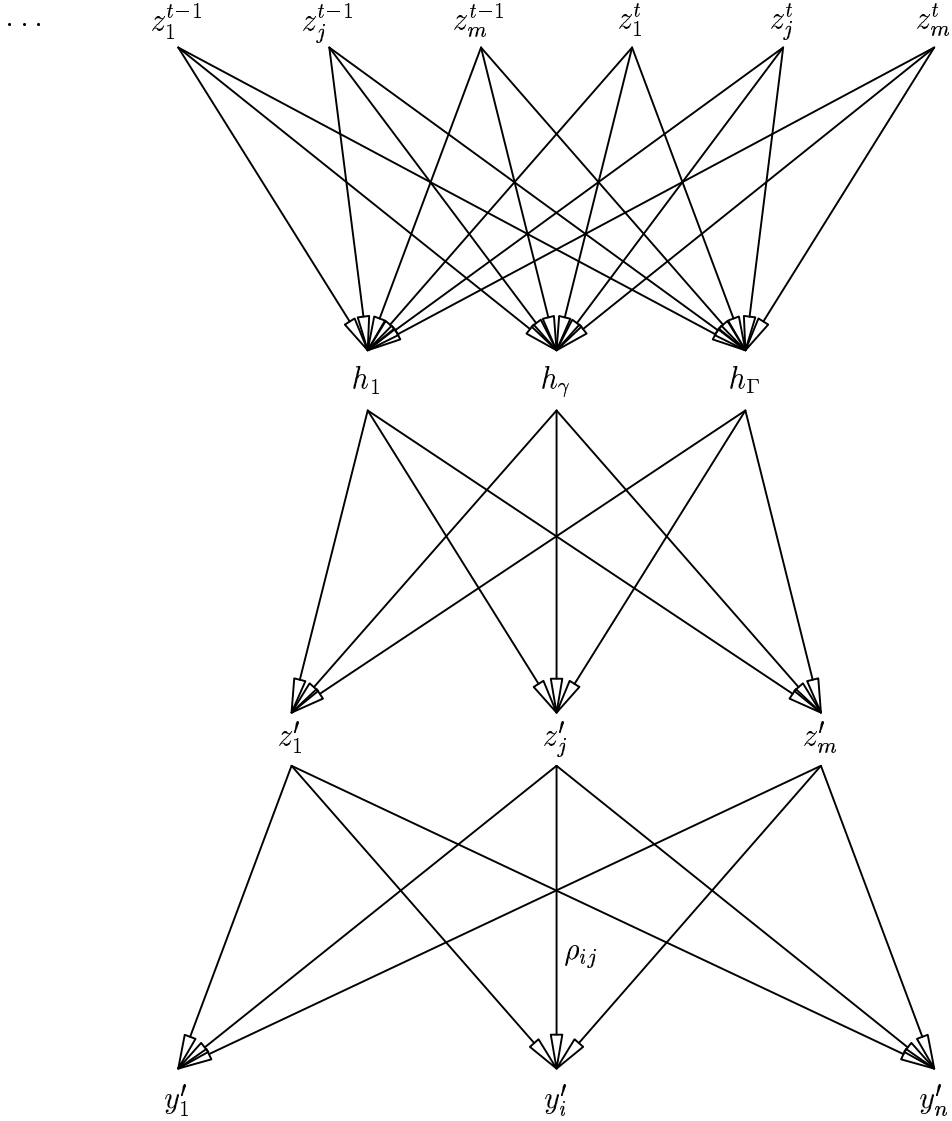


Figure 4.3: Complementary model, to optimize computations in data space.

from the newly updated direct patterns each time, due to the lack of a formula to update  $\beta_{ji}$  from  $\rho_{ij}$ . The full patterns must be re-evaluated:

$$B = D^T M = P(P^T P)^{-1} \tag{4.9}$$

Matrix inversions are computationally expensive, and algorithms like scaled conjugate gradient descent are not possible since the training set changes after each step. For this project the choice was to fall back on steepest gradient descent for each iteration, and not to try to adapt the scaled conjugate gradient algorithm. Hence the direction information between consecutive steps is lost, but then, since the training set changes, it may prove difficult to adapt the scaled conjugate algorithm.

Thus, this second stage is slower than the first, and the lack of efficient training method means the training error isn't reduced much. It has however been noted in

practice that using this second stage does indeed reduce the error committed, and that doing the 2 steps in sequence rather than alternatively gives the best results.

An interesting use of this complementary model could be on-line training, or for diagnostic purposes. The past observations can be used to train a MLP using the first stage algorithm, and then this network created. This can be done by keeping the same MLP layer, and pseudo-inverting the  $\beta_{ji}$  matrix of the first stage to produce the  $\rho_{ij}$  used here according to Equation (4.9). Predictions in data space can be computed, and as new observations become available, it is possible to update the model. If no new observations are available, it is still possible to make a prediction without the analysis step. It is also possible to reset the model to a new value if the error gets too large, by repeating the first stage on a new training set and inverting the result.

On-line training and updated models can also deal with non-stationarity in the data. In the method presented here, the mean and variance of the series would have to be recomputed when resetting the model.

# Chapter 5

## An artificial data set

The algorithms presented here all have problems dealing with the high dimension of the data. Even when reducing the size of the working space to the order of hundred variables using EOFs, this difficulty remains.

In order to overcome this problem, a small artificial data set was created, and comparisons between the different algorithms were undertaken.

### 5.1 The Lorenz equations

To build the artificial data set, a sufficiently unstable process was needed to simulate the real prediction difficulties. A well known set of equations in meteorology which exhibit sensitivity to initial conditions are the Lorenz equations. Their analytic expression is:

$$\frac{dx}{dt} = 10(y - x) \tag{5.1}$$

$$\frac{dy}{dt} = 28x - y - xz \tag{5.2}$$

$$\frac{dz}{dt} = xy - \frac{8}{3}z \tag{5.3}$$

The dynamics of those equations are driven by an attractor, of which a 3D representation is shown in Figure 5.1.

These equations were integrated using the Ordinary Differential Equation Solver of the software package Scilab. This routine "automatically selects between non-stiff predictor-corrector Adams method and stiff Backward Differentiation Formula (BDF) method. It uses non-stiff method initially and dynamically monitors data in order to decide which method to use"<sup>1</sup>. This solver was chosen over a simple Runge-Kutta algorithm, which was implemented first. In fact, since noise is added later to the computed values so as to simulate the real input noise on the meteorological measurements, the precision required for the trajectories and thus the method choice wasn't that important.

---

<sup>1</sup>Text cited from the Scilab ode function help file.

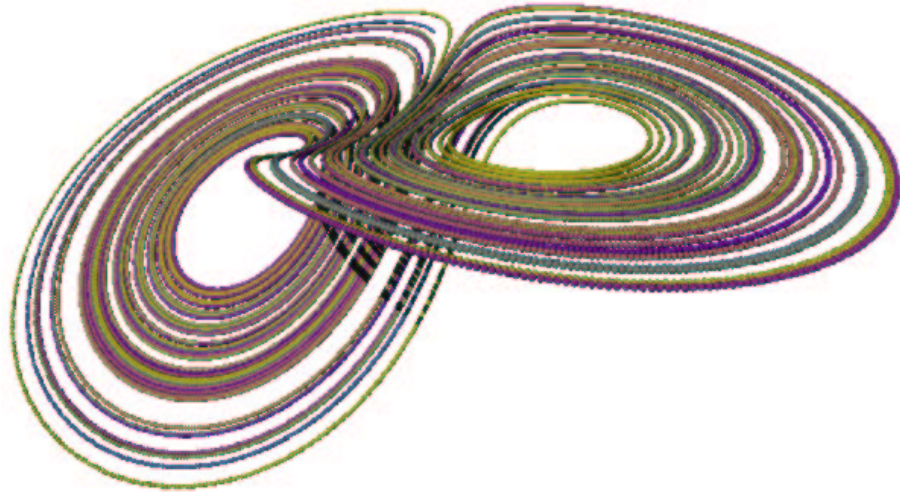


Figure 5.1: 3D representation of the Lorenz attractor.

The starting point for the series was chosen close to the attractor, but the first 5000 iterations were discarded nevertheless to ensure a non-transitory trajectory. The 10000 next points were chosen for the training set, and 5000 others for the test set. The step size chosen corresponds roughly to 60 points per loop.

Once the series were computed, a  $20 \times 3$  projection matrix was randomly chosen to consider this trajectory in a 20-dimensional space. The matrix was tested to be non degenerated. Finally, random noise<sup>2</sup> was added in this space, because there is noise in the real data set, and also because it renders the data fully 20-dimensional.

## 5.2 Results

### 5.2.1 Empirical Orthogonal Functions

The reconstruction error experiment presented in Section 3.1.1 was tested on this artificial data set. The results are presented in Figure 5.2.

There is a distinct break at 3 components, suggesting the attractor has been captured correctly. The residual error tails off and is zero when the EOF space equals the data space, with 20 dimensions. The EOFs were defined on the training set, but span the validation set equally well. This isn't surprising considered the way those series were generated; they lie on the same attractor.

On the other hand, this property wasn't so well verified in the real world data set. There, the reconstruction error on the validation set was distinctly higher than on the training set. See Figure 3.2 and the corresponding section.

In any case, when projecting the validation series of the artificial data on the EOFs, the emerging structure corresponds to the attractor. Figure 5.3 shows the projection in the 3 planes defined by the 3 first EOFs, and the corresponding 3D representation.

<sup>2</sup>Gaussian distribution, 0 mean and variance 1.

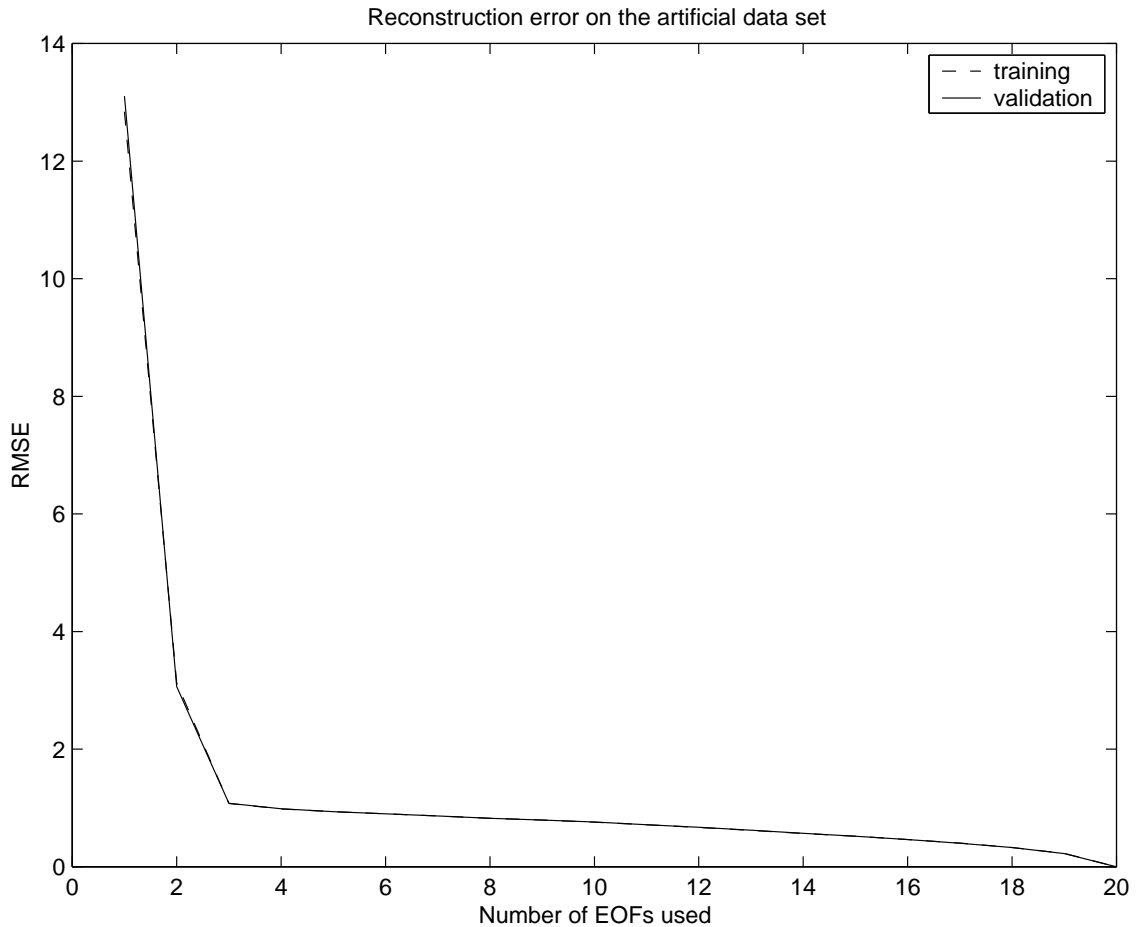


Figure 5.2: EOF reconstruction error on the artificial data.

It is interesting to observe how the directions selected by the EOF process lie along the attractor. Along EOF 1, the values alternate from negative to positive regardless of the loop that the current point is in. The second EOF defines two modes, one for each attractor loop. EOF 3 represents the last dimension. One can suspect the orthogonality condition to constrain EOF 3 to a slight distortion, since the shape of the attractor suggests an orthogonal basis isn't the most appropriate.

### 5.2.2 2-stage PIP algorithm

Even on such a small data set, with only 20 dimensions, back-propagation is necessary for the computations, for speed reasons. The 2-stage algorithm presented in Section 4.2 was run instead of the full PIP problem. So as to increase the convergence speed, the starting point for the reduced dimension space was chosen to be the one defined by the EOFs. The one-step ahead prediction structure is shown on Figure 5.4.

The basis vectors, still called PIPs for convenience<sup>3</sup>, are altered, both in norm and

<sup>3</sup>This may be an improper usage of the PIP terminology. No proof was given whether the 2-stage algorithm converges to the same basis vectors, or not, as the full PIP problem.

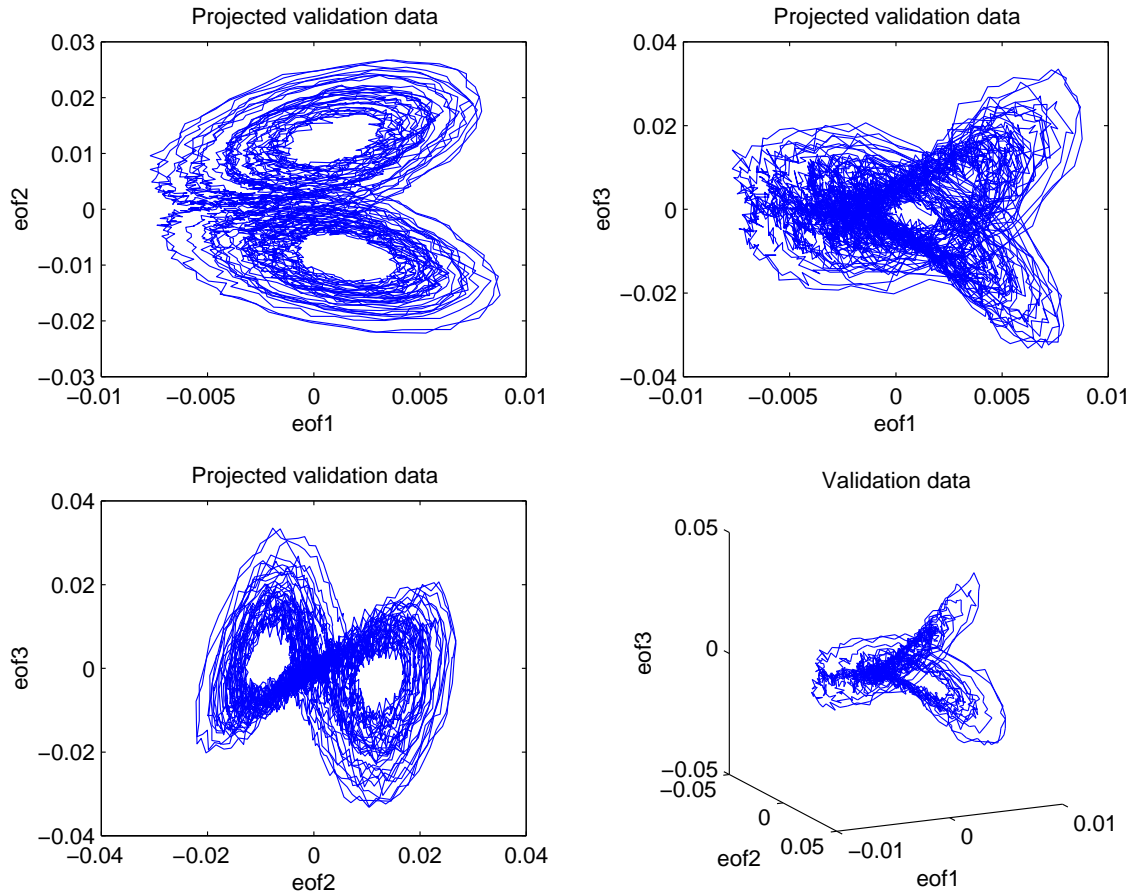


Figure 5.3: Artificial data structure as captured by the EOFs.

direction. The basis isn't orthonormal anymore. The same remarks as for the EOFs are still valid, though. This means the process hasn't gone too far from its starting point, and the ordering of the dimension captured is the same. This also means the starting point is important, and EOFs were already performing quite well.

### 5.2.3 Prediction comparison

The methods compared are a 5-order Auto-Regressive model run on the EOF space, a MLP with lag 5 on the same subspace, and the 2-stage algorithm on the full 20-dimensional space but with the same MLP layer. Values were also computed for a persistence prediction, which is commonly used as a benchmark in weather forecasting, and consists in using the current values as an estimate for the future values. The equivalent of the climatology, labelled as "no prediction", where only the seasonal component is taken as the approximate for the prediction, was also computed using the mean value of the series. Results are presented in Figure 5.5.

Of all the models, only the MLP gives worse results than the persistence. This may be due to a problem during training, but repeated experiments with different starting points and number of hidden units give approximately the same curve. Another

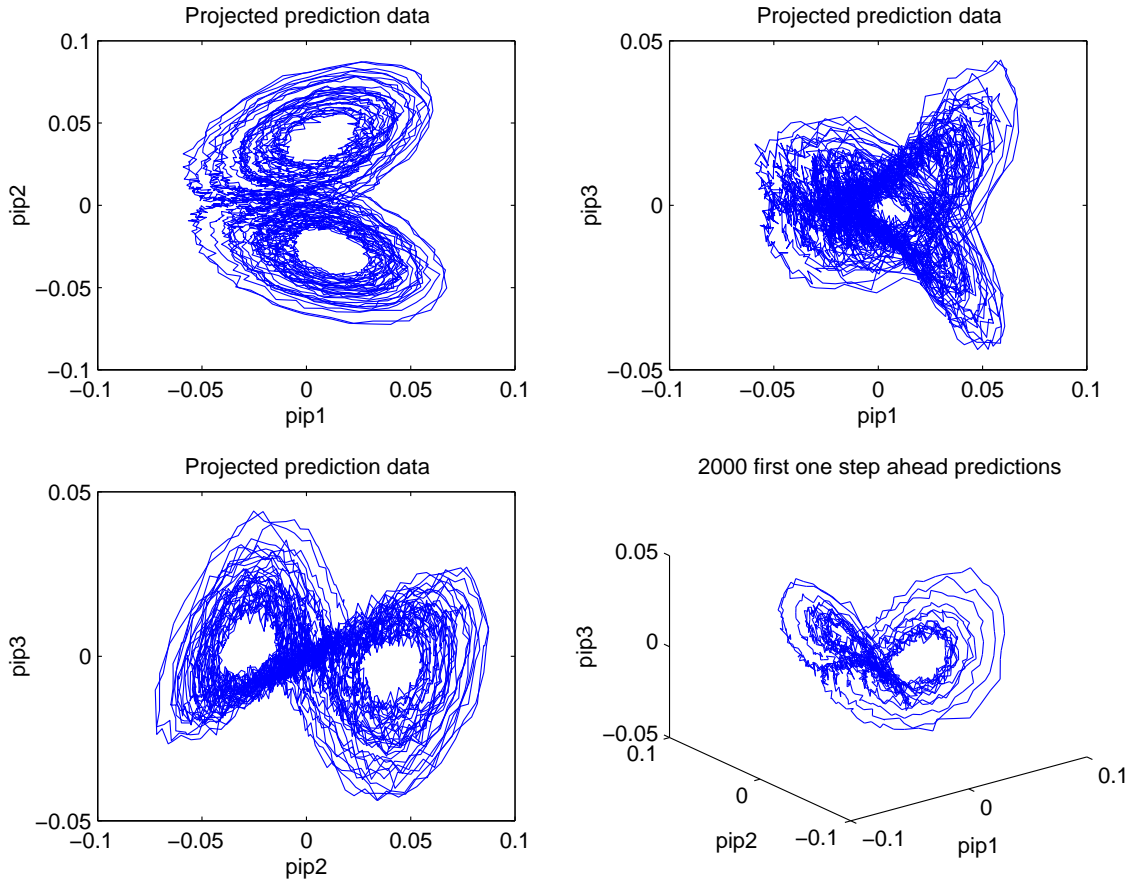


Figure 5.4: One-step ahead prediction structure.

explanation could then be the number of free parameters in the model considered, and effect of local minima on the training process. In the long term<sup>4</sup>, the MLP still ends beating the persistence, and doesn't blow up above the "no prediction" climatology equivalent.

The 2-stage algorithm is comparable to the MLP, both were run using the same structure (number of hidden units, especially). Thus, the increase in performance is due to the combined optimization of the subspace together with the model. This validates the idea the projection chosen has to take in account the dynamics, or at least the evolution model chosen, rather than being based only on the statistical distribution of the data.

The Auto-Regressive model gives the best results on this experiment. The corresponding curve stays below the 2-stage algorithm one, but has nearly the same shape with a constant difference. No explanation was found for this phenomenon, but a hypothesis, untested, could be the existence of a 'perfect' prediction method, modeled approximately by the AR(5) and the 2-stage algorithm together.

In practice, it appears that simple models like the AR(5) used here can do better than more complex ones which are theoretically more powerful.

<sup>4</sup>A reminder how to make multiple step ahead predictions is given in Appendix B.3.

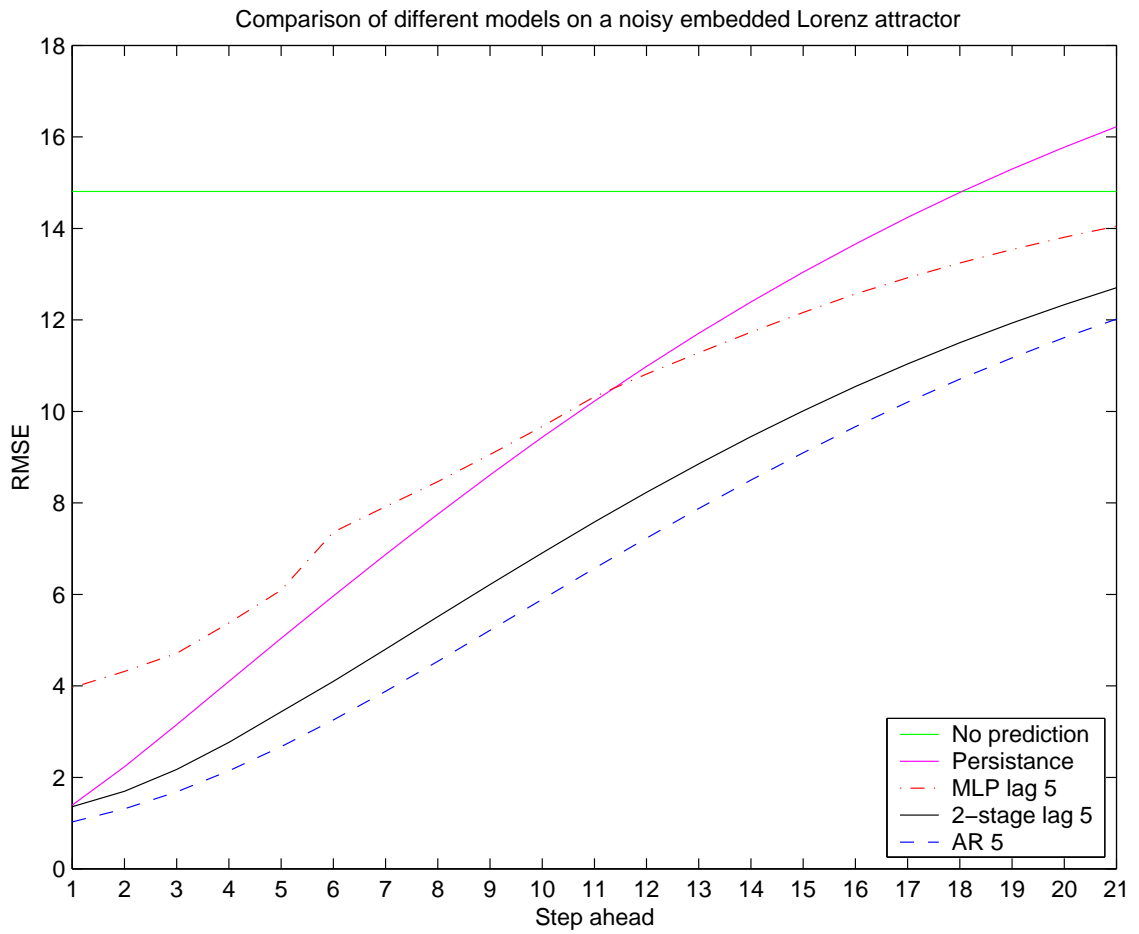


Figure 5.5: Comparison of prediction methods on a Lorenzian system.

# Chapter 6

## Real data

### 6.1 Selected data

The data set presented in Section 1.2 contains 6 variables. Among those, the wind velocities have a vector field nature, and are too different from geopotential height and temperature. They are kept apart. Relative humidity could have been used, but it looks to have a very different scale of variability<sup>1</sup> and with temperature and geopotential height on 3 levels, the data space dimension is already 7680. This is about the resources limit for the NCRG servers, and thus the logical choice. Temperature and geopotential height are also usually considered very important and informative variables.

EOF computations were carried out as detailed in Section 3.1.

As done on the Lorenzian system, different models were investigated for forecasting future values. The climatology was computed by fitting a sine with period 1 year to each component, as presented in 3.1.2. A 5-order autoregressive model and a multi-layer perceptron were used in the 150 EOF space, and the data reconstructed.

The 2-stage algorithm was also run. Unfortunately, the data dimension was still too high to enable it to converge in a reasonable time. With respect to this, a projection on 500 EOFs was performed as a pre-processing treatment on the data, in the hope the information loss was sufficiently low. Figure 3.1, the reconstruction error curve on the real data set, indicates this corresponds to the plateau part, thus presumably only noise isn't captured with 500 EOFs. See Section 3.1.1. The results are presented in Figure 6.1 for the geopotential height at 500mb. The 5 other variables behave similarly.

The AR(5) model gives the best results. It remains below the persistence prediction. This means the dimension reduction and forecasting were good enough to give a better information on the future of the series than merely keeping today's values as an estimate for tomorrow.

The multi-layer perceptron starts above the persistence, but even when it beats it at 3-day ahead, the prediction is so poor as to be the same level as the climatology. Interestingly, the MLP does even give worse results than the climatology. This means the MLP makes worse predictions than using the yearly averaged values as an estimate

---

<sup>1</sup>Please refer to Appendix A for data plots.

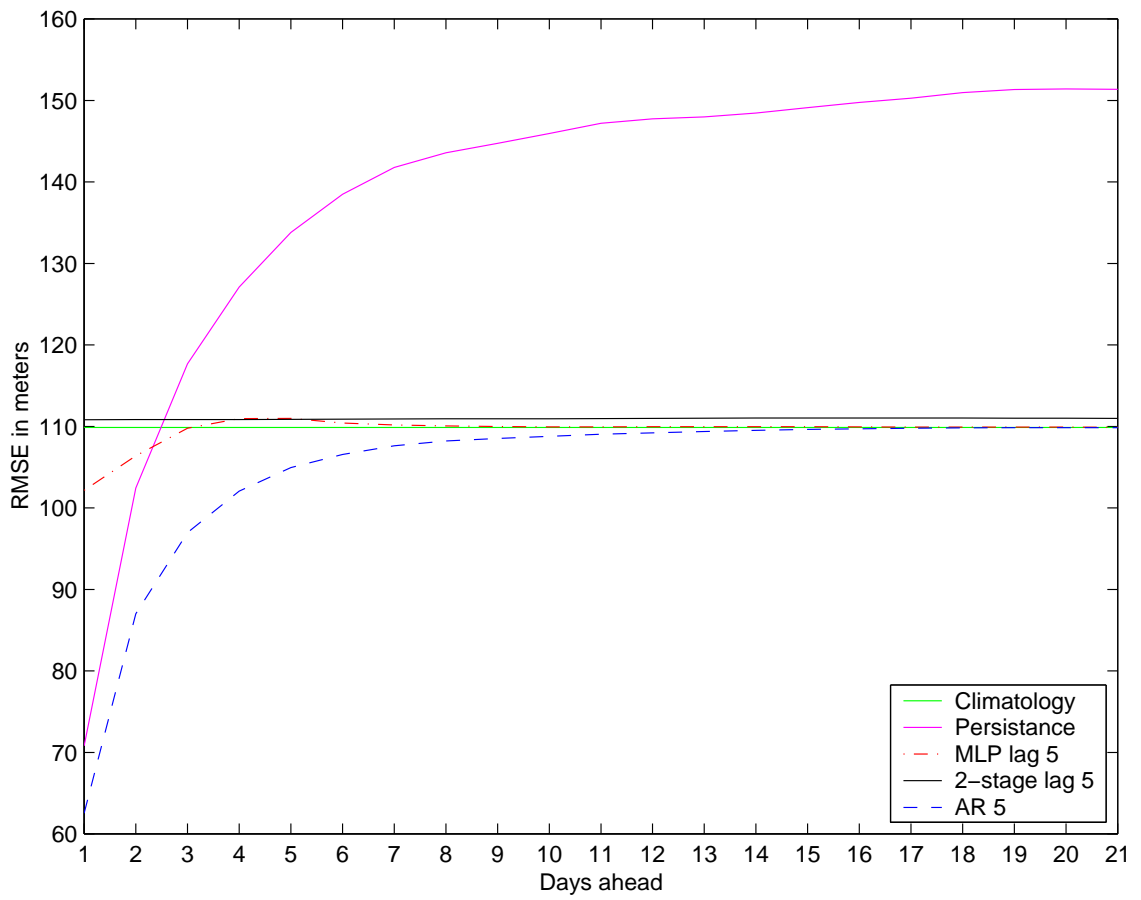


Figure 6.1: Comparison of different forecasting models on real data.

for the predicted values. The curve then joins the climatology, which means that predictions many step ahead are damped down to 0 and only the climatology remains.

As for the artificial data set, the results for the AR(5) are better than the MLP. Unfortunately the 2-stage algorithm simply could not be trained due to computational power limitations, and gives a nearly constant bad result for all forward predictions in time.

Nevertheless, all the methods presented here do not compete with the predictions from the ECMWF<sup>2</sup> computed on full atmospheric models with 4D variational methods, and powerful computers. There, the predictions for geopotential height at 500mb start at 10m for the first day-ahead, and reach 100m after 10 days [1].

An interesting conclusion is that after 2 weeks, all methods seem to reach climatology level, so cannot do better than a yearly averaged prediction. There is still work to do to increase the forecasts accuracy in the 2-3 weeks medium range domain.

In the light of those results, another experiment was carried on. Sub-sampling the data allows more dimension reduction, and is presented in the next section.

## 6.2 Sub-sampled real data

Since the full data set is too large for the 2-stage algorithm to converge in a reasonable time, an idea is to sub-sample it. The aim here is to be able to compare the different methods on real data, instead of the artificial data set presented in Chapter 5. The variable chosen was geopotential height at 500mb, a standard level in meteorology, alone. This brings the dimension of the data to 1280. The region selected was sub-sampled every 5 degrees in space, which further decreases the dimension down to 320. The training data set was limited to the 5 years from 1984 to 1989, the validation set remaining the same last five years, 1990 to 1994.

As before, the MLP and 2-stage algorithm were run with the same parameters, number of hidden units and weight decay coefficient especially. The observed difference can thus be attributed to the effect of optimizing the reduced dimension space basis together with the forecasting model, as for the results on the Lorenzian system in Section 5.2.3.

Results are presented in Figure 6.2. For the predictions before 7 days ahead, the 2-stage algorithm performs slightly, but not really significantly, better than the MLP. At 4 days ahead, the error is above the climatology, thus not really relevant. More importantly, the MLP ends up predicting a zero value, hence the climatology after data reconstruction, whereas the 2-stage algorithm still produces results. This may be attributed to the capture of part of the data dynamics, and suggests a comparison to the persistence prediction instead of the climatology.

All the algorithms do better than the persistence, which wasn't the case for the original, not sub-sampled, data set experiment in the previous section. This probably shows the importance of a complete training, since the neural networks give better

---

<sup>2</sup>European Center for Medium-Range Weather Forecasting

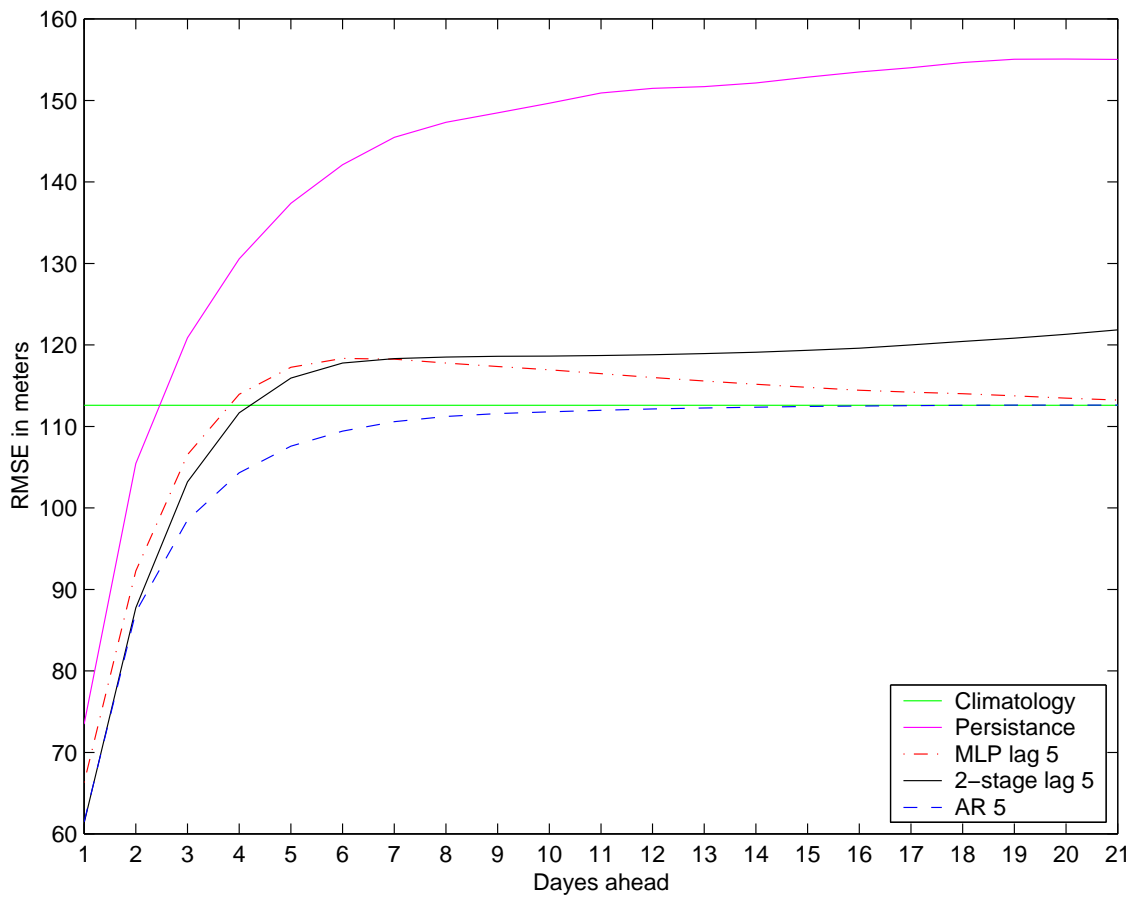


Figure 6.2: Comparison of different forecasting models on subsampled data.

results despite the loss of information inherent to the sub-sampling operation. This also validates the use of these algorithms, a better prediction is obtained than the immediate computations of persistence or climatology. Nevertheless, the 3 algorithms considered don't compete with the ECMWF results in [1] using all available data previously mentioned.

The AR(5) gives the best results. However, for the first 2 days ahead, the 2-stage algorithm does equally well. This is encouraging, but more experiments would be needed to confirm or invalidate the ability of the 2-stage algorithm to potentially produce better forecasts than simpler models.

# Chapter 7

## Conclusion

The aim of this project was to study the possibility of reducing to dimension of models and data sets in weather forecasting. Several points were made, concerning both aspects, and a new algorithm was presented to decouple the PIPs problem in a neural network context.

Working on multiple variables simultaneously in an EOF reduction enables us to take in account part of the relation that exists between them, and may allow a greater dimension reduction. The choice is made on the acceptable threshold for the reconstruction error. Figure 3.1 illustrates this effect. If a low reconstruction error is important, then treating each variable separately is necessary. In that case, EOFs have to be defined for each variable, and the total number of components, the number of variables times the number of EOFs retained for each, will be significantly higher than when considering all the variables together. In this case, the minimum error committed is higher, but EOFs are common to all the variables.

Another lesson of Chapter 3 concerns the data reduction problem itself. When predictions are made with the reduced data set, a purely statistical dimension reduction algorithm based on the distribution of the data may not be enough. The method chosen has to take in account the internal dynamics of the data and estimate the projection concurrently. The projection may otherwise not be significant with respect to the evolution models considered.

In the light of the comparison results between the different algorithms presented in this document, the edge between practice and theory can be observed once more. Simple models like the AR(5) can do better than complicated ones theoretically more powerful. However, when the models are comparable like the MLP and 2-stage algorithm, the effect of optimizing the projection together with the prediction model can be measured. To do that, all the parameters defining the prediction model, in this case the MLP and the corresponding layer of the 2-stage algorithm, are identical. In particular, in all the computations, the number of hidden units and the weight decay coefficient were identical for the 2 networks. No validation set was used in this project, out of sample predictions were felt to be sufficient. The difference in the results could be attributed to the subspace basis optimization, and since the 2-stage algorithm gives better results than the MLP in EOF space, the theory of optimizing jointly model and

subspace is verified.

Concerning the 2-stage algorithm, the idea of optimizing the error committed in the reduced subspace in a first time may be applied with other models than a neural network. This is a rather interesting extension to the PIP method, in the sense that it can improve the computation time required to carry out the experiments. However, convergence is not guaranteed to define the same subspace as the PIPs.

Other limiting factors were encountered. The region selected might be too small, and boundary effects could have perturbed the experiments. Similarly, external factors not taken in account may have influenced the results. Those drawbacks are related to the tradeoff between precision and computational power required, it wasn't possible to increase the data dimension. The numerical problems encountered are presented in Appendix B, but the main concerns were speed and memory requirements. A specific C++ library<sup>1</sup> was created to address these problems, and used for the computations.

The general impression emerging from this project is that there seems to be no efficient way to reduce the dimension in terms of the error committed, and that unfortunately the huge models dealing with all the data available are required for precision. On the other hand, reducing the dimension of the models and data sets can be a way to carry out experiments with limited resources. More work is necessary to determine whether the precision obtained on reduced dimension models is sufficient, but with some improvement this could provide a way to compute ensemble predictions cheaply.

Extensions to this project could be considered in this area, the ultimate goal being to improve the error to an acceptable point, even if not as good as what can be obtained using the full models, while keeping the computations affordable for a small computing infrastructure. The alternative approach could be to further extend the more complicated models, and run them in the powerful prediction centers, in the hope of increasing the accuracy of the forecasts.

The idea of a scale-dependent representation of the atmosphere also came up at early stages of this project, and could be an alternative study. Future works could consider the problem along these lines, possibly with wavelet decompositions and fractal extrapolations, so as to represent more faithfully the nature of the atmosphere.

In any case, this project can serve as a basis for further experiments.

---

<sup>1</sup>This library is available at <http://CheapMatrix.sourceforge.net>

# Appendix A

## Data plots

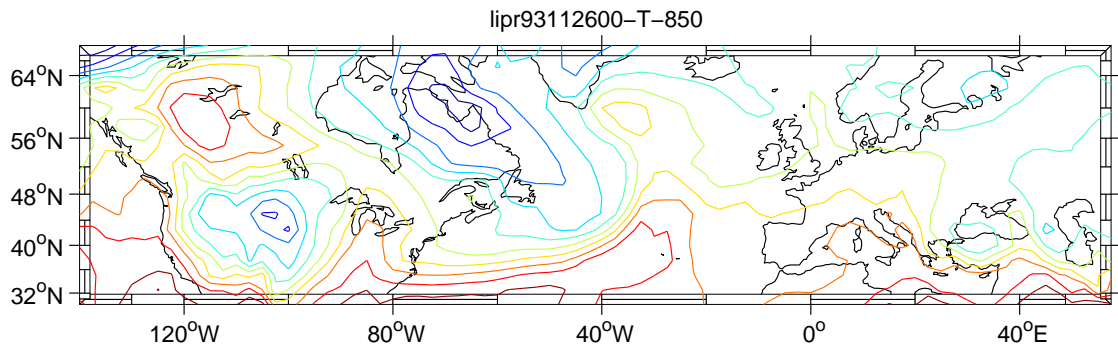


Figure A.1: Temperature at 850mb on November 26<sup>th</sup>, 1993.

Figure A.1 gives the temperature contour levels of November, 26<sup>th</sup>, 1993. Maxima/minima can be identified, and the contours around them aren't just smooth transitions but rather occur in sharp gradients.

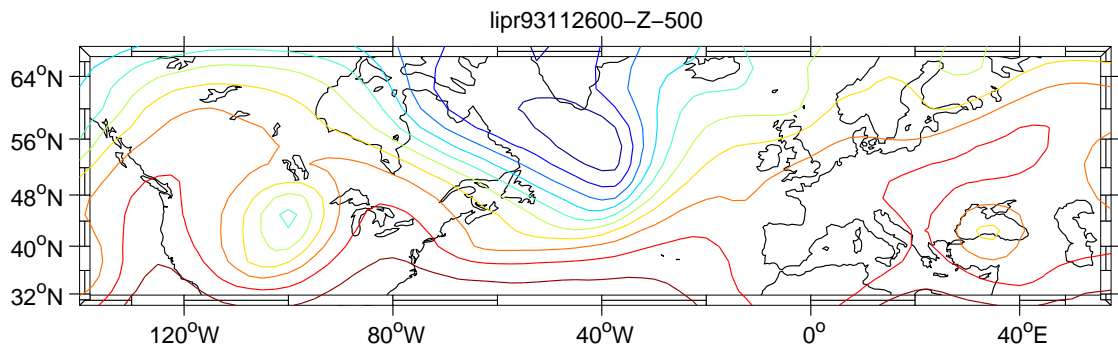


Figure A.2: Geopotential height at 500mb on November 26<sup>th</sup>, 1993.

Figure fig:lipr93112600-Z-500 shows the geopotential height at 500mb at the same as Figure A.1. Though it isn't the same variable, centers can be found at approximately

## APPENDIX A. DATA PLOTS

the same places, hinting at a possible correlation between them. The field at  $500mb$  is also much smoother, a feature typical of the atmosphere.

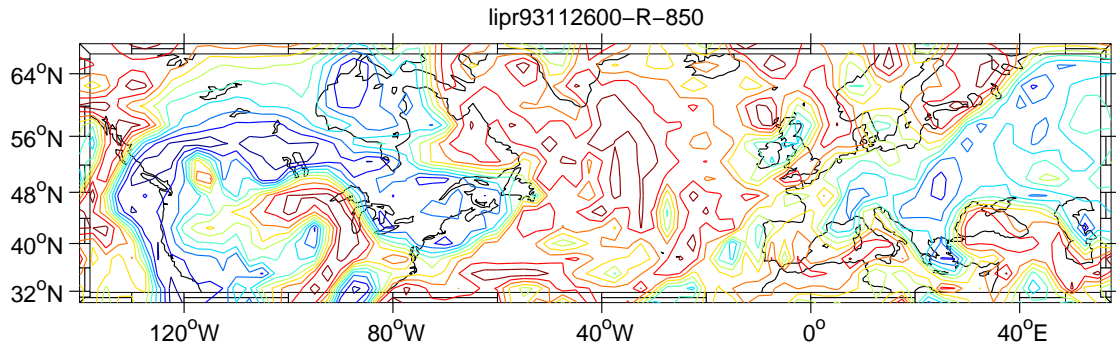


Figure A.3: Relative humidity at  $850mb$  on November 26<sup>th</sup>, 1993.

The relative humidity has much shorter characteristic scales than the other variables, without any clearly identifiable center. This is one reason why this present study didn't take it into account.

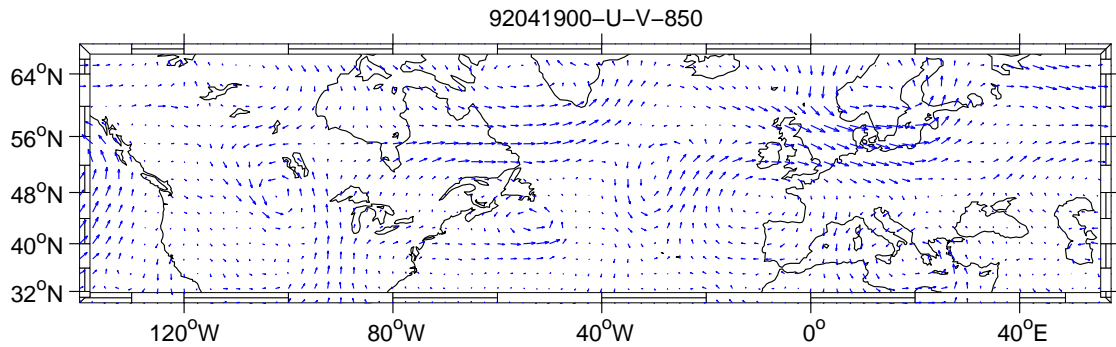


Figure A.4: Wind vector field at  $850mb$  on April 4<sup>th</sup>, 1992.

$u, v$  wind velocities are plotted as a vector field, and because of that nature have also been kept apart. The interesting point on this representation is the appearance of a west-east wave-like propagation for the winds.

# Appendix B

## Coding issues

### B.1 Presentation

It's not surprising to discover that once more, data size and the speed of computations are the main problems. The implementation of the algorithms presented in this document must address these factors, in addition to usual coding issues like code re-usability.

Early attempts to use the popular interpreter MATLAB were rapidly discarded as the memory usage was poorly managed. Additionally, MATLAB isn't famous for its speed, especially when performing iterative processing. A search over the Internet for more appropriate tools wasn't successful.

A solution was thus specially designed for this project. It had to be able to handle the memory efficiently, be easy to use and reuse, and allow scientific computing. As customary in computer sciences, 'reinventing the wheel' combines both a waste of time and a need to solve well-known problems. Adding up those constraints, the natural solution was to create a library itself interfaced to a pre-existing package.

The choice retained was the excellent public domain Linear Algebra Package, LAPACK<sup>1</sup>, as a back-end for typical problems like Singular Value Decomposition which are required for this project<sup>2</sup>.

The C++ language was retained, for its efficiency, its popularity, and because Object Oriented Programming reduces the re-usability and ease-of-use constraints. The C translation of LAPACK was preferred over the original FORTRAN code for obvious interfacing reasons, though with a little effort the C++ library written for this project is now compatible and can be linked with FORTRAN code too. This enables platform specific implementation of the BLAS<sup>3</sup> to be used seamlessly, a great benefit on parallel architectures for instance.

The library itself contains a Matrix and a Vector classes, with all arithmetic operations and standard functions overloaded. These classes were designed so that a

---

<sup>1</sup>LAPACK and many other scientific public domain libraries can be found on the Netlib repository, at <http://www.netlib.org>.

<sup>2</sup>In the case of SVD, this routine was needed to compute the EOFs.

<sup>3</sup>Basic Linear Algebra System, the core routines used by LAPACK.

huge matrix may be loaded partially in memory at a given time, transparently, thus drastically reducing the memory consumption<sup>4</sup>. Extensions exist: a utility module, a link to LAPACK functions, and an optimizer framework. Features were extended beyond what's strictly necessary for this project and allow comfortable computing in C++, but functions are still missing in other domains. As usual in Open-Source development<sup>5</sup>, the reader is free to contribute! The library is located at <http://CheapMatrix.sourceforge.net>.

## B.2 Algorithm implementation example

A fairly complete documentation with examples is packaged with the library, and can also be consulted on-line. The programs used for this project can be found at <http://nicolas.brodu.free.fr/mscr>.

As an example, the following piece of code was used to remove the seasonal components. This isn't an introductory example at all, yet shows the philosophy behind the optimizer framework.

```
// Model to fit a sine with the given period to a data vector
// The model is of the form S(t) = alpha * sin (omega * t + phi);
class SineModel : public OptimizeTarget
{
protected:
    double omega;
    // Evaluate error and gradients on this series
    MatrixType evaluate(const Matrix& data, const Matrix& target,
EvaluateTarget what = EVALUATE_BOTH);
public:
    SineModel(double _omega);
};

SineModel::SineModel(double _omega) : omega(_omega)
{
    // Set initial values to something random
    values = randn(2);
    gradients = zeros(2);
}

// Evaluate error and gradients on this series
MatrixType SineModel::evaluate(const Matrix& data, const Matrix& target,
EvaluateTarget what)
```

---

<sup>4</sup>At the expense of the mass memory (hard drive) space, and a decrease in performance when swapping.

<sup>5</sup>This library is released under the GNU Library Public License. See <http://www.gnu.org>.

## APPENDIX B. CODING ISSUES

```
{
    Vector common = omega * data + values(2);
    Vector common2 = sin(common);
    Vector common3 = values(1) * common2 - target;
    if (what & OptimizeTarget::EVALUATE_GRADIENT) {
        gradients(1) = dot(common3, common2); // alpha derivative
        gradients(2) = dot(common3, values(1) * cos(common)); // phi
    }
    if (what & OptimizeTarget::EVALUATE_ERROR) {
        return 0.5 * dot(common3, common3); // error
    } else return 0;
}
```

The vectors are created and manipulated like any C++ object, operations and functions like `cos` can be applied to them directly. Utility functions are available, for example `randn(2)` creates a 2-dimensional vector whose elements are drawn from a Gaussian distribution with mean 0 and variance 1. The model provides its own method `evaluate` to compute the error function, its gradient, or both, according to the need of the optimizer<sup>6</sup>. No extraneous operations will be requested, and calculations common to the error function and its gradient can be reused with this system, improving the optimizer speed. The actual optimization call, together with the removal of the seasonal component from the data, is:

```
const double omega = 2.0*M_PI/365.25;
SineModel sine(omega);
SCG scg; // Scaled Conjugate Gradient optimizer
scg.option(1) = 100; // For SCG, number of iterations
Matrix params(2,m);
for (int i=1; i<=m; i++) {
    scg.optimize(sine, range(1,t),
        training(i,all).transpose()-mean_correction(i));
    Vector alphi = sine.copy_values();
    // Correct all the data (extend to the validation set as well)
    data(i,all) -= alphi(1)*sin(omega*range(1,n).transpose()+alphi(2))
        + mean_correction(i);
    params(all,i).copy(alphi);
    cout << "." << flush;
    if (!(i%10)) cout << " " << flush;
    if (!(i%50)) cout << i << endl;
}
cout << " " << m << endl;
```

---

<sup>6</sup>For the reader not familiar with C programming, `EVALUATE_BOTH` is a binary mask combining `EVALUATE_GRADIENT` and `EVALUATE_ERROR`.

To help the reader, the notation `data(i,all)` is equivalent to the MATLAB `data(i,:)`, and means the  $i^{\text{th}}$  row of the Matrix `data`. `range(1,n)` is equivalent to the MATLAB `1:n` and means all natural numbers from 1 to  $n$  included. More information about the library syntax can be found in the online documentation.

### B.3 Notions revisited in practice

This section presents some caveats and tips for the programmer, when implementing some of the notions presented in this document.

**EOF** Empirical Orthogonal Functions are presented in Section 2.1.2, with the practical algorithm using SVD<sup>7</sup>. This is exactly what has been done for this project. The useful property to remember is that EOFs are orthogonal, with norm 1. This means projecting the data is just a matter of multiplying by a transpose matrix, with some care concerning the singular values. More exactly, the SVD of a matrix  $D$  is in the form  $D = U * S * V^T$ , so  $V^T = S^{-1} * U^T * D$  with  $S$  a square matrix whose diagonal elements can be stored in a vector in practice. The EOF vectors are the columns of  $U$ , the associated series the lines of  $V^T$ . The reconstruction from a selection of EOF components is straightforward, but must also take in account the singular values.

**Optimization** The first step is to code the error function considered, using the model parameters to optimize. Very often, part of the computations can be reused for the error gradient. Thus, it is a good idea to modify the optimization algorithm to calculate simultaneously the error function and its gradient whenever possible. This has been done in this project, the code for scaled conjugate gradients was adapted from the MATLAB package NETLAB<sup>8</sup>, and this C++ version includes the joint computation of error and gradient improvement.

**RMSE** The Root Mean Square Error for a variable is averaged over all error values of this variable. One just has to remember to do so on the squared values, in a sum of square error, before computing the square root of the result. This is especially important when a variable consists in several separated series, like for example temperature at 200mb in this project<sup>9</sup>, that includes 1280 such series, one for each grid point.

**Step-ahead** To compute a prediction several steps ahead, the principle is to move a window along the values and the predictions already computed. More exactly, if the model considered uses the 5 last values for the series  $x$  to compute the next one, then the 1-step ahead prediction  $x'_6$  will be computed with  $x_1, x_2, x_3, x_4, x_5$ .

---

<sup>7</sup>Singular Value Decomposition.

<sup>8</sup>NETLAB is a neural network tool box written by Ian T Nabney & Christopher M Bishop (1996, 1997, 1998). It is available at <http://www.ncrg.aston.ac.uk/netlab/index.html>

<sup>9</sup>Any variable at any level, for that matter.

## APPENDIX B. CODING ISSUES

The 2-step ahead  $x'_7$  will be the value obtained using  $x_2, x_3, x_4, x_5, x'_6$ , the 3-step ahead  $x'_8$  with  $x_3, x_4, x_5, x'_6, x'_7$ , and so on. As mentioned, a buffer window of 5 elements can be used to store the values needed for the prediction, and each step forward shifts out the leftmost value in the buffer. The last prediction is introduced by the right, and the process repeats.

# Bibliography

- [1] The ECMWF operational implementation of four-dimensional variational assimilation. I: Experimental results with simplified physics. *Quarterly Journal of the Royal Meteorological Society*, 126:1143–1170, April 2000.
- [2] U. Achatz and G. Schmitz. On the closure problem in the reduction of complex atmospheric models by PIPs and EOFs: A comparison for the case of a two-layer model with zonally symmetric forcing. *Journal of the Atmospheric Sciences*, 54:2452–2474, 1996.
- [3] DLT Anderson. *Data Assimilation in Ocean Models*. Workshop on non-linear aspects of data assimilation, European Centre for Medium-Range Weather Forecasts, September 1997.
- [4] Chris Chatfield. *The Analysis Of Time Series, an Introduction*. Chapman & Hall, fifth edition, 1996.
- [5] K. Hasselmann. PIPs and POPs: The reduction of complex dynamical systems using principal interaction and oscillation patterns. *Journal of Geophysical Research*, 93:11,015–11,021, 1988.
- [6] James R. Holton. *An Introduction to Dynamic Meteorology*. Academic Press, London, 1979.
- [7] David E. Rumelhart Paul Smolensky, Michael C. Mozer. *Mathematical perspectives on neural networks*. Lawrence Erlbaum associates, Publishers, 1996.
- [8] F.M. Selten. Toward an optimal description of atmospheric flow. *Journal of the Atmospheric Sciences*, 50:861–877, 1993.
- [9] Hans Von Storch and W. Zwiers. *Statistical analysis in climate research*. Cambridge universit press, 1999.
- [10] JM Wallace and PV Hobbs. *Atmospheric Science, An Introductory Survey*. Academic Press, London, 1977.