UCL Université catholique de Louvain

École polytechnique de Louvain (EPL)

Early fault detection in power plants

Dissertation presented by
Pierre-Paul MOUCHET

for obtaining the Master's degree in
Mathematical Engineering

Supervisor(s)
Julien HENDRICKX

Reader(s)
Bertrand HAUT, Michel VERLEYSEN

Academic year 2016-2017

LINMA2990: Master thesis

Early fault detection in power plants

Pierre-Paul Mouchet

Academic year 2016-2017

Supervisor Prof. Julien Hendrickx

Readers Prof. Michel Verleysen,
PhD. Bertrand Haut

Acknowledgments

I would like to thank my supervisor Prof. Julien Hendrickx for his help and availability as well as his wise remarks and comments throughout the achievement of my master thesis.

I also would like to thank PhD. Bertrand Haut and Mr. Olivier le Fevere de ten Hove from Engie for providing working data, explanations on wind turbines, help and research directions. Without their assistance, the study of fault detection methods on wind turbines could not have been carried out.

Eventually, I would like to thank Prof. Michel Verleysen and PhD. Bertrand Haut for being the readers of my master thesis.

Pierre-Paul Mouchet

Contents

List of Symbols and Abbreviations

ANN	Artificial neural network
CUSUM	Cumulative sum
d_1	number of system's input signals
d_2	number of system's output signals
DE	driven-end
Δ	A similarity
$\mathbb{E}$	Expectation
E•F	Set of wind turbines E1F, E2F, E3F, E4F, E5F
E•RC	Set of wind turbines E1RC, E2RC, E3RC, E4RC, E5RC, E6RC
E•RY	Set of wind turbines E0RY, E1RY, E2RY, E3RY, E4RY, E5RY, E6RY
E•V	Set of wind turbines E1V, E2V, E3V, E4V, E5V, E6V
EWMA	Exponentially weighted moving average
$F_{a,b}$	Fisher-Snedecor distribution with parameters a ans b
g	Impulse response of a linear single input single output system
G	A linear single input single output system
KPI	Key performance indicator
MEWMA	Multivariate exponentially weighted moving average
μ	Mean
$\mathcal{N}$	Normal law
NDE	non-driven-end
NN	Neural net
PCA	Principal components analysis
RMSE	Root-mean-square error
RR	Rejection region
Σ	Variance-covariance matrix
σ	Standard deviation
SPRT	Sequential probability ratio test
$T^2_{p,m}$	Hotelling's T-squared distribution with parameters p and m
u_i	system's $i-th$ input signal
v	A stochastic signal
y_i	system's $i-th$ output signal

List of Figures

List of Tables

Introduction

The daily oversight of most industrial processes implies to collect data measurements of many signals. Those measurements are gathered in huge databases often called historians. As explained in [22], tremendous progresses where made before the early 2000s in the daily and routinely computer monitoring of industrial processes. Nevertheless, few interest had been carried out to the anticipation and management of abnormal events which often remained the task of human operators. In 2002, [22] estimated the annual costs of the unexpected events occurring in industrial processes for the British economy to 27 billion dollars. Consequently, it is no surprise that interest in the development of automated fault detection tools has increased for the last decades.

Therein, growing attention has been recently paid to machine learning and big data techniques for the monitoring of complex processes. This is because those tools often require few information about the monitored process except a large set of data.

In the power industry, relatively recent interest has been shown for this kind of fault detection tools (see for example [26], [32], [35]) with some notable particular attention for the monitoring of wind turbines. According to [26], costs of operations and maintenance in wind turbines are rather high while their reliability is perfectible. Moreover, those costs are expected to increase with the age of the wind turbine. For a wind farm over 20 years, [26] estimates that 10 to 15 % of the generated income is spent on operation and maintenance.

So far, we have not properly defined what will be considered as a fault for the sequel. According to [18], '*A fault is an unpermitted deviation of at least one characteristic property (feature) of the system from the acceptable, usual, standard condition*'. This definition remains broad (we still need to define the *standard condition*) and being more precise will depend on the application.

In practice, the following terminology is often encountered in literature and will be used now and then in the sequel. The *fault detection problem (FD)* is the problem of telling whether or not a new point of measurement is faulty. The *fault detection and isolation problem (FDI)* consists of fault detection along with determination of the occurring fault.

This dissertation will be divided in 5 chapters. Chapters 1 to 3 aim to provide an overview of popular fault detection tools with particular attention paid to the power industry. In chapters 4 and 5, we will implement fault detection methods for data from wind turbines to get an insight of what can be achieved with those methods. Notice that chapters 1 and 2 will introduce and describe techniques that won't necessarily be used in the following chapters. Nonetheless, we thought it was legitimate to talk about those methods in the framework of proceeding to an overview of model-based schemes (indeed, most of those were demonstrated to provide nice results for fault detection in power plants and we could not claim to deliver a survey on the fault detection field without including them).

In chapter 1, we will introduce basic methods for the monitoring of a signal or a set of signals. Those methods do not assume the existence of any model linking some of the signals. Instead, stationarity and/or Gaussianity hypothesis will often be required.

In chapter 2, we will introduce the popular model-based fault detection. Model-based fault detection requires having a model for predicting some signal. Faults on this signal are then detected by comparing the prediction of the model and the true measurement. This comparison will often be performed using methods of chapter 1.

In chapter 3, we will try to investigate some of the commercial software performing fault detection. However, we will fail in fairly comparing those software because most of the contacted companies refused to disclose much information about their software. All that could be done is mentioning some proof concepts that were carried out by those companies. Nevertheless, this chapter will justify some of the content of the 2 previous ones.

In chapter 4, we will implement model-based fault detection methods on wind turbines using neural networks. This will require building models for some specific compounds of the wind turbines. Identifying the useful input and output signals for those models will rely on the physical understanding of the system (filter method) as well as on a few tests (wrapper method). As argued in [8], it is impossible to build an original training set faithfully representing all the possible incoming operating conditions for the next months or years. This is among others due to some compounds of the system being subject to aging and/or being replaced. Therefore, the models of chapter 4 will constantly be learned again and again every time we have a new time period where we can ensure no fault has occurred. We will come to the conclusion that model-based fault detection detects some faults at least one month earlier than simple methods of chapter 1.

Eventually, chapter 5 will investigate the possibility of building a fault detection method without having any knowledge on the relationships that may exist between the measured signals. So, we do not have to guess the features that may be related to each other (as in chapter 4) anymore. Having such an idealistic tool will only be feasible at the price of assuming that the process is static (actually, we will explain how the dynamic could be taken into account but it will require having an idea of the order of the dynamic relationships). Therefore, we will use an auto-encoder network. Since this is a more generic tool, fault detection using this auto-encoder will be no more efficient than model-based methods of chapter 4 but the difference between the two approaches will often remain small. Finally, we will see that this method will only perform fault detection but won't be suited for fault isolation (we will be able to tell whether a fault has occurred or not but we will not always be able to identify the fault).

In conclusion, through this dissertation, we aim at giving an overview of the different approaches to data-based fault detection and providing some proof of efficiency.

Chapter 1

Process monitoring

In this chapter, we will present some basic existing tools for process monitoring. All of those tools proceed to fault detection by analyzing the statistical behavior of the measured signals. In this chapter, we will work directly with the measured signals. Next chapter will introduce methods to build new signals onto which methods of this chapter may be applied.

We will first analyze one dimensional signals in section 1.1. Then section 1.2 will be devoted to multidimensional signals. Most of the methods presented here are studied in [19] and [18]. Examples of applications to fault detection can be found in [1], [32]. From an industrial point of view, those methods are used in [34] (SmartSignal) and [36] (STEAG) among others.

1.1 Univariate process monitoring

In this section, we start with some simple fault detection methods which will form the basis for some of the more complicated methods we will see further. Methods introduced here perform fault detection by directly analyzing a monitored signal denoted by v. The section ends with the presentation of a very popular tool: Shewhart control charts.

1.1.1 Limit checking and trend checking

The simplest fault detection tool consist of imposing lower and upper bounds for the monitored signal v. Simple limit checking and trend checking ([18]) verify that

$$v_{\min} < v(t) < v_{\max},$$
$$\dot{v}_{\min} < \dot{v}(t) < \dot{v}_{\max},$$

for some lower and upper thresholds $v_{\min}, v_{\max}, \dot{v}_{\min}, \dot{v}_{\max}$ chosen thanks to expertise. Here, $\dot{v}$ denotes the trend of v with respect to time for the particular observed realization of the signal v and can be estimated numerically (e.g. using finite differences). An alarm will be triggered whenever one of the bounds is violated.

However, the choice of the bounds require having some knowledge of the signal and this method only take into account the actual value of the signal. Some changes may be undetected by such a method. For example, a modification in the mean of the signal may not be detected if the range is not modified. Figure 1.1 highlights another drawbacks of this method: there are many false alarms making it impossible to truly tell when the fault occurred. We will see later (figures 1.2, 1.3, 1.4) that there is no reason for launching alarms before abscissa 400. In conclusion, figure 1.1 underline the need for more powerful methods more sensitive to the trends of the signal and less sensitive to the volatility of the signal.

1.1.2 Statistical change detection

We will now make some hypothesis on the stochastic signal v to allow estimating the plausibility of the observed realization and considering simultaneously signal values at different time instants.

Let us suppose the monitored stochastic signal v is quasi-stationary and has some ergodicity properties. Quasi-stationarity requires the limits

$$\lim_{N\to\infty} \frac{1}{N} \sum_{t=1}^{N} \mathbb{E}[v(t)] \text{ and } \lim_{N\to\infty} \frac{1}{N} \sum_{t=1}^{N} \mathbb{E}[v(t)v(t+\tau)]$$

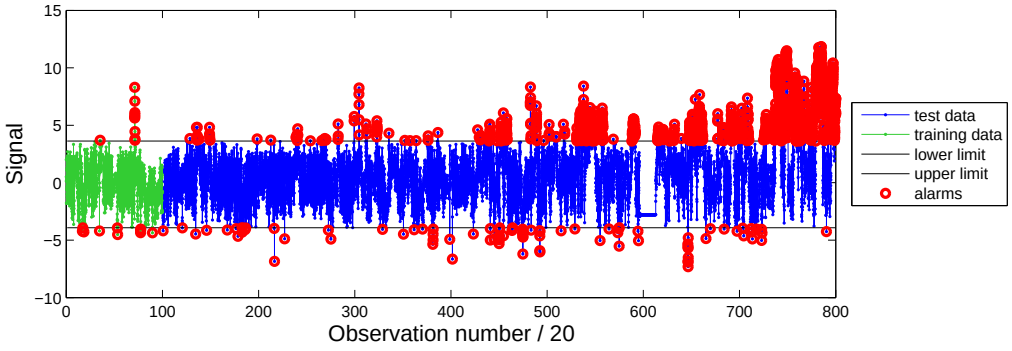


Figure 1.1: Limit checking for a residual signal. That signal is taken from an example of chapter 4. The abscissa axis has been scaled (there are 20 points on every unitary interval of the abscissa axis) so that abscissa are consistent with figures 1.2, 1.3, 1.4 and 1.5. Upper and lower limits have been set to the quantiles 0.995 and 0.005 of the training data (so false alarm rate is 1%). We can guess a change in the signal distribution has occurred but can not tell when.

exist for any τ. The signal v is ergodic with respect to the first moment if

$$\frac{1}{T}\sum_{t=1}^{T} v(t) \to \lim_{N\to\infty} \frac{1}{N}\sum_{t=1}^{N} \mathbb{E}[v(t)] \ \ w.p.\ 1 \ \ when\ T \to \infty,$$

and with respect to the second moment if

$$\frac{1}{T}\sum_{t=1}^{T} v(t)v(t+\tau) \to \lim_{N\to\infty} \frac{1}{N}\sum_{t=1}^{N} \mathbb{E}[v(t)v(t+\tau)] \ \ w.p.\ 1 \ \ when\ T \to \infty, \ \ for\ every\ \tau.$$

Thus, we can now replace averages in the phase state by averages over time. Then, we can develop fault detection methods based on the hypothesis that a fault will lead to changes in the characteristics of v (e.g. mean - resp. variance - if v is ergodic with respect to the first - resp. second - moment). The distribution parameters can be estimated using all the previous observations and updating online (e.g. $\hat{\mu}(T) = 1/T\sum_{k=1}^{T} v(k)$ for the mean), using a sliding window (e.g. $\hat{\mu}(T) = \sum_{k=T-w}^{T} v(k)$ for the mean) or using an exponentially decaying window (e.g. $\hat{\mu}(T) = \sum_{k=1}^{T} \lambda^{T-k} v(k)$ - with $0 < \lambda < 1$ - for the mean).

One can next use limit checking, confidence intervals or hypothesis tests (which are closely related to confidence intervals) on the built estimator. The two last options require having some knowledge on the distribution of $v(t)$. Table 1.1 summarizes some of the most common confidence intervals and tests. Those can be used on measurements taken from a stationary process. If we do not know anything about the distribution of $v(t)$ and $v(t)$ is stationary, we should opt for an empirical confidence interval established thanks to a set of non-faulty training data. This approach will be used in chapter 4.

Finally, notice that we have to find a compromise between the confidence level for the interval or the significance level of the test α and the size of the detected changes.

Tool	**Hypothesis**	**Interval or test and rejection region**
confidence interval on the mean	$X \sim \mathcal{N}(\mu, \sigma)$	$\left[\bar{x} - t_{n-1,\alpha/2}\frac{S}{\sqrt{n}}; \bar{x} + t_{n-1,\alpha/2}\frac{S}{\sqrt{n}}\right]$
confidence interval on the variance	$X \sim \mathcal{N}(\mu, \sigma)$	$\left[\frac{(n-1)S^2}{\chi^2_{n-1,\alpha/2}}; \frac{(n-1)S^2}{\chi^2_{n-1,1-\alpha/2}}\right]$
large sample confidence interval on θ	$\frac{\hat{\theta}-\theta}{\sigma_{\hat{\theta}}} \sim \mathcal{N}(0,1)$	$\left[\hat{\theta} - z_{\alpha/2}\sigma_{\hat{\theta}}; \hat{\theta} + z_{\alpha/2}\sigma_{\hat{\theta}}\right]$
hypothesis test on the mean	$X \sim \mathcal{N}(\mu, \sigma)$	$H_0 : \mu = \mu_0$, $H_1 : \mu \neq \mu_0$, RR: $\left\{\left\lvert\frac{\bar{x}-\mu_0}{S/\sqrt{n}}\right\rvert > t_{n-1,\alpha/2}\right\}$
hypothesis test on the variance	$X \sim \mathcal{N}(\mu, \sigma)$	$H_0 : \sigma^2 = \sigma_0^2$, $H_1 : \sigma^2 \neq \sigma_0^2$, RR: $\left\{\frac{(n-1)S^2}{\sigma_0^2} > \chi^2_{n-1,\alpha/2}\right\} \cup \left\{\frac{(n-1)S^2}{\sigma_0^2} < \chi^2_{n-1,1-\alpha/2}\right\}$
large sample hypothesis test on θ	$\frac{\hat{\theta}-\theta}{\sigma_{\hat{\theta}}} \sim \mathcal{N}(0,1)$	$H_0 : \theta = \theta_0$, $H_1 : \theta \neq \theta_0$, RR: $\left\{\left\lvert\frac{\hat{\theta}-\theta}{\sigma_{\hat{\theta}}}\right\rvert > z_{\alpha/2}\right\}$

Table 1.1: Common confidence intervals and hypothesis tests on a sample of independent realizations of a random variable X.

1.1.2.1 Shewhart control charts

Control charts or Shewhart control charts (see [19], [30]) are tools that can be used to analyze stationary stochastic signals. They are helpful in fault detection to check whether a signal (often a residual signal or a KPI, see later) deviates from its normal behavior. Suppose we observe a stationary stochastic signal denoted by $v : \Omega \times \mathbb{Z} \to \mathbb{R}$ on a period of n successive measurements. This will be called a sample. Using those n measurements, we then build a statistic Q summarizing the behavior of the process on that particular period. The value of the statistic will determine whether the system was under normal operating mode or not during that period. For this purpose, we should previously establish lower and upper acceptable bounds for the statistic. Often, those bounds are percentiles derived from the theoretical distribution of the statistic. A control chart is a plot of the value of the statistic over different periods (i.e. over different non-overlapping samples of size n). One usually add a line representing the theoretical mean of the statistics Q. The upper and lower limits are often taken 3 standard deviations above and below the mean (this can be motivated by Chebyshev's inequality). The choice of the length n of the periods is important and depends on the application and the desired statistic Q. The idea is that a smaller n may lead to earlier fault detection (the statistic will contains mostly information from the very last measurements) but is more likely to induce false alarms. The probability of a false alarm can often be deduced from the value of n, the distribution of Q and the lower and upper limits. We look here at some well-known control charts.

1.1.2.1.1 Xbar control chart A very basic example of control chart is the following. Suppose that the stationary stochastic process $v : \Omega \times \mathbb{Z} \to \mathbb{R}$ is Gaussian with mean μ and standard variation σ. We believe that the occurrence of a fault will lead to a significant change in the mean of the process. Let us take the average as the statistic and denote by $v_j^{(i)}$ the j-th measurement ($1 \leq j \leq n$) of sample i ($1 \leq i \leq m$). So, for every sample $1 \leq i \leq m$, the value of the statistic is

$$\bar{v}_i = \frac{1}{n} \sum_{j=1}^{n} v_j^{(i)},$$

which is normally distributed with mean μ and variance $\sigma/\sqrt{n}$. We can take $\mu - 3\sigma/\sqrt{n}$ and $\mu + 3\sigma/\sqrt{n}$ as the lower and upper limits. Doing so, we know that the probability of a false alarm is less than 0.3%. Nevertheless, we do not have any idea of the performance of this method for detecting faults (it depends on the size of the change in the mean). Often, the Gausianity hypothesis will be violated. If n is large enough, the use of this control chart can still be justified by the central limit theorem (if the measurements are independent identically distributed). An example of a Xbar control chart is shown on figure 1.2.

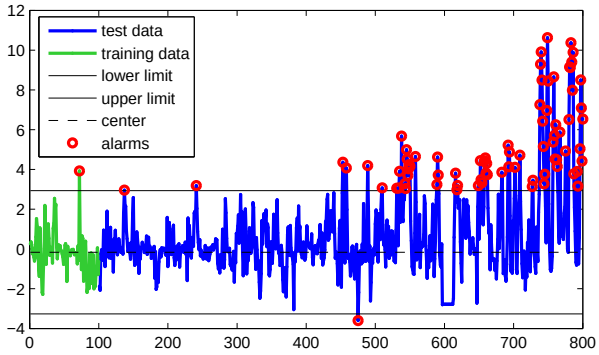


Figure 1.2: Control chart for a residuals signal. That signal is taken from an example of chapter 4. We can see recurrent alarms starting from abscissa 538 indicating a change in the signal mean. The upper and lower limits have been set to $\mu - 3\sigma/\sqrt{n}$ and $\mu + 3\sigma/\sqrt{n}$. Values of μ and σ are estimated with the training set. There are 800 samples of size 20.

Notice that the parameters of the distribution (e.g. μ and σ herein) are often unknown. In this case, we need to estimate those parameters thanks to another set of data. If we are given another set of r samples of size n to estimate those parameters, we should proceed as follows (the j-th measurement of sample i is denoted

$\tilde{v}_j^{(i)}$)

$$\bar{\tilde{v}}_i = \frac{1}{n}\sum_{j=1}^{n} \tilde{v}_j^{(i)}, \quad 1 \leq i \leq r,$$

$$s_i^2 = \frac{1}{n-1}\sum_{j=1}^{n} (\tilde{v}_j^{(i)} - \bar{\tilde{v}}_i)^2, \quad 1 \leq i \leq r,$$

$$\bar{v} = \frac{1}{r}\sum_{i=1}^{r} \bar{\tilde{v}}_i,$$

$$s^2 = \frac{1}{r}\sum_{i=1}^{r} s_i^2 = \frac{1}{rn-r}\sum_{i=1}^{r}\sum_{j=1}^{n} (\tilde{v}_j^{(i)} - \bar{\tilde{v}}_i)^2.$$

Notice that s^2 is the pooled variance estimator. Other methods, using the sample range for example, can be used to estimate σ (see [19] chapter 3).

Control charts as described above do not take into account the entire sequence of points but only the last one. As a consequence, small changes (less than $1.5\sigma_Q$ in the literature) may stay undetected. To detect small (or slow) changes, we will consider several simultaneous periods. The two following paragraphs introduce methods that use present and past values of the statistic.

1.1.2.1.2 CUSUM control chart Cumulative sum (CUSUM) control chart is a sequential analysis (as we will see, the size of the sample test is not fixed in advance) tool. CUSUM control charts allow to detect smaller changes than classic control charts do by taking into account the trend of the process. Suppose we built a statistic Q (we denote by $\mu_Q^{(0)}$ the theoretical mean of Q, which may be estimated using another data set) and we believe that the occurrence of a fault will lead to a small change in the mean of the statistic. Let us denote by q_i the statistic value computed with the sample i. For every sample i, we compute the sum of all the differences between the statistics values and the statistic mean up to period i, i.e.

$$C_i = C_{i-1} + q_i - \mu_Q^{(0)} = \sum_{j=1}^{i} (q_j - \mu_Q^{(0)}),$$

starting with $C_0 = 0$. The CUSUM is used to monitor the statistic mean. The values plotted on the control chart are

$$C_i^U = \max(0, C_{i-1}^U + q_i - (\mu_Q^{(0)} + k)),$$
$$C_i^L = \max(0, C_{i-1}^L - q_i + (\mu_Q^{(0)} + k)).$$

Because of the max functions, past values of the process will be forget at some time if the process is under normal operating mode. This tool is more sensitive to changes in the mean because C_i^U (resp. C_i^L) will accumulate successive positive (resp. negative) deviations in the mean. We now have to decide for the values k and the upper and lower limits $\mu_Q^{(0)} + H$ and $\mu_Q^{(0)} - H$. We will suppose that the statistic Q follows a normal distribution (this hypothesis is discussed in section 1.1.2.1.4). Let us take $k = d\sigma_Q$ and $H = h\sigma_Q$. According to [30], $d = 1/2$ and $h = 4$ or 5 is a good choice based on the abundant literature studying the CUSUM control charts performances. The strategy to choose those parameters is the following. First, we should choose k equal to the size of the shifts we want to detect divided by σ_Q. Then, we choose H such that the number of false alarms is acceptable. This is done by imposing some in-control average run length (ARL), which is the number of time units between two false alarms (see [30] table 9.3 p.408 for ARL values for different k and H). However, in many practical applications (which is the case for figure 1.3), we do not have enough knowledge on the fault to proceed in that way and we can not proceed otherwise than setting those parameters empirically.

If C_i^U becomes bigger than the upper limit or if $-C_i^L$ becomes smaller than the lower limit, the process is out of control. Figure 1.3 shows an example of CUSUM control chart. Observe that recurrent alarms appear earlier (approximately 100 abscissa earlier) than on figure 1.2.

1.1.2.1.3 EWMA control chart Exponentially weighted moving average control charts are another tool to detect small changes in the statistic. For some $0 < \lambda \leq 1$, starting from $z_0 = \mu_Q^{(0)}$ (or an estimation of μ_0 computed with another independent data set), we compute

$$z_i = \lambda q_i + (1-\lambda) z_{i-1} = z_0(1-\lambda)^i + \lambda \sum_{j=1}^{i} (1-\lambda)^{i-j} q_j.$$

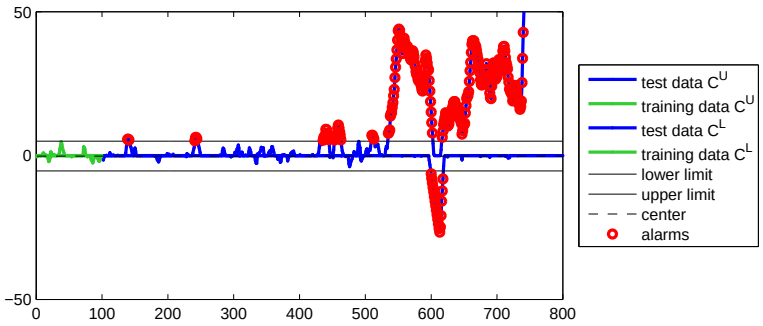


Figure 1.3: CUSUM control chart for a residuals signal. That signal is taken from an example of chapter 4. We can see recurrent alarms starting from abscissa 436 indicating a change in the signal mean. The upper and lower limits have been set to $\mu_Q^{(0)} - 5\sigma_Q$ and $\mu + 5\sigma_Q$. Values of $\mu_Q^{(0)}$ and σ_Q are estimated with the training set. There are 800 samples of size 20. Observe that recurrent alarms appear earlier than on figure 1.2.

So, we can see that old values become progressively insignificant as i increases because of exponentially decreasing weights. We can verify that the sum of the weights equates to one since

$$(1-\lambda)^i + \lambda\sum_{j=1}^{i}(1-\lambda)^{i-j} = (1-\lambda)^i + \lambda\sum_{l=0}^{i-1}(1-\lambda)^l = (1-\lambda)^i + \lambda\frac{1-(1-\lambda)^i}{1-(1-\lambda)} = 1.$$

So, $\mathbb{E}[z_i] = \mu_Q^{(0)}$ and

$$\begin{aligned}\mathrm{Var}[z_i] &= \mathrm{Var}[z_0(1-\lambda)^i + \lambda\sum_{j=1}^{i}(1-\lambda)^{i-j}q_j] = \lambda^2\sum_{j=1}^{i}(1-\lambda)^{2(i-j)}\mathrm{Var}[q_j] = \lambda^2\sum_{j=0}^{i-1}((1-\lambda)^2)^l\sigma_Q^2 \\ &= \lambda^2\frac{1-(1-\lambda)^{2i}}{1-(1-\lambda)^2}\sigma_Q^2 = \lambda\frac{1-(1-\lambda)^{2i}}{2-\lambda}\sigma_Q^2.\end{aligned}$$

If every z_i is supposed to follow a normal distribution (this will be the case if the statistic Q is normally distributed), we can take the following lower and upper limits (notice that the limits values now vary with i and tend to some constant value as i goes to infinity)

$$\begin{aligned}U_i &= \mu_Q^{(0)} + k\sigma_Q\sqrt{\lambda\frac{1-(1-\lambda)^{2i}}{2-\lambda}}, \\ L_i &= \mu_Q^{(0)} - k\sigma_Q\sqrt{\lambda\frac{1-(1-\lambda)^{2i}}{2-\lambda}}.\end{aligned}$$

We now have to choose the values of k and λ. Recommendations found in the literature for those parameters do not always match and the choice should depend on the application. According to [30], common values for λ vary between 0.05 and 0.25 and $k = 3$ or a little less (2.8) is often used as 3 standard deviation of the mean is often the limit choice for control charts. Those recommendations work fine if the normality assumption holds. If this assumption is not verified, empirical values should be chosen to reach some given fault alarm rate under normal conditions. This is the case on figure 1.4. Observe that the recurrent alarms appears earlier than on figure 1.2 but a little later than on figure 1.3.

1.1.2.1.4 On the normality assumption The examples above always assume gaussianity of the statistic Q to provide theoretical upper and lower bounds. However, this assumption is not required for every control chart. For example, there are control charts dealing with Student's t-distribution and chi-squared distributions for Q. Notice that for CUSUM and EWMA control charts, if every q_i is, for example, the sum of many independent identically distributed random variables taken from sample i (say the average $\bar{x}_i$ for example), then q_i tends to be normally distributed. Finally, if the distribution of Q is not known (which is the case in many practical applications), one can still compute empirical lower and upper bounds with another set of data. Both approaches have been used in chapter 4 and 5.

1.1.2.2 Sequential probability ratio test

We pay particular attention to this test since it turns out to be mentioned in several articles (e.g. [25] and [6]) and patents (e.g. [34]) to perform fault detection. The main advantage of this test is that it is a sequential

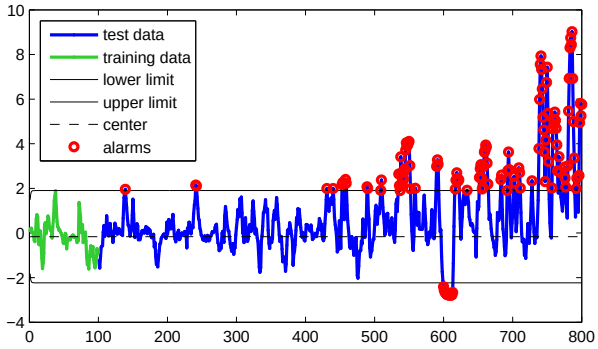


Figure 1.4: EWMA control chart for a residuals signal. That signal is taken from an example of chapter 4. We can see recurrent alarms starting from abscissa 440 indicating a change in the signal mean. The upper and lower limits have been set to $\mu_Q^{(0)} - 4\sigma_Q$ and $\mu + 4\sigma_Q$. Values of $\mu_Q^{(0)}$ and σ_Q are estimated with the training set. There are 800 samples of size 20. Observe that recurrent alarms appear earlier than on figure 1.2 but approximately at the same time as on figure 1.3.

test, meaning that the sample test size is not fixed in advance. Instead, data arriving online are added to the sample until we can choose between the null hypothesis or the alternative hypothesis. Moreover, it has been shown (see [31]) that this test is the most powerful one to choose between two probability distribution functions. The drawback is that this test require knowing the probability distribution functions for the stationary process under normal (f_0) and faulty (f_1) conditions (since the process v is stationary, the pdf of $v(t)$ is identical for every t). In that regard, the patent [34] proposes the two following solutions.

- Suppose both probability density functions f_0 and f_1 are Gaussian. Parameters of the probability distribution function f_0 can be easily estimated based on collected data. However, if we do not have data collected under faulty conditions, the probability distribution function f_1 can not be estimated that way. It is proposed in [34] to take it identical to f_0 but with a positive or negative (according to the type of change in the pdf we want to monitor) shift in the mean equal to 1 standard deviation.
- Estimate both probability density functions f_0 and f_1 using methods such as kernel density estimation or cubic splines. Nevertheless, this require having data collected under faulty conditions.

Let us now explain that test. Suppose observations of some stationary random process $v : \Omega \times \mathbb{Z} \to \mathbb{R}$ arrive one after another. So, the sequence $(v(t))_{t\in\mathbb{Z}}$ is a sequence of identically distributed random variables. We will also suppose that those variables are independent. We want to decide as soon as possible between the two hypothesis

$$H_0 : \text{the pdf for X(t) is } f_0(x),$$
$$H_1 : \text{the pdf for X(t) is } f_1(x).$$

Let us remember that the proportion of type I and type II errors are

$$\alpha = P(\text{accept } H_1|H_0),$$
$$\beta = P(\text{accept } H_0|H_1),$$

and the power of the test is $P(\text{reject } H_0|H_1) = 1 - \beta$. Given observations $v_1, v_2, \ldots, v_n$, we will use the log-likelihood ratio test. The log-likelihood ratio is given by

$$\Lambda_n = \ln \frac{p(v_1, v_2, \ldots, v_n|H_1)}{p(v_1, v_2, \ldots, v_n|H_0)} = \ln \frac{p(v_1|H_1)p(v_2|H_1)\ldots p(v_n|H_1)}{p(v_1|H_0)p(v_2|H_0)\ldots p(v_n|H_0)} = \ln \frac{f_1(v_1)f_1(v_2)\ldots f_1(v_n)}{f_0(v_1)f_0(v_2)\ldots f_0(v_n)}.$$

Let us choose a significance level α and compute η such that

$$P(\Lambda_n \leq \eta|H_0) = \alpha.$$

The log-likelihood ratio test will reject H_0 if $\Lambda_n \leq \eta$. According to Neyman-Pearson lemma (see [31]), this is the most powerful test for choosing between H_0 and H_1.

Let us now introduce the sequential log-likelihood ratio test. Notice now that

$$\Lambda_n = \ln \frac{p(v_1, v_2, \ldots, v_n|H_1)}{p(v_1, v_2, \ldots, v_n|H_0)} = \ln \frac{p(v_1|H_1)p(v_2|H_1)\ldots p(v_n|H_1)}{p(v_1|H_0)p(v_2|H_0)\ldots p(v_n|H_0)} = \Lambda_{n-1} + \ln \frac{p(v_n|H_1)}{p(v_n|H_0)} = \Lambda_{n-1} + \ln \frac{f_1(v_n)}{f_0(v_n)}.$$

This recurrence formula is used by Wald in [38] to transform the log-likelihood ratio test into a sequential test: the value of the log-likelihood ratio is updated each time a new observation arrives until we can choose between H_0 and H_1. It is usually considered that

- if $\Lambda_n > \ln \frac{1-\beta}{\alpha}$, then H_1 is true,
- if $\Lambda_n < \ln \frac{\beta}{1-\alpha}$, then H_0 is true,
- if $\ln \frac{\beta}{1-\alpha} \leq \Lambda_n \leq \ln \frac{1-\beta}{\alpha}$, then continue sampling,

where the threshold values are demonstrated in [38]. In the framework of fault detection, sequential probability ratio test can be used on a stochastic signal. In this purpose, it is necessary to ensure that the signal is stationary under normal operating mode and that its pdf at a given time switch from f_0 to f_1 when a fault occurs. Moreover, we need to estimate the marginal probability distributions f_0 and f_1. To detect a change from f_0 to f_1, we will frequently start new sequential probability ratio tests. We may have several test with different starting points running at the same time. As soon as one of those tests concludes that H_1 is true, we know that the change has occurred.

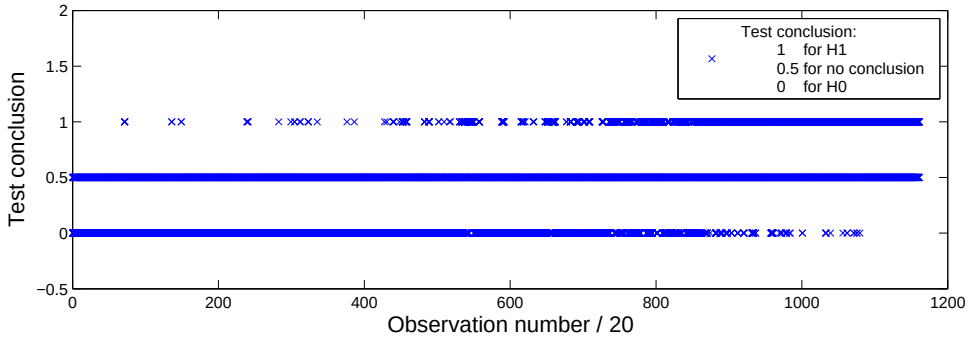


Figure 1.5: Result of the sequential probability ratio test for a residuals signal. That signal is taken from an example of chapter 4. The density probability distribution function f_0 is estimated using kernel density estimation. The density probability distribution function f_1 is a translation of f_0 by 4 standard deviations to the right (i.e. the mean is thus increased by 4 standard deviations). We decided to run only one sequential test at a time and to wait until it finishes before launching a new test. Value of α and β is 0.01. The abscissa axis has been scaled (since there are 20 more points on this figure) so that abscissa are consistent with figures 1.2, 1.3 and 1.4 (though, we display on a longer time frame since results are displayed until 1200 and not 800). Here, an alarm is launched when we conclude to H_1. Recurrent alarms becomes to be unsettling at abscissa 450 indicating a change in the signal mean. Observe that recurrent alarms appear earlier than on figure 1.2 but approximately at the same time as on figure 1.3 and 1.4.

1.1.3 Conclusion

In this section, we presented some methods regularly encountered for detecting changes in uni-dimensional signals. Those methods are often studied under the hypothesis that the signal is Gaussian. If this is not the case, nothing prevent us from using those methods but we do not have any warranty about the efficiency anymore. So, we also provided an insight of how those methods performs with respect to one another on some non-Gaussian signal thanks to figures 1.2, 1.3, 1.4 and 1.5.

1.2 Multivariate process monitoring

In section 1.1, we have considered the monitoring of univariate processes. We now consider multivariate process monitoring. So, we are given L stochastic signal $v_l : \Omega \times \mathbb{R} \to \mathbb{R}$ with $1 \leq l \leq L$. We will see how some of the tools introduced in section 1.1 can be extended to multidimensional processes. All along this section, we will apply some of the presented methods to an example of chapter 4. This is an example of fault detection in a two dimensional signal. Figure 1.6 introduces both signals.

1.2.1 Perfectibility of univariate process monitoring

A first idea could be to use univariate methods on each of those signals independently. Figure 1.7 shows the result of applying EWMA to both signals v_1 and v_2 of our example separately. As explained in the caption, three time periods for occurrence of the faults are identified.

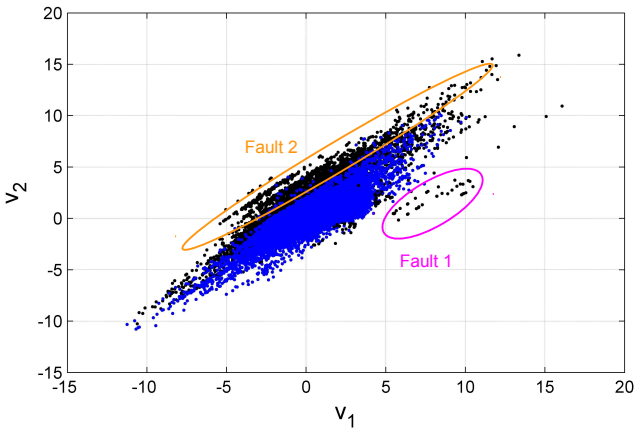
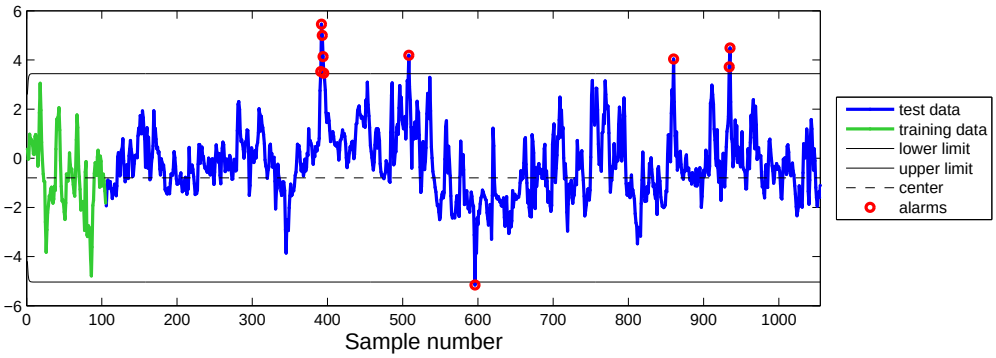
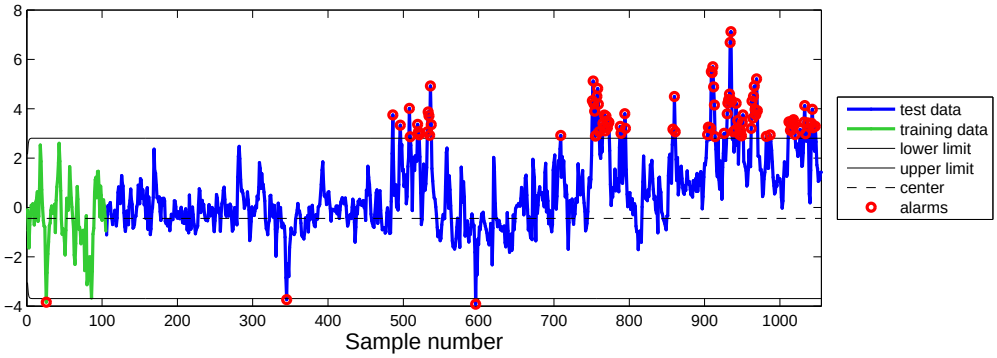


Figure 1.6: Plot of signal v_1 versus signal v_2. Blue points represent a training set of which most points are not faulty. Black points (of which many are behind blue points) are a test set in which we can identify two main faults that we will refer to as fault 1 and fault 2. Fault 1 corresponds to too high values of v_1 (with respect to v_2). Fault 2 corresponds to too high values of v_2 (with respect to v_1).

However, this is less powerful than methods taking simultaneously into accounts all of those signals. Indeed, one can imagine situations in which values of $v_1(t)$ and $v_2(t)$ are plausible taken one by one but not plausible together (this is the case for some points of the fault 2 on figure 1.6). Therefore, tools introduced for univariate process monitoring are generalized in higher dimensions. Though we will not detect new faults doing so for our particular example, faults may be detected earlier.



(a)



(b)

Figure 1.7: Univariate EWMA control charts with $\lambda = 0.25$ and sample size 20. Lower and upper bounds are set empirically based on the test set. (a) EWMA for the signal v_1. There are no recurrent alarms except from abscissa 391 to 396. This correspond to the time of occurrence of the type 1 faults described above (see figure 1.6). (b) EWMA for the signal v_2. There are recurrent alarms from abscissa 486 to 537 and from abscissa 751. This correspond to the time of occurrence of the type 2 faults described above (see figure 1.6).

1.2.2 Multivariate statistical change detection

In this section, we will introduce two tools for multivariate process monitoring. Those will be used in chapter 5.

1.2.2.1 Hotelling T^2 control chart

This control chart apply to multivariate Gaussian signals and defines a hyperellipsoidal inside which the process is thought to be under control. This is the most simple multivariate control chart.

Let us consider a L-variate Gaussian random variable V with mean vector μ and covariance matrix Σ. Let $v^{(1)}, \dots, v^{(n)}$ be a sample drawn from that multivariate Gaussian distribution. Then

$$\chi^2_L = n(\bar{v} - \mu)\Sigma^{-1}(\bar{v} - \mu)$$

follows a chi-squared distribution with L degrees of freedom. Imposing an upper limit $\chi^2_{L,\alpha}$ on that random variable defines a hyperellipsoidal inside which the process is considered under control. However, in practice μ and Σ are unknown and need to be estimated.

If we are given m samples of n vectors $v^{(i1)}, \dots, v^{(in)}$ with $1 \leq i \leq m$, we estimate μ by $\bar{\bar{v}}$ and Σ by S as follows

$$\begin{aligned}
\bar{v}_k^{(i)} &= \frac{1}{n}\sum_{j=1}^{n} v_k^{(ij)}, && 1 \leq i \leq m, 1 \leq k \leq L,\\
s^{2\,(i)}_{kl} &= \frac{1}{n-1}\sum_{j=1}^{n}(v_k^{(ij)} - \bar{v}_k^{(i)})(v_l^{(ij)} - \bar{v}_l^{(i)}), && 1 \leq i \leq m, 1 \leq k, l \leq L,\\
\bar{\bar{v}}_k &= \frac{1}{m}\sum_{i=1}^{m} \bar{v}_k^{(i)}, && 1 \leq k \leq L,\\
\bar{s}^2_{kl} &= \frac{1}{m}\sum_{i=1}^{m} s^{2\,(i)}_{kl}, && 1 \leq i \leq m, 1 \leq k, l \leq L,\\
\bar{\bar{v}} &= \left[\bar{\bar{v}}_k\right]_{k=1}^{L},\\
S &= \left[\bar{s}^2_{kl}\right]_{k,l=1}^{L}.
\end{aligned}$$

Then, given a new sample with mean $\bar{v}$, the statistic

$$T^2 = n(\bar{v} - \bar{\bar{v}})S^{-1}(\bar{v} - \bar{\bar{v}})$$

follows a Hotelling's T-squared distribution $T^2_{L,mn-m}$ ($mn - m$ is the number of remaining degrees of freedom after estimating the means). Then, $\frac{mn-m-L+1}{m(n-1)L}T^2$ follows a F-distribution $F_{L,mn-m-L+1}$. The upper control limit for T^2 is thus $\frac{(m+1)(n-1)L}{mn-m-L+1}F_{\alpha,L,mn-m-L+1}$. Notice that the case $n = 1$ should be treated apart and the upper control limit on T^2 in this case is $\frac{(m+1)(m-1)L}{m^2-mL}F_{\alpha,L,m-L}$. More details can be found in [30] (chapter 11).

We will not show the results of applying Hotelling's test to our example because this lead to too many false alarms. This is because the Gaussianity hypothesis is not verified here. It is thought in literature that EWMA control charts are less sensitive to the Gaussianity hypothesis. So, let us investigate multivariate EWMA.

1.2.2.2 MEWMA

Multivariate exponentially weighted moving average control charts are the multivariate generalization of EWMA control charts introduced previously. MEWMA control chart for the mean first computes

$$\begin{aligned}
z^{(i)} &= \lambda\bar{v}^{(i)} + (1 - \lambda)z^{(i-1)},\\
z^{(0)} &= \mu,
\end{aligned}$$

with $0 < \lambda \leq 1$. The control chart represents

$$\begin{aligned}
T_i^2 &= {z^{(i)}}^T \Sigma_i^{-1} z^{(i)},\\
\Sigma_i &= \frac{\lambda}{2 - \lambda}\left(1 - (1 - \lambda)^{2i}\right)\Sigma.
\end{aligned}$$

Σ and μ can again be estimated as in section 1.2.2.1. Recommendations for the choice of λ and the upper limit can be found in [30]. Figure 1.8 presents the results of applying MEWMA to our example.

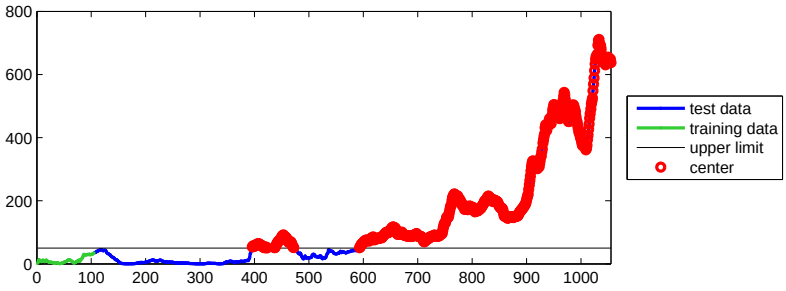


Figure 1.8: MEWMA for example of figure 1.6 using $\lambda = 0.025$ and $H = 50$ (see [30] for more details on how to choose those parameters). There are recurrent alarms from abscissa 396 to 472 and starting from abscissa 593. When comparing to figure 1.7, we should notice that

- The sudden high values around abscissa 390 is detected later and alarms last longer because of the optimal parameter λ being smaller.
- The alarms taken place around abscissa 468 on figure 1.7b are not launched anymore. This is because signal v_1 also takes high values at that abscissa (see figure 1.7a) and the value of v_2 is not alarming anymore with respect to the value of v_1).
- The increase in the mean of v_2 is detected much earlier (593 as compared to 751).

1.2.3 Conclusion

We saw that multivariate process monitoring overpowers univariate process monitoring (that will be shown in chapter 5 too). We also noticed that the mentioned multivariate methods assume Gaussianity for the signals. If it is not the case, we can either hope that the control limits will still lead to good results or try to adapt the limits. Taking quantiles from a training set collected under normal conditions is one solution. Given a training sample $q^{(1)}, \dots, q^{(n)}$ of the chart statistic, we can also estimate the probability density function at a new point q using the multivariate kernel density estimation

$$f(q) \approx \frac{1}{n} \sum_{i=1}^{n} K_H(q - q^{(i)}),$$
$$K_H = |H|^{-1/2} K(H^{-1/2} q),$$

and decide the point q is faulty when density is too low (K is a kernel and H a symmetric positive definite bandwidth matrix). That approach will be used in chapter 5 too.

Chapter 2

Model-based fault detection

In model-based fault detection (see for example [18], [22], [24]), faults are detected by taking advantage of relations between the measured variables. Those relations are often deduced from the underlying physics of the system. This approach is sometimes called analytical redundancy because we will build a model which is redundant with the measurements.

The first step of model-based fault detection is to identify a feature (state variable, parameter, ...) or a set of features that will be modified by the occurrence of a fault (i.e. the feature(s) value(s) should be disturbed by the fault). Afterwards, we focus on building a model that will compute an estimation for the feature(s) value(s) if the system is in normal operating mode. In the sequel, we will denote by $y_1, y_2, \ldots, y_{d_2}$ the output features to estimate and $u_1, u_2, \ldots, u_{d_1}$ the input features to our model.

The inputs to this model can be any of the past and present measurements of the signals $u_1, u_2, \ldots, u_{d_1}$ and any of the past measurements of the signals $y_1, y_2, \ldots, y_{d_2}$ (some models even take as input the measured value at time t of the feature(s) to estimate). For any given $t \in \mathbb{Z}$, let us introduce $u(t) = [u_1(t), u_2(t), \ldots, u_{d_1}(t)]$ and $y(t) = [y_1(t), y_2(t), \ldots, y_{d_2}(t)]$. The output of the model is an estimation (sometimes called the reference value) for the present value(s) of the feature(s) $y_1, y_2, \ldots, y_{d_2}$, i.e.

$$\hat{y}(t) = f\left(u(t), u(t-1), u(t-2), \ldots, y(t-1), y(t-2), \ldots\right).$$

Finally, a symptom is generated using the estimation(s) and the measured value(s) of the feature(s). The terminology key performance indicator (KPI) is sometimes used to designate symptoms. Faults are detected by analyzing the symptoms (using methods of chapter 1).

Examples of symptoms are the difference or the quotient between the estimation and the measured value. The difference between the estimation and the measurement is called the residual. If the model is good, the residuals should be 'close' to 0 when the system is under normal operating mode. The residuals should be 'far' from 0 when a fault has occurred and causes the feature to deviate from its normal value.

Process model identification We now have to explain how we can determine the model denoted by f here above. The model can be obtained either by fully theoretical modeling or experimental modeling. Both approaches can be used simultaneously and compared.

Using theoretical modeling suppose we have full understanding of the system. We usually end up with systems of ODE or PDE. The model is then often simplified (linearization, order reduction, approximation). In experimental modeling, a model is built based on observations of the process in normal operating mode and require having collected measurements of the input and output signals. Theoretical modeling is also known as white-box modeling. Experimental modeling is known as black-box modeling. Gray-box modeling is in between (e.g. we know the model up to some parameters which must be experimentally computed).

(a) **White-box modeling**: the model is derived based on theoretical knowledge.

(b) **Gray-box modeling**: we have a model for the process up to some parameters that need to be estimated. For example, for time-invariant parameter estimation of linear processes, correlation method or least squares method (either on the equation error or the output error, see [18]) can be used. For time-variant parameter estimation of linear processes, the method of weighted least squares can be used. For nonlinear processes, there are a few models studied in literature.

(c) **Black-box modeling**: since we suppose we do not know anything about the process, black-box modeling require having tools that allow to approximate any nonlinear mapping. Artificial neural networks (ANNs) and supervised learning are widely used for that purpose since this does not require any knowledge about the process and the ANN is theoretically able to approximate any (static) nonlinear relationship to any

accuracy. The drawback of most of the popular neural network structures is that there are only suitable for approximating static relationships (i.e. of the type $y(t) = f(u(t))$). Indeed, popular ANNs are memoryless. There are two possibilities to deal with that problem.

(1) **External dynamic**: this consist of increasing the dimension of the input space by taking as input to our system at time t the values $u_1(t), \dots, u_1(t-n_1), u_2(t), \dots, u_2(t-n_2), \dots, u_{d_1}(t), \dots, u_{d_1}(t-n_{d_1})$ and $y_1(t), \dots, y_1(t-m_1), y_2(t), \dots, y_2(t-m_2), \dots, y_{d_2}(t), \dots, y_{d_2}(t-m_{d_2})$.

(2) **Internal dynamic**: this consist of using neural nets that differ from the classic neural nets in the sense that either the connections or the neurons contain delay operators. We will see that the learning step becomes much more complex.

Finally, according to [32], the dynamic of the system can sometimes be neglected, mainly when

(1) the data are collected at high frequency with respect to the system characteristic frequencies

(2) the system has a control loop with constant target (in case the target is piecewise constant on long time intervals, the dynamic can also be neglected; the system will be badly modeled during transitions). This is the case in [32] for gas power plants.

This chapter will summarize several well-known identification methods in the framework of fault detection. It is not intended to draw up an exhaustive list of those methods. This means we do not claim to provide a complete overview of all the system identification methods. Instead, we will focus on some common methods that can be find in industrial patents or article and that appears to be efficient for our purpose. We will distinguish between linear (section 2.1) and nonlinear (section 2.2) processes. However, we will pay much more attention to nonlinear processes since those are most commonly encountered in our framework.

2.1 Linear processes

In this section, we will provide an example of parametric and an example of non-parametric methods for linear system identification (more methods can be found in [18], [22]). However, those methods are barely encountered for fault detection (though some applications can be found in [18]) because linear systems are not common. Nevertheless, to insist linear processes can be useful in the framework of fault detection, we will end the section with a method more often encountered in fault detection and based on principal component analysis.

In the sequel, G will denote a linear single input single output system. Then, we can use classic system identifications methods for linear processes.

2.1.1 A non-parametric method

Let us introduce the correlation function method. Suppose signals u and y are quasi-stationary. We can compute the average correlation and cross-correlation function

$$r_u(\tau) = \lim_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} u(t)u(t-\tau), \qquad r_{yu}(\tau) = \lim_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} y(t)u(t-\tau),$$

which can be estimated by

$$\hat{r}_u(\tau) = \frac{1}{T} \sum_{t=0}^{T-1} u(t)u(t-\tau), \qquad \hat{r}_{yu}(\tau) = \frac{1}{T} \sum_{t=0}^{T-1} y(t)u(t-\tau),$$

for finite samples of size T. Now, we remember that

$$y(t) = G(q)u(t) + v(t) = \sum_{k=0}^{\infty} g(k)u(t-k) + v(t),$$

which implies that

$$\begin{aligned} r_{yu}(\tau) &= \lim_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} \left(\sum_{k=0}^{\infty} g(k)u(t-k) + v(t) \right) u(t-\tau) \\ &= \lim_{T\to\infty} \sum_{k=0}^{\infty} g(k) \frac{1}{T} \sum_{t=0}^{T-1} u(t-k)u(t-\tau) + \lim_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} v(t)u(t-\tau) \\ &= \lim_{T\to\infty} \sum_{k=0}^{\infty} g(k) \frac{1}{T} \sum_{t=0}^{T-1} u(t-k)u(t-\tau) = \sum_{k=0}^{\infty} g(k) \lim_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} u(t-k)u(t-\tau) = G(q)r_u(\tau), \end{aligned}$$

where we supposed that $v(t)$ and $u(t-\tau)$ are uncorrelated and we can exchange the limit and the sum because

$$\sum_{k=0}^{\infty} g(k) \frac{1}{T} \sum_{t=0}^{T-1} u(t-k)u(t-\tau)$$

converges (if the system is BIBO stable) and

$$\sum_{k=0}^{\infty} g(k) \lim_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} u(t-k)u(t-\tau)$$

converges (if the system is BIBO stable) since u is quasi-stationary. Finally, we can approximate the impulse response $g(t)$ by solving the truncated linear system

$$\hat{r}_{yu}(\tau) = \sum_{k=0}^{\infty} \hat{g}(k)\hat{r}_u(\tau - k).$$

2.1.2 Parametric methods

Parametric methods perform grey-box modeling. Indeed, we need to have an idea of possibles models for the process. The set of considered models will be called the model set. Well-known models for single input single output systems are given in table 2.1. Given measurements from time 0 to time T for the input signal u and the output signal y, computing the coefficients of the polynomials A, B, C, D is often formulated as solving the optimization problem

$$\min \frac{1}{T} \sum_{t=1}^{T} (y(t) - \hat{y}(t))^2,$$

where $\hat{y}(t)$ is the estimation for $y(t)$ given polynomials $\hat{A}, \hat{B}, \hat{C}, \hat{D}$, input signal u up to time t and output signal y up to time $t-1$. There exists other ways to compute the coefficients of the polynomials, such as statistical methods (e.g. maximizing the likelihood of output signal y given the input signal u). One should pay attention to the fact that the input signal must be informative enough to correctly identify the parameters of the model. This may not be easy because, in the framework of fault detection, we can not always choose the input signals that will be applied during the learning stage.

Box-Jenkins	$y(t) = \frac{A(q)}{B(q)}u(t) + \frac{C(q)}{D(q)}w(t)$
ARARMAX	$A(q)y(t) = B(q)u(t) + \frac{C(q)}{D(q)}w(t)$
ARMAX	$A(q)y(t) = B(q)u(t) + C(q)w(t)$
ARMA	$A(q)y(t) = C(q)w(t)$
ARX	$A(q)y(t) = B(q)u(t) + w(t)$
IIR	$A(q)y(t) = B(q)u(t)$
FIR	$y(t) = B(q)u(t) + w(t)$

Table 2.1: Different structures for parametric model identification. $A(q), B(q), C(q), D(q)$ are polynomials (of any order) in q^{-1} and w is a white noise.

In [41], identification is performed using an ANNARIMA model. This means the output is supposed to be the sum of a nonlinear system (identified using artificial neural networks) and a linear system (identified using an ARIMA model, which is a special case of the ARMA model of table 2.1).

For multiple input multiple output systems, the models of table 2.1 can be extended. The output $y_i(t)$ being a function of all the previous and present inputs and all the previous outputs.

Up to now, we considered open-loop systems. Suppose we have a closed-loop system with feedback $-F$ and reference signal r. Suppose also there is a delay in either G or F and that r and $v = Hw$ are independent signals. Then, one can distinguish between direct and indirect methods. Direct methods identify the system based on u and y without paying attention to the control loop. In this case, one can recover the true model if it belongs to the considered model set. Indirect methods identify the system having the reference signal r as input and y as output. Then, we can recover the true model G (H does not need to belong to the considered model set) if F is known.

2.1.3 A black-box model : principal component analysis

In this section, we will use PCA to reduce dimension by modeling linear relationships between measured signals. Fault detection with PCA (see [18]) can be considered as black-box modeling as we will explain later.

Let us first explain how PCA should be applied and then how this can be useful for fault detection. Suppose we observe a set of L signals $x_1, x_2, \ldots, x_L$ and that we know there exist linear relationships between these signals. For any $k \in \mathbb{Z}$, let us introduce the matrix

$$X_k = \begin{bmatrix} x_1(k-n) & x_2(k-n) & \ldots & x_L(k-n) \\ x_1(k-n+1) & x_2(k-n+1) & \ldots & x_L(k-n+1) \\ \vdots & \vdots & \ddots & \vdots \\ x_1(k) & x_2(k) & \ldots & x_L(k) \end{bmatrix},$$

which is thus a matrix containing the n last observations of the L signals. Let us now choose some external dynamic parameter d and build the matrix

$$X = \begin{bmatrix} X_1 & X_2 & \ldots & X_d \end{bmatrix}.$$

We will assume the signals contained in this matrix have been scaled to have zero mean (this hypothesis is required to apply PCA) and unit variance (this is required because the signals may be expressed in different units). We apply singular value decomposition to matrix X and obtain unitary matrices U and V and diagonal matrix S such that

$$X = USV^T.$$

The decorrelated PCA variables are given by $Y = XV$ so that the correlation matrix

$$\Sigma_Y = \frac{1}{n}Y^TY = \frac{1}{n}V^TVS^TU^TUSV^TV = \frac{1}{n}S^TS = \Lambda$$

is diagonal. Λ is the diagonal matrix with the eigenvalues of the correlation matrix Σ_X (sorted in decreasing order) on the diagonal. So, the first PCA variables bear most of the variance of the process. The fraction of the variance borne by the first P PCA variables is

$$\frac{\sum_{i=1}^{P}\lambda_i}{\sum_{i=1}^{L}\lambda_i}.$$

Those variables are given by $Y_P = Y = XV_P$, where V_P designate the restriction of V to its first P columns. The back-transformed variables are given by $\tilde{X} = YV_P^T = XV_PV_P^T$.

We will now explain how PCA can be useful to fault detection. The idea is to use PCA to identify the number of linear constraints between the variables. First of all, the PCA model has to be learned thanks to a training set. This means that we determine the matrix parameter P and the matrix V_P. The value of P is selected by looking at the eigenvalues so that we conserve some percentage of the total variance. Given a new point x, we can now compute the corresponding P first PCA variables $y = xV_P$. The following two step approach has been proposed in [1] to determine whether x is a faulty point or not ([29] proposes a similar approach).

1. In the PCA state, we should check that y belongs to the subspace covered by points under normal operating mode (if not, that means that the linear relationships that should hold between variables are violated). In [1], good results are obtained by assuming $T = y^T\Lambda^{-1}y$ follows a Hotelling's T-squared distribution. One can use a Hotelling T^2 control chart (see section 1.2.2.1). What we gain here compared to proceeding to multivariate signal analysis in the signal space is that the dimension has been reduced.

2. Compute the back-transformed variable $\tilde{x} = xV_PV_P^T$ and check that the residual $r = x - \tilde{x}$ is under control (if not, that means that the original point x was projected on a point in the low dimensional space that does not contain enough information to allow accurate reconstruction, which should not occur under normal operating conditions if the model is correctly identified). In that purpose, [1] proposes an interval of confidence for monitoring the statistic $Q = r^Tr$.

This method can be seen as model-based with $\tilde{x}$ as the model prediction. Notice that matrix X could be build in a different way: we could choose a different external dynamic parameter d_l for every signal x_l.

This approach will be partially used in chapter 4 where we will also try to generalize this method to nonlinear relationships between the signals.

2.2 Nonlinear processes

In this section, we will give an overview of some methods for non-linear process monitoring. There exist many different methods for identification of nonlinear dynamic systems. Therefore, we will focus on methods for which practical applications to fault detection can be found in literature.

2.2.1 Grey-box modeling using non-linear models

Popular tools for non-linear dynamic system identification are Volterra and Wiener series which can be seen as non-linear generalization of the impulse response method. However, those classical approaches are computationally heavy and does not suit our purpose.

Therefore, methods representing non-linear behavior using polynomials have been studied. For example, for SISO systems, [12] proposes to use a generalized Hammerstein model polynomial in the input signal

$$A(q^{-1})y(k) = B_1(q^{-1})u(k) + B_1(q^{-1})u^2(k) + \cdots + D_1(q^{-1})w(k),$$

with w a white noise. The main advantage of those methods is that there are linear in the parameters. Thus, finding those parameters is easy using, for example, a least square method. However, those methods are limited in practice because a particular form as to be assumed for the model.

2.2.2 Black-box modeling

Those tools are the most popular for model identification in the framework of fault detection. Many applications can be found, as well from an academic point of view as from a commercial point of view.

2.2.2.1 Feed-forward neural network

In this section, we will present some neural networks based approaches to perform model-based fault detection. Neural networks are very popular black-box models for nonlinear regression. We will here focus on feed-forward neural networks because of the efficiency of the learning algorithms for those (recurrent networks are discussed in section 2.2.2.4).

Suppose we want to identify a nonlinear process with input signals $u_1, u_2, \ldots, u_{d_1}$ and output signals $y_1, y_2, \ldots, y_{d_2}$. The general model for a L-layer feed-forward neural network is the following

$$\begin{aligned} u_i^{(l)} &= \phi_i^{(l)} \left(\sum_{j=1}^{D_{l-1}} u_j^{(l-1)} w_{ij}^{(l-1)} + b_i^{(l-1)} \right), && 1 \leq l \leq L \\ u_i^{(0)} &= u_i, && 1 \leq i \leq d_1, \\ \hat{y}_i &= u_i^{(L)}, && 1 \leq i \leq d_2, \end{aligned}$$

where $D_0 = d_1$ and $D_L = d_2$ are the dimensions of the input and output layer, $D_1, \ldots, D_{L-1}$ are the dimensions of the hidden layers, matrices $w^{(l)}$ are parameters, vectors $b^{(l)}$ are bias terms and $\phi^{(l)}$ is the activation function of the i-th unit of the l-th layer. Generally, activation functions are linear, sigmoidal or hyperbolic tangent. Figure 2.1 shows a 3-layer neural network. This neural-network is feed-forward because connections does not form cycles. The model is optimized on the parameters w and b. Therefore, we will make use of the scaled conjugate gradient algorithm. The number of layers and number of hidden units in every layer are meta-parameters that have to be investigated too (see chapter 4).

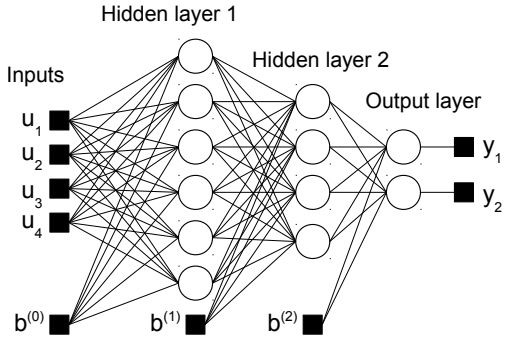


Figure 2.1: A 3-layer feed-forward neural network with bias vectors.

A reason why those tools are popular is the universal approximation theorem which state that any continuous function defined on a compact subset of $\mathbb{R}^{d_1}$ can be approximated to any given precision with a feed-forward neural network made of only two layers

$$u \mapsto \sum_{i=1}^{D_1} w_i^{(2)} \phi \left[\sum_{j=1}^{D_0} u_j w_{ji}^{(1)} + b_i^{(1)} \right] + b^{(2)},$$

where ϕ is a non-constant, bounded, monotonically increasing continuous function. The drawback of this theorem is that number of units D_0 may be very large. Consequently, learning the parameters w may require a lot of data and time.

Notice that neural networks as described above can only approximate static mappings. We can again use external dynamic to approximate dynamic mappings. Furthermore, we can add to the input signals the delayed output signals to approximate nonlinear autoregressive exogenous systems as (see [7])

$$y(k) = f(y(k-1), \dots, y(k-n_y), u(k), u(k-1), \dots, u(k-n_u)) + e(k).$$

This approach is developed in [35] for wind turbine fault detection with particular attention paid to temperature signals. In that article, three methods for fault detection in wind turbines using either linear regression or feed-forward neural networks (autoregressive or not) are compared. It is come to conclusion that autoregressive models (lag values n_y and n_u are typically 1 or 2) lead to earlier fault detection for slowly changing signals (such as temperature signals).

In [13], it is argued that using RBF neural networks along with CUSUM control charts to detect faults in the heat transport system of a nuclear reactor is efficient.

2.2.2.2 Similarity-based modeling

Similarity-based modeling (see [32],[16]) is an important model-based black-box method because it is a very general approach and it encompasses many of the popular machine learning and pattern recognition methods for classification, clustering and approximation (see [11] for more details). The basic idea of similarity-based modeling is to find points of the training set similar (or to rank the training set points by order of similarity) to the new test point we are interested in. Classification, clustering or approximation is then performed using information from the training set and giving more importance to (or exclusively considering) points highly similar to our new point. The freedom in the choice of the similarity will lead to many different methods. Notice also that we can defines similarity between different objects of the same type (so we can define similarities between matrices, between vectors, between strings, ...). We can find practical applications and example of such similarities in [16].

There is no clearly established definition for a similarity. So, we will give here an example and highlight properties that should be verified. For vectors, the mapping $\Delta : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^+$ is a similarity on $\mathbb{R}^d$ if it is

- symmetric: $\forall x, y \in \mathbb{R}^d$, $\Delta(x, y) = \Delta(y, x)$,
- maximum in 0: $\forall x \in \mathbb{R}^d$, $\Delta(x, y)$ is maximum if $x = y$,
- monotonically decreasing with $\|x - y\|$, for $x, y \in \mathbb{R}^d$.

In practice, $1/(1 + \|x - y\|)$ is a popular similarity.

Suppose we have a static process with input signals $u_1, u_2, \dots, u_{d_1}$ and output signals $y_1, y_2, \dots, y_{d_2}$. As a training set, we are given n input-output vectors pairs $(u^{(1)}, y^{(1)}), (u^{(2)}, y^{(2)}), \dots, (u^{(n)}, y^{(n)})$ with

$$\begin{aligned} u^{(i)} &= \begin{bmatrix} u_1^{(i)} & u_2^{(i)} & \dots & u_{d_1}^{(i)} \end{bmatrix}^T, && 1 \le i \le n, \\ y^{(i)} &= \begin{bmatrix} y_1^{(i)} & y_2^{(i)} & \dots & y_{d_2}^{(i)} \end{bmatrix}^T, && 1 \le i \le n. \end{aligned}$$

Then, for a new input vector u, the similarity-based model is given by

$$\begin{aligned} \hat{y} &= Y \frac{w}{1^T w}, \\ w &= S^{-1} s, \\ S_{ij} &= \Delta(u^{(i)}, u^{(j)}), && 1 \le i, j \le n, \\ s_i &= \Delta(u^{(i)}, u), && 1 \le i \le n, \\ Y &= \begin{bmatrix} y^{(1)} & y^{(2)} & \dots & y^{(n)} \end{bmatrix}, \end{aligned}$$

with Δ the similarity over vectors defined above. So, the estimation $\hat{y}$ is a weighted average of all the training outputs $y^{(1)}, y^{(2)}, \ldots, y^{(n)}$. Notice that the similarity-based model interpolates the points $(u^{(i)}, y^{(i)})$ (notice therefore that $Sw = s$). This approach has been developed in [32] where it is used along with Hotelling's test to perform fault detection in a natural gas power generation plant. Since the model is static, that work focus on points where the system is at stable operating points. Attention is also carried out to make the training set as small as possible while keeping it representative of the process so that S and s are of smaller dimensions (remember that we need to solve the system $Sw = s$). The need to make that training set as small as possible is one major drawback of similarity based-modeling.

As stated above, this model is purely static. We can make it dynamic by taking previous inputs vectors into account. Let us select some external dynamic parameter d big enough to capture the dynamic of the system (notice that we will need d additional input-output pairs denoted by $(u^{(-d+1)}, y^{(-d+1)}), \ldots, (u^{(0)}, y^{(0)})$). Similarities won't be computed between two input vectors anymore but between two sequences of $d+1$ input vectors. Given a new input-output pair $(u^{(h)}, y^{(h)})$ (with $h > n + d$ if we do not want to spill on the training set) we compute an approximation $\hat{y}$ to $y^{(h)}$ by

$$\begin{aligned}
\hat{y} &= Y \frac{w}{1^T w}, \\
w &= S^{-1} s, \\
S_{ij} &= \sum_{k=0}^{d} \alpha_k \Delta(u^{(i-k)}, u^{(j-k)}), && 1 \leq i, j \leq n, \\
s_i &= \sum_{k=0}^{d} \alpha_k \Delta(u^{(i-k)}, u), && 1 \leq i \leq n, \\
Y &= \begin{bmatrix} y^{(1)} & y^{(2)} & \ldots & y^{(n)} \end{bmatrix},
\end{aligned}$$

where Δ is the similarity between vectors defined above and the weights α_k (decreasing with k) allow to give more importance to similarity between most recent measurements. Notice that we choose to sum similarities between corresponding vectors to obtain similarities between two sequences of vectors. [16] explores many other possibilities for the choice of the similarity between sequences of vectors.

2.2.2.3 Kernel regression

In this section, we present model-based fault detection based on kernel regression. This approach looks close to similarity-based modeling (both are often presented together) and it can be shown kernel regression using Gaussian kernel is a particular case of similarity-based modeling. Kernel regression is interesting because it has an evident interpretation in terms of probabilities. We will first explain kernel regression for static processes and then explain how to take the dynamic into account (see [16] and [10] for more).

A kernel function K is a non-negative symmetric function $K : \mathbb{R}^d \to \mathbb{R}$, such that $\int_{\mathbb{R}^d} K(x)\,\mathrm{d}x = 1$. A typical kernel is the Gaussian kernel $K(x) = \frac{1}{(\sqrt{2\pi}\sigma)^d} \exp\left(\frac{-\|x\|^2}{2\sigma^2}\right)$.

Inferential kernel regression In this paragraph, we will build a predictive model for a process with inputs signals $u = [u_1, u_2, \ldots, u_{d_1}]$ and outputs signals $y = [y_1, y_2, \ldots, y_{d_2}]$. Suppose we are given a set of n inputs-outputs vector pairs $(u^{(1)}, y^{(1)})$,$(u^{(2)}, y^{(2)})$,$\ldots$,$(u^{(n)}, y^{(n)})$ of the process. Given a new input vector u, we estimate the output vector y by taking the conditional expectation

$$\hat{y} = \mathbb{E}[y|u] = \int_{\mathbb{R}^{d_2}} y p(y|u)\,\mathrm{d}y = \int_{\mathbb{R}^{d_2}} y \frac{p(y,u)}{p(u)}\,\mathrm{d}y.$$

Using the kernel density estimation with bandwidth h for the probability density functions,

$$\begin{aligned}
\hat{p}(u) &= \frac{1}{nh} \sum_{i=1}^{n} K\left(\frac{u - u^{(i)}}{h}\right), \\
\hat{p}(u,y) &= \frac{1}{nh^2} \sum_{i=1}^{n} K\left(\frac{u - u^{(i)}}{h}\right) K\left(\frac{y - y^{(i)}}{h}\right),
\end{aligned}$$

we get the Nadaraya-Watson kernel regression

$$\hat{y} = \int_{\mathbb{R}^{d_2}} \frac{y}{h} \frac{\sum_{i=1}^n K\left(\frac{u-u^{(i)}}{h}\right) K\left(\frac{y-y^{(i)}}{h}\right)}{\sum_{j=1}^n K\left(\frac{u-u^{(j)}}{h}\right)} \mathrm{d}y = \sum_{i=1}^n \frac{K\left(\frac{u-u^{(i)}}{h}\right)}{\sum_{j=1}^n K\left(\frac{u-u^{(j)}}{h}\right)} \int_{\mathbb{R}^{d_2}} \frac{y}{h} K\left(\frac{y-y^{(i)}}{h}\right) \mathrm{d}y$$
$$= \sum_{i=1}^n \frac{K\left(\frac{u-u^{(i)}}{h}\right)}{\sum_{j=1}^n K\left(\frac{u-u^{(j)}}{h}\right)} \int_{\mathbb{R}^{d_2}} (hy' + y^{(i)}) K(y') \mathrm{d}y' = \frac{\sum_{i=1}^n y^{(i)} K\left(\frac{u-u^{(i)}}{h}\right)}{\sum_{j=1}^n K\left(\frac{u-u^{(j)}}{h}\right)}$$

since the kernel K is symmetric around 0 and integrates to 1. Notice that if the kernel function is the Gaussian kernel, this is equivalent to normalized radial-basis function network. Non-linear regression with radial basis function networks optimizes the model

$$y(x) = \sum_{i=1}^Q w_i^R \exp\left[-\frac{\|u - c_i\|_2^2}{2(h\sigma_i)^2}\right] + (w^L)^T u + w_0^B,$$

on the parameters w^R, w^L, w_0^B to minimize the error on the training set (notice that the linear and constant term are not always used). The centroids c_i should be chosen using a vector quantization algorithm on the training data (e.g. competitive learning or frequency sensitive learning). The number of centroids Q and the bandwidth h are hyper-parameters. The normalized RBF model is

$$y(x) = \sum_{i=1}^Q w_i^R \frac{\exp\left[-\frac{\|u - c_i\|_2^2}{2(h\sigma_i)^2}\right]}{\sum_{j=1}^Q \exp\left[-\frac{\|u - c_j\|_2^2}{2(h\sigma_j)^2}\right]}.$$

So, if we choose the $Q = n$ centroids c_i to be the data points $x^{(i)}$, the optimal weights for minimizing the error on the training set are given by $w_i^R = y^{(i)}$. Inferential kernel regression for fault detection is studied in [10] where it is compared to some other methods.

Auto-associative kernel regression In the sequel, we will consider a process of which we measure some characteristic signals. We denote by $x \in \mathbb{R}^d$ the vectors of signals. As in inferential kernel regression, given vectors $x^{(1)}, x^{(2)}, \ldots, x^{(n)}$ and a new vector x, we can estimate $\mathbb{E}[x|x]$ by

$$\hat{x} = \frac{\sum_{i=1}^n x^{(i)} K\left(\frac{x-x^{(i)}}{h}\right)}{\sum_{j=1}^n K\left(\frac{x-x^{(j)}}{h}\right)}.$$

This may seem strange or silly since we know that $\mathbb{E}[x|x] = x$. However, it should be understood here that we want to model a complex process while keeping as few information as possible in memory. Auto-associative kernel regression can be used to compress information. Suppose that $\hat{x}$ well approximates the true vector x. Then, if n is smaller than the dimension of x, it is interesting keeping in memory the values of $K(\frac{x-x^{(i)}}{h})$ for $i = 1, \ldots, n$ instead of x. In fault detection applications, we expect the above estimator will approximate correctly a non-faulty input vector x (reconstruction under normal operating mode) and badly a fault input vector x since the auto-associative model is learned under normal operating mode. However, there is no warranty that reconstruction will in practice be bad for faulty input vectors because those vectors belong to a region of the space that was not modeled. Thus, we might be very unlucky and have that the reconstruction method is also good at extrapolation. This is why we must stay cautious with some parts of the approach proposed in [16] and [4] where it is proposed to simply check for the residuals $x - \hat{x}$.

An advantage of the auto-associative method is that there is no distinction between input and output signals so we do not need to know anything about the process. In chapter 4, we will try to build an auto-associative model for fault detection using a neural network with a bottleneck layer which should ensure no faulty points are correctly modeled anymore.

Capturing the dynamic with external dynamic As we already suggested, identifying a dynamic process using external dynamic amounts to identifying a static process of which inputs at time t are all inputs up to time t and outputs up to time $t-1$. However, we will only consider a finite number of them. Let us introduce

$$U^{(i)} = \begin{bmatrix} u^{(i)} & u^{(i-1)} & \ldots & u^{(i-k)} \end{bmatrix}, \quad Y^{(i)} = \begin{bmatrix} y^{(i)} & y^{(i-1)} & \ldots & y^{(i-k)} \end{bmatrix},$$

for some k big enough to capture the dynamic of the system (notice that we need k additional input-output pairs $(u^{(-k+1)}, y^{(-k+1)})), \ldots, (u^{(0)}, y^{(0)}))$. Given a new input-output pair (u, y), we build the corresponding matrices U and Y and we use inferential kernel regression (auto-associative kernel regression can be make dynamic in a similar way)

$$\hat{Y} = \frac{\sum_{i=1}^{n} Y^{(i)} \tilde{K}\left(\frac{U - U^{(i)}}{h}\right)}{\sum_{j=1}^{n} \tilde{K}\left(\frac{U - U^{(j)}}{h}\right)},$$

where we need to define a kernel function $\tilde{K}$ on our matrices. A basic idea is to write that kernel as a weighted sum of kernel functions of which inputs are the columns of the matrix (a sum of kernel functions is a kernel function)

$$\tilde{K}(X) = \sum_{j=1}^{n} w_j K(x_j).$$

If we want to give more importance to the most recent measurements, the weights should decrease as k increases. Several kernel functions for matrices are claimed in patent [16].

2.2.2.4 Methods with internal dynamic

To be complete, we will now look at some existing neural network models to internalize the dynamic (see [33],[9]) though most of the modeling methods encountered in literature make use of external dynamic, as explained in [33]. Let us consider a nonlinear process with input signals $u_1, u_2, \ldots, u_{d_1}$ and output signals $y_1, y_2, \ldots, y_{d_2}$.

Fully recurrent networks This is the most general form of recurrent networks. The inputs to the network at time k are $u_1(k), u_2(k), \ldots, u_{d_1}(k)$. There are d_2 output units $y_1(k+1), y_2(k+1), \ldots, y_{d_2}(k+1)$. The network contains lots of other units and is fully connected. Each connection contains a delay operator z^{-1}. Notice that the neurons are identical to the neurons used in the previously introduced static neural networks.

However, fully recurrent networks are not that used in practice because of many drawbacks. There are many weights (since the network is fully connected) and the training of the network is slow and complex. Moreover, as noticed in [33], since there is a delay operator on every connection, we can not fix seperately the number of hidden units and the dynamic order of the system. We will not go further into details given no interesting application examples were found.

Partially recurrent networks Partially recurrent networks are simplified recurrent networks. Elman and Jordan partially recurrent networks are the most famous ones (see [33]). Those networks consist of a feedforward neural network plus a context layer. The context layer model the internal state of the process and can just be seen as a connection line with a delay operator. Neurons of the other layers are still identical to the neurons of the static neural networks introduced before. The main advantage of those networks is that they can be learned using back-propagation algorithms.

Another partially recurrent model is the recurrent multi-layer perceptron, which consist of a multi-layer perceptron with additional links between units of the same layer. Those additional links are links with delay operator. Notice however that learning the model is time consuming it is not feedforward anymore.

Again, few applications to fault detection can be found in the literature.

Locally recurrent globally feedforward networks Locally recurrent globally feedforward (LRGF) neural networks are feedforward neural networks of which synapses are modified (so we do not need delay links anymore). In classical neural networks, synapses are linear function of their inputs. In LRGF networks, synapses are linear systems. FIR and IIR synapses are studied in [3]. Suppose we have a L layer feedforward network, each layer l having N_l neurons. We denote by f the activation function and $z_i^l(t)$ the output of neuron i of layer l at time t. Then, for FIR synapses, we have

$$z_k^{l+1}(t) = f\left(\sum_{i=1}^{N_l} B_{ik}^{l+1}(q) z_i^l(t)\right),$$

$$B_{ik}^{l+1}(q) = \sum_{j=0}^{M} b_{ikj}^{l+1} q^{-j},$$

and for IIR synapses

$$z_k^{l+1}(t) = f\left(\sum_{i=1}^{N_l} \frac{B_{ik}^{l+1}(q)}{A_{ik}^{l+1}(q)} z_i^l(t)\right),$$
$$A_{ik}^{l+1}(q) = 1 - \sum_{j=1}^{M} b_{ikj}^{l+1} q^{-j}.$$

The basic idea is thus that the neuron can remember its pasts inputs and outputs. Notice that the value of M is the same for every neuron but introducing different values for every neuron is possible. Each time an input-output pair is presented, the training algorithm minimize the instantaneous error criterion

$$\frac{1}{2}\|y(t) - z^L(t)\|^2$$

Training algorithms are described in [3] and look very similar to backward propagation algorithm for training of classical MLP networks (notice that the training data should now be presented organized from the lowest to the highest value of t). For FIR synapses, convergence to a local minimum is guaranteed in [3]. For IIR synapses, there is no convergence guarantee.

2.3 Conclusion

In this chapter, we presented some of the linear and nonlinear methods used for model-based fault detection and referred to articles and patents presenting results using those tools. For nonlinear systems, methods using external dynamic turns out to be much more popular than methods with internal dynamic for which few applications to fault detection can be found. This is probably due to the fact that good results (see for example [35]) are obtained using external dynamic with low dynamic order. In chapter 4, we will use neural networks with external dynamic to perform fault detection on wind turbines.

Chapter 3

Commercial software

In this chapter, we will briefly summarize some of the research that were done in order to gather information about commercial software performing fault detection in power plants. In that purpose, we took contact with 9 different companies susceptible to provide fault detection software products. Table 3.1 summarizes our search. Unfortunately, few of the contacted companies took time to answer in a detailed and expanded way (we decided not to insist if no interesting answer could be obtained after two or three attempts depending on the company).

Nevertheless, we managed to collect valuable information for some of them, either by searching their websites and consulting patents or thanks to mails and phone calls with the contact persons mentioned in table 3.1. For some of those companies, we find it is interesting to expose the collected information here because they are someway related to the previous chapters and may justify their contents.

Eventually, notice that it will be impossible to fairly compare those software with each other because we have very few points of comparison. For some software, it is possible to find examples of fault detection and savings generated but those are useless for comparison because they are drawn from very distinct situations. A good comparison strategy would have been to run all of those software on some test data to evaluate how each of them performs. However, this was unfeasible since we could not get access to the software products.

In the rest of this chapter, we will summarize the information that we managed to collect from some of the companies mentioned in table 3.1. There will be some overlap with chapter 2, to which we will often refer.

Company name	**Software**	**Contact person(s)**	**Disclosed information**
Algorithmica technologies	AFP	/	No reply
CAMO	/	/	Multivariate data analysis, not much experience in power plants
Cassantec	IPS	Moritz von Plate	None except a few folders, see below
General Electric	SmartSignal	Jonathan Kosa	None except available patents, see below
GREENBYTE	Breeze	/	No reply
PEPITe	DATAmaestro	Philippe Mack	Machine learning models available on: `http://docs.mydatamaestro.com`
Schneider Electric	PRiSM	/	No reply
STEAG	SR::SPC	Fabio Wagner, Dennis Braun	See below
Tangent Works	/	/	No reply

Table 3.1: Summary of the contacted companies and replies.

3.1 General Electric - SmartSignal

Presentation General Electric (GE) is an American company active in a wide variety of sectors among which energy. GE Digital is a branch of GE working on applications of IIoT[1] (GE's cloud platform for IIoT is called Predix).

Solution Among a wide variety of products, GE digital proposes a software performing fault detection called SmartSignal. SmartSignal applies to different fields such as aviation, power plants, chemistry, manufacturing and finances. SmartSignal technologies are protected by about 45 patents (see for example [34], [39], [28], [16]).

[1] Industrial Internet of Things (IIoT) designate the application of machine learning and big data algorithms to industrial processes.

In practice, GE builds what they called a digital twin (see [14]) modeling the different parts of the plant (basically, this is model-based fault detection - see chapter 2). The digital twin can be used for different purposes such as operations optimization (e.g. dispatch optimization), performance optimization and equipment supervision (e.g. fault detection). Therefore, GE's offer falls within a wider framework than simple fault detection called the digital transformation of electricity.

According to [14], the digital twin is built using both physic based models, experimental knowledge, and artificial intelligence and depicts the behavior of the plant under normal operating mode. Some of the techniques used for modeling are mentioned in patents [39] and [16]. Explanations on some of those techniques (e.g. neural networks, similarity-based modeling and kernel regression) can be found in chapter 2. In patent [16], it is suggested to use extended kernel functions (of which inputs are arrays of data vectors) for similarity-based modeling and kernel regression to include the time domain information in the model as explained in chapter 2 (see for example sections 2.2.2.3 and 2.2.2.2).

Faults are detected and diagnosed by comparing the value measured by sensors with predictions generated by the digital twin, just as explained in chapter 2. In particular, SmartSignal claims a fault detection method using sequential probability ratio test (see section 1.1.2.2) and cubic splines for approximation of the residuals probability density function in [34] and a fuzzy classification method for fault diagnosis in [28].

Efficiency and customer stories On average, GE claims that they can decrease unplanned downtime by 5%. GE pretends to be able to detect a wide variety of faults thanks to deep field experience. Several customer stories can be found in the GE documentations (see [15]).

According to [15], the savings for a 'typical 550MW combine-cycle combustion turbine plant ' are close to \$680M a year. In a 445MW Irish CCGT power plant, 2.28MM€ annual savings were generated using GE solution. Scottish and Southern Energy equipped its thermal plants with GE solution (for a total of 1026 equipped assets in over 11 different locations) and estimates the gain due to early fault detection to more than £3MM a year.

In conclusion, though it is impossible to evaluate its true value with the disclosed information, the examples above tend to demonstrate the efficiency of the SmartSignal.

3.2 STEAG - SR::SPC

Presentation STEAG (Steinkohle-Elektrizität AG) is a German energy producer founded in 1937 and mainly active in Germany. As an energy producer, STEAG is interested in optimizing power plants productivity (maintenance optimization, outage planning, ...). Aside, STEAG also offers solutions for external customers. STEAG has been working on fault detection systems for 20 years, the main products being SR::SPC for fault detection and SR::EAGLE for fault isolation.

Solution In [36], STEAG explains that fault detection and isolation using SR::SPC is performed through the four following steps (which is nothing else but model-based fault detection).

1. **Data filtering**. Before investigating the faulty character of the data measured by sensors, those data are filtered. Selection filtering removes measurements collected during shut down and start up periods, load change periods, faulty process periods, ... Outlier filtering removes measurements that are irrelevant but not due to any fault (e.g. outliers due to automatic sensors cleaning).

2. **Key performance indicators**. Reference values are generated using either a physical approach or a data-based model (STEAG recommends artificial neural networks in [36]). Then, normalized key performance indicators (KPIs) are computed using the reference values and the measured values. Those KPIs should only depend on the process quality (normal or faulty). The particularity here being that the KPIs are normalized (dimensionless) values. For example, one could divide the reference value by the measured value. Building a consistent KPI require field experience. For example, STEAG has built about 30 KPIs for fault detection in wind turbines. The sensitivity of the KPI to fault occurrence is the only quality criterion to satisfy.

3. **Analysis of the data**. Statistical analysis is performed on the KPIs to detect faults. The main tools for statistical process control are control charts (see chapter 1). SR::SPC builds the control charts using the KPIs. SR::EAGLE performs fault isolation using intelligent event trees to detect several causes to the top event (building an event tree require having good experimental knowledge of the system).

4. **Trend detection and prediction**. Beyond fault detection, STEAG proposes to predict when some limit threshold will be reached based on the trend of the KPI.

Efficiency and customer stories Several examples of fault detection are provided in [36] and [37] among which bearing vibrations due to the crack of the shaft of a boiler feed pump, air ingress in condenser, increase in wind turbine gear bearing temperature, increase in wind turbine gear oil pressure and increase in wind turbine main bearing temperature. STEAG claims having increase the productivity of a 1.8MW German wind farm by 1.6%. According to [37], 171000€ a year could be saved by avoiding unplanned downtimes on a wind farm of 36 wind turbines (this does not take into account other related savings such as avoided damages and better maintenance planning). We refer to [37] for more details on estimated savings.

3.3 Others

For most of the other software, we struggled to get any interesting information. Here is a short survey of what we could learn by browsing the available documentation and websites.

Cassantec (Cassandra Technologies) is a Swiss company founded in 2007 and offering services in assets management, malfunction detection, maintenance scheduling and lifetime estimation. Cassantec has developed a tool for wind turbines monitoring called IPS WT (for Intelligent Prognostic System Wind Turbine). Here is the reply we get from Moritz von Plate from Cassantec: '*We differentiate ourselves from others in that we prognosticate malfunctions rather than just giving early warning indicators. We calculate when you can expect which malfunction with which probability, hence providing an explicit timeline until malfunction (weeks, months) rather than just an early warning that does not say when. We don't disclose more information about our algorithms than what you can find on our website and in the attached flyer*'.

Algorithmica technologies is a German company focusing on the application of advanced machine learning to industrial processes. In that regard, the software Advanced Failure Predictor (AFP) was designed for fault detection. Algorithmica technologies claims AFP can increase the plant availability by 2% and reduce maintenance costs by 20 %.

GREENBYTE is a company working exclusively on renewable energy. They propose the software Breeze for wind turbines monitoring, control and planning. From the information we could get, Breeze has an artificial intelligence based approach that seems to be efficient at detecting faults (they have several artificial neural networks for monitoring different parts of a wind turbine). However, no answer was given to our attempts to reach them to get more information.

PEPITe is a Belgian company that proposes help for data mining with their software called DATAmaestro. Access to some of the machine learning models they use is provided (see table 3.1). According to Philippe Mack (PEPITe CEO), fault detection is reached by combining data driven models with domain knowledge. This tools seems not to be at the level of the aforementioned solutions yet.

For the last 2 software (Instep Prism by Schneider Electric and Tangent Works), we did not get any information worth to expose here.

3.4 Conclusion

In conclusion, we get very few information on the commercial software. However, thanks to those information and the analysis of two software products done above, we can suspect that most of those software rely on the same theoretical principles mentioned above. What should differentiate them is the field experience some of those companies have acquired.

This chapter at least suggests that an efficient fault detection tool can lead to consequent savings. Finally, the only reliable way to compare those software would have been to run those on some known data set (that should include occurrences of the faults we are interested in) and to confront their results. Nevertheless, this could not be done since we did not get access to any of those software.

Chapter 4

Model-based fault detection in wind turbines

In this chapter, we will try to build model-based fault detection methods for wind turbines. In that purpose, we will use (among others) some of the tools presented in the previous chapters. In section 4.1, we will monitor the gearbox temperature. In section 4.2, we will focus on temperatures measurements around the generator. We will come to the conclusion that model-based fault detection (see chapter 2) outrivals straightforward statistical analysis (see chapter 1) for early fault detection. Let us start with a brief description of the data.

Data description We are provided with data from 24 wind turbines. The 24 wind turbines are denoted by E0RY, E1RY, E2RY, E3RY, E4RY, E5RY, E6RY, E1RC, E2RC, E3RC, E4RC, E5RC, E6RC, E1F, E2F, E3F, E4F, E5F, E1V, E2V, E3V, E4V, E5V, E6V. Data contains measurements of 48 signals every 10 minutes. All of those signals are not necessarily useful in our case. So, when identifying a model, we will make our choice among the 34 following signals (complementary information on some of those signals are given in appendix A page 54)

- **Wind/orientation**: wind speed, nacelle deviation, nacelle position,
- **Hub**: hub speed, main bearing temperature, blade A pitch position, blade B pitch position,blade C pitch position, blade A set value, blade B set value, blade C set value,
- **Gearbox**: gearbox lube oil temperature, gearbox lube oil pressure, gearbox axial global vibration, high speed gearbox bearing temperature,
- **Generator**: generator speed 1, generator speed 2, generator DE (Driven-End) bearing temperature, generator NDE (Non-Driven-End) bearing temperature, generator winding temperature A, generator winding temperature B, generator cooling air temperature,
- **Transformer**: transformer temperature,
- **Output power**: active power, reactive power, power factor,
- **Output voltage**: phase voltage A, phase voltage B, phase voltage C,
- **Output current**: phase current A, phase current B, phase current C,
- **Other**: ambient temperature, nacelle temperature.

Data filtering Before proceeding to model identification, we will always filer the data. This filtering stage should remove

1. points where data were not collected (NAN entries)
2. points where the wind turbine is not operating (and sometimes points corresponding to start-ups and shutdowns - in short, points where the rotation speed is low).

Error To compute values of the model parameters, we will minimize the root-mean-squared error (RMSE)

$$\sqrt{\frac{1}{N}\sum_{t=1}^{N}\left(y(t)-\hat{y}(t)\right)^2},$$

which will also constitute a first indicator of the performance of the model.

Alarms In the sequel, we will often talk about 'recurrent alarms' being launched. Since this terminology may seem rather broad or fuzzy, it will always be accompanied by at least one figure illustrating the alarms. Notice that the first alarm of a sequence of recurrent alarms will always be chosen as the reference point when comparing different methods.

4.1 Increasing high speed gearbox bearing temperature

The fault we will consider in this section is an increase in the high speed gearbox bearing temperature (or similarly in the bearing lubricant oil temperature) which may indicate a degradation in the lube oil quality. The input shaft of the gearbox is the main shaft driven by wind. The output shaft is the generator shaft. The role of the gearbox is to increase the rotation speed so that the generator rotation frequency is high enough. This goal is reached by simply multiplying the rotation speed by approximately 90 (so, we should use the term multiplier rather than gearbox). A schematic representation of the gearbox can be found in figure 4.3 page 29.We are here interested in the temperature of the gearbox output shaft bearing.

Figure 4.1 shows an example of increasing high speed gearbox bearing temperature on machine E5F. In this section, we will build several methods to detect this increase in temperature and compare them. Obviously, we are interested in detecting the fault earlier than feasible by simply looking at the curve of figure 4.1. At this stage, we can not give some precise day for the appearance of the increase in the temperature. In the sequel, we will first proceed to straightforward fault detection by analyzing the temperature signal thanks to univariate statistical change detection (see section 1.1.2) in section 4.1.1. Afterwards, we will proceed to model-based fault detection in section 4.1.2. We will come to the conclusion that the model-based approach allows to detect the fault almost one month earlier in some cases.

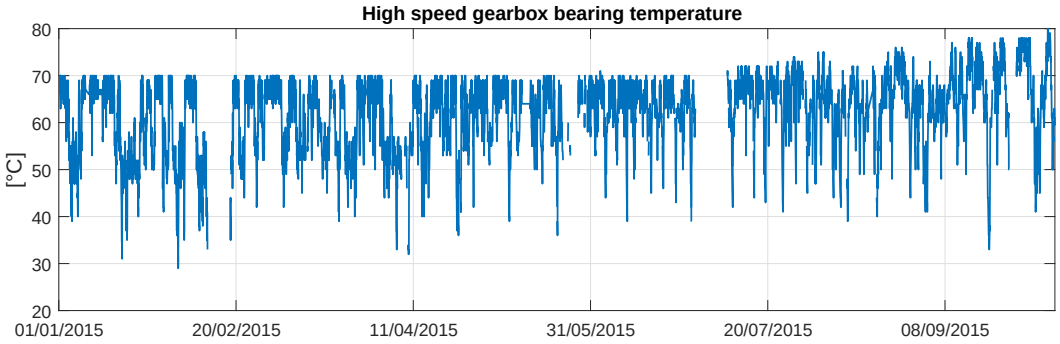


Figure 4.1: Increasing high speed gearbox bearing temperature for wind turbine E5F.

4.1.1 Straightforward univariate statistical change detection

We will here directly analyze the high speed gearbox bearing temperature signal (using univariate statistical change detection, see section 1.1.2) to provide a first idea for the time of the fault detection. Based on figure 4.2, we see that the fault occurs before July. So, we will hope to detect the fault earlier with more advanced methods.

4.1.2 Model-based fault detection

The goal here is to build a model predicting the high speed gearbox bearing temperature in the hope that analysis of the residuals will lead to earlier fault detection. Let us first identify features that may be useful in that purpose. Here, we will first select interesting features based on the knowledge we have of the system: among all the signals measured in the data, one can identify several signals more or less related to the gearbox. Figure 4.3 summarizes the signals we will consider. To select the signals we will use as inputs to our model among those signals, we will firstly remove some inconsistent signals from figure 4.3 and then remove signals that do not improve the quality of the predictive model. It is obvious that we can not use the gearbox lube oil temperature as an input to our system since it is directly linked to the bearing temperature (i.e. an increase in the bearing temperature goes hand in hand with an increase in the lube oil temperature). Besides, we can choose equivalently between the hub speed and the generator speed and remove the other one because those signals are always linearly linked (see above). Eventually, we will be reluctant to include the main shaft bearing metal temperature and the generator DE bearing temperature as input features because those could be faulty as well, leading to bad prediction.

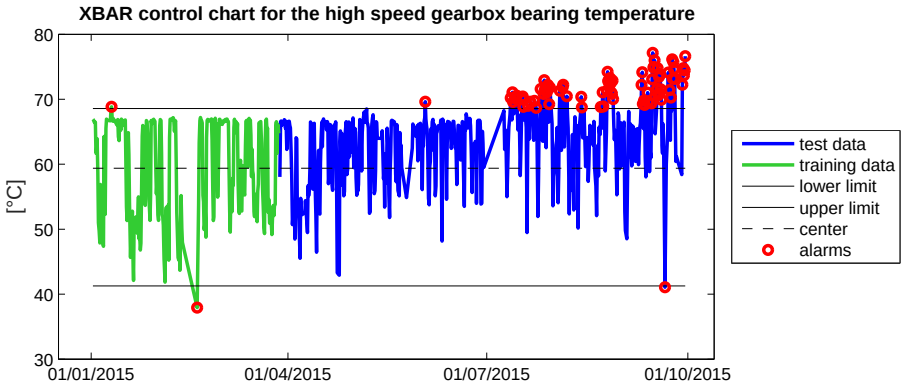


Figure 4.2: Xbar control chart for the high speed gearbox bearing temperature. The signal shown is an average over 400 minutes (40 measurements) and the false alarm rate is 1%. Those two parameters were arbitrarily selected but other values do not lead to sharper conclusions. There are recurrent alarms starting from 11 July 2015 18:32. Notice that recurrent alarms could maybe be observed earlier if we had data between 29 June 2015 and 08 July 2015.

Machine	Recurrent alarms XBAR	Date of correction
E1F	12/07/2015 00:12	03/06/2016 13:59
E2F	/	/
E3F	03/05/2015 23:43	29/06/2015 14:03
E4F	/	/
E5F	11/07/2015 18:32	12/10/2015 16:56
E0RY	/	/
E1RY	/	/
E2RY	/	/
E3RY	/	/
E4RY	/	/
E5RY	30/05/2016 17:28	13/10/2016 13:31
E6RY	/	/

Table 4.1: Alarms for machines E•F and E•RY using univariate statistical change detection. This provides an idea of the faulty machines but is not efficient at early fault detection. The recurrent alarms column gives the date from which recurrent alarms are observed. The date of correction is the last time we observe the wind turbine is faulty in the data. This does not mean this is the day the fault was effectively detected by the operator (surely it was detected earlier but the operator deliberately decided to let it run until repair were performed).

It is clear that the gearbox bearing temperature is influenced by the ambient temperature (we can check that not including the ambient temperature as an input feature leads to poor model identification). To get a training set representative of all the possible ambient temperatures, it would be necessary to span this training set on a whole year (or at least half a year). Since we are provided with data collected on one or two years, performing the learning step on a whole year (and taking the ambient temperature as an input to the system too) would leave too few data for the test step. So, we won't be able to build that ideal model here because our learning set has to span only on a maximum of three months (because, as we will see later, some faults occur after three months). Consequently, we will have to find some trick to deal with that problem. This is done in section 4.1.2.4 by progressively increasing the training set.

In the sequel, we will adjust and compare three different neural network models (see chapter 2). We will consider the three following models: a simple nonlinear input-output model, an auto-regressive model and an auto-regressive model with past estimations as input to the model. For validation and for tuning of the meta-parameters of the models (the number of hidden units and the number of past inputs and outputs to take into account to model the dynamic), we will use wind turbine E4F, since this is one of the few non-faulty machine for which we have data on at least two entire years. So, meta-parameters will be identified using training and test sets spanning on a full year. Afterwards, a model using those meta-parameters will be learned and run on wind turbines E•F and E•RY. Finally, notice that we may have introduced some optimism for wind turbine E4F by tuning meta-parameters thanks to that wind turbine (i.e. the model may be better for that wind turbine). As we will see later, the model does not necessarily performs better for that wind turbine (see table 4.3).

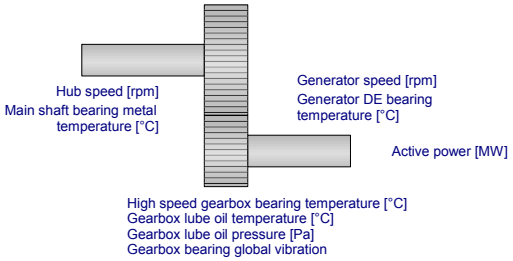


Figure 4.3: Schematic gearbox and some of the measured signals.

4.1.2.1 model 1: nonlinear input-output

For predicting the high speed gearbox bearing temperature, we first build a model with external dynamic of the type

$$\hat{y}(t) = f(u_1(t), u_1(t-1), \ldots, u_1(t-n_1), u_2(t), u_2(t-1), \ldots, u_2(t-n_2), \ldots, u_{d_1}(t), u_{d_1}(t-1), \ldots, u_{d_1}(t-n_{d_1})),$$

where y is the output signal and $u_1, u_2, \ldots, u_{d_1}$ are the input signals (which we will precise below). The parameters $n_1, n_2, \ldots, n_{d_1}$ give the external dynamic.

Based on table 4.2, we see that only the generator rotation speed, the ambient temperature and the active power are useful input signals. Indeed, the gearbox lube oil pressure does not contribute to lower the RMSE. So does the gearbox bearing global vibration (though it is not shown in the table for conciseness). Remember that we decided not to consider the generator DE bearing temperature and the main shaft bearing temperature (or more generally temperature signals other than the ambient temperature) as inputs because we want a model that is not sensitive to the occurrence of a fault on those bearings.

Notice that the values of $u_1, u_2, \ldots, u_{d_1}$ may be high. Since temperatures are varying slow with respect to the generator speed, this seems consistent: lots of past rotation speeds may influence the present temperature. An auto-regressive model may help to decrease those parameters values.

Because the sampling for measurements is 10 minutes, we do not have any information on the rotation speed in between two measurements with this model, which may have a bad influence on the performance of the model. This is a second reason why we are interested in testing an auto-regressive model.

4.1.2.2 model 2: nonlinear auto-regressive with external exogenous inputs (NARX)

For predicting the high speed gearbox bearing temperature, we now build a model with external dynamic of the type

$$\begin{aligned}\hat{y}(t) =& f(u_1(t), u_1(t-1), \ldots, u_1(t-n_1), u_2(t), u_2(t-1), \ldots, u_2(t-n_2), \ldots, u_{d_1}(t), u_{d_1}(t-1), \ldots, u_{d_1}(t-n_{d_1}),\\ & \ldots, y(t-1), y(t-2), \ldots, y(t-m_1)),\end{aligned}$$

where y is the output signal and $u_1, u_2, \ldots, u_{d_1}$ are the input signals. The parameters $n_1, n_2, \ldots, n_{d_1}, m_1$ give the external dynamic. We expect the auto-regressive model will lead to a notably smaller RMSE than the input-output model 1 presented above. We can observe that in table 4.2.

Notice that the parameters $n_1, n_2, \ldots, n_{d_1}, m_1$ are now much smaller as compared to model 1 (especially for the generator speed signal). That can be explained as follows: some of the history needed by model 1 is now 'stored ' in the gearbox temperature inputs $y(t-1), \ldots, y(t-m_1)$.

4.1.2.3 model 3: NARX with previous estimations

The last model we will consider for predicting the high speed gearbox bearing temperature is identical to model 2 except that we use the estimation $\hat{y}$ rather than the true signal y as input to our model, i.e.

$$\begin{aligned}\hat{y}(t) =& f(u_1(t), u_1(t-1), \ldots, u_1(t-n_1), u_2(t), u_2(t-1), \ldots, u_2(t-n_2), \ldots, u_{d_1}(t), u_{d_1}(t-1), \ldots, u_{d_1}(t-n_{d_1}),\\ & \ldots, \hat{y}(t-1), \hat{y}(t-2), \ldots, \hat{y}(t-m_1)),\end{aligned}$$

where y is the output signal and $u_1, u_2, \ldots, u_{d_1}$ are the input signals. The parameters $n_1, n_2, \ldots, n_{d_1}, m_1$ give the external dynamic. The motivation for considering such a model is that we want to test whether deviations between y and $\hat{y}$ will be spotted earlier than with the auto-regressive model. We were expecting this model will lead to a smaller RMSE than the input-output model 1 but higher RMSE than auto-regressive model 2. However, this model scores rather bad (see table 4.2). The explanation is that we try keeping the parameters $n_1, n_2, \ldots, n_{d_1}, m_1$ low while the inputs $\hat{y}(t-1), \ldots, \hat{y}(t-m_1)$ are not close enough to the true values $y(t-1), \ldots, y(t-m_1)$. Increasing those parameters will lead to an optimal model similar to model 1 with $\hat{y}$ being an useless input. Since this model scores no better than model 1 or 2, we won't use it in the sequel.

	Model description	Input feature(s)	Opt. ext. dyn. para.	RMSE training set	RMSE test set
model 0	k-nearest neighbors $k = 1$	Generator speed	0	0	3.65
model 1	Nonlinear input-output 2-layer NN 2 hidden units	Generator speed	0 : 15	2.22	2.71
		Generator speed Ambient temperature	0 : 15 0 : 5	2.08	2.43
		Generator speed Ambient temperature Active power	0 : 15 0 : 5 0 : 5	1.83	2.19
		Generator speed Ambient temperature Active power Lube oil pressure	0 : 15 0 : 5 0 : 5 0 : 5	1.84	2.20
model 2	NARX 2-layer NN 3 hidden units	Generator speed Gearbox temperature	0 : 3 1	0.91	1.02
		Generator speed Ambient temperature Gearbox temperature	0 : 3 0 : 2 1	0.88	0.99
		Generator speed Ambient temperature Active power Gearbox temperature	0 : 3 0 : 2 0 : 2 1	0.85	0.94
model 3	NARX with estimations 2-layer NN 3 hidden units	Generator speed Gearbox temperature	0 : 10 1	2.16	2.78
		Generator speed Ambient temperature Gearbox temperature	0 : 3 0 : 3 1	2.27	2.73
		Generator speed Ambient temperature Active power Gearbox temperature	0 : 3 0 : 3 0 : 3 1	1.85	2.41

Table 4.2: Root mean squared error using different models on wind turbine E4F (only the best scores among the different tested models are presented here). The external dynamic parameter gives the number of previous values of the input or output signals that will be fed to the model (i.e. $n_1, n_2, \ldots, n_{d_1}$ and m_1). The whole data are split in two to constitute training and test sets. Training set spans from 01/01/2015 to 01/01/2016. Test set spans from 01/01/2016 to 01/01/2017. Since the RMSE is not a relative error criterion, let us mention that the maximal and minimal values of the high speed gearbox bearing temperature are 70°C and 11°C and observe that the very simple model 0 (kNN with $k = 1$ and no external dynamic) lead to a RMSE of 3.65. We should also mention that the measured temperatures are integer values (so, the accuracy of the best predictive model is of the same order as the accuracy of the measurements). Notice that model 1 require high values of parameters $n_1, n_2, \ldots, n_{d_1}$. Auto-regressive model 2, as expected, scores better than model 1. Model 3 scores surprisingly bad. An explanation for that may be that the true temperature measurement $y(t-1)$ contains much more information than the estimation $\hat{y}(t-1)$ (i.e. $\hat{y}(t-1)$ is not close enough to $y(t-1)$).

4.1.2.4 Fault detection using the models: progressive increase of the training set

Now we have identified optimal values of the meta-parameters for our 3 models thanks to table 4.2, we will proceed to fault detection. We will only use models 1 and 2 introduced above since those seem the give the best predictions. Besides, we will progressively increase the training set with new time periods as soon as we can guarantee a time period contains no fault.

In table 4.2, we used one of the few wind turbines for which we have data on 2 whole years without faults in the high speed gearbox bearing temperature. For most of the other wind turbines, we can not afford learning models on one full year because of the lack of data (or the lack of data before the occurrence of a fault). So, in the sequel, we will proceed as follow

1. Suppose we have non-faulty data for the first N months, say for example $N = 2$ or 3 months (notice that this is an assumption needed for any fault detection method),
2. Use those data to learn a model (this model may be poor if the training set of point 1 is too small),

3. Select a number of months M, say for example 2 or 3,

4. Monitor the system online:
   If no faults was detected (using the model learned on the training set) during the months $N+1$ to $N+M$ included:

   - add month $N+1$ to the training set and learn the model again,
   - $N \leftarrow N+1$,
   - continue monitoring.

In other words, we require that no fault is detected during the following $M-1$ months before adding month $N+1$ to the training set. This ensure we won't add to the training set a month containing undetected early faults indicators. Indeed, we expect the undetected faults of month $N+1$ will become more and more critical and easier detectable in the following $M-1$ months.

Notice that this approach amounts to simulating the actual implementation of a fault detection tool from the very first use of the turbine. Though we chose the month as a time unit above, we could imagine the same scheme with days or weeks as time units.

The main drawback of this approach is that we might have many false alarms while the learning set is too small. Indeed, if the initial value for N is too small, the training set may not be representative enough of all the possible states of the process. This may lead to false alarms because model prediction will be bad for new unknown but non-faulty points. So, the choice of the initial value for N is critical.

A second drawback is that some faults may be detected later, especially during the first months of the implementation of this tool if N is not high enough. Indeed, because of the model being not very accurate, we may not be able to detect small changes in the residuals due to a fault.

4.1.2.5 Results

Table 4.3 presents results of applying the method explained in 4.1.2.4 (with $N = 3$ and $M = 2$) on a whole year (year 2015 for machines E•F and year 2016 for machines E•RY) to all of the wind turbines for which we have enough data (we do not have enough data for machines E•V and E•RC in 2016). Recurrent alarms are reported. Notice that we did not use all the data points we were provided with. Indeed, we used a filter to avoid doing predictions when the ambient temperature was never previously observed (i.e. is not part of the training set). Indeed, since this signal depends on the season, the number of initial months N could never be less than 12 (or 6 if we consider half a year is representative of a full year) without this filter.

There are two points we should discuss now.

1. **The ability to correctly identify the system and the false alarms due to bad identification.** Non-faulty wind turbines E2F, E4F, E0RY, E2RY, E3RY, E6RY lead to few false alarms and overall 'good ' predictions with model 1 and model 2. Non-faulty wind turbine E1RY is well identified with model 2 but not with model 1 (false alarms). Identification of non-faulty wind turbine E4RY leads to some alarms in mid-2016 because of some never experienced temperatures (74°C though temperatures is usually under 70°C). For machine E3RY, prediction is close to the measurements until the end of the year where a lack of data seems to indicate a change in the system characteristic (maybe due to external intervention). Only those alarms are not reported in the table. In conclusion, model 2 gives nice results for 6 out of 7 non-faulty wind-turbines. Figure 4.4 presents the results of monitoring wind turbine E2F on year 2015. We can observe there are few false alarms.

2. **The ability to detect faults.** Increase in the high speed gearbox bearing temperature are detected on machines E1F, E3F, E5F and E5RY (it is interesting to compare this table with table 4.1). For machine E5F, model-based fault detection allow to detect the increase in temperature almost one month earlier (see also figure 4.2 where the fault was detected on 11/07/2015) than statistical change detection applied to the temperature signal. For machine E1F, statistical change detection applied to the temperature signal will give recurrent alarms starting from 12/07/2015. So, model-based fault detection detect the fault approximately one month earlier too. For machines E5RY and E3F, the fault occurs too early in the data to discuss the early fault detection nature. Figure 4.5 presents results of monitoring wind turbine E5F on the year 2015. We can observe recurrent alarms starting from 11/06/2015. Finally, notice that there is no rule telling whether or not model 1 is better than model 2 and whether or not XBAR control charts are better than EWMA control charts. This is because it depends on the nature of the fault (e.g. whether the fault is brutal or not).

Besides, we should discuss our choice of values for M and N. The initial number of months in the training set was taken to $N = 3$ because a smaller value may lead to a too small training set (in particular, it will contain too few different values for the ambient temperature signal). We took $M = 2$ because we want M to be

as big as possible (so that we ensure no faulty data were included in the training set) while still being able to identify a good predictive model for lots of points (remember that a filter is used, rejecting points with ambient temperatures never experienced in the training set). So, we have a trade-off here between the risk of considering the early symptoms of a fault as normal and the accuracy of the model. We picked $M = 2$ because $M = 3$ does not lead to earlier fault detection while the accuracy of the model with $M = 3$ is a little less than with $M = 2$.

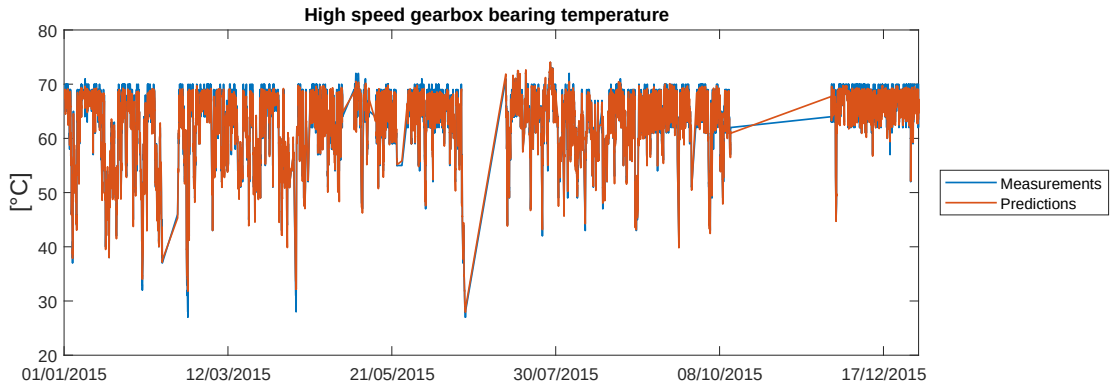
Eventually, we should notice there may be some instability in the results, especially during the first few months. Indeed, we may observe that some isolated points are badly predicted with this model. Those points may be considered as hard points. The prediction is bad because we picked N rather small and the N first months contain thus few hard points. In short, when running the model several times, we may obtain different false alarms. This is because the scaled conjugate gradient algorithm in charge of learning the model converges to a local minimum, that may depend on the randomly chosen starting point. Nevertheless, this does not have an influence on the occurrence of recurrent false alarms (but the reader should be aware that he may get slightly different results when running the codes available in appendix B).

Machine	Model	RMSE	Recurrent alarms XBAR	Recurrent alarms EWMA
E1F	1	2.48	15/06/2015 06:01	13/06/2015 20:38
	2	1.09	12/07/2015 04:02	13/06/2015 21:58
E2F	1	3.02	/	/
	2	1.11	/	/
E3F	1	4.78	01/04/2015 04:00	01/04/2015 10:40
	2	1.34	17/04/2015 03:20	17/04/2015 10:00
E4F	1	3.00	/	/
	2	0.87	/	/
E5F	1	3.05	14/06/2015 11:08	14/06/2015 11:08
	2	1.26	15/06/2015 20:42	11/06/2015 02:18
E0RY	1	2.14	/	/
	2	1.25	/	/
E1RY	1	1.82	/	/
	2	0.77	/	/
E2RY	1	1.51	/	/
	2	1.03	/	/
E3RY	1	1.74	/	/
	2	0.91	/	/
E4RY	1	2.77	/	/
	2	1.10	/	/
E5RY	1	2.59	02/03/2016 05:32	01/03/2016 22:52
	2	0.84	01/03/2016 10:42	01/03/2016 16:42
E6RY	1	1.49	/	/
	2	1.14	/	/

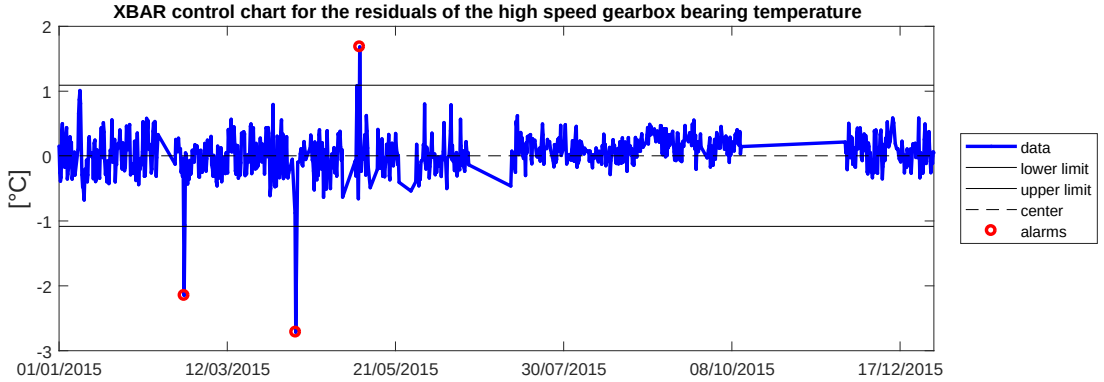
Table 4.3: Alarms for all machines using online monitoring with $N = 3$ (except for machines E3F and E5RY where we used $N = 2$) and $M = 2$. The change in the residuals is detected using either mean-standard deviation or EWMA control charts. For machines E3F and E5RY, we had to shorten the initial learning set since the fault occurs very early (having data for the end of the year 2014 would lead to earlier fault detection). Some further minor adaptations had to be done for those machines too (so, results for those machines may perhaps be improved).

4.1.3 Conclusion

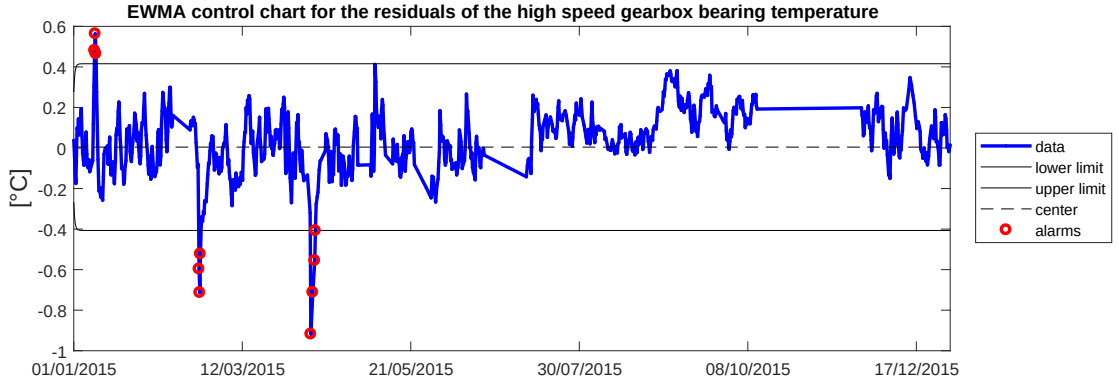
In conclusion, we built here a predictive model performing early fault detection by exploiting physical knowledge of relationships linking different signals. Thanks to this model, we were able to detect faults one month earlier than a simple univariate analysis. A gain of one month seems simultaneously interesting and realistic when compared to the time elapsed until the fault is fixed. The time elapsed between the first alarms using univariate analysis and the reparation is typically three or four months (see table 4.1). So, a one month gain is not negligible.



(a)

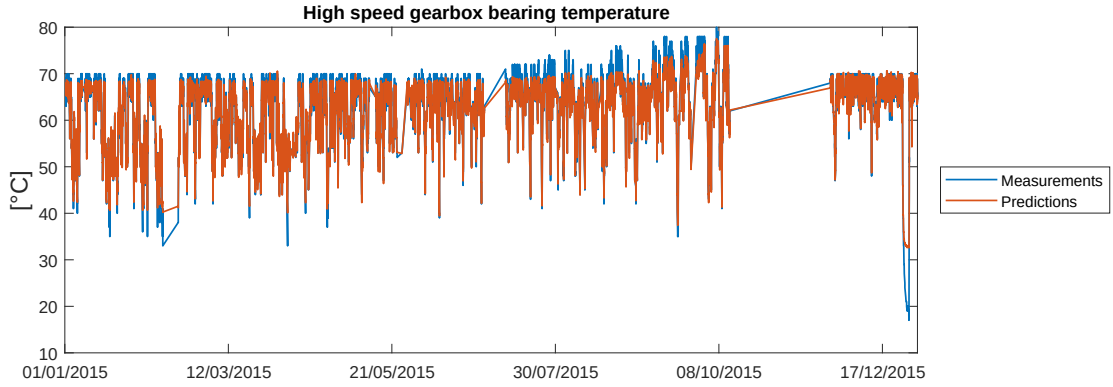


(b)

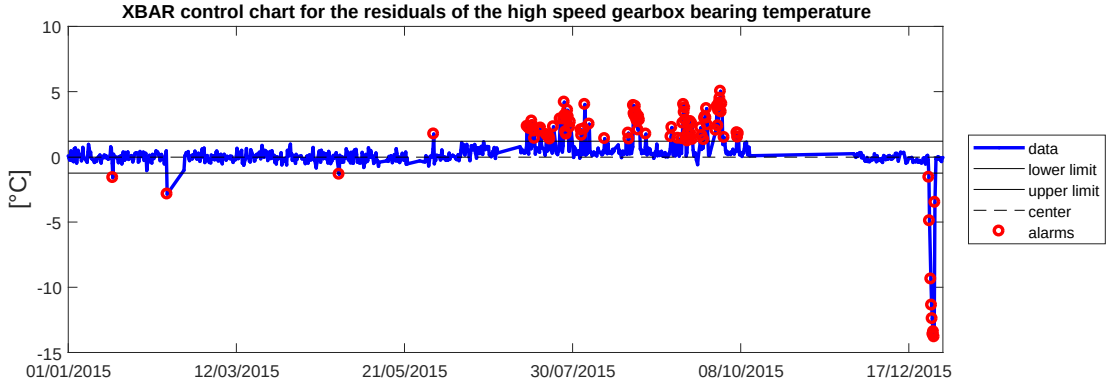


(c)

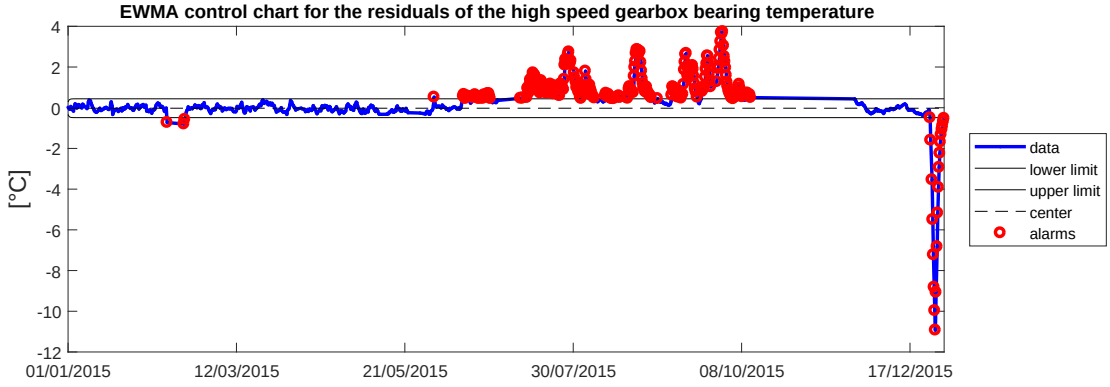
Figure 4.4: Outputs of applying the online monitoring method ($N = 3, M = 2$) to machine E2F using model 2 on the year 2015. (a) Measurements and two-months ahead predictions (b) Xbar control chart for the residuals using samples of size 40 (lower and upper limits are 3 standard deviations from the mean) (c) EWMA control charts for the residuals using samples of size 40 and the following parameters: $\lambda = 0.25$ and $k = 3$ (see section 1.1.2.1.3 for signification of those parameters). No recurrent alarms are observed.



(a)



(b)



(c)

Figure 4.5: Outputs of applying the online monitoring method ($N = 3, M = 2$) to machine E5F using model 2 on the year 2015. (a) Measurements and two-months ahead predictions. Notice that we did not stop online updating of the training set though recurrent alarms were observed (this is why the model is still able to output high temperatures values after 2 faulty months). (b) Xbar control chart for the residuals using samples of size 40 (lower and upper limits are 3 standard deviations from the mean). There are recurrent alarms starting from 15/06/2015. (c) EWMA control charts for the residuals using samples of size 40 and the following parameters: $\lambda = 0.25$ and $k = 3$ (see section 1.1.2.1.3 for signification of those parameters). There are recurrent alarms starting from 11/06/2015.

4.2 Model for the generator

In this section, we will build a model-based fault detection tool for the temperature signals related to the generator. We will consider the four following temperature signals: the generator driven-end (DE) temperature, the generator non-driven-end (NDE) temperature, the winding temperature A and the winding temperature B. The two first signals are measurements of two bearing temperatures while the two last ones are two measurements of the winding temperature. Again, input signals to the models will be selected based on physical knowledge on the one hand and on tests on the other hand. We will see that the generator bearing rotation speed and the ambient temperature will be selected. Outputs signals will alternatively be the two bearing temperatures (driven end and non-driven end bearings) and the two winding temperatures. We will eventually highlight an abnormal behavior for the generator NDE bearing temperature on wind turbine E3F.

4.2.1 Generator DE and NDE bearing temperatures

Here, we will identify an auto-regressive model for the generator DE and NDE bearing temperatures. Though those two signals are highly correlated (correlation of 0.95), we will build a model for each of those two signals. Another solution could be to build a model for monitoring one of the two signals only and use the correlation to monitor the second signal.

We will use an approach identical to what we have done in section 4.1.2.4 where the training set was progressively increased with new periods once we can ensure no fault has occurred during the considered period.

4.2.1.1 Model identification

For each of the generator DE and NDE bearing temperature signals, we will identify an auto-regressive model

$$\begin{aligned}\hat{y}(t) =& f(u_1(t), u_1(t-1), \dots, u_1(t-n_1), u_2(t), u_2(t-1), \dots, u_2(t-n_2), \dots, u_{d_1}(t), u_{d_1}(t-1), \dots, u_{d_1}(t-n_{d_1}),\\ & \dots, y(t-1), y(t-2), \dots, y(t-m_1)),\end{aligned}$$

where y is the output signal and $u_1, u_2, \dots, u_{d_1}$ are the input signals. The parameters $n_1, n_2, \dots, n_{d_1}, m_1$ give the external dynamic. Optimal values for those parameters as well as for the net hyper-parameters are identified using data from machines E•F. Table 4.4 presents optimal values found for those parameters and justify why we do not select features such as the active power that barely decrease the RMSE. This table only shows results for predicting the generator DE bearing temperature. The conclusions are similar for the generator NDE bearing temperature (remember that both signals are highly correlated and it is thus consistent that they are well predicted by very similar models).

	Model description	Input feature(s)	Opt. ext. dyn. para.	RMSE	
				training set	test set
model 1	NARX 2-layer NN 3 hidden units	Generator speed Ambient temperature Generator DE temperature	0 : 3 0 : 2 1	0.49	0.50
		Generator speed Ambient temperature Generator DE temperature Active power	0 : 3 0 : 2 1 0 : 2	0.47	0.48

Table 4.4: Model performances for the generator DE bearing temperature. The training set is made of data from wind turbine E4F on year 2015. The test set is the data from machine E4F during year 2016. The feature active power does not contribute to strongly decrease the RMSE and will not be selected. We did not find other non-temperature features decreasing the RMSE.

4.2.1.2 Application and progressive extension of the learning set

Because of the lack of data, we can not learn the model on a full year for every wind turbine. So, we will again proceed as in section 4.1.2.4, by progressively increasing the learning set. A month will be added to the learning set if no fault is detected during the two following months. The model is learned again every time we add data to the learning set. For example, for machines E•F, the initial training set will span from 01/07/2015 to 01/03/2016. We will then let the model run until 01/08/2016. So, the initial training set is 8 months long (notice however that we do not have data from 12/10/2015 to 24/11/2015, so we only have 7 months of training data) and the monitoring will last for 5 months.

Results are given in table 4.5 for wind turbines E•F and E•RY. We can see one anomaly is detected. Figure 4.7 shows the monitoring of the residuals for a healthy wind turbine (E5F) while figure 4.8 shows the monitoring for a faulty wind turbine (E3F) on the same period. For that wind turbine, the NDE bearing temperature turns out to increase by the end of April 2016 as suggested by figure 4.8(b). On figure 4.6, we can see that NDE bearing temperature curve tends to be above the DE bearing temperature curve starting from the end of April though both curves were very close before. Notice also that the change in the NDE bearing temperature signal is sudden. Indeed, recurrent alarms on figure 4.8(b) are sudden and of high intensity. So, we are not facing a slowly developing anomaly. Figure 4.6 seems to confirm that the NDE bearing temperature deviates suddenly from the DE bearing temperature. Consequently, other types of control statistics specially developed for slowly developing changes are not helpful here and do not lead to earlier detection, as we can see from table 4.5.

Eventually, notice that a simple method performing fault detection thanks to correlation between the two signals will launch alarms as early as the above method (this was expected based on figure 4.6). The only advantage of the above method is that we can identify which of the two signals is faulty.

4.2.1.3 Conclusion

In conclusion, for the particular observed anomaly, we can not present consequent results in terms of early fault detection because some simple method (comparing the DE and NDE bearings temperatures) lead to similar performances and the fault looks rather sudden. However, this method is better for fault identification since the simple method is not able to tell which of the 2 temperatures is faulty.

Machine	Signal	RMSE	Recurrent alarms XBAR	Recurrent alarms EWMA
E1F	DE Temp.	0.48	/	/
	NDE Temp.	0.49	/	/
E2F	DE Temp.	0.51	/	/
	NDE Temp.	0.50	/	/
E3F	DE Temp.	0.47	/	/
	NDE Temp.	0.63	06/07/2016 01:16	25/04/2016 01:16
E4F	DE Temp.	0.51	/	/
	NDE Temp.	0.57	/	/
E5F	DE Temp.	0.52	/	/
	NDE Temp.	0.53	/	/
E0RY	DE Temp.	0.50	/	/
	NDE Temp.	0.47	/	/
E1RY	DE Temp.	0.51	/	/
	NDE Temp.	0.59	/	/
E2RY	DE Temp.	0.48	/	/
	NDE Temp.	0.53	/	/
E3RY	DE Temp.	0.43	/	/
	NDE Temp.	0.46	/	/
E4RY	DE Temp.	0.49	/	/
	NDE Temp.	0.48	/	/
E5RY	DE Temp.	0.51	/	/
	NDE Temp.	0.53	/	/
E6RY	DE Temp.	0.50	/	/
	NDE Temp.	0.49	/	/

Table 4.5: Results for monitoring of the generator DE and NDE bearing temperatures by progressively increasing the learning set. Notice that the fault detection is much easier using EWMA control chart than XBAR control chart. This is because the XBAR control chart is not able to detect a change in the mean relatively small with respect to the standard deviation.

4.2.2 Generator winding temperature

Models similar to the models presented in section 4.2.1 were developed for predicting the generator winding temperatures A and B. Those models appeared to perform well for prediction too. Nevertheless, no faults were detected using those models on wind turbines E•F and E•RY (which was expected because the data does not seem to contain any anomaly linked to those signals). Consequently, we won't go deeper into details here.

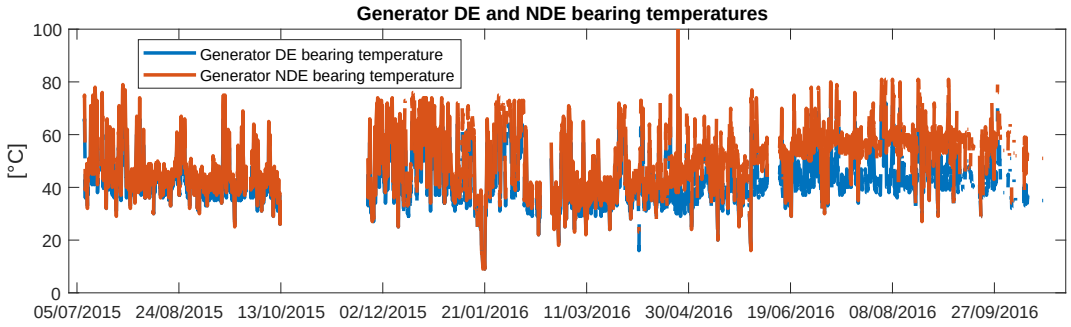


Figure 4.6: Generator DE and NDE bearing temperatures for wind turbine E3F. Observe that the signals tend to deviate from each other starting from late April 2016. The NDE temperature seems to be high with respect to the DE temperature.

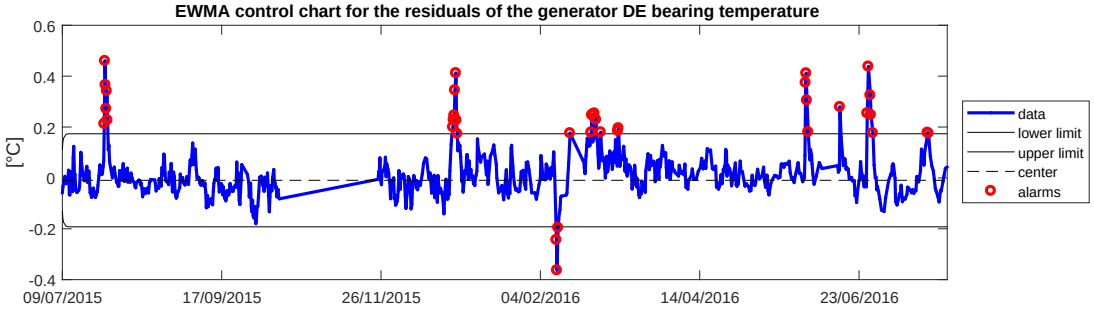
4.3 Conclusion

In this chapter, we developed model-based tools for fault detection in wind turbines. Though we focused mainly on temperature signals (because we knew from the start some of them contain faults), this approach can be used for many of the signals introduced at the beginning of the chapter.

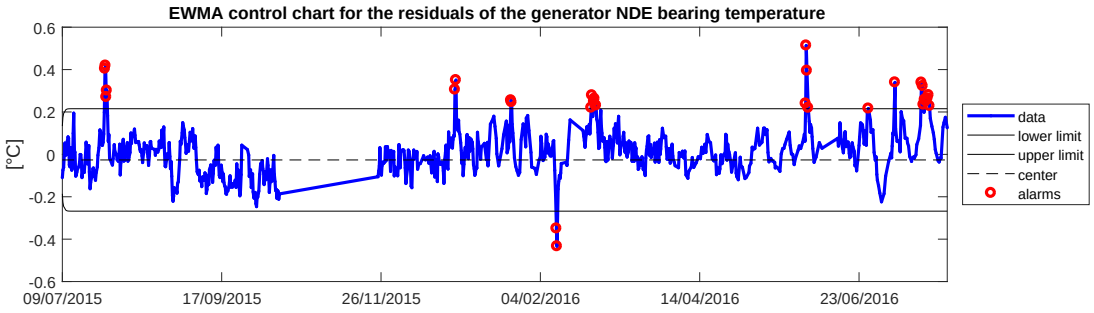
The first step of model-based fault detection is the identification of the model. We saw in section 4.1 the superiority of model-based approach on univariate statistical change detection. We also compared the performances of different models (see table 4.2). It turns out that the model with the smallest RMSE often lead to the earliest fault detection, as expected. However, in many cases, the difference with a model with a higher RMSE was not so consequent (3 days at maximum, which is not huge with respect to the time scale). Indeed, though the variance of the residuals is higher, the change in the residuals distribution because of the fault is still easily detectable.

The second step of model-based fault detection is the analysis of the residuals. In that regard, table 4.2 shows that we can sometimes detect a fault a few days earlier thanks to a well-chosen analysis method.

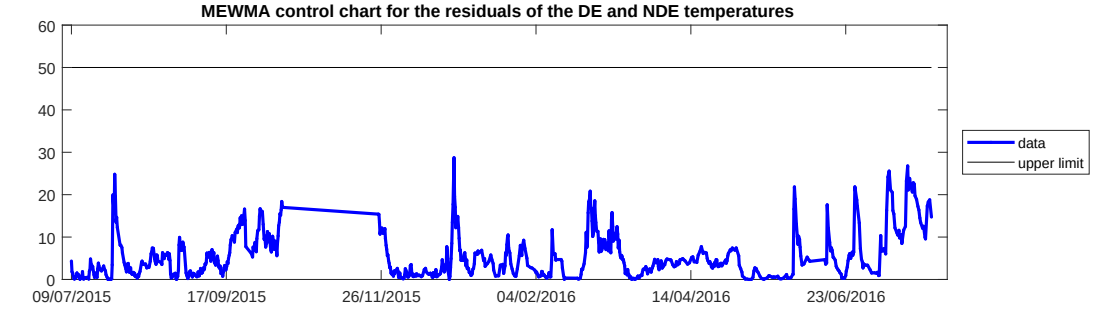
In conclusion, for the particular faults and the particular models studied in this chapter, the major gain in fault detection are due to the use of a model (few gains are made thanks to the use of more complicated statistical analysis tools).



(a)

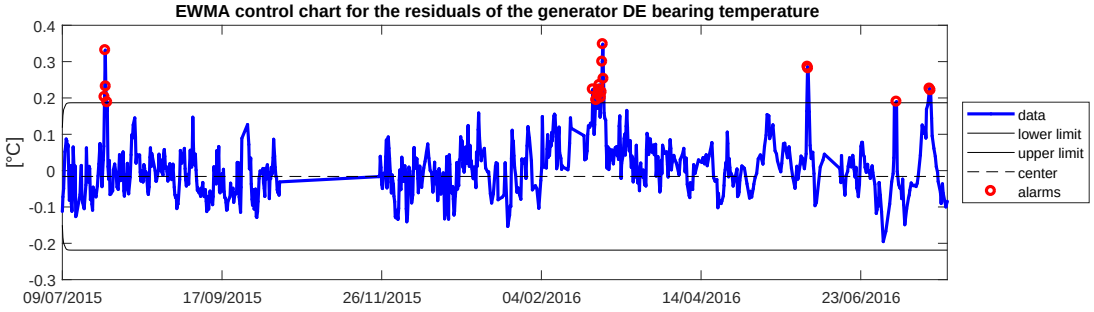


(b)

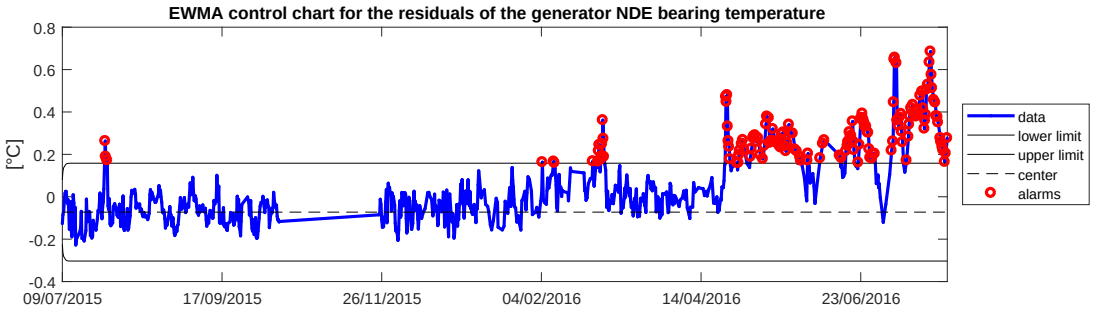


(c)

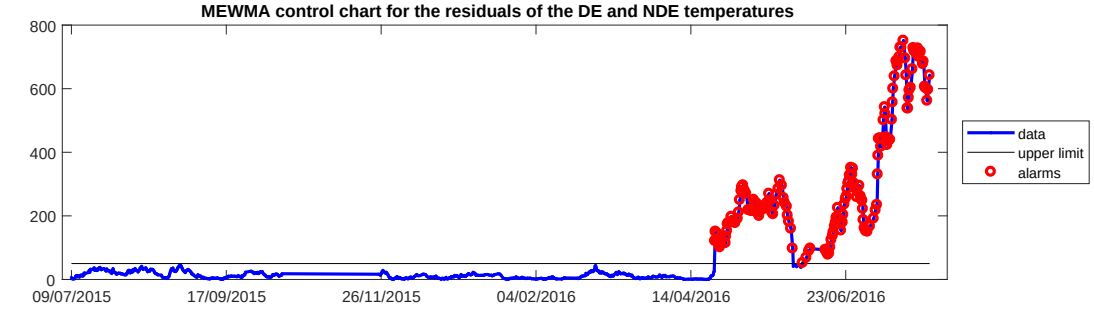
Figure 4.7: Control charts for the residuals of wind turbine E5F. (a) EWMA control chart for the residuals of the generator DE bearing temperature and (b) EWMA control chart for the residuals of the generator NDE bearing temperature. (c) MEWMA control chart for the simultaneous montioring of both residuals. There are a few alarms and no recurrent alarms. The sample size is always $n = 40$ and $\lambda = 0.25$.



(a)



(b)



(c)

Figure 4.8: Control charts for the residuals of wind turbine E3F. (a) EWMA control chart for the residuals of the generator DE bearing temperature and (b) EWMA control chart for the residuals of the generator NDE bearing temperature. (c) MEWMA control chart for the simultaneous montioring of both residuals. There are recurrent alarms starting from 25/04/2015 01:16 for the two last charts.

Chapter 5

Blind fault detection

In the previous chapter, we performed model-based fault detection relying mostly on physical understanding of the system to identify signals that may be related with each other and to build a model for some components of the wind turbine. We are now interested in building a method for fault detection without having *a priori* knowledge on the relationships between the measured signals. Such a generic method could thus be applied on any process. This will be done at the price of neglecting the dynamic of the system. As we will see, this method will detect faults no earlier than methods of chapter 4 (which was expected since we do not use the knowledge we might have on the system).

In that purpose, [1] proposes an approach for automatic identification of linear relationships between measured signals and performs fault detection using principal component analysis (see section 2.1.3 for more details). Suppose all the signals have zero mean and unit variance. Applying PCA to the matrix of measurement amounts to a linear projection and enables to identify signals combinations bearing most of the variance. By looking at the eigenvalues of the correlation matrix, we can identify the number of signals to select in the transformed variables space to keep some percentage of the total variance. Applying PCA back-transformation to the selected signals, we should recover the original signals up to some errors introduced by the dimension reduction in the PCA variables space.

[1] proposes the following method. First, the PCA model is learned on non-faulty training data. Let us denote by X the measurement matrix (a row contains measurements of every signals at a given time). Then, obtaining the PCA model consists of computing the matrix V using the singular value decomposition $X = USV^T$. If V_P is the restriction of V to its first P columns, the first P PCA variables are given by XV_P and the back transformed variables by $XV_PV_P^T$. Once we have identified matrix V_P thanks to training set, fault detection is performed in two steps. For a new input point x

1. Compute the PCA variables xV_P and check that there are in the subspace covered by points under normal operating mode (if not, that means that the linear relationships that should hold between variables are violated).

2. Compute the back-transformed variables $xV_PV_P^T$ and check that the residuals are under control (if not, that means that the original point x was projected on a point in the low dimension space that does not contain enough information to allow accurate reconstruction, which should not occur under normal operating conditions if the model is correctly identified).

Notice that we do not have to care about extrapolation when using this approach. Indeed, if the point x does not fall into the subspace covered by the training data, either PCA variables will lead to fault detection or the residuals will be abnormal. In conclusion, this approach performs fault detection in two steps using a kind of linear bottleneck.

This idea can be used to extend this approach to nonlinear relationships. Principal components analysis will be replaced by an auto-encoder neural network with a bottleneck layer. The number of units in that bottleneck being the equivalent of the number of principal components P kept above (choosing the number of units will require looking at the curve of the RMSE as function of the number of units). The outputs of the network should be equal to the inputs. Fault detection will be performed in the exact same way as explained above (i.e. a check in the low dimension space along with monitoring the residuals).

In the sequel, we will use both approaches (linear in section 5.1 and nonlinear in section 5.2) in cascade to monitor linear and nonlinear relationships. Section 5.1 will help reduce the dimension of the input space of the auto-encoder of section 5.2. Notice however that we can not apply method of [1] because it is suppose the PCA variables are independent (or equivalently that the only existing relationships between the variables in the input space are linear) so that there is no more relations between the PCA variables (so, the data in the PCA variables space look completely random). In our case, there would remain relations between PCA variables

because of nonlinear relationships between input features. As we will see, we will be lucky enough to identify all the linear relationships using correlation only (that means that all the linear relationships are one-to-one). If it was not the case, we would have to use the PCA variables as input to the auto-encoder network of section 5.2 rather than the original signals (but we would loose the interpretability of the signals).

Eventually, notice that we will only consider points where the system is under stable operating mode. In other words, we will only keep points where the system has been running for a while so that we can neglect the dynamic. This will be done by requiring that the rotation speed on the last two hours is above a threshold. Doing so, we hope to suppress some of the dynamical effects and the influence of external weather conditions (ambient temperature) on most of the features. So, in the sequel, we will consider the system is static. That means that we will only be able to check for static relationships (if some dynamic relationship is violated, we won't be able to see it).

5.1 Linear dimension reduction using correlation and PCA

We are originally provided with a set of 34 signals measurements (see chapter 4 page 26 for data description). We will here identify linear relationships between those signals. Let us first identify linear relationships involving only two features using correlation. Table 5.1 summarizes the results of computing correlation between features. One can identify some highly correlated features. Taking advantage of this correlation, we can already drop 15 features (keeping one feature for every line of the table and dropping the other features of the table).

Let us use PCA on the provided data (using the 34 features) to check whether there exists linear relationships involving more than two signals that we could not identify with the correlation. Based on figure 5.1, we conclude that the first 19 principal components bear most of the variability of the process. There is a kind of elbow at 19 and the following singular values drop more or less to 0. We could discuss the exact number of principal components to drop (maybe someone will argue we could drop more than 19 principal components). However, this is not important here. Indeed, if we missed a linear relation, this relation will be identified by the auto-encoder in section 5.2 (remember that the goal here is mainly to reduce the dimension to make things easier in section 5.2).

It means that the dimension can be reduced to 19 features by taking advantage of linear relationships. Since this was already done using correlation, we conclude that no more features will be dropped.

Eventually, we should discuss the possibility of not detecting a fault due to the above dimension reduction. Indeed, a fault affecting only dropped features may stay undetected. For this reason, we should constantly check whether the linear relation between the dropped feature and the kept feature still holds. Figure 5.2 illustrates that check for the relation between the generator DE and NDE bearing temperatures. Figure 5.2b presents a healthy machine while on figure 5.2a there are many points out of the 99% confidence interval for regression. Finally, observe that

- either one of the two signals is faulty (or both signals are faulty and the linear relation does not hold anymore), in which case the fault is detected here by investigating the linear relation.
- or both signals are faulty so that the linear relation still holds (which may seem improbable), in which case the fault will be detected by investigating nonlinear relations in the next section.

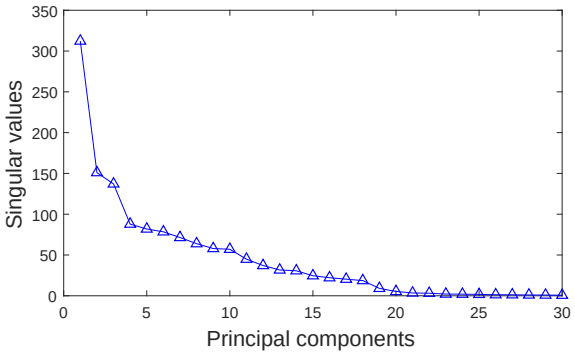
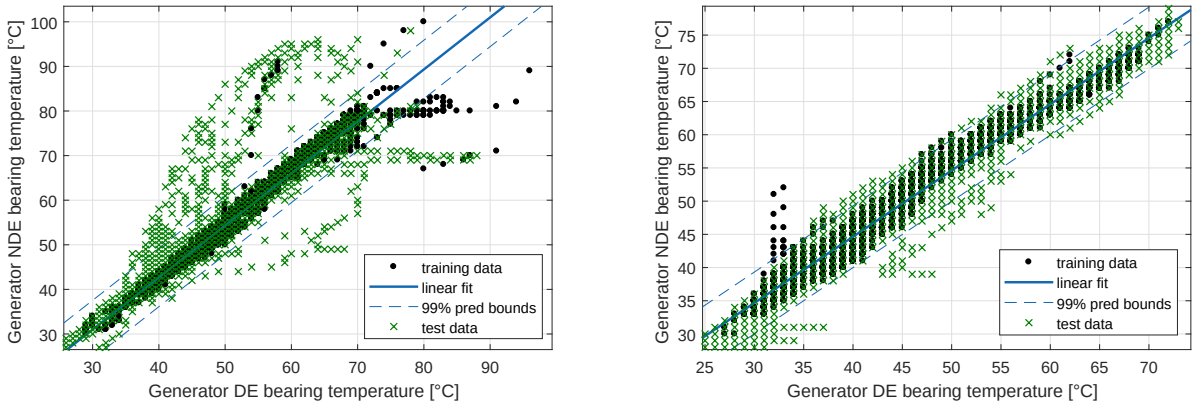


Figure 5.1: Singular values of the data matrix. We observe an elbow at 19 principal components. The 19 first principal components capture 99.96% of the variability of the data.

Features	correlation
Active power, phase current A, phase current B, phase current C	>0.997
Blade A pitch position, blade B pitch position, blade C pitch position, blade A set value, blade B set value, blade C set value	>0.999
Phase voltage A, phase voltage B, phase voltage C	>0.999
Hub speed, generator speed 1, generator speed 2	>0.999
Generator cooling air temperature, generator winding temperature A, generator winding temperature B	>0.997
Generator DE bearing temperature, generator NDE bearing temperature,	>0.970

Table 5.1: Highly correlated features. Notice that some other features are highly correlated too (correlation higher than 0.9) but we decided not to use this information to further reduce dimension here. We can keep only one feature per line of the table. So, 16 features can be dropped.



(a) (b)

Figure 5.2: Linear regression for predicting the generator NDE bearing temperature from the generator DE bearing temperature (machine E1RC on figure 5.2a and E2RC on figure 5.2b). 99% confidence intervals are shown. Many points are out of the confidence interval for machine E1RC.

5.2 Nonlinear dimension reduction using an auto-encoder

We are now interested in highlighting the nonlinear static relationships that may exist between the features. As explained in the introduction of the chapter, this will be done by mapping to a lower dimension space while still being able to reconstruct the original signals. Fault detection will be done in two steps.

1. Monitoring in the lower dimension space.
2. Monitoring the residuals after reconstruction of the signals.

Consequently, we need a method allowing to compute the back transformation to the original input space from the lower dimensional space. Moreover, it is require that this method has an internal model since it will be used to map test points that were not known at the training stage. Those two requirements strongly restrict the set of available methods.

An auto-encoder network satisfies both of those requirements. An auto-encoder is a network of which outputs should reproduce as accurately as possible the inputs and of which one layer is a bottleneck layer. The number of units of that layer is the number of input features minus the number of relations between those features. Figure 5.3 represents the used auto-encoder network. Let us take a closer look at this network.

5.2.1 Description of the auto-encoder

The used auto-encoder networks is here described into details. We will discuss the inputs, the methodology for fault detection, the number of hidden units and the identification of the process.

Input signals The dimension of the input space is 15. Although there were 19 features remaining after identifying the linear relationships in section 5.1, we decided to drop

- the nacelle position signal (which gives the orientation of the nacelle in degrees from 0 to 360) because it is not useful in fault detection (only the nacelle deviation, which is its orientation with respect to the wind direction, is useful) and is independent of the other signals. This signal is drop only to reduce the input space dimension and the computation time but could easily be added to the model (the dimension of the bottleneck layer should then be increased by 1).

- the ambient temperature, the nacelle temperature and the main bearing temperature features because those features are still strongly varying with the seasons. As we explained above, we can not afford learning on a full year. Since the training set can not contain data from every season, we will drop those features not to extrapolate. This means that we won't be able to detect increase in the main bearing temperature. For other temperatures signals, the fact that we only consider points were the wind turbine is under stable operating mode makes those temperatures features much less sensitive to ambient temperature. As we will see, we will be able to correctly reconstruct those features.

Notice that those four features were dropped based on physical knowledge. Consequently, we are not performing blind fault detection anymore. However, we did not had the choice in our particular case as just explained above. If we were provided with bigger data sets spanning over several years and could pay less attention to computation time, we could take those features into account and pursue with an input dimension 19 instead of 15.

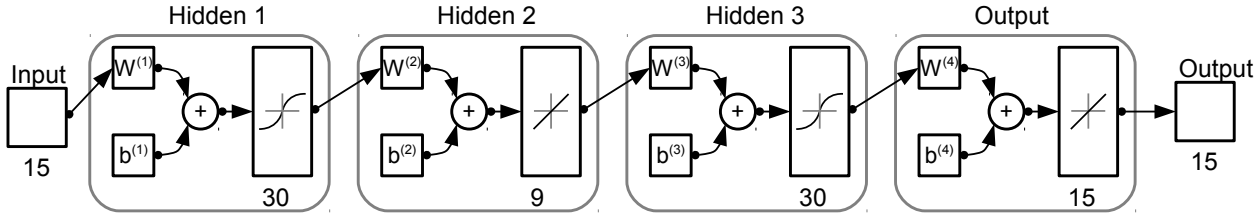


Figure 5.3: The auto-encoder network. The dimension of the input and output is 15. The bottleneck layer dimension is 9.

Number of units in the bottleneck layer Figure 5.4 is used to choose the number of hidden units in the bottleneck layer. If $u_1, u_2, \ldots, u_{d_1}$ are the input signals and $\hat{u}_1, \hat{u}_2, \ldots, \hat{u}_{d_1}$ the output signals, this figure shows

$$\frac{1}{d_1}\sum_{i=1}^{d_1}\frac{1}{U_i}\sqrt{\frac{1}{N}\sum_{t=1}^{N}(u_i(t)-\hat{u}_i(t))^2},$$

where U_i is the maximal absolute value of signal U_i on the training set and the test set is supposed to span from time 1 to time N. This is thus the scaled average RMSE on the test set. From figure 5.4, we conclude that 9 hidden units in the bottleneck layer is a consistent choice: a higher number of units does not decrease the error (error is approximately stable starting from 9 hidden units). In conclusion, we map a 15 dimensions space to a 9 dimensions space.

It means that our system is well represented by a set of 9 signals. We might expect the internal dimension of our system to be less than 9. For example, one may expect the wind signal is enough to predict all the other signals (and so the internal dimension would be one). However, we should remind that

- some signals (e.g. temperature signals) depend dynamically on the wind signal. Since we neglected the dynamic, the wind signal is not enough to predict such signals, which lead to an increase in the bottleneck layer dimension.

- some signals are not linked to the wind speed. The nacelle deviation (the orientation of the nacelle with respect to the wind direction), for example, is not linked to the wind speed. The wind speed is not sufficient either to predict the gearbox lube oil pressure. This is also the case for some electrical signals.

Fault detection As mentioned above, fault detection is performed in two steps. Let us explained it in a more detailed way. When feeding a faulty point to the auto-encoder network, there are two possibilities.

1. The corresponding point in the low dimension space is outside the subspace defined by the training points in the low dimension space. In that case, we do not have any indication on the performance of the reconstruction. That means this type of fault has to be detected in the low dimension space. Indeed, the reconstruction may be bad (in which case the fault will also be detected at step 2) but it may also be good

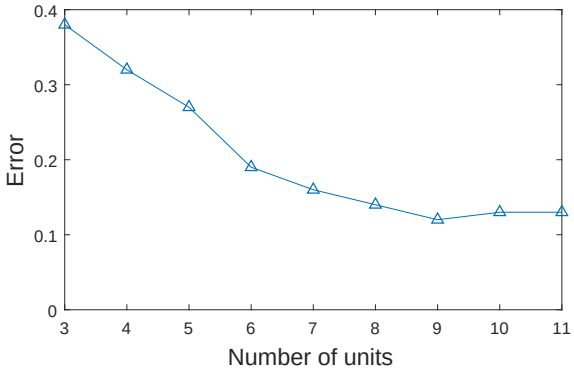


Figure 5.4: Cumulative reconstruction error as function of the number of units in the bottleneck layer. 9 units seems to be a consistent choice since the error does not decrease anymore for higher number of units.

(this is very improbable since the point is outside the training region but this could happen in theory). To estimate whether a point x of the low dimension space is inside the region covered by training data $x^{(1)}, x^{(2)}, \ldots, x^{(D)}$ in that space, we will use a multivariate kernel density estimation (this is actually an estimation of the probability density function in the low dimension space)

$$\frac{1}{D}\sum_{i=1}^{D}|H|^{-1/2}K_H\left(H^{-1/2}(x-x^{(i)})\right) = \frac{1}{D(2\pi)^{d_2}}\sum_{i=1}^{D}|H|^{-1/2}\exp\left[(x-x^{(i)})^T H^{-1}(x-x^{(i)})\right],$$

where H is a symmetric positive definite bandwidth matrix, the chosen kernel K_H is the Gaussian kernel and d_2 is the dimension of the lower dimension space. For the bandwidth matrix H, we will use Silverman's rule of thumb ([20])

$$\begin{aligned} H_{ii} &= \left(\frac{4}{(d_2+2)D}\right)^{1/(d_2+4)}\sigma_i, && 1 \leq i \leq d_2, \\ H_{ij} &= 0, && 1 \leq i,j \leq d_2, \end{aligned}$$

where σ_i is the standard deviation of the i-th variable.

Afterwards, we will choose a threshold $p \in \mathbb{R}_0^+$. Point x will be considered as a novelty if the density at point x is less than p. An example of threshold is the 0.01-quantile of the densities of the training set (i.e. the densities associated with $x^{(1)}, x^{(2)}, \ldots, x^{(D)}$).

We should now justify why we did not proceed to fault detection in the original input space using the technique mentioned above (so, estimating the probability density function in the input space). There are two reasons for that.

- The dimension has been lowered (by the end of the day, we can say the dimension has been lowered from 30 to 9), the computation time is thus reduced and the number of points with respect to the dimension has increased (which is generally a good thing in regression to avoid the so called 'empty space problem').
- The cloud of points seems to present a less complex structure in the low dimensional space (see figure 5.5). Indeed, the subspace created by the cloud of points in the input space may be relatively complex because of the nonlinear relationships between the features (see figure 5.6). In the low dimensional space, we end up with relatively independent features (notice however that nothing guarantees that).

2. The reconstruction is bad though no fault was detected in the low dimension space. The output point of the auto-encoder is far from the input point. So, in the residuals space, the residual will be detected as abnormally far from the origin. If no alarms were raised at step 1, this means that the input point is mapped to a point of the lower dimension space that does not normally correspond to the input point. The error is detected when mapping back to the original space. Notice that we have to proceed to multivariate process monitoring in the residuals space. Indeed, univariate process monitoring on each of those signals is not necessarily sufficient as explained in chapter 1. Furthermore, a fault on one particular signal may influence the reconstruction of other signals only. Indeed, although there is a warranty that reconstruction will be bad, there is no warranty on the signal(s) that will be badly reconstructed. Consequently, we will proceed to fault detection in the residuals space using MEWMA (see section 1.2.2.2).

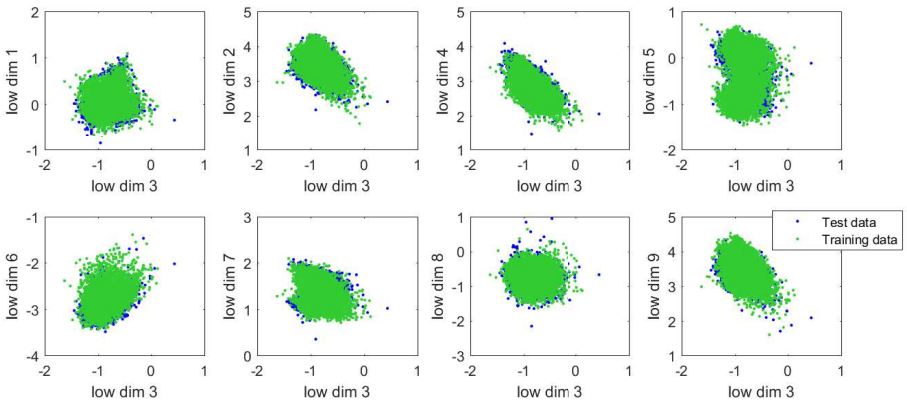


Figure 5.5: Example of scatter plots in the low dimension space focusing on dimension 3. No relationship is highlighted between the low dimension features using scatter plots, only clouds of data points appear. Test points (in blue) are plotted under the training points (in green) so that we can see whether new points are mapped to a region of the space where there are no training points.

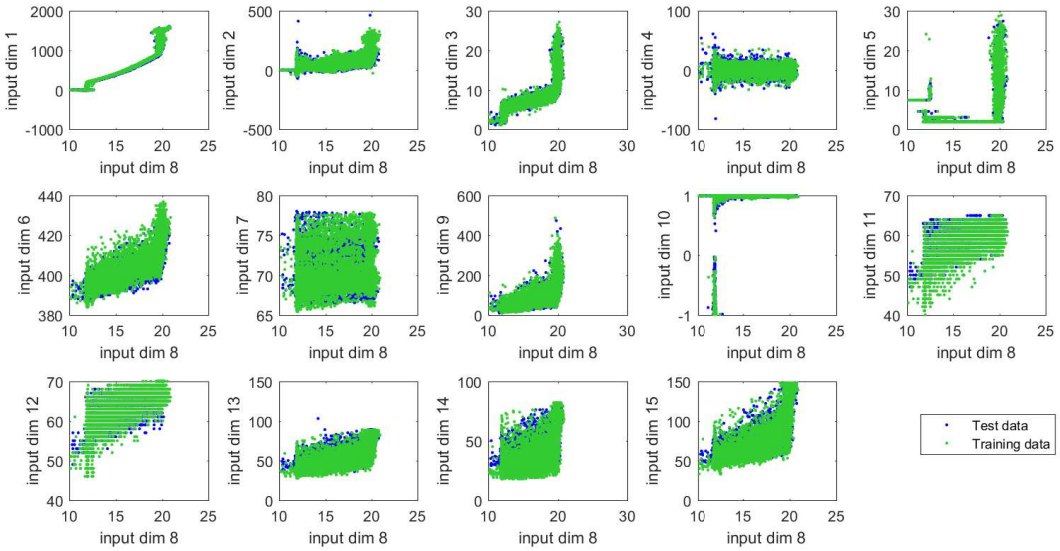


Figure 5.6: Example of scatter plots in the input space focusing on dimension 8. Relationships are highlighted between the input features using scatter plots. Test points (in blue) are plotted under the training points so that we can see whether there are test points outside the training set.

Identification step Let us explain how we proceeded to identify the parameters of the auto-encoder. We did not train an auto-encoder for each of the wind turbines. Indeed, we could have built an auto-encoder for every wind turbine. However, it turns out that this method performs rather bad because exploitable data of the training set of a given machine do not necessarily represent all possible operating states susceptible to occur later (in other words, the training set is not representative enough). As a consequence, the model will perform bad for some non-faulty but never previously observed points because we have no warranty on the validity of the model outside the region covered by the training set. Therefore, we will rather build a generic model applicable to any of the wind turbines. The training set will be made of the data of several wind turbines. The main advantage of this approach is that the training set contains now more various operation states. The main drawback is an increase in the RMSE when predicting. Indeed, though the prediction error has strongly decreased for some points (i.e. points where the previous model was extrapolating), it has slightly increased for most of the points. This seems consistent since this general model as to hold for all wind turbines, though they may have different behaviors (i.e. different internal parameters). So, we are now able to provide a prediction for a larger set of points but the quality of the prediction has decreased. A second drawback of this method is that we have to ensure that a larger training set contains no faults.

5.2.2 Application

In this section, we will apply the method introduced above to the wind turbines E•F on the one hand and E•RY on the other hand. We consider wind turbines E•F together because the time periods for which data are exploitable are identical for all wind turbines with this tag (and similarly for wind turbines with tags E•RY).

Matlab codes for this section can be found in appendix B.4 page 66. Before presenting the results, we should explain a major difficulty that was encountered when trying to identify an efficient model using the data. For every wind turbine, there are time periods where the data collection was interrupted. It often happens that a model that was efficient at prediction before such an interruption will lead to an automatic bias after the interruption (maybe due to some component being changed or to sensors calibration). Consequently, we tried to focus on time periods that do not contain such interruptions so that we do not have alarms due to the apparition of an automatic bias. In practice, we should learn the model again after an interruption took place (the former model could still be used until we have enough data to identify the new model but it will be less efficient for detecting faults).

Wind turbines E•F Let us investigate how the model introduced in the previous section performs for detecting faults on wind turbines E•F. Those wind turbines are considered together because the time periods for which exploitable data are available are identical. Our training set will spans from 01/01/2015 to 1/04/2015 and will be composed of data from wind turbines E1F, E2F, E4F and E5F (E3F is excluded because a fault occurs very early in 2015 - see table 4.3). Test set will span from 01/04/2015 to 01/07/2015. So, we have three months of training data and three months of test data. Table 5.2 presents the results for that time period. Observe that multivariate statistical analysis often detects faults earlier than univariate statistical analysis (except for one fault).

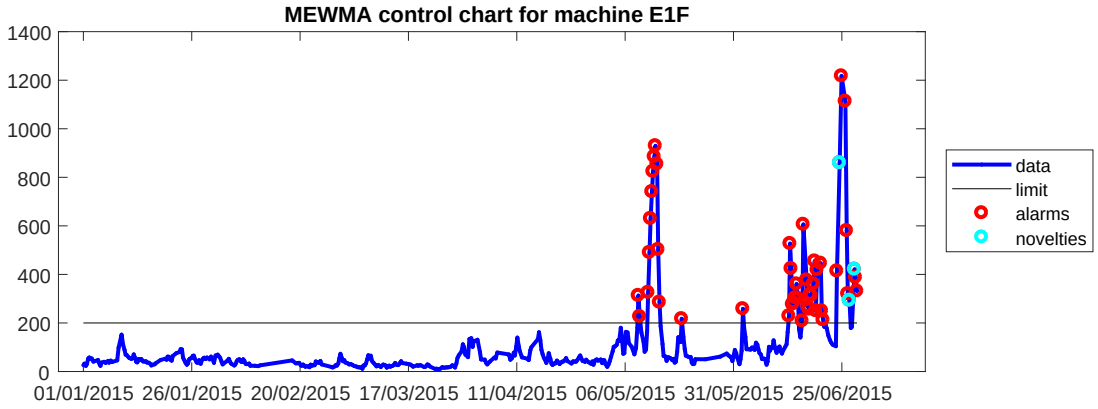
It is interesting to compare this table with table 4.3 where model-based fault detection was used. As expected, we can observe the fault on machine E5F is detected later (9 days later) than on table 4.3, which is consistent since our model is generic (valid for several wind turbines) and we did not use physical knowledge and dynamic. Faults on machines E1F and E3F are detected on the same day as on table 4.3. Notice that the alarms due to high gearbox lube oil pressure on machine E5F are probably not a fault. Those are simply due to some high pressure values that were never experienced in the training set. In practice, once we observe this was not a fault (i.e. once things are normal again), we could add those points to the training set so that it becomes more representative.

Eventually, figure 5.7 represents the monitoring of the residuals using multivariate statistical analysis (remember that the residuals are in a space of dimension 15) for faulty wind turbine E1F and healthy wind turbines E2F and E4F.

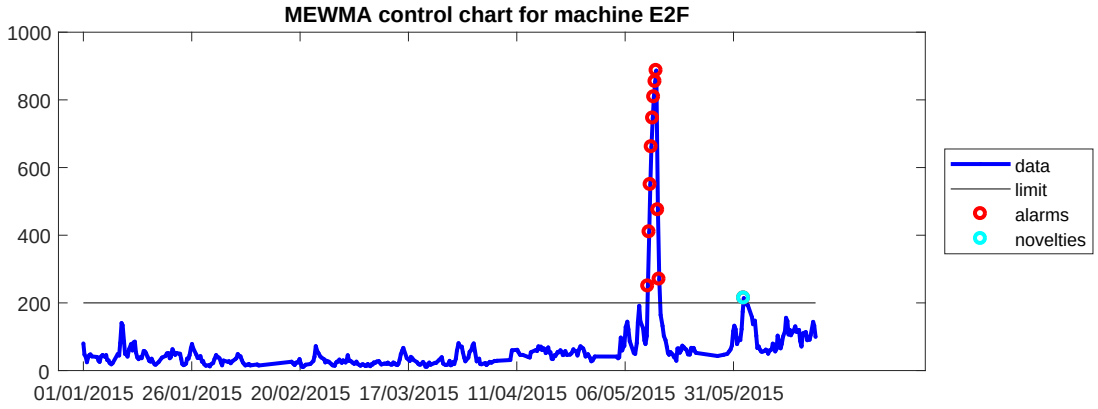
Machine	Average scaled RMSE	Recurrent alarms univariate XBAR	Recurrent alarms univariate EWMA	Recurrent alarms MEWMA	Cause
E1F	0.12	14/06/2015 22:08	14/06/2015 12:18	13/06/2015 20:38	Gear. bear. temp.
E2F	0.12	/	/	/	/
E3F	0.38	29/04/2015 14:32	17/04/2015 10:20	01/04/2015 06:30	Gear. bear. temp.
E4F	0.12	/	/	/	/
E5F	0.13	19/04/2015 19:01 /	19/04/2015 19:01 22/06/2015 15:37	20/04/2015 19:31 20/06/2015 23:31	Gear. bear. press. Gear. bear. temp.

Table 5.2: Recurrent alarms for machines E•F. Notice that RMSE is high for machine E3F because the fault occurs early in the test set (RMSE from 01/01/2015 to 01/04/2015 is small: 0.14), which is not the case for the other machines. Faults are detected no earlier than in the previous chapter using model-based fault detection (see table 4.3 page 32).

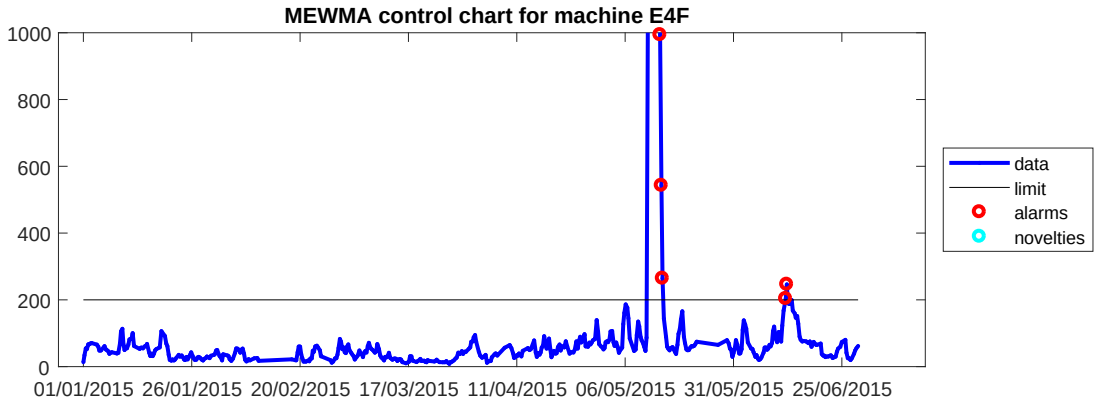
Wind turbines E•RY For wind turbines E•RY, we learned a generic model on a training set made of data from wind turbines E2RY, E4RY and E6RY from 01/01/2016 to 01/07/2016. The test set spans from 01/07/2016 to 01/03/2017. Results are shown in table 5.3. We only have recurrent alarms for wind turbine E2RY from 29/11/2016 to 11/12/2016 because the gearbox lube oil pressure is higher than it has ever been before. However, this does no seem to be a fault since the pressure drops again after some time. So, our training set may miss some points with high gearbox lube oil pressure.



(a)



(b)



(c)

Figure 5.7: MEWMA control charts for the residuals. Parameters values are 0.3 for λ and 200 for the upper limit. The alarms around 12/05/2015 are due to some fault in the collected data. Indeed, lots of sensors are stuck for a while on the same measurement value. Consequently, several successive residuals are identical which lead to the detection of an automatic bias. Alarms for machine E1F are due to an increase in the high speed gearbox bearing temperature (see section 4.1).

Machine	Average scaled RMSE	Recurrent alarms univariate XBAR	Recurrent alarms univariate EWMA	Recurrent alarms MEWMA	Cause
E0RY	0.13	/	/	/	/
E1RY	0.15	/	/	/	/
E2RY	0.15	29/11/2016 20:04	01/12/2016 08:34	27/11/2016 03:23	Gear. bear. press.
E3RY	0.16	/	/	/	/
E4RY	0.14	/	/	/	/
E5RY	Not enough non-faulty data for estimation of S and μ				
E6RY	0.17	/	/	/	/

Table 5.3: Recurrent alarms for machines E•RY. For machine E2RY, there are alarms because of a never previously experienced high pressure from 27/11/2015 to 11/12/2015 (we guess this is not a fault since there are no more alarms beyond that day). For wind turbine E5RY, the fault occurs too early (see table 4.3) and we do not have enough non-faulty data to accurately estimate statistical properties of the residuals.

5.3 Conclusion

In this chapter, we tried to build a generic method for fault detection that does not require any knowledge about the monitored process. We decided to restrict to static systems. We performed fault detection on a static process in two stage. We first identified faults by investigating linear relationships between features. Simultaneously, we reduced the dimension of the input space. We then proceeded to fault detection by investigating the violation of nonlinear static relationships between features using an auto-encoder.

This approach turns out to effectively detect faults. However, when comparing results with those of the previous chapter, we saw that this method detect faults later than a more specific model-based approach.

Finally, we saw that this method does not provide any information about the origin of the fault. This is consistent because, without any knowledge on the process, we can not say which signal (resp. signals) is (resp. are) abnormal with respect to the others.

Conclusion

As explained in the introduction, this dissertation had two main goals. The first one was to carry out a literature review of some popular approaches to fault detection. The second one was to perform fault detection on data collected in wind turbines.

Chapter 1 presented an overview of methods encountered in literature for monitoring signals by univariate and multivariate statistical analysis. We came to the conclusion that all methods are not equivalent (so, one of them may lead to better results) and that multivariate analysis outrivals univariate analysis. Notice that most of those methods require having some hypothesis on the analyzed signals. Often, it is required that the signal is stationary and is a sequence of Gaussian random variables. We get around that strong hypothesis in chapter 4 and 5 by analyzing the mean of the residual signal on successive time windows and using the central limit theorem.

In chapter 2, we introduced model-based fault detection. Model-based fault detection can be summarized in two step. First, a model representing the process is built using data collected under normal operating conditions. Then, at the monitoring stage, the model is used to compute an estimation for the output of the process. Faults are detected by comparing the estimation with the measurement of the output signal. In that purpose, methods introduced in chapter 1 are useful. Two of the methods introduced in chapter 2 will be fully or partially used in chapters 4 and 5 on the wind turbines data.

Chapter 3 was dedicated to an overview of the commercial software performing fault detection. For two of them we got exploitable information and could refer to some methods of chapter 2. For those software, we could easily get convinced their use can lead to some savings in terms of maintenance planning and unexpected down-times and breakdowns (those savings are still hard to estimate since they hardly depends on the application). For most of the other software, access to detailed explanations about the main principles of the software was denied. Therefore, we could not reach the original goal of having a fair comparison of those software. As mentioned in chapter 3

In chapter 4, we performed model-based fault detection on data collected in wind turbines. Therefore, we used artificial neural networks to train a model based on past non-faulty data. In section 4.1.2, we build an input-output model and an auto-regressive model. It turns out that, though the error was much smaller for the auto-regressive model, both models detected faults approximately at the same time. This is because both models were accurate enough to capture the change in the residuals distribution. In practice, we would prefer the auto-regressive model because a smaller dimension of the neural net input space is required.

In chapter 4, because of the lack of data, we also investigated the influence of progressively increasing the learning set with new time periods once we can guarantee no fault was developing during those periods. Thanks to this approach, we could efficiently predict the high speed gearbox bearing temperature and detect an increase in this signal up to one month earlier than with straightforward statistical analysis (see table 6.1). We also observed that the drawback of starting with a small learning set and progressively increasing it is not that the model won't be accurate for the first few predictions (that would indicate that the model was badly identified). Instead, it is that the model can sometimes not be used because the input point is not in the region of the input space covered by the training points. In our case, this was due to the variation of the ambient temperature through the year. So, either we can not give any prediction, or we can give an accurate prediction (the larger the initial training set, the more points we can predict). In short, the size of the initial training set was someway imposed so that it contains enough different ambient temperatures allowing us to predict enough points during the incoming months.

Another advantage of this progressive increase in the learning set is that it allows to take into account the aging of the machinery. Indeed, there are industrial systems of which behavior is modified as they get older. Constantly adding recent points to the training set helps maintaining an accurate model. The drawback is that every time we added points to the training set, the model had to be learned again. Furthermore, we could imagine dropping old points when new points are added to the training set (this was not done in our case because the training set size remained acceptable).

In section 4.1, we came to the conclusion that model-based fault detection (chapter 2) allows to detect an increase in the high speed gearbox bearing temperature up to one month earlier than straightforward statistical analysis (chapter 1). Results are summarized in table 6.1. A one month gain in fault detection seems interesting

with respect to the time between the fault detection by statistical analysis and the date of correction which is 2 and 3 months later for wind turbines E3F and E5F. However, for wind turbines E1F and E5RY, a one month gain in fault detection may not be crucial because they still ran for at least 6 months until repairs were done. Notice that the operator of the wind turbines could not provide us with some file reporting the occurred faults and the detection date by the tool that is currently used for monitoring the turbines. So, we can not say whether the model developed in chapter 4 performs better or not.

In chapter 5, we tried to build a method for detecting faults without having any understanding about the process (i.e. we will make some hypothesis on the process but we will not require having knowledge about how the measured signals may be related). Of course, we were not able to build an idealistic method performing fault detection on any processes without further assumptions. We will make the hypothesis that the process is static (as explained below, this hypothesis could actually be replaced by a less restrictive hypothesis stipulating that the dynamic order of the process is less than a known positive integer n). Therefore, we identified an auto-encoder model for fault detection in a static process. The auto-encoder contained a bottleneck layer of small dimension which represents the number of features necessary to accurately reconstruct the original point. This amounts to finding the static relationships between the input features. Notice that this method only allows to check that static relationships are verified (so we can not detect a fault leading to the violation of a dynamic relationship). Taking dynamic relationships into account could be done by using external dynamic (see page 14) and increasing the input space dimension. However, this require having an idea of how many previous inputs should be taken into account and will consequently increase the dimension of the input space and slow the computations.

The basic idea of the auto-encoder of chapter 5 is that a faulty point will lead to either an inconsistent point in the small dimension space or a bad reconstruction of the original point. The second case appears to be the most common one. When the first case occurs, we are not able to say whether the point is a new non-faulty point or a faulty point. For fault detection in the low dimension space, we estimated the probability density function with kernel density estimation. In the residuals space, we used methods for multivariate statistical analysis introduced in chapter 1.

The advantage of the method used in chapter 5 is that it does not require having a deep understanding of the process. Indeed, in chapter 4, choosing input features that may help to predict the output features was often based on the understanding of the process (notice that exploring all the possible combinations of features was unfeasible in practice due to the number of features).

Last column of table 6.1 gives the detection date for the increase in the high speed gearbox bearing temperature. The table allows to compare the performance of the method used in chapter 5 with the method of chapter 4. We can see that faults are detected later or not significantly earlier with methods of chapter 5. For machine E5F, the fault is even detected 9 days later with methods of chapter 5. We expected the method of chapter 5 to perform worse than the method of chapter 4 since they did not took advantage of the knowledge on the system and on the dynamic. As shown in table 6.1, it turns out to be the case.

Finally, notice that auto-encoders built in chapter 5 are tuned using data from several wind turbines. So, there are generic auto-encoders applicable to several wind turbines. The advantage of doing so is that we increase the training set and are able to correctly predict a larger variety of points. Nevertheless, the global error increases slightly because the individual characteristics specific to a given wind turbines are not modeled by a generic model.

In conclusion, through this dissertation, we tried to give an overview of some popular methods for fault detection in chapters 1, 2 and 3 while chapters 4 and 5 were dedicated to the implementation of fault detection tools to get an idea of the efficiency of fault detection methods. Some further work may be dedicated to evaluating the financial earnings that may be generated by capitalizing on the time gain in fault detection to evaluate the usefulness of investing in fault detection tools.

Machine	Date of correction	First alarm		
		Statistical analysis chapter 4	Model-based FD chapter 4	Blind FD chapter 5
E1F	03/06/2016 13:59	12/07/2015 00:12	13/06/2015 20:38	13/06/2015 20:38
E3F	29/06/2015 14:03	03/05/2015 23:43	01/04/2015 10:40	01/04/2015 06:30
E5F	12/10/2015 16:56	11/07/2015 18:32	11/06/2015 02:18	20/06/2015 23:31
E5RY	13/10/2016 13:31	30/05/2016 17:28	01/03/2016 16:42	/

Table 6.1: Alarms for an increase in the high speed gearbox bearing temperature (results from chapters 4 and 5). The first alarm columns give the date from which recurrent alarms are observed. The date of correction is the last day we observe the wind turbine is faulty in the data. This does not mean this is the day the fault was effectively detected by the operator (surely it was detected earlier but the operator deliberately decided to let it run until repair were performed). We did not have enough data before the fault to adjust the Blind FD model for machine E5RY.

Bibliography

[1] D.G. Alvarez, *Fault detection using Principal Component Analysis (PCA) in a Wastewater Treatment Plant (WWTP).*

[2] P. Amyotte, R. Ferdous, F. Khan, R. Sadiq, B. Veitch, *Fault and event Tree Analysis for Process Systems Risk Analysis: Uncertainty Handling Formulation*, Risk Analysis, Vol 31, No 1, 2011.

[3] A.D. Back, A.C. Tsoi, *FIR and IIR Synapses, a New Neural Network Architecture for Time Series Modeling*, 1991.

[4] P. Baraldi, F. Di Maio, P. Turati, E. Zio, *Robust signal reconstruction for condition monitoring of industrial components via a modified Auto Associative Kernel Regression method*, Mechanical Systems and Signal Processing, Elsevier, 2015, 60-61, pp.29-44.

[5] J.C. Bezdek, R. Ehrlich, W. Full, *The fuzzy c-means clustering algorithm*, Computer & Geosciences, Vol. 10, No. 2-3, pp. 191-203, 1984.

[6] Y. Chetouani, *A sequential probability ratio test (SPRT) to detect changes and process safety monitoring*, Université de Rouen, 2013.

[7] F.N. Chowdhury, A. Fekih, H. Xu, *Neural Networks Based System Identification Techniques for Model Based Fault Detection of Nonlinear Systems*, Department of Electrical and Computer Engineering, University of Louisiana at Lafayette, 2005.

[8] E. Davis, J.W. Hines, *Lessons Learned From the U.S. Nuclear Power Plant On-Line Monitoring Programs.*

[9] G. MT. D'Eleuteriob, A.P. Trischlera, *Synthesis of recurrent neural networks for dynamical system simulation*, University of Toronto, Institute for Aerospace Studies, 2010.

[10] I. Diaz, J. Hollmen, *Residual Generation and Visualization for Understanding Novel Process Conditions.*

[11] W. Duch, *Similarity-based methods: a general framework for classification, approximation and association*, Department of Computer Methods, Nicholas Copernicus University, 2000.

[12] E. Eskinat, S. H. Johnson, W.L. Luyben, *Use of Hammerstein Models in Identification of Nonlinear Systems*, Lehigh University, 1991.

[13] Wm.J. Garland, K.P. Leger, W.F.S. Poehlman, *Fault Detection and Diagnosis using Statistical Control Charts and Artificial Neural Network*, McMaster University.

[14] GE Power Digital Solutions, *GE Digital Twin Analytic Engine for the Digital Power Plant*, 2016.

[15] GE Power Digital Solutions, *The Value of GE SmartSignal powered by Predix Solutions to Combined-Cycle Plants.*

[16] J.P. Herzog, *Monitoring method using kernel regression modeling with pattern sequences*, US Patent 8620853, SmartSignal Corporation, 2013.

[17] J. House, W.Y. Lee, D.R. Shin, *Classification techniques for Fault Detection and Diagnosis of an air-handling unit*, ASHRAE Transactions, 1999.

[18] R. Isermann, *Fault-Diagnosis Systems, An Introduction from Fault Detection to Fault Tolerance*, Springer, 2006.

[19] J.M. Jobe, S.B. Vardeman, *Statistical Methods for Quality Assurance: Basics, Measurement, Control, Capability, and Improvement*, 2nd Ed., Springer, 2016.

[20] M.C. Jones, J.S. Marron, S.J. Sheath, *A Brief Survey of Bandwidth Selection for Density Estimation*, Journal of the American Statistical Association, March 1996, Vol. 91, No. 433.

[21] K. Jyoti, S. Singh, *Data Clustering Approach to Industrial Process Monitoring, Fault Detection and Isolation*, International Journal of Computer Applications, 2011.

[22] S.N. Kavuri, R. Rengaswamy, V. Venkatasubramanian, K. Yin, *A review of process fault detection and diagnosis, Part I: Quantitative model-based methods*, Elsevier, 2002.

[23] S.N. Kavuri, R. Rengaswamy, V. Venkatasubramanian, *A review of process fault detection and diagnosis, Part II: Qualitative models and search strategies*, Elsevier, 2002.

[24] S.N. Kavuri, R. Rengaswamy, V. Venkatasubramanian, K. Yin, *A review of process fault detection and diagnosis, Part III: Process history based methods*, Elsevier, 2002.

[25] X.R. Li, L. Wang, *Fault detection using sequential probability ratio test*, Power Engineering Society 1999 Winter Meeting, 1999.

[26] Y. Li, B. Lu, X. Wu, Z. Yang, *A Review of Recent Advances in Wind Turbine Condition Monitoring and Fault Diagnosis*, 2009.

[27] K.A. Loparo, Y. Maki, *A Neural-Network Approach to Fault Detection and Diagnosis in Industrial Processes*, 1997.

[28] R. Marcell, C. Stoecker, S.W. Wegerich, *Fuzzy classification approach to fault pattern matching*, US Patent 7941701, SmartSignal Corporation, 2011.

[29] J. Mina, C. Verde, *Fault Detection Using Dynamic Principal Component Analysis by Average Estimation*, 2005.

[30] D.C. Montgomery, *Introduction to Statistical Quality Control*, 6th Ed., John Wiley & Sons Inc., 2009.

[31] J. Neyman, E.S. Pearson, *On the Problem of the Most Efficient Tests of Statistical Hypotheses*, Philosophical Transactions of the Royal Society of London, 1933.

[32] M.E. Orchard, R. Paredes, F.A. Tobar, L. Yacher, *Anomaly Detection in Power Generation Plants using Similarity-based Modeling and Multivariate Analysis*, American Control Conference, 2011.

[33] K. Patan, *Artificial Neural Networks for the Modelling and Fault Diagnosis of Technical Processes*, Springer, 2008

[34] R.M. Pipke, S.W. Wegerich, *Residual signal alert generation for condition monitoring using approximated SPRT distribution*, US Patent 6975962, SmartSignal Corporation, 2005.

[35] M. Schlechtingen and I.F. Santos, *Comparative analysis of neural network and regression based condition monitoring approaches for wind turbine fault detection*, Department of Mechanical Engineering, Technical University of Denmark, 2010.

[36] U. Steinmetz, H. Spliessgardt, *Data Evaluation to Detect Faults and Problems in Power Plant Operation*, STEAG Energy Services GmbH, 2014.

[37] F. Wagner, *Increasing wind farm profitability through advanced SCADA data analysis enhanced by engineering RCA*, STEAG Energy Services GmbH, 2015.

[38] A. Wald, *Sequential test of statistical hypothesis*, 1945.

[39] S.W. Wegerich, X. Xu, *System state monitoring using recurrent local learning machine*, US Patent 7403869, SmartSignal Corporation, 2008.

[40] Y. Zhang, *An artificial Neural Network Approach to Transformer Fault Diagnosis*, Virginia Polytechnic Institute, 1996.

[41] F. Zheng and S. Zhong, *Time series forecasting using a hybrid RBF neural network and AR model based on binomial smoothing*, 2011.

Appendix A

Data description

In this appendix, we will give a brief explanation on the content of the data from the wind turbines used in chapter 4 and 5.

As mentioned in chapter 4, we focused on 34 signals. Here are those signals with their meaning.

- **Wind/orientation**:
  - wind speed [m/s]
  - nacelle deviation [°]: orientation of the nacelle with respect to the wind direction
  - nacelle position [°]: orientation of the nacelle
- **Hub**:
  - hub speed [rpm]: rotation speed of the hub
  - main bearing temperature[°C]: temperature of the main bearing
  - blade A/B/C pitch position [°]: position of the blade A/B/C from 0°to 90°
  - blade A/B/C set value [°]: setpoint value for the position of the blade A/B/C from 0°to 90°
- **Gearbox**:
  - gearbox lube oil pressure [hPa]: pressure of the gearbox lubricant oil
  - gearbox lube oil temperature[°C]: temperature of the gearbox lube oil
  - high speed gearbox bearing temperature[°C]: temperature of the high speed shaft gearbox bearing
  - gearbox axial global vibration: measure of the vibrations
- **Generator**:
  - generator speed 1 and 2 [rpm]: 2 measurements of the high speed shaft rotation speed
  - generator DE bearing temperature [°C]: temperature of the high speed shaft generator driven-end bearing temperature
  - generator NDE bearing temperature [°C]: temperature of the high speed shaft generator non-driven-end bearing temperature
  - generator winding temperature A and B [°C]: 2 measurements of the generator winding temperature
  - generator cooling air temperature [°C]: temperature of the cooling air for the generator winding
- **Transformer**: transformer temperature [°C],
- **Output power**:
  - active power [MW]: output active power $P = |S| \cos \phi$ with $S = VI^*$ the complex power and ϕ the phase of voltage relative to current
  - reactive power [Mvar]: output reactive power $S = |S| \sin \phi$ with $S = VI^*$ the complex power and ϕ the phase of voltage relative to current
  - power factor: ratio of the active power (P) to the apparent power ($|S|$) $\cos \phi = P/|S|$
- **Output voltage**: phase voltage A/B/C [°]
- **Output current**: phase current A/B/C [°]
- **Other**: ambient temperature [°C], nacelle temperature [°C]

Appendix B

Matlab codes

This appendix contains the principal Matlab programs used for chapters 4 and 5 along with some comments.

B.1 Codes of section 4.1.1

This code is used to proceed to univariate fault detection and generate figure 4.2.

Univariate_FD

```
load('E5F.mat') % change the data (choose between E.F and E.RY, e.g. E0RY and E4F for non-faulty turbines, E5F or E1F for faulty machines)
year = 2015; % use 2015 for data E.F and 2016 for data E.RY

Y = data.sig(:,43);

abs_all = SelectDataNotNan(Y,1,1e6,data,0,3,1000,-1000);
abs_all = abs_all(((data.timevec(abs_all)>=datenum(year,1,1)).*(data.timevec(abs_all)<=datenum(year+1,1,1)))>0);
Y = Y(abs_all);
TimePred = data.timevec(abs_all);

%% 3.3 XBAR control chart
colordata = [0.2 0.8 0.2];
colortest = 'b';
ntr = 12000; % number of points for estimating mu and S
n = 40; % number of measurements in every sample of the controlchart

residuals_train = Y(1:ntr);
residuals_test = Y(ntr+1:end);
abs_train = TimePred(1:ntr);
abs_test = TimePred(ntr+1:end);
% Select data
L = n*floor(size(residuals_train,1)/n);
ctl_cht_train = residuals_train(1:L,:);
P = n*floor(size(residuals_test,1)/n);
ctl_cht_test = residuals_test(1:P,:);
% Select time abscissae
ctl_cht_train_matrix_time = reshape((abs_train(1:L)),n,L/n)';
ctl_cht_train_matrix_time = ctl_cht_train_matrix_time(:,end);
ctl_cht_test_matrix_time = reshape((abs_test(1:P)),n,P/n)';
ctl_cht_test_matrix_time = ctl_cht_test_matrix_time(:,end);

figure
ctl_cht_train_matrix = reshape(ctl_cht_train,n,L/n)';
ctl_cht_test_matrix = reshape(ctl_cht_test,n,P/n)';
mu = mean(ctl_cht_train);
sigma = std(ctl_cht_train);
false_alarm_rate = 0.01; % if we want to set upper and lower limits to a quantile
ULm = quantile(mean(ctl_cht_train_matrix,2),1-false_alarm_rate/2);
LLm = quantile(mean(ctl_cht_train_matrix,2),false_alarm_rate/2);

XBARControlChart(ctl_cht_test_matrix,ctl_cht_train_matrix,[LLm mu ULm],ctl_cht_test_matrix_time,ctl_cht_train_matrix_time,colortest,colordata);
title('XBAR␣control␣chart␣for␣the␣high␣speed␣gearbox␣bearing␣temperature');
ylabel('\fontsize{14}[C]')
```

B.2 Codes of section 4.1.2

Here are the Matlab codes for section 4.1.2 detecting increases in the high speed gearbox bearing temperature. The following script can be run on machine E•F and E•RY to get results of section 4.1.2. It will simulate the monitoring of a wind turbine on a whole year.

Model_Script

```
% Script for monitoring of a wind turbine with 2-months ahead predictions

%% 1. Initialization
% Training and test machines (RC and V: not enough data):
head_train = {'E2F.mat'}; % training data
head_test = 'E2F.mat'; % test data

% Initialize dates
year = 2015; % starting year (2015 for E.F, 2016 for E.RY)
end_train_month = 3; % last month for initial training set starting from 01/01/year
step = 2; % number of healthy months before adding a new month to training data
end_simu = 11; % number of months to simulate starting from 01/01/year
model = 2; % model number (1 or 2)

end_train = datenum(year,end_train_month+1,1,0,0,0); % end of training day
start_month = end_train_month+1; % starting month for the online monitoring
end_test = datenum(year,start_month+step,1,0,0,0); % initialization for the first day we should consider extending the training set
% Initialization:
SignPred = []; % will contain the step-month predictions
TimePred = []; % will contain time indexes of the step-month predictions
ResiPred = []; % will contain residuals of the step-month predictions
AbsiPred = []; % will contain abscissae for residuas and predictions

load(head_test); % load test machine

%% 2. Simulation
for m = start_month:end_simu
    fprintf('Start MLP with end of training: 28-29-30-31/%d/%d \n',m-1,year);
    if m==start_month
        [Predi,Resid,TimeP,AbsiP] = Model_Fun(head_train,head_test,end_train,end_test,model); % compute predictions
        abs_m = TimeP<=end_test; % select data until m+step-1 month
    else
        [Predi,Resid,TimeP,AbsiP] = Model_Fun(head_train,head_test,end_train,end_test,model); % compute predictions
        abs_m = ((TimeP>datenum(year,m+step-1,1,0,0,0)).*(TimeP<=datenum(year,m+step,1,0,0,0)))>0; % select data on m+step-1 month
    end
    SignPred = [SignPred; Predi(abs_m)];
    ResiPred = [ResiPred; Resid(abs_m)];
    TimePred = [TimePred; TimeP(abs_m)];
    AbsiPred = [AbsiPred; AbsiP(abs_m)];
    end_train = datenum(year,m+1,1,0,0,0); % add a training month
    end_test = datenum(year,m+1+step,1,0,0,0); % add a test month
end
RMSE = std(ResiPred); % compute RMSE

%% 3. Visualization
% 3.1 Signal and predictions
figure
plot(data.timevec(AbsiPred),data.sig(AbsiPred,43),'linewidth',1); hold on;
plot(TimePred,SignPred,'linewidth',1)
xlim([datenum(year,1,1,0,0,0) TimePred(end)]);
set(gca,'XTick',datenum(year,1,1,0,0,0):70:TimePred(end));
datetick('x','dd/mm/yyyy','keeplimits','keepticks');
title('High speed gearbox bearing temperature');
ylabel('\fontsize{14}[C]')
legend('Measurements','Predictions','Location','EastOutside')
% 3.2 Residuals
figure
plot(TimePred,ResiPred,'linewidth',1);
datetick('x','dd/mm/yyyy','keeplimits','keepticks');
ylabel('Residuals');

% 3.3 XBAR control chart
colordata = [0.2 0.8 0.2];
colortest = 'b';
ntr = 6000; % number of points for estimating mu and S
n = 40; % number of measurements in every sample of the controlchart

residuals_train = ResiPred(1:ntr);
residuals_test = ResiPred(ntr+1:end);
abs_train = TimePred(1:ntr);
abs_test = TimePred(ntr+1:end);
% Select data
L = n*floor(size(residuals_train,1)/n);
ctl_cht_train = residuals_train(1:L,:);
P = n*floor(size(residuals_test,1)/n);
ctl_cht_test = residuals_test(1:P,:);
% Select time abscissae
ctl_cht_train_matrix_time = reshape((abs_train(1:L)),n,L/n)';
ctl_cht_train_matrix_time = ctl_cht_train_matrix_time(:,end);
ctl_cht_test_matrix_time = reshape((abs_test(1:P)),n,P/n)';
ctl_cht_test_matrix_time = ctl_cht_test_matrix_time(:,end);

figure
ctl_cht_train_matrix = reshape(ctl_cht_train,n,L/n)';
ctl_cht_test_matrix = reshape(ctl_cht_test,n,P/n)';
```

```
mu = mean(ctl_cht_train);
sigma = std(ctl_cht_train);
% false_alarm_rate = 0.01; % if we want to set upper and lower limits to a quantile
% ULm = quantile(mean(ctl_cht_train_matrix,2),1-false_alarm_rate/2);
% LLm = quantile(mean(ctl_cht_train_matrix,2),false_alarm_rate/2);
ULm = mu+3*std(mean(ctl_cht_train_matrix,2));
LLm = mu-3*std(mean(ctl_cht_train_matrix,2));
XBARControlChart(ctl_cht_test_matrix,ctl_cht_train_matrix,[LLm mu ULm],ctl_cht_test_matrix_time,
    ctl_cht_train_matrix_time,colortest,colordata);
title('XBAR control chart for the residuals of the high speed gearbox bearing temperature');
ylabel('\fontsize{14}[C]')
legend('data','lower limit','upper limit','center','alarms','Location','EastOutside')

% 3.4 EWMA control chart
figure
EWMAControlChart(ctl_cht_test_matrix,ctl_cht_train_matrix,ctl_cht_test_matrix_time,ctl_cht_train_matrix_time
    ,colortest,colordata);

title('EWMA control chart for the residuals of the high speed gearbox bearing temperature');
ylabel('\fontsize{14}[C]')

legend('data','lower limit','upper limit','center','alarms','Location','EastOutside')
```

The following function identifies the specified model (model 1 or 2 of section 4.1.2) using training data and computes estimations for test data.

Model_Fun

```
function [Predi,Resid,TimeP,AbsiP] = Model_Fun(head_dat,head_test,end_train,end_test,model)
% MODEL_FUN: used data in head_dat and prior to end_train to build a
% predictive model for data in head_test between end_train and end_test.
% INPUTS: - head_dat: cell of training machines names
%         - head_test: test machine name (string)
%         - end_train: end of training set (double)
%         - end_test: end of test set (double)
%         - model: model number (1 or 2)
% OUTPUTS: - Predi: predictions on training and test sets
%          - Resid: residuals on training and test sets
%          - TimeP: time abscissae for Predi and Resid
%          - AbsiP: abscissae for Predi and Resid in the test data

%% 1. Select data
dat = cell(length(head_dat),1);
for l=1:length(head_dat)
    load(head_dat{l});
    dat{l} = data;
end

rec_start = 12; % parameter for number of previous ambient temperatures measurements to consider
% number of hidden units in the hidden layer / could be a vector:
if model==1; nhidden = 2;
elseif model==2; nhidden = 3;
else; error('Invalid model value: model should be 1 or 2');
end
its = 1000; % maximal number of iterations for the scaled conjugate gradient algorithm


%% 2. Target features
head_tar = {'HIGH_SPEED_GEARBOX_BEARING_TEMPERATURE_1'};
n_tar = length(head_tar);
% autoregressive parameters vector for signals of head_tar, should be the size of head_tar;
if model==1; n_aut = zeros(n_tar,1);
else; n_aut = 1*ones(n_tar,1);
end
% initialize indexes of targeted features
ind_tar = zeros(n_tar,1);

%% 3. Input features
head_sel = {'HUB_SPEED' 'AMBIENT_TEMPERATURE_1' 'ACTIVE_POWER'};
n_sel = length(head_sel);
% external dynamics parameter for signals of head_sel / should be the size
% of head_sel:
if model==1; n_rec = [15 5 5];
else; n_rec = [3 2 2]; end
% find indexes of selected features
ind_sel = zeros(n_sel,1);


%% 4. Training stage
% 4.1 Initialize the nets
net = cell(length(nhidden),1); % each cell will contain a net
TR = cell(length(nhidden),1); % each cell will contain a training record
for i=1:length(nhidden)
    net{i} = fitnet(nhidden(i)); % initialize a 2-layer MLP
    net{i}.trainFcn = 'trainscg'; % comment for trainlm
    net{i}.trainParam.epochs = its; % matlab default is 1000
    net{i}.divideParam.trainRatio = 1;
    net{i}.inputs{1}.processFcns = {}; % suppress normalization of the network, we already proceeded to
        normalization
```

```
    net{i}.outputs{2}.processFcns = {}; % suppress normalization of the network, we already proceeded to normalization
end
% 4.2 build full Xtrain matrix considering all training machines
Xtrain = [];
Ytrain = [];
for h=1:length(dat) % loop on the training machines
    % find indexes of selected features for machine h
    data = dat{h};
    for i=1:n_sel
        if ~isempty(find(strcmp(head_sel{i},data.header'),1))
            ind_sel(i) = find(strcmp(head_sel{i},data.header'),1);
        end
    end
    ind_sel = ind_sel(ind_sel>0);
    for i=1:n_tar
        ind_tar(i) = find(strcmp(head_tar{i},data.header'),1);
    end
    ind_tar = ind_tar(ind_tar>0);

    % Initialize bounds on the ambient temperature
    amb_temp = 'AMBIENT_TEMPERATURE_1';
    feat_num_rec = find(strcmp(amb_temp,data.header'),1);
    feat_num_unbound = 1000; % no bound
    feat_num_infunbound = -1000; % no bound
    feat_num_bound = 0; % no bound
    feat_num_infbound = 1000;

    % Build X and Y
    X = data.sig(:,ind_sel);
    Y = data.sig(:,ind_tar);

    % increase input matrix dimensions / external dynamics
    start_time = max([n_rec n_aut])+1;
    Xnew = [];
    % for input signals
    for l=1:n_sel
        for j=0:n_rec(l) % start from 0 because we take signal at time t
           Xnew = [Xnew X(start_time-j:end-j,l)];
        end
    end
    % for output signals
    for l=1:n_tar
        for j=1:n_aut(l) % start from 1 because we do not take signal at time t
           Xnew = [Xnew Y(start_time-j:end-j,l)];
        end
    end
    X = [zeros(start_time-1,size(Xnew,2)); Xnew];

    % find valid abscissae
    abs_all = SelectDataNotNan([X Y],start_time,size(X,1),data,rec_start,feat_num_rec,feat_num_unbound,feat_num_infunbound);
    abs_train = abs_all(data.timevec(abs_all)<end_train);
    mean_vec = mean(X(abs_train,:),1);
    max_vec =  std(X(abs_train,:));
    Xn = X-repmat(mean_vec,length(X(:,1)),1); % centering
    Xn = Xn./repmat(max_vec,length(X(:,1)),1); % normalize
    Xtrain = [Xtrain; Xn(abs_train,:)]; % update training matrix with data from this machine
    mean_vec = mean(Y(abs_train,:),1);
    max_vec =  std(Y(abs_train,:));
    Yn = Y-repmat(mean_vec,length(Y(:,1)),1); % centering
    Yn = Yn./repmat(max_vec,length(Y(:,1)),1); % normalize
    Ytrain = [Ytrain; Yn(abs_train,:)]; % update training matrix with data from this machine

    feat_num_bound = max(max(data.sig(abs_train,feat_num_rec)),feat_num_bound); % update temperatures bounds
    feat_num_infbound = min(min(data.sig(abs_train,feat_num_rec)),feat_num_infbound); % update temperatures bounds
end

% 4.3 Learn model with Xtrain
myperm = randperm(size(Xtrain,1)); % data permutation
X_mlp = {Xtrain(myperm,:)}; % permute data
Y_mlp = {Ytrain(myperm,:)}; % permute data
for i=1:length(nhidden)
    fprintf('Starting training MLP for : M_hidden = %d, max_its = %d \n',nhidden(i),its);
    [net{i},TR{i}] = train(net{i}, X_mlp{1}', Y_mlp{1}'); % train the MLP
    fprintf('Finished training MLP for : M_hidden = %d, max_its = %d \n',nhidden(i),its);
end

%% 5. Test stage
load(head_test); % load test machine

% 5.1 find indexes of selected features for test machine
for i=1:n_sel
    if ~isempty(find(strcmp(head_sel{i},data.header'),1))
        ind_sel(i) = find(strcmp(head_sel{i},data.header'),1);
    end
end
ind_sel = ind_sel(ind_sel>0);
for i=1:n_tar
    ind_tar(i) = find(strcmp(head_tar{i},data.header'),1);
```

```
end
ind_tar = ind_tar(ind_tar>0);

% 5.2 Build X and Y
X = data.sig(:,ind_sel);
Y = data.sig(:,ind_tar);

% 5.3 increase input matrix dimensions / external dynamics
start_time = max([n_rec n_aut])+1;
Xnew = [];
% for input signals
for l=1:n_sel
    for j=0:n_rec(l) % start from 0 because we take signal at time t
        Xnew = [Xnew X(start_time-j:end-j,l)];
    end
end
% for output signals
for l=1:n_tar
    for j=1:n_aut(l) % start from 1 because we do not take signal at time t
        Xnew = [Xnew Y(start_time-j:end-j,l)];
    end
end
X = [zeros(start_time-1,size(Xnew,2)); Xnew];

% find valid abscissae
abs_all = SelectDataNotNan([X Y],start_time,size(X,1),data,rec_start,feat_num_rec,feat_num_bound,
    feat_num_infbound);
abs_all = abs_all(data.timevec(abs_all)<=end_test);
abs_train = abs_all(data.timevec(abs_all)<end_train);
abs_test = abs_all(data.timevec(abs_all)>=end_train);
% Normalize X:
mean_vec = mean(X(abs_train,:),1);
max_vec =  std(X(abs_train,:));
Xn = X-repmat(mean_vec,length(X(:,1)),1); % centering
Xn = Xn./repmat(max_vec,length(X(:,1)),1); % normalize
Xtest = Xn(abs_test,:);
Xtrain = Xn(abs_train,:);
% Normalize Y:
mean_vec = mean(Y(abs_train,:),1);
max_vec =  std(Y(abs_train,:));
Yn = Y-repmat(mean_vec,length(Y(:,1)),1); % centering
Yn = Yn./repmat(max_vec,length(Y(:,1)),1); % normalize
Ytest = Yn(abs_test,:);
Ytrain = Yn(abs_train,:);

% Compute model outputs
Y_mlp = {Ytest Ytrain Yn(abs_all,:)};
X_mlp = {Xtest Xtrain Xn(abs_all,:)};
abs_mlp = {abs_test abs_train abs_all};
nsample_mlp = length(Y_mlp);
errMLP = cell(nsample_mlp,n_tar);
errMLPcum = cell(nsample_mlp,1);
for k=1:nsample_mlp
    errMLPcum{k} = zeros(length(nhidden),1);
end
Ypred = cell(nsample_mlp,length(nhidden),1); % each cell will contain a prediction matrix
residuals = cell(nsample_mlp,length(nhidden),1); % each cell will contain a residual matrix
for i=1:length(nhidden)
        fprintf('Starting computing MLP outputs for : data = %s, M_hidden = %d, max_its = %d \n',head_dat{h
            },nhidden(i),its);
        for k=1:nsample_mlp
            [Ypred{k,i}] = sim(net{i},X_mlp{k}')'; % computes the MLP's outputs on inputs Xtest{k} (net(X)
                could be used too)
            Ypred{k,i} = Ypred{k,i}.*repmat(max_vec,size(Ypred{k,i},1),1);
            for l=1:n_tar
                Ypred{k,i}(:,l) = Ypred{k,i}+repmat(mean_vec,size(Ypred{k,i},1),1);
            end
            residuals{k,i} = data.sig(abs_mlp{k},ind_tar)-Ypred{k,i}; % compute residual matrix
            for l=1:n_tar
                errMLP{k,l}(i) = norm(residuals{k,i}(:,l))/sqrt(length(Y_mlp{k}(:,1))); % compute error
            end
            for l=1:n_tar
                errMLPcum{k}(i) = errMLPcum{k}(i) + errMLP{k,l}(i)/n_tar; % average error
            end
        end
        fprintf('Finished computing MLP outputs for : data = %s, M_hidden = %d, max_its = %d \n',head_dat{h
            },nhidden(i),its);
end

Predi = Ypred{end,1};
Resid = residuals{end,1};
TimeP = data.timevec(abs_mlp{end});
AbsiP = abs_mlp{end};

end
```

This function is used to filter the data. It should remove NAN values as well as points where the signal `feat_num` is out of the training bounds (`inf_bound` and `up_bound`) and points where the wind turbine has not been operating at any time during the last hour.

SelectDataNotNan

```
function [ind] = SelectDataNotNan(X,ind_start,n,data,rec_start,feat_num,up_bound,inf_bound,dir)
% SELECTDATANOTNAN: select n non NaN points in x starting from ind_start
% INPUT: - x: data
%        - ind_start: starting index
%        - n: number of points to select
%        - rec_start: number of previous measurements to reject at startup
%        - dir: selected points are right to ind_start unless dir = 1
%        - feat_num: feature number for selection
%        - up_bound: upper bound for selection
%        - inf_bound : lower bound for selection
% OUTPUT: - ind: index vector of the selected points
if nargin < 9; dir = 0; end
cur_pos = ind_start;
ind = zeros(n,1);
m = length(X(:,1));
if dir == 0
    i = 1;
    while i<=n && cur_pos<=m
        if ~isnan(sum(X(cur_pos,:))) && all(data.sig(max(cur_pos-rec_start,1):cur_pos,feat_num) <= up_bound)
              && all(data.sig(max(cur_pos-rec_start,1):cur_pos,feat_num) >= inf_bound) && all(data.sig(max(
             cur_pos-5,1):cur_pos,30) > 000)
            ind(i) = cur_pos;
            i = i+1;
        end
        cur_pos = cur_pos+1;
    end
else
    i = n;
    while i>=1 && cur_pos>0
        if ~isnan(sum(X(cur_pos,:)));
            ind(i) = cur_pos;
            i = i-1;
        end
        cur_pos = cur_pos-1;
    end
end
ind = ind(ind>0);
end
```

This function builds Xbar control charts.

XBARControlChart

```
function [] = XBARControlChart(data_matrix_test,data_matrix_train,limits,time_test,time_train,color_test,
    color_train)
% XBARCONTROLCHART: XBAR control chart for data_matrix_test
% INPUTS: - data_matrix_test: test data
%         - data_matrix_train: training data
%         - limits: vector containing the lower limt, center and upper limit
%         - time_train: time vector for training data
%         - time_test: time vector for test data
%         - color_test: color for test data
%         - color_train: color for training data
signtest = mean(data_matrix_test,2);
signtrain = mean(data_matrix_train,2);
time = [time_train; time_test];
sign = [signtrain; signtest];

plot([time_train; time_test],[signtrain; signtest],'color',color_test,'marker','.','linewidth',2); hold on;
% plot(time_test,signtest,'color',color_test,'marker','.','linewidth',2); hold on;
% plot(time_train,signtrain,'color',color_train,'marker','.','linewidth',2);
plot([time(1) time(end)],[limits(1) limits(1)],'k'); hold on;
plot([time(1) time(end)],[limits(end) limits(end)],'k');
plot([time(1) time(end)],[limits(2) limits(2)],'k--');

xlim([time(1) time(end)]);
set(gca,'XTick',(min(time):70:max(time)));
datetick('x','dd/mm/yyyy','keeplimits','keepticks');

% Plot faults
ind_fault = ((sign<limits(1)) + (sign>limits(3)))>0;
sign_fault = sign(ind_fault);
time_fault = time(ind_fault);
plot(time_fault,sign_fault,'color','r','LineStyle','none','marker','o','Markersize',5,'linewidth',2);

%legend('test data','training data','lower limit','upper limit','center','alarms','Location','EastOutside')
end
```

This function builds EWMA control charts.

EWMAControlChart

```
function [] = EWMAControlChart(data_matrix_test,data_matrix_train,time_test,time_train,color_test,
    color_train,U,L)
% EWMACONTROLCHART: EWMA control chart for data_matrix_test
% INPUTS: - data_matrix_test: test data
%         - data_matrix_train: training data
%         - time_train: time vector for training data
```

```
%          - time_test: time vector for test data
%          - color_test: color for test data
%          - color_train: color for training data
%          - U,L (vectors): upper and lower limits if we do not want to use default
%          EWMA limits
lambda = 0.25; % 0.25
k = 3;

signtest = mean(data_matrix_test,2);
signtrain = mean(data_matrix_train,2);
mu = mean(signtrain);
sigma = std(signtrain);

sign = [signtrain; signtest];
sign(1) = lambda*sign(1);
for i=2:length(sign)
    sign(i) = lambda*sign(i)+(1-lambda)*sign(i-1);
end
time = [time_train; time_test];
ind_train = 1:length(time_train);
ind_test = ind_train(end)+(1:length(time_test));
ind = 1:length(time);

signtest = sign(ind_test);
signtrain = sign(ind_train);

% Compute limits:
if nargin<8
U = mu + k*sigma*sqrt(lambda*(1-(1-lambda).^(2*ind))/(2-lambda));
L = mu - k*sigma*sqrt(lambda*(1-(1-lambda).^(2*ind))/(2-lambda));
end

plot([time_train; time_test],[signtrain; signtest],'color',color_test,'marker','.','linewidth',2); hold on;
%plot(time_train,signtrain,'color',color_train,'marker','.','linewidth',2);
%plot(time_test,signtest,'color',color_test,'marker','.','linewidth',2);
plot(time,U,'k');
plot(time,L,'k');
plot([time(1) time(end)],[mu mu],'k--');

xlim([time(1) time(end)]);
set(gca,'XTick',(min(time):70:max(time)));
datetick('x','dd/mm/yyyy','keeplimits','keepticks');

% Plot faults
ind_fault = ((sign<L') + (sign>U'))>0;
sign_fault = sign(ind_fault);
time_fault = time(ind_fault);
plot(time_fault,sign_fault,'color','r','LineStyle','none','marker','o','Markersize',5,'linewidth',2);

%legend('test data','training data','lower limit','upper limit','center','alarms','Location','EastOutside')
end
```

B.3 Codes of section 4.2

Here are the Matlab codes for section 4.2 detecting increases in the generator non-driven-end bearing temperature. The following script performs online monitoring.

Script_Model

```
% Script for monitoring of a wind turbine with 2-months ahead predictions
% To use with data E.F or E.RY
% Not to use with E.RC and E.V (not enough data)

%% 1. Initialization
% Training and test machines (RC and V: not enough data):
head_train = {'E5F.mat'}; % replace by E3F for results for the faulty machine
head_test = head_train{1};
year = 2015; % starting year (2015 for E.F and 2016 E.RY)
step = 2; % number of healthy months before adding a new month to training data

end_train_month = 15; % first month of test counting from 01/01/2015
start_month = end_train_month+1;
end_simu = start_month+2; % end of simulation
begin_train = datenum(year,7,1,0,0,0); % begin of training day
end_train = datenum(year,end_train_month,1,0,0,0); % end of training day / begin of test day
end_test = datenum(year,start_month+step,1,0,0,0);


% head_train = {'E6RY.mat'};
% head_test = head_train{1};
% year = 2016;
% end_train_month = 4; % first month of test counting from 01/01/2015
% start_month = end_train_month+1;
% end_simu = start_month+2; % end of simulation
% begin_train = datenum(year,1,1,0,0,0); % begin of training day
```

```
% end_train = datenum(year,end_train_month,1,0,0,0); % end of training day / begin of test day
% end_test = datenum(year,start_month+step,1,0,0,0);

%% 2. Simulation

head_tar = {'GENERATOR_DE_BEARING_TEMPERATURE' 'GENERATOR_NDE_BEARING_TEMPERATURE'}; % list of 2 features to
      predict
n_tar = length(head_tar);
ind_tar = zeros(n_tar,1);

load(head_test); % load test machine
% Get indexes of target signals:
for i=1:n_tar
    ind_tar(i) = find(strcmp(head_tar{i},data.header'),1);
end
ind_tar = ind_tar(ind_tar>0);

for l=1:n_tar
    % Initialization:
    SignPred = []; % will contain the step-month predictions
    TimePred = []; % will contain time indexes of the step-month predictions
    ResiPred = []; % will contain residuals of the step-month predictions
    AbsiPred = []; % will contain abscissae for residuas and predictions
    for m = start_month:end_simu % 12
        current_month = mod(m,12);
        add_year = floor(m/12);
        fprintf('Start MLP with end of training: 28-29-30-31/%d/%d \n',current_month,year+add_year);
        if m==start_month
            [Predi,Resid,TimeP,AbsiP] = Model_Fun(head_train,head_test,begin_train,end_train,end_test,
                head_tar(l));
            abs_m = ((TimeP>=begin_train).*(TimeP<=end_test))>0;
        else
            [Predi,Resid,TimeP,AbsiP] = Model_Fun(head_train,head_test,begin_train,end_train,end_test,
                head_tar(l));
            abs_m = ((TimeP>datenum(year,m+step-1,1,0,0,0)).*(TimeP<=datenum(year,m+step,1,0,0,0)))>0;
        end
        SignPred = [SignPred; Predi(abs_m)];
        ResiPred = [ResiPred; Resid(abs_m)];
        TimePred = [TimePred; TimeP(abs_m)];
        AbsiPred = [AbsiPred; AbsiP(abs_m)];
        end_train = datenum(year,m+1,1,0,0,0);
        end_test = datenum(year,m+1+step,1,0,0,0);
    end
    Resi_Fin{l} = ResiPred;
    Sign_Fin{l} = SignPred;
    Time_Fin{l} = TimePred;
    Absi_Fin{l} = AbsiPred;
end


[Absi_Comm,A_ind,B_ind] = intersect(Absi_Fin{1},Absi_Fin{2}); % common abscissae to predictions 1 and 2
Resi_Comm = [Resi_Fin{1}(A_ind) Resi_Fin{2}(B_ind)]; % corresponding residuals
Sign_Comm = [Sign_Fin{1}(A_ind) Sign_Fin{2}(B_ind)]; % corresponding signals
Time_Comm = [Time_Fin{1}(A_ind) Time_Fin{2}(B_ind)]; % corresponding signals
RMSE = std(Resi_Comm); % compute RMSE



%% 3. Visualization
% 3.1 Signal and predictions
for l=1:n_tar
    figure
    plot(data.timevec(Absi_Comm),data.sig(Absi_Comm,ind_tar(l)),'linewidth',1); hold on;
    plot(data.timevec(Absi_Comm),Sign_Comm(:,l),'linewidth',1)
    xlim([begin_train Time_Comm(end)]);
    set(gca,'XTick',begin_train:70:Time_Comm(end));
    datetick('x','dd/mm/yyyy','keeplimits','keepticks');
    title(lower(strrep(head_tar{l},'_',' ')));
    ylabel('\fontsize{14}[C]')
    legend('Measurements','Predictions','Location','EastOutside')
end

% 3.2 Residuals
for l=1:n_tar
    figure
    plot(data.timevec(Absi_Comm),Resi_Comm(:,l),'linewidth',1)
    xlim([begin_train Time_Comm(end)]);
    set(gca,'XTick',begin_train:70:Time_Comm(end));
    datetick('x','dd/mm/yyyy','keeplimits','keepticks');
    title(['Residuals of ' lower(strrep(head_tar{l},'_',' '))]);
    ylabel('\fontsize{14}[C]')
    legend('Residuals','Location','EastOutside')
end

%% 3.3 XBAR control charts
colordata = [0.2 0.8 0.2];
colortest = 'b';
ntr = 10000; % number of points for estimating mu and S
n = 40; % number of measurements in every sample of the controlchart

residuals_train = Resi_Comm(1:ntr,:);
```

```
residuals_test = Resi_Comm(ntr+1:end,:);
abs_train = Time_Comm(1:ntr);
abs_test = Time_Comm(ntr+1:end);
% Select data
L = n*floor(size(residuals_train,1)/n);
ctl_cht_train = residuals_train(1:L,:);
P = n*floor(size(residuals_test,1)/n);
ctl_cht_test = residuals_test(1:P,:);
% Select time abscissae
ctl_cht_train_matrix_time = reshape((abs_train(1:L)),n,L/n)';
ctl_cht_train_matrix_time = ctl_cht_train_matrix_time(:,end);
ctl_cht_test_matrix_time = reshape((abs_test(1:P)),n,P/n)';
ctl_cht_test_matrix_time = ctl_cht_test_matrix_time(:,end);

for l=1:n_tar
    ctl_cht_train_matrix = reshape(ctl_cht_train(:,l),n,L/n)';
    ctl_cht_test_matrix = reshape(ctl_cht_test(:,l),n,P/n)';
    mu = mean(ctl_cht_train(:,l));
    sigma = std(ctl_cht_train(:,l));
%    false_alarm_rate = 0.01;
%    ULm = quantile(mean(ctl_cht_train_matrix,2),1-false_alarm_rate/2);
%    LLm = quantile(mean(ctl_cht_train_matrix,2),false_alarm_rate/2);
    ULm = mu+3*std(mean(ctl_cht_train_matrix,2));
    LLm = mu-3*std(mean(ctl_cht_train_matrix,2));
    figure
    XBARControlChart(ctl_cht_test_matrix,ctl_cht_train_matrix,[LLm mu ULm],ctl_cht_test_matrix_time,
        ctl_cht_train_matrix_time,colortest,colordata);
    title(['XBAR control chart for the residuals of the ' lower(strrep(head_tar{l},'_',' '))]);
    ylabel('\fontsize{14}[C]')
    legend('data','lower limit','upper limit','center','alarms','Location','EastOutside')
end

%% 3.4 EWMA control charts
for l=1:n_tar
    ctl_cht_train_matrix = reshape(ctl_cht_train(:,l),n,L/n)';
    ctl_cht_test_matrix = reshape(ctl_cht_test(:,l),n,P/n)';
    mu = mean(ctl_cht_train(:,l));
    sigma = std(ctl_cht_train(:,l));
    figure
    EWMAControlChart(ctl_cht_test_matrix,ctl_cht_train_matrix,ctl_cht_test_matrix_time,
        ctl_cht_train_matrix_time,colortest,colordata);
    title(['EWMA control chart for the residuals of the ' lower(strrep(head_tar{l},'_',' '))]);
    ylabel('\fontsize{14}[C]')
    legend('data','lower limit','upper limit','center','alarms','Location','EastOutside')
end

%% 3.5 MEWMA control chart

ind = 1;
L = floor(length(residuals_train)/n);
q_train = zeros(L,size(residuals_train,2));
time_train = zeros(L,1);
for l=1:L
    abs_h = (ind-1)*n+(1:n);
    q_train(l,:) = mean(residuals_train(abs_h,:));
    time_train(l) = abs_train(abs_h(end));
    ind = ind+1;
end
ind = 1;
L = floor(length(residuals_test)/n);
q_test = zeros(L,size(residuals_test,2));
time_test = zeros(L,1);
for l=1:L
    abs_h = (ind-1)*n+(1:n);
    q_test(l,:) = mean(residuals_test(abs_h,:));
    time_test(l) = abs_test(abs_h(end));
    ind = ind+1;
end
MEWMAControlChart(q_test,q_train,time_test,time_train,colortest,colordata)
legend('data','upper limit','alarms','Location','EastOutside')
title(['MEWMA control chart for the residuals of the DE and NDE temperatures'])
```

The following function identifies the auto-regressive model with the training data and computes estimations for the test data.

Model_Fun

```
function [Predi,Resid,TimeP,AbsiP] = Model_Fun(head_dat,head_test,begin_train,end_train,end_test,head_tar)
% MODEL_FUN: used data in head_dat and prior to end_train to build a
% predictive model for data in head_test between end_train and end_test.
% INPUTS: - head_dat: cell of training machines names
%         - head_test: test machine name (string)
%         - begin_train: start of training set (double)
%         - end_train: end of training set (double)
%         - end_test: end of test set (double)
%         - head_tar: targeted feature
% OUTPUTS: - Predi: predictions on training and test sets
%          - Resid: residuals on training and test sets
%          - TimeP: time abscissae for Predi and Resid
%          - AbsiP: abscissae for Predi and Resid in the test data
```

```
14
15  %% 1. Select data
16  dat = cell(length(head_dat),1);
17  for l=1:length(head_dat)
18      load(head_dat{l});
19      dat{l} = data;
20  end
21
22  rec_start = 12; % parameter for number of previous ambient temperatures measurements to consider
23  nhidden = [3]; % number of hidden units in the hidden layer / could be a vector:
24  its = 1000; % maximal number of iterations for the scaled conjugate gradient algorithm
25
26  %% 2. Target features
27  n_tar = length(head_tar);
28  % autoregressive parameters vector for signals of head_tar, should be the size of head_tar;
29  n_aut = 1*ones(n_tar,1);
30  % initialize indexes of targeted features
31  ind_tar = zeros(n_tar,1);
32
33  %% Explanatory features
34  head_sel = {'GENERATOR_SPEED_1' 'AMBIENT_TEMPERATURE_1'}; % 'ACTIVE_POWER'
35  n_sel = length(head_sel);
36  n_rec = [3 2];%[9 2 2]; % external dynamics parameter for signals of head_sel;
37  % initialize indexes of selected features
38  ind_sel = zeros(n_sel,1);
39
40  %% 4. Training stage
41  % 4.1 Initialize the nets
42  net = cell(length(nhidden),1); % each cell will contain a net
43  TR = cell(length(nhidden),1); % each cell will contain a training record
44  for i=1:length(nhidden)
45      net{i} = fitnet(nhidden(i)); % initialize a 2-layer MLP
46      net{i}.trainFcn = 'trainscg'; % comment for trainlm
47      net{i}.trainParam.epochs = its; % matlab default is 1000
48      net{i}.divideParam.trainRatio = 1;
49      net{i}.inputs{1}.processFcns = {}; % suppress normalization of the network, we already proceeded to
            normalization
50      net{i}.outputs{2}.processFcns = {}; % suppress normalization of the network, we already proceeded to
            normalization
51  end
52  % 4.2 build full Xtrain matrix considering all training machines
53  Xtrain = [];
54  Ytrain = [];
55  for h=1:length(dat) % loop on the training machines
56      % find indexes of selected features for machine h
57      data = dat{h};
58      for i=1:n_sel
59          if ~isempty(find(strcmp(head_sel{i},data.header'),1))
60              ind_sel(i) = find(strcmp(head_sel{i},data.header'),1);
61          end
62      end
63      ind_sel = ind_sel(ind_sel>0);
64      for i=1:n_tar
65          ind_tar(i) = find(strcmp(head_tar{i},data.header'),1);
66      end
67      ind_tar = ind_tar(ind_tar>0);
68
69      % Initialize bounds on the ambient temperature
70      amb_temp = 'AMBIENT_TEMPERATURE_1';
71      feat_num_rec = find(strcmp(amb_temp,data.header'),1);
72      feat_num_unbound = 1000; % no bound
73      feat_num_infunbound = -1000; % no bound
74      feat_num_bound = 0; % no bound
75      feat_num_infbound = 1000;
76
77      % Build X and Y
78      X = data.sig(:,ind_sel);
79      Y = data.sig(:,ind_tar);
80
81      % increase input matrix dimensions / external dynamics
82      start_time = max([n_rec n_aut])+1;
83      Xnew = [];
84      % for input signals
85      for l=1:n_sel
86          for j=0:n_rec(l) % start from 0 because we take signal at time t
87             Xnew = [Xnew X(start_time-j:end-j,l)];
88          end
89      end
90      % for output signals
91      for l=1:n_tar
92          for j=1:n_aut(l) % start from 1 because we do not take signal at time t
93             Xnew = [Xnew Y(start_time-j:end-j,l)];
94          end
95      end
96      X = [zeros(start_time-1,size(Xnew,2)); Xnew];
97
98      % find valid abscissae
99      abs_all = SelectDataNotNan([X Y],start_time,size(X,1),data,rec_start,feat_num_rec,feat_num_unbound,
            feat_num_infunbound);
100     abs_train = abs_all(((data.timevec(abs_all)<end_train).*(data.timevec(abs_all)>=begin_train))>0);
101     mean_vec = mean(X(abs_train,:),1);
```

```
    max_vec =  std(X(abs_train,:));
    Xn = X-repmat(mean_vec,length(X(:,1)),1); % centering
    Xn = Xn./repmat(max_vec,length(X(:,1)),1); % normalize
    Xtrain = [Xtrain; Xn(abs_train,:)]; % update training matrix with data from this machine
    mean_vec = mean(Y(abs_train,:),1);
    max_vec =  std(Y(abs_train,:));
    Yn = Y-repmat(mean_vec,length(Y(:,1)),1); % centering
    Yn = Yn./repmat(max_vec,length(Y(:,1)),1); % normalize
    Ytrain = [Ytrain; Yn(abs_train,:)]; % update training matrix with data from this machine

    feat_num_bound = max(max(data.sig(abs_train,feat_num_rec)),feat_num_bound); % update temperatures bounds
    feat_num_infbound = min(min(data.sig(abs_train,feat_num_rec)),feat_num_infbound); % update temperatures
        bounds
end

% 4.3 Learn model with Xtrain
myperm = randperm(size(Xtrain,1)); % data permutation
X_mlp = {Xtrain(myperm,:)}; % permute data
Y_mlp = {Ytrain(myperm,:)}; % permute data
for i=1:length(nhidden)
    fprintf('Starting training MLP for : M_hidden = %d, max_its = %d \n',nhidden(i),its);
    [net{i},TR{i}] = train(net{i}, X_mlp{1}', Y_mlp{1}'); % train the MLP
    fprintf('Finished training MLP for : M_hidden = %d, max_its = %d \n',nhidden(i),its);
end

%% 5. Test stage
load(head_test); % load test machine

% 5.1 find indexes of selected features for test machine
for i=1:n_sel
    if ~isempty(find(strcmp(head_sel{i},data.header'),1))
        ind_sel(i) = find(strcmp(head_sel{i},data.header'),1);
    end
end
ind_sel = ind_sel(ind_sel>0);
for i=1:n_tar
    ind_tar(i) = find(strcmp(head_tar{i},data.header'),1);
end
ind_tar = ind_tar(ind_tar>0);

% 5.2 Build X and Y
X = data.sig(:,ind_sel);
Y = data.sig(:,ind_tar);

% 5.3 increase input matrix dimensions / external dynamics
start_time = max([n_rec n_aut])+1;
Xnew = [];
% for input signals
for l=1:n_sel
    for j=0:n_rec(l) % start from 0 because we take signal at time t
        Xnew = [Xnew X(start_time-j:end-j,l)];
    end
end
% for output signals
for l=1:n_tar
    for j=1:n_aut(l) % start from 1 because we do not take signal at time t
        Xnew = [Xnew Y(start_time-j:end-j,l)];
    end
end
X = [zeros(start_time-1,size(Xnew,2)); Xnew];

% find valid abscissae
abs_all = SelectDataNotNan([X Y],start_time,size(X,1),data,rec_start,feat_num_rec,feat_num_bound,
    feat_num_infbound);
abs_all = abs_all(data.timevec(abs_all)<=end_test);
abs_train = abs_all(((data.timevec(abs_all)<end_train).*(data.timevec(abs_all)>=begin_train))>0);
abs_test = abs_all(data.timevec(abs_all)>=end_train);
% Normalize X:
mean_vec = mean(X(abs_train,:),1);
max_vec =  std(X(abs_train,:));
Xn = X-repmat(mean_vec,length(X(:,1)),1); % centering
Xn = Xn./repmat(max_vec,length(X(:,1)),1); % normalize
Xtest = Xn(abs_test,:);
Xtrain = Xn(abs_train,:);
% Normalize Y:
mean_vec = mean(Y(abs_train,:),1);
max_vec =  std(Y(abs_train,:));
Yn = Y-repmat(mean_vec,length(Y(:,1)),1); % centering
Yn = Yn./repmat(max_vec,length(Y(:,1)),1); % normalize
Ytest = Yn(abs_test,:);
Ytrain = Yn(abs_train,:);

% Compute model outputs
Y_mlp = {Ytest Ytrain Yn(abs_all,:)};
X_mlp = {Xtest Xtrain Xn(abs_all,:)};
abs_mlp = {abs_test abs_train abs_all};
nsample_mlp = length(Y_mlp);
errMLP = cell(nsample_mlp,n_tar);
errMLPcum = cell(nsample_mlp,1);
for k=1:nsample_mlp
    errMLPcum{k} = zeros(length(nhidden),1);
```

```
end
Ypred = cell(nsample_mlp,length(nhidden),1); % each cell will contain a prediction matrix
residuals = cell(nsample_mlp,length(nhidden),1); % each cell will contain a residual matrix
for i=1:length(nhidden)
        fprintf('Starting computing MLP outputs for : data = %s, M_hidden = %d, max_its = %d \n',head_dat{h
            },nhidden(i),its);
        for k=1:nsample_mlp
            [Ypred{k,i}] = sim(net{i},X_mlp{k}')'; % computes the MLP's outputs on inputs Xtest{k} (net(X)
                could be used too)
            Ypred{k,i} = Ypred{k,i}.*repmat(max_vec,size(Ypred{k,i},1),1);
            for l=1:n_tar
                Ypred{k,i}(:,l) = Ypred{k,i}+repmat(mean_vec,size(Ypred{k,i},1),1);
            end
            residuals{k,i} = data.sig(abs_mlp{k},ind_tar)-Ypred{k,i}; % compute residual matrix
            for l=1:n_tar
                errMLP{k,l}(i) = norm(residuals{k,i}(:,l))/sqrt(length(Y_mlp{k}(:,1))); % compute error
            end
            for l=1:n_tar
                errMLPcum{k}(i) = errMLPcum{k}(i) + errMLP{k,l}(i)/n_tar; % average error
            end
        end
        fprintf('Finished computing MLP outputs for : data = %s, M_hidden = %d, max_its = %d \n',head_dat{h
            },nhidden(i),its);
end

Predi = Ypred{end,1};
Resid = residuals{end,1};
TimeP = data.timevec(abs_mlp{end});
AbsiP = abs_mlp{end};

end
```

B.4 Codes of section 5.2

Here are the codes used for identifying the auto-encoder of section 5.2. The following script proceed to the training step and test step. Then, it plots individual Xbar and EWMA control charts as well as a MEWMA control chart.

Script_Blind

```
%% 1. Training stage
head_dat = {'E1F.mat' 'E2F.mat' 'E4F.mat' 'E5F.mat'};
begin_train = datenum(2015,1,1,0,0,0);
end_train = datenum(2015,4,1,0,0,0);

[net,head_sel,n_rec,rec_start,nhidden,nlay] = Model_Fun(head_dat,begin_train,end_train);


%% 2. Test stage
head_test = 'E2F.mat'; % E5F can be tried too (it is not necessary to run point 1 again)
begin_test = datenum(2015,4,1,0,0,0);
end_test = datenum(2015,7,1,0,0,0);

[Xpred,residuals,bottleneckval,abs_mlp,errMLP,errMLPcum] = Simul_Fun(net,head_test,begin_train,end_train,
    begin_test,end_test,head_sel,n_rec,rec_start,nhidden,nlay);

%% 3. Identify improbable bottleneckval for net{1,1}
I = 1;
J = 1;

pts_dens = size(bottleneckval{2,I,J},1);
DAT = bottleneckval{2,I,J}(1:floor(pts_dens/2),:);
DAT2 =  bottleneckval{2,I,J}(floor(pts_dens/2)+1:end,:);
OBJ = bottleneckval{3,I,J};
[n,d] = size(DAT);
b = std(DAT)*(4/n/(d+2))^(1/(d+4));
pdDAT = mvksdensity(DAT,DAT2,'bandwidth',b,'Function','pdf'); % nonparametric distribution
pdOBJ = mvksdensity(DAT,OBJ,'bandwidth',b,'Function','pdf'); % nonparametric distribution
lim = quantile(pdDAT,0.01);
is_out = pdOBJ<lim;

%% 4. Individual XBAR control charts for net{1,1}
load(head_test);
n_sel = length(head_sel);
colordata = [0.2 0.8 0.2];
colortest = 'b';
Units_Blind;
n = 40; % number of measurements in every sample of the controlchart
ctl_cht_indtrain = 2;
ctl_cht_indtest = 1;

residuals_train = residuals{ctl_cht_indtrain,I,J};
residuals_test = residuals{ctl_cht_indtest,I,J};

```

```
% Training data
L = n*floor(size(residuals_train,1)/n);
ctl_cht_train = residuals_train(1:L,:);
P = n*floor(size(residuals_test,1)/n);
ctl_cht_test = residuals_test(1:P,:);
ctl_cht_train_matrix_time = reshape(data.timevec(abs_mlp{(ctl_cht_indtrain)}(1:L)),n,L/n)';
ctl_cht_train_matrix_time = ctl_cht_train_matrix_time(:,end);
ctl_cht_test_matrix_time = reshape(data.timevec(abs_mlp{(ctl_cht_indtest)}(1:P)),n,P/n)';
ctl_cht_test_matrix_time = ctl_cht_test_matrix_time(:,end);
ctl_cht_Is_Out = reshape(is_out(1:P),n,P/n)';

count = 1;
nsubplotperfig = 5;
figure
nstart = 1;
for l=nstart:n_sel

    ctl_cht_train_matrix = reshape(ctl_cht_train(:,l),n,L/n)';
    ctl_cht_test_matrix = reshape(ctl_cht_test(:,l),n,P/n)';

    mu = mean(ctl_cht_train(:,l));
    sigma = std(ctl_cht_train(:,l));

    %     false_alarm_rate = 0.01;
    %     ULm = quantile(mean(ctl_cht_train_matrix,2),1-false_alarm_rate/2);
    %     LLm = quantile(mean(ctl_cht_train_matrix,2),false_alarm_rate/2);
    ULm = mu+3*std(mean(ctl_cht_train_matrix,2));
    LLm = mu-3*std(mean(ctl_cht_train_matrix,2));

    subplot(nsubplotperfig,1,count)
    XBARControlChart(ctl_cht_test_matrix,ctl_cht_train_matrix,[LLm mu ULm],ctl_cht_test_matrix_time,
        ctl_cht_train_matrix_time,ctl_cht_Is_Out,colortest,colordata);

    title(['XBAR control chart for ' lower(strrep(head_sel{l},'_',' ')) ' residuals']);
    ylabel(['\fontsize{14}[' units{l} ']'])
    legend({'test data','training data','lower limit','upper limit','center','alarms','novelties'},'Location
        ','EastOutside','FontSize',8);

    if count==nsubplotperfig
        if l<n_sel; figure; end
        count = 1;
    else
        count = count+1;
    end
end

%% 5. Individual EWMA control charts
count = 1;
figure
for l=1:size(residuals_train,2)
    subplot(nsubplotperfig,1,count)
    EWMAControlChart(ctl_cht_test_matrix(:,l),ctl_cht_train_matrix(:,l),ctl_cht_test_matrix_time,
        ctl_cht_train_matrix_time,ctl_cht_Is_Out,colortest,colordata);

    title(['EWMA control chart for the residuals of the ' lower(strrep(head_sel{l},'_',' '))]);
    if count==nsubplotperfig
        if l<n_sel; figure; end
        count = 1;
    else
        count = count+1;
    end
end

%% 6. MEWMA control chart
D = size(residuals_train,2);
ctl_cht_train_matrix = zeros(L/n,D);
ctl_cht_test_matrix = zeros(P/n,D);
for l=1:L/n
    myabs = (1:n)+(l-1)*n;
    ctl_cht_train_matrix(l,:) = mean(ctl_cht_train(myabs,:));
end
for l=1:P/n
    myabs = (1:n)+(l-1)*n;
    ctl_cht_test_matrix(l,:) = mean(ctl_cht_test(myabs,:));
end
MEWMAControlChart(ctl_cht_test_matrix,ctl_cht_train_matrix,ctl_cht_test_matrix_time,
    ctl_cht_train_matrix_time,ctl_cht_Is_Out,colortest,colordata);

title(['MEWMA control chart for machine ' strrep(head_test,'.mat','')]);
legend('data','limit','alarms','novelties','Location','EastOutside')

xlim([735960 736160])
```

The following function identifies the auto-encoder model using the specified data between the specified training dates.

Model_Fun

```
function [net,head_sel,n_rec,rec_start,nhidden,nlay] = Model_Fun(head_dat,begin_train,end_train)
% MODEL_FUN: used data in head_dat and prior to end_train to build a
% predictive model.
% INPUTS: - head_dat: cell of training machines names
%         - head_test: test machine name (string)
%         - begin_train: start of training set (double)
%         - end_train: end of training set (double)
%         - begin_test: begin of test set
%         - end_test: end of test set (double)
%         - head_tar: targeted feature
% OUTPUTS: - net: a cell array of the identifed nets

%% 1. Load data
dat = cell(length(head_dat),1);
for l=1:length(head_dat)
    load(head_dat{l});
    dat{l} = data;
end

%% 2. Select features

head_sel = {'ACTIVE_POWER' 'REACTIVE_POWER' 'WIND_SPEED_1' 'NACELLE_DEVIATION'...
    'BLADE_A_PITCH_POSITION' ...
    'PHASE_VOLTAGE_A' ...
    'LUBE_OIL_PRESSURE_GEARBOX' 'HUB_SPEED' 'GEARBOX_AXIAL_GLOBAL_VIBRATION' 'POWER_FACTOR' ...
    'GEARBOX_LUBE_OIL_TEMPERATURE' 'HIGH_SPEED_GEARBOX_BEARING_TEMPERATURE_1'...
    'GENERATOR_NDE_BEARING_TEMPERATURE' 'TRANSFORMER_TEMPERATURE_1' 'GENERATOR_WINDING_TEMPERATURE_A'...
    }; % 'MAIN_BEARING_TEMPERATURE_1'



%% 3. Training stage

n_rec = 0; % external dynamics parameter
rec_start = 12; % parameter for steady operating mode
nhidden = 9; % hidden units in layer 2 (could be a vector if we want to identify several models)
its = 1000; % number of iterations for the scaled conjugate gradient algorithm
nlay = 30; % hidden units in layer 1 and 3 (could be a vector if we want to identify several models)

n_sel = length(head_sel);
ind_sel = zeros(n_sel,1);

% Initialize the nets
net = cell(length(nhidden),length(nlay)); % each cell will contain a net
TR = cell(length(nhidden),length(nlay)); % each cell will contain a training record
for i=1:length(nhidden)
    for j=1:length(nlay)
        net{i,j} = fitnet([nlay(j) nhidden(i) nlay(j)]); % initialize a 4-layer MLP
        net{i,j}.layers{2}.transferFcn = 'purelin';
        net{i,j}.trainFcn = 'trainscg'; % comment for trainlm
        net{i,j}.trainParam.epochs = its; % matlab default is 1000
        net{i,j}.inputs{1}.processFcns = {}; % suppress normalization of the network, we already proceeded
            to normalization
        net{i,j}.outputs{4}.processFcns = {}; % suppress normalization of the network, we already proceeded
            to normalization
    end
end


% build full Xtrain matrix
Xtrain = [];
for h=1:length(dat)
    % find indexes of selected features for machine h
    data = dat{h};
    for i=1:n_sel
        if ~isempty(find(strcmp(head_sel{i},data.header'),1))
            ind_sel(i) = find(strcmp(head_sel{i},data.header'),1);
        end
    end
    ind_sel = ind_sel(ind_sel>0);

    % Build Xtrain
    X = data.sig(:,ind_sel);
    % increase matrix dimensions / external dynamics
    A = [zeros(1,length(X(1,:))); X(1:end-1,:)];
    for j=1:n_rec
        X = [X A];
        A = [zeros(1,length(A(1,:))); A(1:end-1,:)];
    end

    % find valid abscissae
    abs_all = SelectDataNotNan(X,1+n_rec,size(X,1),data,rec_start,1000,-1000,ind_sel(end));

    abs_train = abs_all(((data.timevec(abs_all)<end_train).*(data.timevec(abs_all)>=begin_train))>0);
    mean_vec = mean(X(abs_train,:),1);
    max_vec =  std(X(abs_train,:));
    Xn = X-repmat(mean_vec,length(X(:,1)),1); % centering
    Xn = Xn./repmat(max_vec,length(X(:,1)),1); % normalize
    Xtrain = [Xtrain; Xn(abs_train,:)];

end
```

```
90
91  % Learn model bottleneck with Xtrain
92  X_mlp = {Xtrain(randperm(size(Xtrain,1)),:)};
93  nsample_mlp = 1;
94  errMLP = cell(nsample_mlp,n_sel);
95  errMLPcum = cell(nsample_mlp,1);
96  for k=1:nsample_mlp
97      errMLPcum{k} = zeros(length(nhidden),length(nlay));
98  end
99  Xpred = cell(nsample_mlp,length(nhidden),length(nlay)); % each cell will contain a prediction matrix
100 residuals = cell(nsample_mlp,length(nhidden),length(nlay)); % each cell will contain a residual matrix
101 for i=1:length(nhidden)
102     for j=1:length(nlay)
103         fprintf('Start␣training␣MLP␣for␣:␣M␣=␣%d,␣nlay␣=␣%d,␣its␣=␣%d␣\n',nhidden(i),nlay(j),its);
104         [net{i,j},TR{i,j}] = train(net{i,j}, Xtrain', Xtrain(:,1:n_sel)'); % train the MLP
105         for k=1:nsample_mlp
106             [Xpred{k,i,j}] = sim(net{i,j},X_mlp{k}')'; % computes the MLP's outputs on inputs Xtest{k} (net(
                    X) could be used too)
107             residuals{k,i,j} = X_mlp{k}(:,1:n_sel)-Xpred{k,i,j}; % compute residual matrix
108             for l=1:n_sel
109                 errMLP{k,l}(i,j) = norm(residuals{k,i,j}(:,l))/sqrt(length(X_mlp{k}(:,1))); % compute error
110             end
111             for l=1:n_sel
112                 errMLPcum{k}(i,j) = errMLPcum{k}(i,j) + errMLP{k,l}(i,j)/n_sel; % normalized error
113             end
114         end
115         fprintf('Finished␣training␣MLP␣for␣:␣M␣=␣%d,␣nlay␣=␣%d,␣its␣=␣%d␣\n',nhidden(i),nlay(j),its);
116     end
117 end
118
119 end
```

This function evaluates the predictive model at test points in between the specified dates.

Simul_Fun

```
1   function [Xpred,residuals,bottleneckval,abs_mlp,errMLP,errMLPcum] = Simul_Fun(net,head_test,begin_train,
        end_train,begin_test,end_test,head_sel,n_rec,rec_start,nhidden,nlay)
2   % SIMUL_FUN: use the net to monitor the data on head_test from begin_test
3   % to end_test.
4   % INPUTS: - net: net containing the models
5   %          - head_test: test machine name (string)
6   %          - begin_train: start of training set (double)
7   %          - end_train: end of training set (double)
8   %          - begin_test: begin of test set
9   %          - end_test: end of test set (double)
10  %          - head_tar: targeted feature
11  % OUTPUTS: - net: a cell array of the identifed nets
12
13  its = 1000;
14
15  load(head_test);
16  n_sel = length(head_sel);
17  ind_sel = zeros(n_sel,1);
18  % find indexes of selected features for test machine
19  for i=1:n_sel
20      if ~isempty(find(strcmp(head_sel{i},data.header'),1))
21          ind_sel(i) = find(strcmp(head_sel{i},data.header'),1);
22      end
23  end
24  ind_sel = ind_sel(ind_sel>0);
25
26  % Build Xest
27  X = data.sig(:,ind_sel);
28  % increase matrix dimensions / external dynamics
29  A = [zeros(1,length(X(1,:))); X(1:end-1,:)];
30  for j=1:n_rec
31      X = [X A];
32      A = [zeros(1,length(A(1,:))); A(1:end-1,:)];
33  end
34
35  % find valid abscissae
36  abs_all = SelectDataNotNan(X,1+n_rec,size(X,1),data,rec_start,1000,-1000,ind_sel(end));
37
38  % find training and test abscissae
39  abs_train = abs_all(((data.timevec(abs_all)<end_train).*(data.timevec(abs_all)>=begin_train))>0);
40  abs_test = abs_all(((data.timevec(abs_all)>=begin_test).*(data.timevec(abs_all)<=end_test))>0);
41  mean_vec = mean(X(abs_train,:),1);
42  max_vec =  std(X(abs_train,:));
43  Xn = X-repmat(mean_vec,length(X(:,1)),1); % centering
44  Xn = Xn./repmat(max_vec,length(X(:,1)),1); % normalize
45  Xtest = Xn(abs_test,:);
46  Xtrain = Xn(abs_train,:);
47
48  % Compute model outputs and bottleneck variables
49  X_mlp = {Xtest Xtrain Xn(abs_all,:)};
50  abs_mlp = {abs_test abs_train abs_all};
51  nsample_mlp = length(X_mlp);
52  errMLP = cell(nsample_mlp,n_sel);
53  errMLPcum = cell(nsample_mlp,1);
```

```
for k=1:nsample_mlp
    errMLPcum{k} = zeros(length(nhidden),length(nlay));
end
bottleneckval = cell(nsample_mlp,length(nhidden),length(nlay)); % each cell will contain a matrix of output of hidden layer
Xpred = cell(nsample_mlp,length(nhidden),length(nlay)); % each cell will contain a prediction matrix
residuals = cell(nsample_mlp,length(nhidden),length(nlay)); % each cell will contain a residual matrix
for i=1:length(nhidden)
    for j=1:length(nlay)
        fprintf('Starting computing MLP outputs for : data = %s , M = %d, nlay = %d, its = %d \n',head_test, nhidden(i),nlay(j),its);
        for k=1:nsample_mlp
            [Xpred{k,i,j}] = sim(net{i,j},X_mlp{k}')'; % computes the MLP's outputs on inputs Xtest{k} (net(X) could be used too)
            Xpred{k,i,j} = Xpred{k,i,j}.*repmat(max_vec,size(Xpred{k,i,j},1),1) + repmat(mean_vec,size(Xpred{k,i,j},1),1);
            residuals{k,i,j} = X(abs_mlp{k},:)-Xpred{k,i,j}; % compute residual matrix

            outlay1 = tansig(net{i,j}.IW{1}*X_mlp{k}'+repmat(net{i,j}.b{1},1,size(X_mlp{k},1)));
            bottleneckval{k,i,j} = (purelin(net{i,j}.LW{2,1}*outlay1+repmat(net{i,j}.b{2},1,size(X_mlp{k},1)))');

            for l=1:n_sel
                errMLP{k,l}(i,j) = norm(residuals{k,i,j}(:,l))/sqrt(length(X_mlp{k}(:,1))); % compute error
            end
            for l=1:n_sel
                errMLPcum{k}(i,j) = errMLPcum{k}(i,j) + errMLP{k,l}(i,j)/max_vec(l)/n_sel; % normalized error
            end
        end
        fprintf('Finished computing MLP outputs for : data = %s , M = %d, nlay = %d, its = %d \n',head_test, nhidden(i),nlay(j),its);
    end
end

end
```

This function give units of every signals.

Units_Blind

```
units = cell(n_sel,1);

for l=1:n_sel
   if any(strcmp(head_sel{l},{'ACTIVE_POWER' 'REACTIVE_POWER'}))
       units{l} = 'MW';
   elseif any(strcmp(head_sel{l},{'AMBIENT_TEMPERATURE_1' 'AMBIENT_TEMPERATURE_2' 'GENERATOR_WINDING_TEMPERATURE_A' 'GENERATOR_WINDING_TEMPERATURE_B'...
        'GENERATOR_COOLING_AIR_TEMPERATURE' 'GEARBOX_LUBE_OIL_TEMPERATURE' 'MAIN_BEARING_TEMPERATURE_1' 'GENERATOR_DE_BEARING_TEMPERATURE' 'GENERATOR_NDE_BEARING_TEMPERATURE'...
        'TRANSFORMER_TEMPERATURE_1' 'HIGH_SPEED_GEARBOX_BEARING_TEMPERATURE_1' 'HIGH_SPEED_GEARBOX_BEARING_TEMPERATURE_2' 'NACELLE_TEMPERATURE_1'}))
       units{l} = 'C';
   elseif any(strcmp(head_sel{l},{'PHASE_VOLTAGE_A' 'PHASE_VOLTAGE_B' 'PHASE_VOLTAGE_C'}))
       units{l} = 'V';
   elseif any(strcmp(head_sel{l},{'PHASE_CURRENT_A' 'PHASE_CURRENT_B' 'PHASE_CURRENT_C'}))
       units{l} = 'A';
   elseif any(strcmp(head_sel{l},{'WIND_DIRECTION' 'NACELLE_POSITION' 'BLADE_A_PITCH_POSITION' 'BLADE_A_SET_VALUE'...
    'BLADE_B_PITCH_POSITION' 'BLADE_B_SET_VALUE' 'BLADE_C_PITCH_POSITION' 'BLADE_C_SET_VALUE' 'NACELLE_DEVIATION'}))
       units{l} = 'degree';
   elseif any(strcmp(head_sel{l},{'WIND_SPEED_1'}))
       units{l} = 'm/s';
   elseif any(strcmp(head_sel{l},{'LUBE_OIL_PRESSURE_GEARBOX'}))
       units{l} = 'hPa';
   elseif any(strcmp(head_sel{l},{'GENERATOR_SPEED_1' 'GENERATOR_SPEED_2' 'HUB_SPEED'}))
       units{l} = 'rpm';
   elseif any(strcmp(head_sel{l},{'POWER_FACTOR'}))
       units{l} = '-';
   elseif any(strcmp(head_sel{l},{'GEARBOX_AXIAL_GLOBAL_VIBRATION'}))
       units{l} = '-';
   end
end
```

This function is used to filter the data. It should remove NAN values, remove points where the signal `feat_num` is out of the training bounds (`inf_bound` and `up_bound`) and points where the wind turbine has not been operating at high speed at any time for the last `rec_start` measurements.

SelectDataNotNan

```
function [ind] = SelectDataNotNan(X,ind_start,n,data,rec_start,temp_sup,temp_inf,ind_amb_temp,dir)
% SELECTDATANOTNAN: select n non NaN points in x starting from ind_start
% INPUT: - x: data
%        - ind_start: starting index
%        - n: number of points to select
%        - rec_start: number of previous measurements to reject at startup
```

```
%        - dir: selected points are right to ind_start unless dir = 1
% OUTPUT: - ind: index vector of the selected points
if nargin < 9; dir = 0; end
cur_pos = ind_start;
ind = zeros(n,1);
m = length(X(:,1));
if dir == 0
    i = 1;
    while i<=n && cur_pos<=m
        if ~isnan(sum(X(cur_pos,:))) && all(data.sig(max(cur_pos-rec_start,1):cur_pos,30) > 920) && all(data
            .sig(max(cur_pos,1):cur_pos,ind_amb_temp) >=temp_inf) && all(data.sig(max(cur_pos,1):cur_pos,
            ind_amb_temp) <=temp_sup)
            ind(i) = cur_pos;
            i = i+1;
        end
        cur_pos = cur_pos+1;
    end
else
    i = n;
    while i>=1 && cur_pos>0
        if ~isnan(sum(X(cur_pos,:)));
            ind(i) = cur_pos;
            i = i-1;
        end
        cur_pos = cur_pos-1;
    end
end
ind = ind(ind>0);
end
```

Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve **www.uclouvain.be/epl**