Adaptive Fault Detection and Diagnosis Using Parsimonious Gaussian Mixture Models Trained with Distributed Computing Techniques

Thiago A. Nakamura[a,*], Reinaldo M. Palhares[a], Walmir M. Caminhas[a], Benjamin R. Menezes[a], Mário Cesar M. M. de Campos[b], Ubirajara Fumega[c], André P. Lemos[a]

[a] *Federal University of Minas Gerais, Department of Electronics Engineering*
Av. Antônio Calos 6627 - Belo Horizonte - MG - 31270-901 - Brazil
[b] *CENPES, PETROBRAS*
Av. Horácio de Macedo, 950 Cidade Universitária, Ilha do Fundão - Rio de Janeiro - RJ - 21941-915 - Brazil
[c] *REGAP, PETROBRAS*
R. José Dias Diniz, 690 - Betim - MG - 32689-898 - Brazil

Abstract

After a great advance by the industry on processes automation, an important challenge still remains: the automation under abnormal situations. The first step towards solving this challenge is the Fault Detection and Diagnosis (FDD). This work proposes a batch-incremental adaptive methodology for fault detection and diagnosis based on mixture models trained on a distributed computing environment. The models used are from a family of Parsimonious Gaussian Mixture Models (PGMM), in which the reduced number of parameters of the model brings important advantages when there are few data available, an expected scenario of faulty conditions. On the other side, a large number of different models rises another challenge, the best model selection for a given behaviour. For that, it is proposed to train a large number of models, using distributed computing techniques, for only then selecting the best model. The work proposes the usage of the *Spark* framework, ideal for iterative computations. The proposed

*Corresponding author
Email addresses: akionakamura@ufmg.br (Thiago A. Nakamura), rpalhares@ufmg.br (Reinaldo M. Palhares), caminhas@ufmg.br (Walmir M. Caminhas), brm@cpdee.ufmg.br (Benjamin R. Menezes), mariocampos@petrobras.com.br (Mário Cesar M. M. de Campos), fumega@petrobras.com.br (Ubirajara Fumega), andrepaim@ufmg.br (André P. Lemos)

methodology was validated in a simulated process, the Tennessee Eastman Process (TEP), showing good results for both the detection and the diagnosis of faults. Furthermore, numeric experiments show the viability of training a large number of models for the best model selection *a posteriori*.

Keywords: FDD, mixture models, parsimonious Gaussian, distributed computing

1. Introduction

In the last decades, the industrial processes control and automation has had great advances with the usage of computers for controlling complex plants, mainly using the so called regulatory control. Even with all that advancement, however, an important task on the industrial plant management still remains in great part manual, done by human operators. That task involves all necessary actions against a process fault, since the abnormal event detection until its diagnosis, locating the origin of the fault and taking the proper actions in order to return the process to a normal and safe state. All that activity is generally called Abnormal Event Management (AEM) [1].

The human ability to deal with this task has been getting ever more limited, due to several reasons. Firstly, the scope of possible problems in a plant is much too broad, and the amount of data to be monitored is way too high, possibly thousands of features every few seconds. Furthermore, the speed to reach the diagnosis is, oftentimes, essential for both reducing the productivity loss and increasing safety.

Thus, this is the next big challenge on control engineering. As the regulatory control, automatised by computer, has spread around all industry and brought a huge progress regarding product quality and consistency, besides safety and efficiency, the automation of AEM will be of great importance. The automation of the Fault Detection and Diagnosis (FDD) is the first step of the AEM, and it is the main focus of this work.

First of all, it is important to define what is the task and the scope of a

FDD system. A fault in a process it is not necessarily a result of a catastrophic problem in an equipment, it is not even necessary to involve a physical component. A fault can be, for example, a non-optimal operation of the plant or a product off its specification. The root of a problem can be diverse, e.g. sensor calibration, poorly tuned controllers, bad quality raw material, pressure lost in pressurised systems, or even human error. All faults generate symptoms, which are events and/or value variations that allows for a fault to be detected and isolated.

To detect a fault is simply to recognise a problem has occurred, even if the root cause is unknown. The detection is done looking for the symptoms, which can be in a qualitative or quantitative form. To diagnose a fault is to specifically point out what was the problem and possibly indicating possible solutions.

Thereby, the FDD task can be directly associated with human diseases. A disease (or a fault) can rarely be detected by the doctor (the FDD system) without a patience with symptoms. Furthermore, a given set of symptoms can be diagnosed if a disease which causes the observed symptoms is known.

A good FDD system, fast and efficient, is essential of minimize the productivity loss facing an abnormal event and guarantee the safe operation of the plant. To observe the symptoms of a fault, it is necessary a model of the plant, so one can see its behaviour change. Modelling methods for FDD can be generally classified as quantitive-based models, qualitative-based models e historical data based models [1, 2, 3, 4].

Over the years, historical data based models gained a lot of attention. One of the reasons is that those methods do not demand a knowledge of the physics of the process, which is oftentimes unfeasible for complex systems. They rely only on data collected during the plant's operation. Some examples of historical data based methods are neuro-fuzzy [5, 6], artificial neural networks [7], imune systems [8, 9], statistical methods [10, 11, 12, 13] and expert systems [14].

The Multivariate Statistical Process Monitoring (MSPM) for FDD has received considerable attention lately, both the research community and the industry [15]. The base of the MSPM is to find a statistical model for the normal

operation condition, followed by a monitoring statistics and its confidence margin. Big industrial processes can have hundreds of monitored variables in real time, which require a complex model on the FDD system. Most of the times, several of these variables are highly correlated, which lead to a redundancy of information and the possibility of simplifying the models. With the normal condition operation model in hands, the detection is made monitoring the plant continuously, looking for a behaviour that deviates from that model, indicating an abnormal state. The diagnosis is possible if, somehow, the abnormal event is also modelled, making the reoccurrence of a known fault faster to be dealt with.

There are several modelling methods which, by means of some restrictions, simplify the model without significant information loss, making them easier to train and more robust. Simpler models require less training data to reach a good parameter estimation, this is particularly important in the FDD scenario because it is expected to have very scare data of faulty conditions. For example, for decades, the Principal Component Analysis (PCA) for MSPM was subject of intense study and research [16, 17]. Over time, its linear limitation and non-statistical formulation lead to the development of the Probabilistic Principal Component Analysis (PPCA) [10, 18, 19]. The PPCA is part of a set of parsimonious Gaussian models directly related to the factor analysis model [20]. Having a probabilistic model, it is possible to define a Mixture of Probabilistic PCA (MPPCA), which allows for a better modelling on a non-linear feature space that many processes have [21]. Those Gaussian-like mixture models can have several others restrictions on their covariance matrix that can be good alternatives [22].

As important as a good modelling technique is how this model is trained with the available data, whether it is from a normal or a faulty condition, since the parameters initialisation [23, 24, 25], which can lead to a quite different final result, until a fast and efficient training algorithm [26, 27]. Moreover, many statistical models are defined as mixtures of a given distribution, and the choice of the ideal number of components is still an important challenge for the

research community [28, 29, 30].

Creating such complex models, with hundreds of variables, thousands of samples and iterative training, and select the ideal model within several possible structures and restrictions, can be computationally expensive and time consuming. Therefore, it is sought to use parallel and distributed programming techniques, already vastly used in the analysis of the so called *big data* in the computer science community, to deal with this problem [31, 32, 33, 34].

This work is divided in five sections. Section 1 introduces the current scenario for the proposal of this work, as well as presents the motivation and objectives of its execution. A short literature review of the three main themes addressed in this work is present in Section 2. Section 3 shows in more detail the statistical model used, explains the batch-incremental approach used for the FDD task and how the models are trained using a distributed computing system. Validation results are shown and discussed in Section 4. Finally, Section 5 concludes the work and analyse future possibilities of improvements.

2. Literature Review

The proposed work, for better understanding, can be divided into three main themes: the fault detection and diagnosis itself, the statistical modelling method to be used and the implementation in a parallel and distributed programming environment. This section presents a short review of these three themes, addressing concepts, strategies and methodologies from the current literature.

2.1. Fault Detection and Diagnosis

A fault is not necessarily characterised by a catastrophic problem in any part of the plant, nor even is necessary to involve a physical equipment. Every component of the plant and the control system is a potential source of a fault, and the FDD system should be able to monitor and detect the problem independently of where it occurred, as illustrated by Figure 1, even if it is not capable of indicating the location or the cause of the fault.

In general, faults can be classified in three categories [1]:

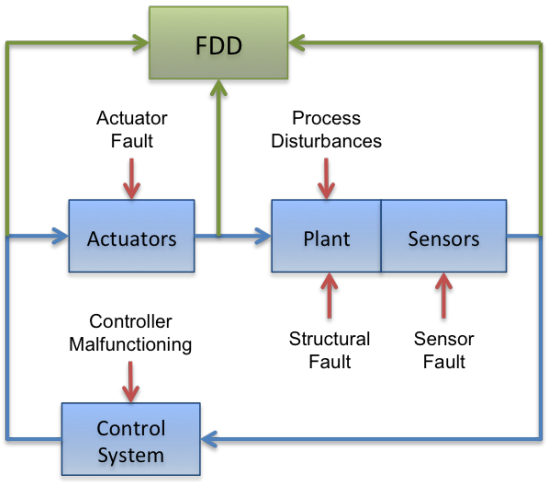


Figure 1: Monitoring scheme of a FDD system.

1. **Process' gross parameter change**: Not all the details of a plant are considered when building a model, and many external factors that were neglected can have an influence in its behaviour. These factors are commonly called exogenous variables. Some examples of this type of disturbances can be a change of a reagent concentration, or heat exchange ratio of a heat exchanger.
2. **Structural change**: A structural change is characterised by a fault in an equipment, resulting in a change in the information flow between the variables. Examples of this class of faults are problems in the controller or a leakage in a pipe.
3. **Sensors and actuators malfunction**: Although a sensor or actuator malfunction does not represent a process failure *per se*, it directly affects its behaviour by compromising the entire control system. A valve gripe is an example of this fault.

Independently of the fault class, it usually is characterised by a change in the distribution of the involved variables. When dealing with a dynamic environment, like most of the real life problems, the learning algorithms must be able to detect or adapt to those changes, usually called concept drifts [35]. In

the FDD context, those changes can represent two scenarios: a change, usually gradual, of the normal operation condition, in which the model should adapt; or the occurrence of a fault, where it should be detected and diagnosed (or learnt, if it is new).

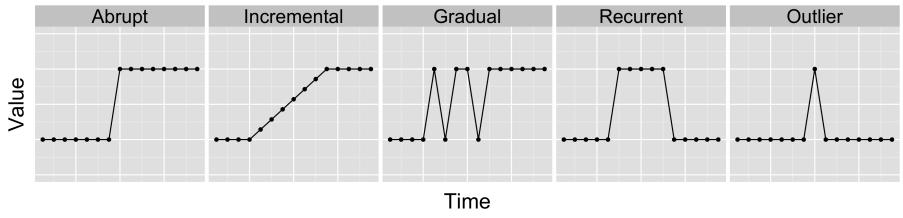


Figure 2: Concept drift types.

The Figure 2 illustrates the different ways a concept drift can occur and the diference between a concept drift and an outlier. For example, equipment breakage will have an *abrupt* change in its behaviour. A natural wear has an *incremental* change, and here it should be decided whether to gradually adapt the model or in fact detect the fault. A fault can also be *gradual*, possibly caused by bad contact, for example. A valve gripe that does not happen all the time and loosens itself after it occurred can be characterised as *recurrent*. In the last case, the *outlier* happens due to some noise or temporary disturbance, and does not represent a concept drift, and this differentiation is one of the challenges of the field.

2.1.1. Desired Characteristic of a FDD System

A good FDD system should have some important characteristics, as explained as follows. Hardly one system will comply with all those characteristics in the best possible way, since many of them present a trade-off between each other. So this set of characteristics are important to enable comparison between different methodologies, presenting a concrete choice of a system given a certain scenario [1].

Fast Detection and Diagnosis. A faulty condition on a process should, ideally, be detected and diagnosed as fast as possible. However, a system which responds quickly to changes is more sensible to high frequency influencies, like noise and outliers, which can lead to a high rate of false alarms.

Isolability. Isolability is the FDD system's ability to distinguish between different failures. Although this characteristic is directly linked with the monitored system, it is sensible to the ability to reject uncertainties of the model. In other words, the ability to identify different faults trades off with the ability of generalize the uncertain behaviour of a given fault.

Robustness. The robustness of a system is characterised by its ability to deal with noise and uncertainties and still deliver a good perfomance. The way the system degrades is also important, the system is more robust if its performance degrades slowly, instead of abruptly.

Novelty Identification. In general, there are a lot of data and information about the normal operation condition of a plant, but the same is not true for abnormal conditions. It is probable that a given problem has never even been seen in practice. A FDD system should be, minimally, able to differentiate a normal from an abnormal condition and, detected the abnormality, differentiate between a known fault or a new one. The lack of data and information about faulty behaviour is an important limiting factor for the novelty identification.

Classification Error Estimate. In practice, it can be important for the user to have a confidence value for the system's diagnosis. That way, the operator can take a safer and better decision, choosing to follow or not the system's recommendation.

Adaptability. It is expected that the behaviour of a process to change with time, be it due to natural wear or external factor, like the season. The FDD system should be able to adapt to these changes as those new information is received.

Explanation. Besides identifying a problem, it would be good if the system could indicate the location of the fault and explain how it began and spread through the plant. It is desirable that the system could justify why it chose a given hypothesis for the origin of the problem and rejected the others. This is specially important when the system has a central role in real time decision making, recommending possible actions by the plant operator, which will evaluate the situation with his expertise and experience.

Modelling Requirement. The necessary effort to build the model to be used by the FDD should be as small as possible. Here, it is taken into consideration the necessary knowledge about the plant, people involved, amount of data, etc.

Computation and Data Storage Requirement. A very complex system can require great computation power and data storage. Although this cost is dwindling, it is still an important factor and should be taken into consideration.

Multiple Fault Idetification. This is a hard task, differentiate two different faults that happen at the same time might be impossible due to the way the variables are naturally linked in the process. Even if it was possible to isolate the behaviours, this combinatory analysis of the possibilities can be prohibitive for big processes.

2.1.2. FDD Methods and Models

A FDD system can be developed under the perspective of several methods, usually separated into: quantitative methods, qualitative methods and historical-data based methods. There are still hybrid methods that aims to mix the advantages of each one. But independently of the method, in general, the diagnosis process can be understood as a series of transformations ou mapping of the process' measurements, based on the *a priori* knowledge and search techniques, as shown in Figure 3 [1].

The Measurement Space does not require any *a priori* knowledge and it is the input of the diagnose system, it is simply the measurements of the plant. The

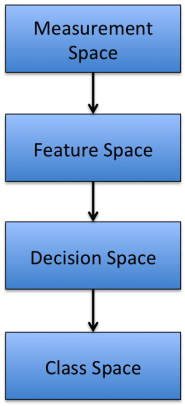


Figure 3: Transformation stack for diagnosis [1].

Feature Space comes the applying some function over the Measurement Space based on a *a priori* knowledge, the measurements are analysed and combined to generate some useful information about the process' behaviour. The Decision Space is usually achieved via an objective function, like error minimisation or simply a limiar function. Finally, the Class Space is the set of possible faults where the given problem is classified and diagnosed.

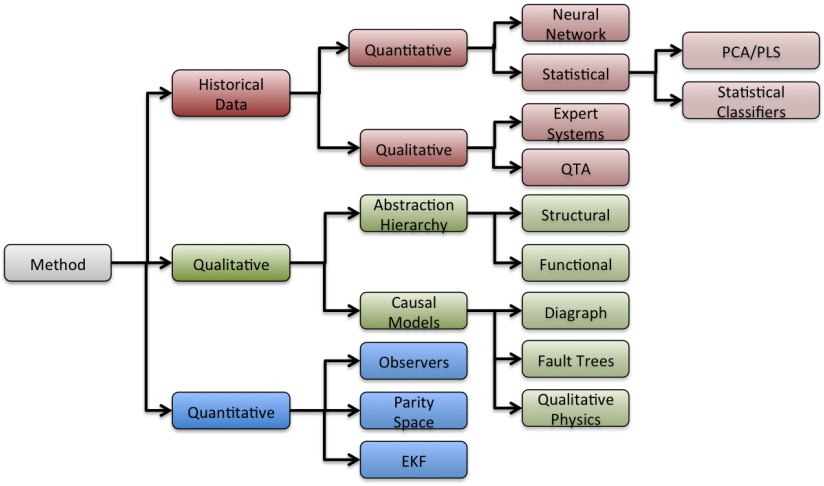


Figure 4: Modelling techniques tree for FDD systems [1].

Figure 4 shows some examples of methods which can be used in the FDD

system. The main focus of this work are methods based on historical data, and some of them will be briefly discussed next. In particular, it was used the statistical classifier, where the base model is described in Section 2.2.7. A general study of other methods can be seen in [1, 2, 3, 4].

2.2. Modelling

Modelling methods for FDD can be, in general, classified in quantitative based models, qualitative based models and historical data based models, as shown in Figure 4. The first two techniques usually require a considerable *a priori* knowledge of the plant, which can, many times, make its usage more difficult.

On the other hand, historical data based models rely only on a big amount of data of the working plant, not needing a profound knowledge of the physics of the process. Thereby, and also because of the fast development of machine learning algorithms and computing power and storage, those methods purely based in data are gaining a lot of space in the industry. In this section, a brief review of some of these methods is presented, finalising with a more detailed approach of the statistical methods, which are used in this work.

In the branch of historical data based models, there are different approaches on how each method transforms the data into information for the diagnosis system. That is also known as characteristic extraction, and can be classified as qualitative or quantitative [3], as also shown in Figure 4.

Two of the mains methods of qualitative characteristic extraction are expert systems and trend analysis. About the quantitative methods, they can still be sub categorised as statistical and non-statistical. Among those, the PCA and the statistical classifiers are most well known, whilst a the artificial neural network is much used non-statistical method.

2.2.1. Expert Systems

Expert systems are based on rules relating characteristics or behaviours of the process, many times being simply *if-then-else* or fuzzy rules. An expert sys-

tem is usually specifically designed for a given process and solves problems for a limited domain. The main advantages of an expert system is the ease in developing, the interpretability of the diagnostic, ability to work under uncertainties, to provide an explanation about the diagnostic and to suggest possible solutions for a fault. The big disadvantage is that the rules are always very specific for the process to which it was designed, besides having a limited representation power.

2.2.2. Trend Analysis

Another important qualitative form of characteristic extraction is the abstraction of the information in the trend of the signals, very used in process monitoring. The QTA aims to model the trend of the collected signals, trying to identify and explain important events that occur in the process, diagnosing the problem or even trying to predict future states.

The qualitative trend representation of the plant allows better comprehension of its behaviour. In the majority of the cases, process malfunctioning present a different trend in the sensor signals. Those trends can be used to identify possible abnormalities, allowing fault detection and classification [3, 36].

2.2.3. Artificial Neural Network

The quantitative characteristic extraction approach for FDD formulates the diagnosis task as a pattern recognition task. In other words, it aims to classify the samples into classes, in general, pre determined.

Statistical and non-statistical methods can be found inside the quantitative approach set. The Artificial Neural Network is the non-statistical method most commonly used, where several architectures, activation functions and learning strategies can be found. The networks with supervised learning are usually trained with the Back Propagation algorithm, and unsupervised learning networks follow the ideia of the Kohonen networks, also known as Self Organising Maps (SOM) [3, 7].

2.2.4. Statistical Models

Finally, the historical-data based models with quantitative characteristics extraction methods can have a statistical approach. The most basic technique is to estimate the parameters of a probability distribution function to model the normal operation condition of the plant and, from that model, monitor its behaviour. New samples can be classified as faulty in case they do not belongs to the normal model, within a given threshold and confidence level [3, 10, 11, 12, 13].

Multivariate statistical techniques, like the Principal Component Analysis (PCA), allows compression and dimensionality reduction without substantial information loss. Furthermore, statistical classifiers, like the Gaussian mixtures, can be directly used as a diagnosis system. Next, a little more detailed explanation about the PCA is given, as well as its relation with the Probabilistic Principal Component Analysis (PPCA) and, in its turn, how the PPCA can be generalised into a broader family of parsimonious Gaussian models.

2.2.5. Principal Component Analysis

The classic PCA is a technique highly spread and used for dimensionality reduction. From a practical point of view, it has two basic definitions that leads to the same formulation [37]. For a given observed data vector of dimension d, $t_{(d)}$[1], the definitions are:

- Projection of the data into orthogonal axis of a sub-space which retains the maximal variance.
  - The projection of the space $\mathbb{R}^d$ into the space $\mathbb{R}^q$, $q < d$, where $\mathbf{x_{(q)}} = W^T_{(d\times q)}(\mathbf{t}_{(d)} - \overline{\mathbf{t}}_{(d)})$ retains the maximal variance.
- Projection of the data into orthogonal axis of a sub-space which optimizes the linear reconstruction of the data.

[1] just during this demonstration, the subscript indicates the vectors and matrixes dimensions for ease of understanding

– Reconstruction of $\hat{\mathbf{t}}_{(d)}$ from the scores $\mathbf{x}_{(q)}$, where $\hat{\mathbf{t}}_{(d)} = W_{(d\times q)}\mathbf{x}_{(q)} + \overline{\mathbf{t}}_{(d)}$ has the smallest reconstruction quadratic error $\sum_{i=0}^{N} \|\mathbf{t}_{i(d)} - \hat{\mathbf{t}}_{i(d)}\|^2$, where N is the number of training samples.

Both definition lead to the same **principal sub-space** defined by $W_{(d\times q)}$, where $W_{(d\times q)}$ is formed by the q eigenvectors associated with the q highest eigenvalues of the sample covariance matrix $S_{(d\times d)} = \sum_{i=1}^{N} (\mathbf{t}_{(d)} - \overline{\mathbf{t}}_{(d)})(\mathbf{t}_{(d)} - \overline{\mathbf{t}}_{(d)})^T/N$. The value q is called number of **principal components** and is set accordingly to the necessary specification of variance retention. Usually the retained variance is set to be 95%, determined by the eigenvalues of $S_{(d\times d)}$ [38].

With the PCA model computed, the real-time monitoring for detecting dissonant behaviours is done with two statistics:

- The Hotelling's T^2, using the q vector of loading selected during modelling, indicating how the data befits the model [39].
- The statistic Q measures random variations in the data, based on the quadratic prediction error, like measurement noise [16].

Both statistics, T^2 and Q, work with confidence levels and thresholds, and detect failures of different natures, but should be used together. There is not a standard method for a combined analysis of them and the linear limitation of the data projections hardens the modelling of non-linear problems, as most of the industrial processes are.

2.2.6. Probabilistic Principal Component Analysis

Even though the use of PCA is highly spread, the usage of a probabilistic approach of the problem brings important advantages [18]:

- Allows easy comparison with different models and other techniques based on probabilistic models;
- The value of the probability density function allows to measure the degree of novelty of a given data regarding the model;

- One single statistic to be monitored;
- Allows to work with missing data (the basis for the variable contribution analysis of a fault);
- It can be easily extended to a mixture of local models.

The Probabilistic Principal Component Analysis (PPCA) is a particular case of the Factor Analysis model, which in turn is a particular case of the Latent Variable model [18]. The Latent Variable model aims to relate observable data $\mathbf{t}_{(d)} \in \mathbb{R}^d$ to a vector of latent variables $\mathbf{x}_{(q)} \in \mathbb{R}^q$, where usually $q < d$, such that:

$$\mathbf{t}_{(d)} = f(\mathbf{x}_{(q)} \mid \mathbf{w}) + \epsilon \tag{1}$$

where $f(\cdot \mid \cdot)$ is any function of the latent variables, $\mathbf{x}_{(q)}$, given the parameters $\mathbf{w}$ and an independent noise ϵ. Defining a probability density function for $\mathbf{x}_{(q)}$ and ϵ it is possible to, from the collected data, estimate the model parameters, ϵ, via maximum likelihood.

The Factor Analysis model is a very common Latent Variable model, where the function $f(\mathbf{x}_{(q)} \mid \mathbf{w})$ is assumed to be a linear function, of the form:

$$\mathbf{t}_{(d)} = W_{(d\times q)}\mathbf{x}_{(q)} + \mu + \epsilon \tag{2}$$

One can already notice the similarity with the classic PCA model, being it a linear transformation of the data. As in the PCA model, it is also assumed that the latent variables are uncorrelated and with unity variance $\mathbf{x}_{(q)} \sim \mathcal{N}(\mathbf{0}, I_{(q)})$. The noise is assumed as having zero-mean and being uncorrelated, $\epsilon \sim \mathcal{N}(\mathbf{0}, \Psi_{(d)})$, where $\Psi_{(d)}$ is diagonal.

Finally, the Probabilistic PCA is the previous Factor Analysis model with the restriction that the noise is isotropic, $\epsilon \sim \mathcal{N}(\mathbf{0}, \sigma^2 I_{(d)})$. Hence, given the latent variables $\mathbf{x}_{(q)}$ the conditional distribution of $\mathbf{t}_{(d)}$ is:

$$p(\mathbf{t}_{(d)} \mid \mathbf{x}_{(q)}) = (2\pi\sigma^2)^{-d/2} e^{-\frac{1}{2\sigma^2}\|\mathbf{t}_{(d)} - W_{(d\times q)}\mathbf{x}_{(q)} - \mu_{(\mathbf{d})}\|} \tag{3}$$

Since it was assumed that $\mathbf{x}_{(q)} \sim \mathcal{N}(\mathbf{0}, I_q)$, it is possible to obtain that the marginal probability distribution of $\mathbf{t}_{(d)}$ is a multivariate normal distribution given by $\mathbf{t}_{(d)} \sim \mathcal{N}(\mu, C)$, where $C = \sigma^2 I_{(d)} + W_{(d\times q)} W^T_{(d\times q)}$, such that:

$$p(\mathbf{t}_{(d)}) = (2\pi)^{-\frac{d}{2}} |C|^{-\frac{1}{2}} e^{-\frac{1}{2}(\mathbf{t}_{(d)}-\mu)^T C^{-1}(\mathbf{t}_{(d)}-\mu)} \tag{4}$$

Comparing with the multivariate normal distribution where the sample covariance matrix, $S_{(d\times d)}$ should be estimated, the covariance matrix of the PPCA model, C, depends only on σ^2 and $W_{(d\times q)}$. Therefore, the PPCA can be understood as a way to reduce the complexity of the model by selecting only the principal components, $q < d$. In another words, the number of free parameters in C, $[dq + 1 - q(q-1)/2]$, of order $O(dq)$, is smaller than the $[d(d+1)/2]$, which is $O(d^2)$, parameters of $S_{(d\times d)}$, if $q < d$.

Finally, with N observed samples of $\mathbf{t}_{(d)}$, it is possible to estimate the parameters $\hat{W}_{(d\times q)}$, $\hat{\mu}$ and $\hat{\sigma}^2$ that define the distribution via statistical methods, e.g., the Maximum Likelihood Estimation (MLE).

Mixtures of PPCA. With a probabilistic model at hand, it is possible to define a mixture of those models. The PPCA mixture model with k components is defined as [19]:

$$p(\mathbf{t}) = \sum_{i=1}^{k} \pi_i p(\mathbf{t}|i) \ , \tag{5}$$

where $p(\mathbf{t}|i) \sim \mathcal{N}(\mu_i, C_i)$ is the distribution of the ith component, in which $C_i = W_i W_i^T + \sigma_i^2 I$, and the weights sum to unity, $\sum_i \pi_i = 1$. Thereby, each of the k components can be understood as a local model, that combined represent the entire set of data. Even though there is no closed-form formula to estimate the parameters of the mixture, the MLE can be achieved iteratively with the Expectation Maximisation (EM) algorithm.

2.2.7. Parsimonious Gaussian Mixture Models

Taking as a base the previously discussed Factor Analysis model, described by Equation 2, and again assuming independent factors and unit variance, $\mathbf{x} \sim$

$\mathcal{N}(\mathbf{0}, I)$, and a noise with zero mean and uncorrelated, $\epsilon \sim \mathcal{N}(\mathbf{0}, \Psi)$, where Ψ is diagonal, the model has a marginal distribution, similar to the PPCA, given by:

$$p(\mathbf{t}) = \mathcal{N}(\mu, WW^T + \Psi) \tag{6}$$

Thereby, one can see that the Factor Analysis model is nothing more than a multivariate Gaussian with restrictions on the covariance matrix. Hence, it can be assumed that mixture of Factor Analysis model has the same formulation as the one described in Equation 5.

With the Factor Analysis mixture model, a family of distributions called Parsimonious Gaussian Mixture Models (PGMM) was developed [20]. Each member of this family applies a set of restrictions on the possible variations of the parameters among its mixture components. In other words, given a mixture with k components, three restrictions can be applied:

1. All the mixture components have the same *loading* matrix, $W_i = W$, $\forall i = 1, 2, ..., k$;
2. All the mixture components have the same noise variance, $\Psi_i = \Psi$, $\forall i = 1, 2, ..., k$;
3. The noise of each component is assumed to be isotropic, $\Psi_i = \sigma_i^2 I$.

Therefore, having three independent restrictions, the PGMM family has eight members, covering all the possible restriction combinations. As shown on subsection 2.2.6, the PPCA is derived from the Factor Analysis model assuming a isotropic error, $\Psi = \sigma^2 I$, which is exactly the third restriction of the PGMM described above. So, the seventh row in the Table 1, the member `UUC` is equivalent to the MPPCA. This relation is illustrated in Figure 5, as well as its relation with the classical PCA.

As discussed on subsection 2.2.6, the PPCA have less parameters to be estimated than a regular Gaussian. The restriction of the PGMM can reduce even more the number of model parameters, as summarised in Table 1. The least restrictive models (`UXX`) have a number of parameters of order $O(kpd)$,

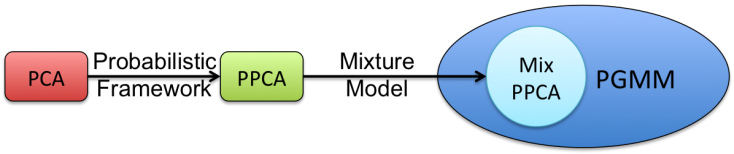


Figure 5: From the PCA to the PGMM family of distributions.

which is already smaller than a regular Gaussian mixture model, with $O(kd^2)$. The most restrictive models (`CXX`), have a number of parameters of order $O(pd)$. This complexity reduction of the models can be crucial on training with scarce data. In the context of FDD, this is important when modelling faulty conditions, where it is expected to have few available data.

Table 1: Covariance structures of the members of the PGMM family.

ID	$W_i = W$	$\Psi_i = \Psi$	$\Psi_i = \sigma_i^2 I$	Covariance Parameters
`CCC`	Constrained	Constrained	Constrained	$[pq - q(q-1)/2] + 1$
`CCU`	Constrained	Constrained	Unconstrained	$[pq - q(q-1)/2] + p$
`CUC`	Constrained	Unconstrained	Constrained	$[pq - q(q-1)/2] + k$
`CUU`	Constrained	Unconstrained	Unconstrained	$[pq - q(q-1)/2] + kp$
`UCC`	Unconstrained	Constrained	Constrained	$k[pq - q(q-1)/2] + 1$
`UCU`	Unconstrained	Constrained	Unconstrained	$k[pq - q(q-1)/2] + p$
`UUC`	Unconstrained	Unconstrained	Constrained	$k[pq - q(q-1)/2] + k$
`UUU`	Unconstrained	Unconstrained	Unconstrained	$k[pq - q(q-1)/2] + kp$

2.2.8. Dynamic Principal Component Analysis

Directly applying the PCA technique on a matrix with the process data is equivalent to create its static model. However, most of the processes and industrial equipment are dynamic by nature, having a more complex relation between the variables than a simple static model is capable of representing. To workaround this limitation, the Dynamic PCA (DPCA) was proposed, where the model is not only obtained with each instant sample, but also with the previous values [40, 41, 42].

To capture all the first order dynamic relation, one can simply consider a unity delay of the sample. In general, the new data matrix used for training the

model will have the following shape:

$$\begin{bmatrix} \mathbf{t}_{(d)}^T(j) & \mathbf{t}_{(d)}^T(j-1) & \cdots & \mathbf{t}_{(d)}^T(j-l) \\ \mathbf{t}_{(d)}^T(j+1) & \mathbf{t}_{(d)}^T(j) & \cdots & \mathbf{t}_{(d)}^T(j+1-l) \\ \vdots & \vdots & \vdots & \vdots \\ \mathbf{t}_{(d)}^T(j+N-1) & \mathbf{t}_{(d)}^T(j+N) & \cdots & \mathbf{t}_{(d)}^T(j+N-1-l) \end{bmatrix} \tag{7}$$

where $\mathbf{t}_{(d)}(j)$ is the vector of dimension d of the process data on time j, N is the number of available samples and l is the number of delayed samples considered to construct the new matrix to build the DPCA model.

Note that the number of variables increases linearly with l and, in general, $l = 1$ or $l = 2$ is enough to model the dynamics of first and second order of the process [43]. A deeper analysis of the variables correlation can be done, in order to find better values of delayed data to use, for example using the correlation of the residues of an ARMA model. Possible sazonal or cyclic dynamics can be an interesting point of analysis. This idea can also be extended to model the process using the PGMM, such that the mixture model will model, together with the static part, the dynamics of the plant.

2.3. Parallel and Distributed Computing

In this section, some parallel and distributed computing concepts are revised, as well as some basic architectures. In the end, the *Hadoop* [44, 45] and the *Spark* [33, 34], two of the currently most used distributed computing frameworks are presented.

2.3.1. Basic Concepts

Parallel computing is the usage of parallel processing to reduce the time taken to execute a single computational problem. The evolution of the parallel computing has gone through several stages, starting with the period which ended in the mid-80's, on the bit-level parallelism, with the evolution of the 4-bit microprocessor to 8-bits, 16-bits until the 32-bits. The increasing number of bits reduces the number of cycles needed to complete 32-bits operations. Today,

there are architectures with 64-bits and even 128-bits, but the evolution in this point forward aims mainly in better representation for floating point and larger address space [46].

From the mid-80's until the mid-90's, instruction parallelism has evolved in the microprocessors, both with the usage of the *pipelines* and the *superscalars* [46]. Here began the real parallel computation, with different instructions being executed at the same time.

Even though a power limit has been reached frequency-wise, following Moore's Law, the processor density has increased in the sense of having multiple processing cores in the same chip, multiplying the processing power but keeping the low consumption, due to the low frequency. Moreover, computer cluster techniques, for the execution of more intense tasks, were developed, representing the beginning of the last stage of the parallel computing evolution. Now, entirely different computations are executed at the same time over the same or different set of data [46].

In general, the parallelism can the achieved in two ways. Firstly, by a centralised multiprocessor, or SMP, which is a highly integrated system where all CPUs share the same global memory. This memory supports communication and synchronisation between processors. Secondly, by multi-computer systems, or computer clusters, which are multiple computers connected through a network, where they interact via messages [46, 47]. This work, and this brief revision, focus in the last one.

The Flynn Taxonomy, represented in the Figure 6, is the most used classification scheme for parallel computing in respect to the way the parallelism is presented in the instruction and data flow. A hardware can support a single or multiple instruction flows handling a single or multiple data flows [48].

The SISD class refers to computers with a single instruction flow and single data flow, commonly seen in uni-processed machines. A processor vector with a single pipeline of control unit able to compute the same operation in several different data sets are classified as SIMD, since they have a single instruction flow but multiple data flow. Machines of the MISD type are rare, and usually

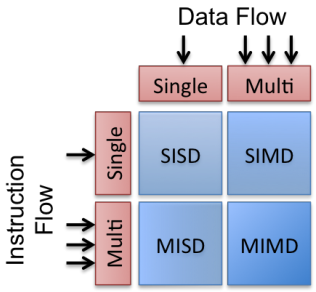


Figure 6: Flynn taxonomy.

have an architecture with multiple function units that compute different operations over the same data at the same time. Finally, the computers categorised as MIMD have multiple instruction and data flow. The majority of the computer architectures nowadays fit into this class, since multicore computers and computer clusters execute, at the same time, different instruction flows over different data flows [47, 48].

The Flynn categories indicates how the hardware handles the parallelism. But it is also important to understand how a task, i.e., part of a code or software, can be parallelised. There are basically two forms of parallelism: the data parallelism and the functional parallelism. The data parallelism happens when independent tasks apply the same operation in different elements of the data set, like in the following pseudo code:

```
int i;
for(i = 0; i < 100; i++)
        a[i] = b[i] + c[i];
```

The sum of the hundred elements of `b` and `c` is done iteratively and allocated to the vetor `a`. Accordingly with the data parallelism, all those operations could be executed at the same time.

The functional parallelism can be achieved with tasks that operates over different data sets or where the result of one does not influence to other, as exemplified in the pseudo-code:

```
a = foo(x);
b = bar(x);
c = a*b;
d = baz(y);
```

In this example, the functions `foo()`, `bar()` and `baz()` can be executed in parallel, since there is no dependency among them. Oftentimes, when writing a code optimised for parallelism, it is necessary to find along the algorithm where the possible places for data and functional parallelism are. The elaboration of a Data Dependency Graph can be helpful in that task [47]. Multiple computers can be connected in different ways through a computer network. This network is used for messages exchange among different processing units, in order to keep the necessary synchronicity during the parallel task.

2.3.2. Beowulf Architecture

In 1994, Thomas Sterling and Don Becker created the first computer cluster made entirely with off the shelf components and open source software. They showed that it was possible to build computer clusters in a way that was a lot cheaper than the supercomputers at the time, which were specifically designed for complex systems and parallel tasks, and still achieve a good performance.

The Beowulf paradigm rapidly dominated the scientific community and became the standard for parallel and distributed computing. However, it is important to notice that even though the cluster is built with off the shelf machines, openly found in the market, a computer cluster differs from working stations connected through a network.

A network of work stations is a collection of disperse computers, typically located at the user's desk. They are usually connected through a Ethernet network and, although they can be used for distributed computing, their main role is to serve the needs of its local user. In general, each machine can have its own operating system and user softwares.

In a computer cluster, the machines are usually located in a single location, usually accessible via network only (without monitors or keyboards, for exam-

ple). Oftentimes those machines are not even accessible for log-in, being used only as slaves in a distributed task. The entire cluster is managed as an entity, not separate machines (although individual faults are possible and expected). One last important difference is that computer cluster have switched network, which is faster and with lower latency, as opposed to the shared network of a conventional computer network [47].

2.3.3. Cluster Based Computing

Nowadays the parallel and distributed computing is well spread and consolidated. Maybe the first big platform for distributed computing development was the Apache Hadoop and the programming model of the MapReduce, based on the system internally developed at Google [44]. Recently, other more modern approaches like the Apache Spark has been gaining a lot of space. In the following sections, both platforms are presented.

Hadoop and the MapReduce. The Apache Hadoop is an open source software platform for distributed storage and processing of large data sets in computer clusters built from off the shelf hardware. All the modules of the Hadoop were designed with a fundamental assumption that individual machines failures are common and the system should be able to automatically deal with them, guaranteeing it regular behaviour and without data loss [44].

The Hadoop can be divided in two cores: the storage, with the Hadoop File System (HDFS) [44, 45], and the processing core, with the MapReduce [49]. The HDFS divides the files in chunks and distribute them in the cluster nodes, with replicas to guarantee availability and consistency. One master node keeps all the meta-data registered, with information of where the chunks are distributed within the cluster. Copies of the master node state are also replicated among several machines, allowing for recovery in case of fault.

To process the data, taking advantage of the data locality, each node process the chunk of data present in its disk, following the MapReduce paradigm. In this programming model, each task has two phases, a *Map* and a *Reduce*. The

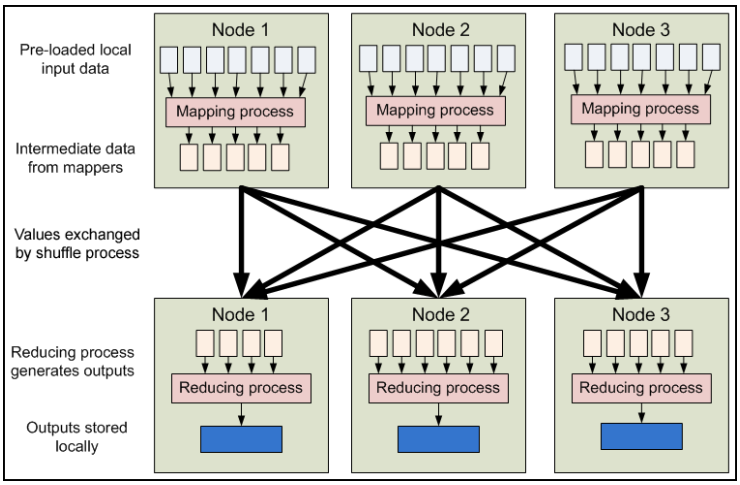


Figure 7: MapReduce programming model (image taken from Yahoo!'s Hadoop tutorial: https://developer.yahoo.com/hadoop/tutorial/module4.html)

Map operation receives a key-pair value as an input and produces another key-value pair, resulted from a filtering or sorting operation, for example, as an intermediate step. The *Reduce* operation receives those intermediate key-value pairs and summarise the final results over each key [49]. The Figure 7 shows the algorithm as a whole. It is important to notice that the system does not keep the data (input, intermediate nor final) in cache. That happens because in the Hadoop's initial design, it was observed that the majority of the tasks the developers aimed to solve, only one passage over the data was necessary, not needing to keep the data in memory [44].

As shown in Figure 7, between the Map and Reduce operation, there may be a Shuffle operation, where the intermediate key-value pairs are transmitted between nodes to optimise Reduce tasks accordingly to the each key. This stage can have a significant network overhead.

Several machine learning algorithms can be adapted to use the Hadoop framework, following the MapReduce paradigm during training [50, 51]. However, machine learning algorithms are in their majority iterative, and the Hadoop's non-caching behaviour may lead to a slow performance due to several disk ac-

cesses. The Spark came to, among other things, solve this problem.

Spark. The Spark, originally developed in the Berkley University California AMPLamp laboratory, is also a platform for distributed data storage and processing. Besides guaranteeing fault tolerance, during Spark's development, two main objectives were pursued differently from Hadoop: efficiency in iterative algorithms and interactive tools for exploratory data analysis [33, 34].

The fundamental data abstraction in Spark is the Resilient Distributed Data (RDD), a logical collection of partitioned data less restrictive than the MapReduce model. A RDD can be create referring to external data or by applying a transformation from another RDD. Users can control an RDD in two aspects: persistency (to store in memory, disk or both and its replication level) and partitioning (number of partitions, also called as level of parallelism). An RDD can be allocated in memory, making its reuse by an iterative algorithm faster [34].

Besides its own cluster manager, the Spark provides support for another systems like Hadoop YARN and Apache Mesos. In relation to the file systems, several systems can be used with Spark, including HDFS, Cassandra, OpenStack Swift and Amazon S3. Finally, the Spark provides support to local execution, e.i., in a pseudo-distributed way, important mainly for testing and developments stages, without the need of a real cluster [34].

3. Methodology

This section describes the proposed adaptive Fault Detection and Diagnosis (FDD) system. The proposed methodology deals with the FDD task as a pattern recognition problem. The patterns to be classified are defined by the plant or equipment's sensors and actuators signals. Each operation condition (normal or faulty) is defined as a class. For each new data sample, it is up to the classifier to estimate the state of the process.

The classifier is incremental, i.e., it is initialised only with the data describing the normal operation condition of the process/equipment. As new operation conditions are detected, those behaviours are modelled and incorporated on

the knowledge base of the classifier. In this way, in case a given operation condition happens more than once, the classifier is able to identify it, i.e., the make an adaptive diagnosis. Although the monitoring statistics of the process can be computed and monitored in real-time sample-by-sample, the actual FDD is made batch-wise, since a window of data is needed to work with the threshold and confidence level based detection and diagnosis as well as for gathering data for training a newly found behaviour.

As previously discussed in the Subsection 2.2.7, the PGMM family gives a big variety of possible structures. To find the best one to model a given dataset, it is proposed to use a distributed computing technique, particularly the *Spark* framework, to train a large number of possibilities and later choose the best fitted model, in a tradeoff with its complexity.

3.1. Modelling

The proposed FDD system has the PGMM family of distributions as its statistical classifier model, as described in Subsection 2.2.7. The advantage of those models is that they have parametric restrictions that limits the number of parameters to be estimated, generating simpler models.

In general, simpler models need less data during its training process to achieve a good final parameter estimation. This characteristic is important for FDD systems because, in practice, data for abnormal conditions are scarce, since those events are, ideally, rare and of short duration.

3.1.1. Training the PGMM

The EM algorithm is an iterative method to estimate the models parameters that maximise the likelihood of the observed data, assuming the existence of non-observable latent variables. The algorithm iterates through two steps: in the first step, *Expectation* (E), the formula for the expected log-likelihood of the data evaluated with the current estimation of the parameters is calculated; on the second step, *Maximisation* (M), we compute the parameters which maximise the

expected log-likelihood formulated on step E. These parameters are used again on the next step E and the algorithm continues to iterate until convergence.

The models of the PGMM family are training using the Alternating Expectation Conditional Maximisation (AECM) algorithm [52], an extension of the EM algorithm, where two different definitions of the latent variables are assumed on the different stages. Each of the eight models have slightly different equations on each step, and their description and formulation can be found on [20]. The implementation of the PGMM was in Python, based on the Machine Learning library Scikit-learn [53], where the regular Gaussian mixtures model was taken as a base model, changing the necessary distribution structures and the AECM algorithm. The computation was distributed using the PySpark, the *Spark* library in Python.

3.1.2. Model Selection

Having several trained models at hand, one should finally choose which of them is the model that best fits the observed data. As discussed earlier, the reduced number of parameters is an important advantage of the parsimonious models, therefore, the metrics to judge the best model should take into account both model complexity and fitness to the data.

The Bayesian Information Criterion (BIC) is a vastly used metric for model selection. It is defined by[20]:

$$BIC = 2\log\left(p(\mathbf{t}|\hat{\theta})\right) - m\log n \tag{8}$$

where $\log\left(p(\mathbf{t}|\hat{\theta})\right)$ is the log-likelihood of the data given the model, $\hat{\theta}$ the estimated parameters of the model, m is the number of parameters and n is the number of training samples. The BIC can be used to choose not only which of the eight structures of the PGMM is the best, but also the ideal number of principal components, q, and the number of mixture components, k.

3.1.3. Monitoring Statistics and its Threshold

The likelihood gives the measurement of how well a data sample fits a given probability distribution. It is this likelihood metric that is going to be used to monitor the condition of the plant, and it can be calculated by:

$$p(\mathbf{t}) = \sum_{i=1}^{k} \pi_i p(\mathbf{t}|i) \ , \tag{9}$$

where $p(\mathbf{t}|i) \sim \mathcal{N}(\mu_i, C_i)$ is the distribution of the i-th component, where $C_i = W_i W_i^T + \Psi_i$, and the weights sum to the unity, $\sum_i \pi_i = 1$.

A threshold can be used with the PGMM to determine if a sample belongs or not to a given model, with a certain level of confidence. For example, it can be used a numeric approach as the Monte Carlo simulations, as follows [54]:

1. Generate N_s samples, $\{\mathbf{t}_j \mid j = 1, ..., N_s\}$, from $p(\mathbf{t})$ given by Eq. 9.
2. Compute the likelihood of those samples as $p(\mathbf{t}_j)$.
3. Sort $p(\mathbf{t}_j)$ of all the $j = 1, ..., N_s$ samples.
4. The confidence bound is given by $h = p(\mathbf{t}_l)$, where $l = \beta N_s$.

Hence, a given data point $\mathbf{t}_i$ is considered out of control, with β-level of significance, if $p(\mathbf{t}_i) < h$, or equivalently, if $-p(\mathbf{t}_i) > -h$. Typically, the number of Monte Carlo samples, N_s, has to be large, and can be determined heuristically. If the training dataset used to fit the model is large enough, the confidence bound can be calculated based on those values.

3.2. Adaptive Fault Detection and Diagnosis

In this section the proposed batch-incremental FDD system is described. An important feature of the proposed FDD system is that it gradually learns new faulty operation conditions of a process by analysing data batches incrementally. The proposed method begins only with the model of a normal operation condition (NOC). The algorithm described in Section 3.1.1 is used to fit the family PGMM and the best is selected using the criteria discussed in Section 3.1.2, describing the normal condition. Every time a new operation condition is detected, a new PGMM is estimated to model this new condition.

Given the NOC PGMM model, process monitoring is performed using the threshold derived in Section 3.1.3. By monitoring the samples' statistics of a given data stream under the normal operation model, deviations from the NOC can be detected, suggesting a fault in the process.

Suppose a new data batch to be analysed, containing N samples. Given that the threshold h of the normal operation PGMM model was obtained with a $100\beta\%$ confidence bound, it is expected that, approximately, $N_f = (1-\beta)N$ samples to be detected outside the threshold, even though the process is operating under normal condition. More formally, the number of samples detected as out of control, N_f, follows a binomial distribution, $N_f \sim \mathcal{B}(N, 1-\beta)$, given that the data is under NOC. Therefore, to actually detect an abnormal operation, the number of out of control samples N_f detected must be greater than the binomial inverse cumulative distribution, also with β level of significance, lets denote $N_f > \mathcal{B}_\beta^{-1}(N, 1-\beta)$.

Before fitting a new PGMM model for the detected fault, the N_f samples are analysed based on all the others PGMM models already created (each one describing one operation condition already observed) to check if the current data is from an already known condition. Suppose that besides the PGMM model for the normal condition, another C mixture models have already been fitted to model C abnormal conditions. The idea is the same as used to check if the data deviates from the NOC. Since all C of the mixture models has its own threshold, h_c, the same test with the binomial distribution is made, for each mixture model.

Therefore, the N_f samples are only used to fit a new PGMM model if none of the already known behaviours can model them well enough. Formally, if N_{fc} is the number of samples in the subset which are outside the threshold h_c of the PGMM model c, a new behaviour is detected if $N_{fc} > \mathcal{B}_\beta^{-1}(N_f, 1-\beta)$, for all $c = 1, 2, ..., C$. If $N_{fc} <= \mathcal{B}_\beta^{-1}(N_f, 1-\beta)$ for at least one c, the detected fault is already known, and the corresponding PGMM model is the one with the smallest N_{fc}.

Given that $(1+C)$ PGMM models already exists, one for the NOC and C

for abnormal behaviours, such that $p(\mathbf{t} \mid c)$ for all $c = 1, 2, ..., C$, the method can be summarised as follows:

1. Given a new data batch, analyse all the N samples from the batch through the statistics $p(\mathbf{t}_i \mid \text{normal})$, detecting N_f out of control samples, under the threshold h.
2. If $N_f <= \mathcal{B}_{\beta}^{-1}(N, 1-\beta)$, the data batch is classified as under normal condition, and the algorithm returns. Else, abnormal condition is detected and analysed in Step 3.
3. Compute $N_{fc} = \sum_{i=1}^{N_f} [p(\mathbf{t}_i \mid c) < h_c]$ for all $c = 1, 2, ..., C$, that is the number of samples in N_f that their statistics are beyond the threshold of each of the known mixture models.
4. If $N_{fc} > \mathcal{B}_{\beta}^{-1}(N_f, 1-\beta)$, for all $c = 1, 2, ..., C$, a new behaviour is detected, go to Step 5. Else, an already known behaviour is detected, classified as $c = \arg\min_c N_{fc}$.
5. Fit a new PGMM model based on the N_f dataset, and compute its threshold h_{c+1} via Monte Carlo simulations. Add the new PGMM model to the FDD system, $C = C + 1$.

These steps can also be seen in the flowchart in the Figure 8.

As previously discussed, even a normal operation condition data set will have some samples detected as abnormal, due to the β level of confidence. To minimize false alarms during these situations, an exponential moving average (EMA) filter is used in the monitoring statistics, adding some time dependency between the samples, a very reasonable assumption in the real-time analysis. The filter is given by:

$$p(\mathbf{t}_i) = (1-\gamma)p(\mathbf{t}_{i-1}) + \gamma p(\mathbf{t}_i) \ , \tag{10}$$

where γ is the filter weight. When $\gamma = 1$, the filtering action is disabled.

3.3. Distributed Training

The training of the large range of possible models, where one would have to choose among different structures, number of principal components and num-

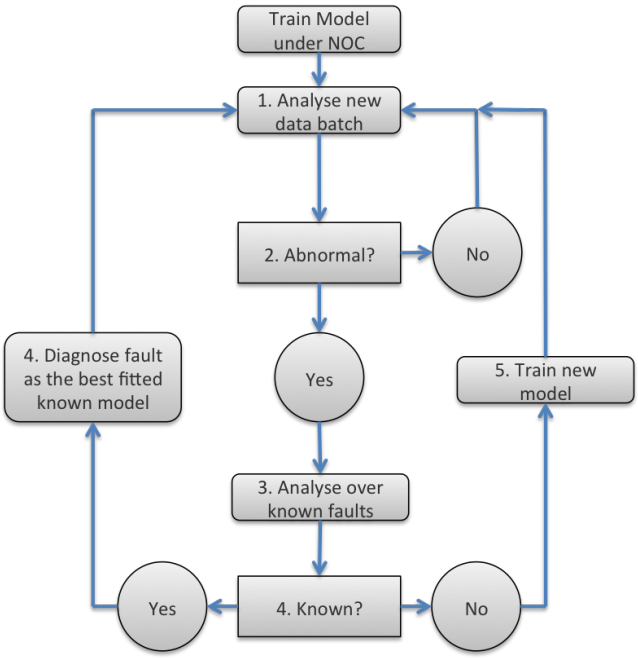


Figure 8: Flowchart of the proposed FDD methodology [55].

ber of mixture components, can represent a intense computational load, which motivated the usage of parallel and distributed computing techniques in this work.

The Apache Spark is one of the most modern frameworks for distributed computing, and was chosen to be used in this work due to its good performance in an iterative workload. The computation parallelism for this task could be achieve by two means:

1. The matrixes computation, mainly those involving high dimensional and cardinality data, can be done in parallel. Indicated for when the processed data can not fit in memory, requiring the data to be distributed over several machines.
2. Different models do not have a dependency with one another, and therefore can be computed entirely in parallel. One example of that situation is on the training of the PGMM family, where the members are independent

from each other.

As the data dealt in this work is not big enough to require the data distribution, it is believed that the overhead for its implementation and computation is not worth it, wherefore only the second method of parallelism was used.

Hence, for the final training, a broadcast variable was created with a training data set, such that all workers have access to it. Next, a vector of hyper-parameters (member of the PGMM family, number of mixture components, k, and number of principal components, q) containing all n desired values to cover the search is randomly distributed among the workers. The random distribution is necessary because each trained model have different complexity, and the randomness aims to balance the load on each worker. Finally, each worker trains the models with their associated hyper-parameters and, at the end, the master collects all values of BIC and chooses the best one.

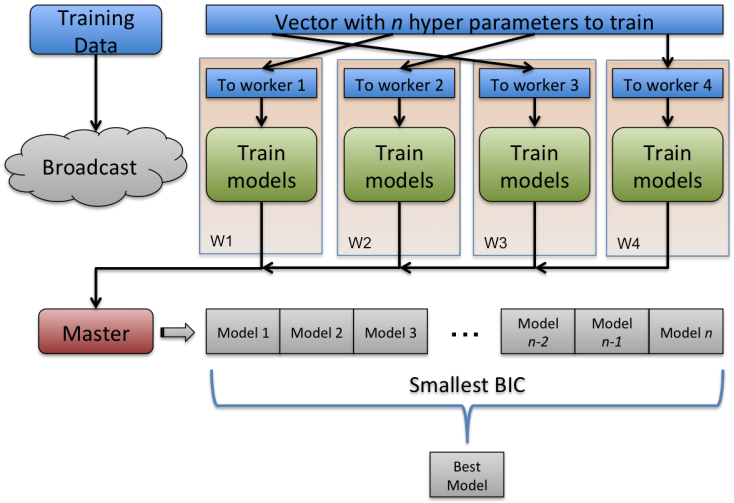


Figure 9: Distributed training scheme.

4. Validation Results

This section presents the experiment results of the proposed method for the fault detection and diagnosis in different scenarios. Besides the methodology

validation, it is also shown a study about that training performance of the models using Spark's distributed computations paradigm.

4.1. System Validation with TEP

This subsection describes the experimental results of the proposed method using computer simulated process data. This simulated scenario gives full control of the faults as well as when those faults happen, allowing for a better and more objective validation of the method.

4.1.1. Tennessee Eastman Process

The proposed method was tested using the Simulink model of the Tennessee Eastman Process (TEP) [56] under a decentralised control strategy [57].

The process has five main units, the reactor, condenser, separator, stripper and compressor, as shown in the Figure 10. The plant has eight components: A, B, C, D, E, F, G and H. The components A, C, D and E are gaseous reactants and B is an inert gas which are fed to the reactor, where the liquids G and H are produced. The component F is a sub-product of the reactions.

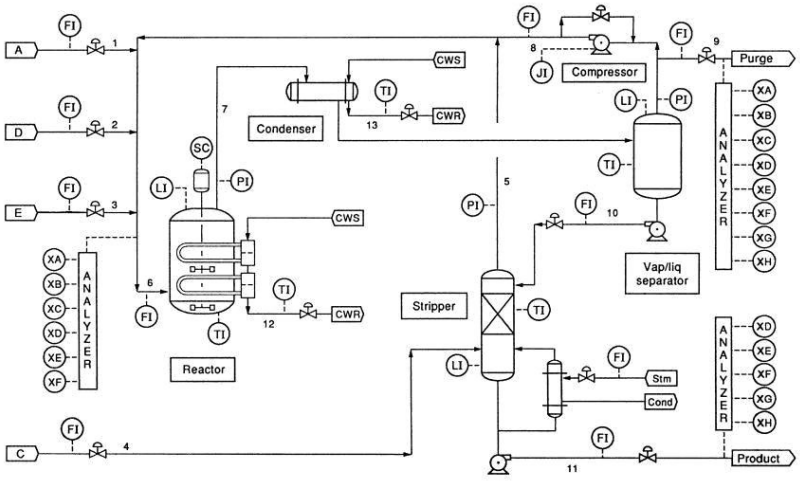


Figure 10: Tennessee Eastman process [56].

The process has 41 measured variables and 12 manipulated variables. Due

to a uniformity in the sample period of 6 minutes, only the first 22 measured variables were used, which are summarised in Table 2. The others 19 measured variables have different sample frequency and are measurements of the plant components, disabling them for direct usage.

Table 2: TEP's measured variables used in the analysis.

Variable	Description
$t(1)$	Feed A (Stream 1)
$t(2)$	Feed D (Stream 2)
$t(3)$	Feed E (Stream 3)
$t(4)$	Total Feed (Stream 4)
$t(5)$	Recycle Flow (Stream 8)
$t(6)$	Reactor Feed Rate (Stream 6)
$t(7)$	Reactor Pressure
$t(8)$	Reactor Level
$t(9)$	Reactor Temperature
$t(10)$	Purge Rate (Stream 9)
$t(11)$	Product Separator Temperature
$t(12)$	Product Separator Level
$t(13)$	Product Separator Pressure
$t(14)$	Product Separator Underflow (Stream 10)
$t(15)$	Stripper Level
$t(16)$	Strippel Pressure
$t(17)$	Stripper Underflow (Stream 11)
$t(18)$	Stripper Temperature
$t(19)$	Stripper Steam Flow
$t(20)$	Compressor Work
$t(21)$	Reactor Cooling Water Outlet Temperature
$t(22)$	Separator Cooling Water Outlet Temperature

Table 3: Tenneessee Eastman process faults.

Fault ID	Fault Description	Type	Detection Delay	No. Mixture Components	No. Principal Components	PGMM Structure
1	A/C feed ratio, B composition constant (Stream 4)	Step	2	13	6	UUC
2	B composition, A/C ratio constant (Strem 4)	Step	5	4	2	UCU
8	A, B, C feed composition (Stream 4)	Random variation	8	14	9	CUC
10	C feed temperature	Random variation	20	3	3	UCU
11	Reactor cooling water inlet temperature	Random variation	3	7	20	CUC
12	Condenser cooling water inlet temperature	Random variation	11	2	3	UCU
13	Reaction kinetics	Slow drift	12	16	9	CUC
14	Rector cooling water valve	Sticking	4	2	4	UCU
17	Unknown	Unknown	17	9	2	UCU
18	Unknown	Unknown	39	6	2	UCU
19	Unknown	Unknown	30	3	3	UCU
20	Unknown	Unknown	26	11	20	CUC

4.1.2. Isolated Experiments

The simulated model has 21 operation modes, corresponding to one normal operation and 20 faulty conditions. On the first set of experiments, always initialising the FDD system with the NOC model, each fault was applied twice in a row, so the system could firstly detect and learn the new behaviour and later diagnose it.

The majority of the operation modes were detected and diagnosed correctly by the proposed method. The considered faults are summarised in the Table 3, together with the time, in samples, to detect the fault after its start, the best number of mixture components and of principal components and the best member of the PGMM family. To get the best model, the range covered by the distributed training for all cases was from 1 to 16 number of components, 1 to 20 principal components and all 8 members of the PMM family, leading to a total of 2560 trained models per scenario. On each case, the model with the smallest BIC was selected.

Among the unlisted faults in the Table 3, the Fault 6 was unstable, being unfeasible for evaluation. The other unlisted faults did not show a significant change in the distribution of the 22 used measured variables on stationary state, when comparing with the NOC. In general, it was only detected a quick transitory behaviour on the instant the fault started, also making them unfeasible for testing.

4.2. Sequential Experiment

In the next experiment, a closer to a real scenario was tested. Starting again with just the model for the NOC, it was applied a sequence of faults, as illustrated by the dashed curve in Figure 11. It was applied, in sequence, the faults 13, 14, 13, 11, 11, and 14, covering the main types of faults in the process, as described in Table 3, always returning the plant to the NOC between the faults. Thus, it is expected that with each new fault a new model is trained and on reoccurrence of the same fault, the system should be able to diagnose it.

The results can also be seen in Figure 11. The blue dashed line shows the

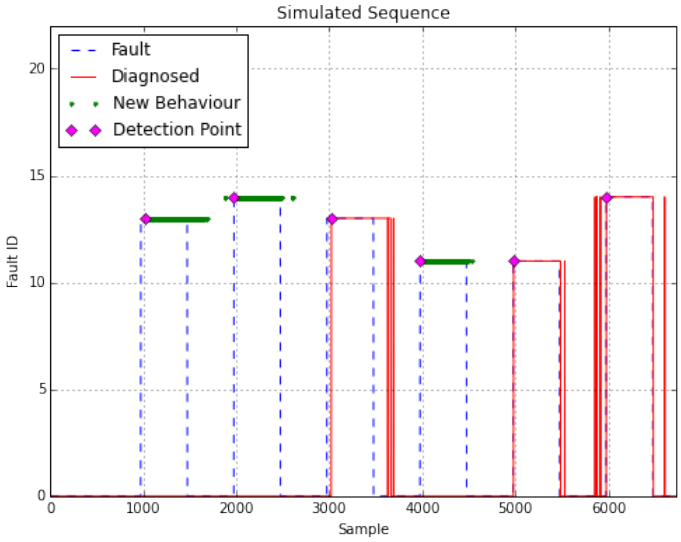


Figure 11: Results of the applied fault sequence.

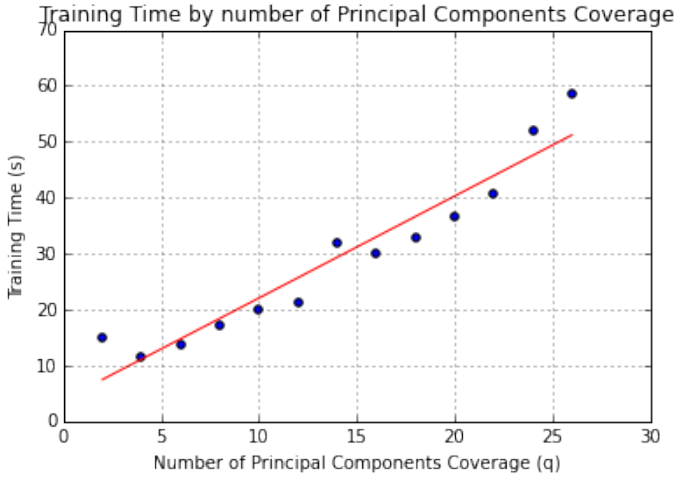
operation condition at each instant, where the NOC is indexed as zero. The solid red line shows the output diagnostic system. The green dots represent the samples used for training a new model, note that they show up only on the first occurrence of a given fault, which is when the model is trained. Finally, the magenta diamond shows when the fault has been detected by the system.

Analysing the obtained results, the methodology has presented the expected behaviour, detecting and diagnosing the fault correctly.

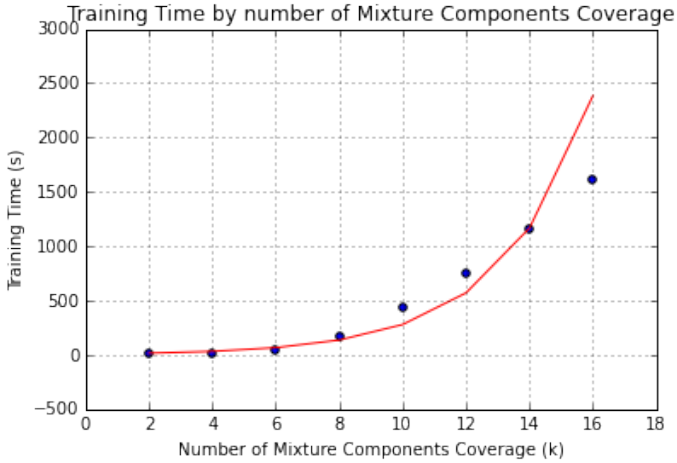
4.3. Training Experiments

Besides the proposed FDD method, this work also proposed the usage of distributed computing for training the models. This section presents some numbers related to their training performances. Keeping the number of cores fixed, Figures 12a and 12b show how the time taken to train all models varies accordingly with the coverage of the number of principal components and number of mixture components, respectively.

For the test considering the number of principal components, shown in Figure 12a, an artificially generated data from a Gaussian mixture model with 5



(a) Principal components coverage.



(b) Mixture components coverage.

Figure 12: Training time varying with models hyper-parameters coverage.

components, 30 features and 10000 samples was used. Keeping the coverage for the number of mixture components fixed, from 1 to 2, as well as the coverage for all eight PGMM structures. The coverage for the number of principal components was varied from 1 to 2 until from 1 to 28. In other words, for each test point, the number of trained models increases linearly from $2 \times 8 \times 2 = 32$

until $2 \times 8 \times 28 = 448$, where in the point the maximum number of principal components covered is increased. As expected, it can be seen that the time taken to train all the model increases approximately linearly.

A similar analysis is made for the number of mixture components, keeping, in turn, the coverage for the principal components fixed, from 1 to 2, and varying the coverage of the number of mixture components from 1 to 2 until from 1 to 16. In this case, as illustrated in Figure 12b, the time taken to train the model increases approximately exponentially. This is the expected behaviour because besides increasing the number of models, their complexity increases in the same rate, taking to an exponential growth of the time to train the set.

In a final test, keeping the same coverage of the trained models, always with the same data set, the number of *Spark* tasks was altered, essentially increasing the parallelism level. Figure 13 shows the obtained result.

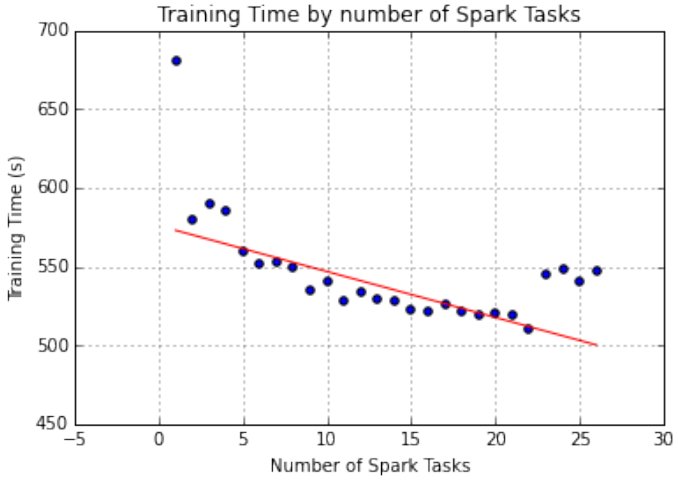


Figure 13: Training time varying the number of Spark tasks.

Firstly, one can see that with only one task, the performance is much worse, because the *Spark master* and *worker* are executed in the same core, which reduces the performance. For two tasks until the number of cores of the cluster, which was 24, the training time reduces approximately linearly with the number of tasks. From the maximum number of cores, there is a saturation, which is

expected, since the parallelism can not be higher than the number of cores in the cluster.

Although the obtained result showed a behaviour close to the expected, the decreasing in the training time was not as big as one would expect, which is a important point for further investigation. The high overhead for the distributed process can have an important role in the observed behaviour, it is expected that in a scenario with more data, where the overhead time would get proportionally smaller, a more significant improvement on the training time would be seen.

5. Conclusion

In this work, it was proposed a Fault Detection and Diagnosis (FDD) system based on statistical models. The FDD system is based on a family of distributions of Parsimonious Gaussian Mixture Models (PGMM) where the reduced number of parameters can be useful in a scenario where few data is available, such as in faulty operation conditions. Although there is a reduced number of parameters in the model, there is a huge number of possible models, and choosing the right one to represent a given operation condition is still an open issue in the research community. For that, it was also proposed to used distributed computing techniques to train a large number of different models and, later, choosing the best one.

The numerical results showed a good performance of the proposed FDD method, detecting and diagnosing correctly a fault. Furthermore, using the power of distributed computing appears to be a good solution for training a larger number of models, specially in a scenario where computation power has been getting ever so inexpensive.

The proposed FDD system can be improved in two main fronts. The first would be in developing a probabilistic contribution analysis of the variables, allowing not only the detection of a fault, but also indicating of the most responsible variables, which would help the operator in diagnosing the problem. Furthermore, an online adaptive training of the models would help increase the

automation level of the system, since it would be able to identify natural changes of the process.

During the development of this work, it was not necessary the usage of data parallelism, since they were not large enough to justify its usage. But the development of a training strategy involving both data and processing parallelism could bring significant improvement in the training time, specially if the data gets very large.

Finally, the robustness of the method still needs further validation on a real process scenario, where greater noise and uncertainties make the FDD task more difficult.

Acknowledgement

The authors acknowledge the support of Petrobras, CNPq (National Counsel of Technological and Scientific Development) and FAPEMIG (Research Foundation of the State of Minas Gerais).

References

[1] V. Venkatasubramanian, R. Rengaswamy, K. Yin, S. N. Kavuri, A review of process fault detection and diagnosis: Part I: Quantitative Model-based Methods, Computers & Chemical Engineering 27 (2003) 293–311. doi:10.1016/S0098-1354(02)00160-6.

[2] V. Venkatasubramanian, R. Rengaswamy, S. N. Kavuri, A review of process fault detection and diagnosis: Part II: Qualitative models and search strategies, Computers & Chemical Engineering 27 (2003) 313–326. doi:10.1016/S0098-1354(02)00161-8.
URL http://www.sciencedirect.com/science/article/pii/S0098135402001618

[3] V. Venkatasubramanian, R. Rengaswamy, S. N. Kavuri, K. Yin, A review of process fault detection and diagnosis: Part III: Process history

based methods, Computers & Chemical Engineering 27 (2003) 327–346. doi:10.1016/S0098-1354(02)00162-X.
URL http://www.sciencedirect.com/science/article/pii/S009813540200162X

[4] A. Shui, W. Chen, P. Zhang, S. Hu, X. Huang, Review of fault diagnosis in control systems, 2009 Chinese Control and Decision Conference (2009) 5324–5329doi:10.1109/CCDC.2009.5195065.
URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5195065

[5] L. F. Mendonça, J. M. C. Sousa, J. M. G. Sá da Costa, An architecture for fault detection and isolation based on fuzzy methods, Expert Systems with Applications 36 (2009) 1092–1104. doi:10.1016/j.eswa.2007.11.009.

[6] A. Lemos, W. Caminhas, F. Gomide, Adaptive fault detection and diagnosis using an evolving fuzzy classifier, Information Sciences 220 (2013) 64–85. doi:10.1016/j.ins.2011.08.030.
URL http://linkinghub.elsevier.com/retrieve/pii/S002002551100449X

[7] B. Samanta, Gear fault detection using artificial neural networks and support vector machines with genetic algorithms, Mechanical Systems and Signal Processing 18 (2004) 625–644. doi:10.1016/S0888-3270(03)00020-7.

[8] C. A. Laurentys, R. M. Palhares, W. M. Caminhas, Design of an artificial immune system based on Danger Model for fault detection, Expert Systems with Applications 37 (7) (2010) 5145–5152. doi:10.1016/j.eswa.2009.12.079.
URL http://dx.doi.org/10.1016/j.eswa.2009.12.079

[9] C. A. Laurentys, R. M. Palhares, W. M. Caminhas, A novel artificial immune system for fault behavior detection, Expert Systems with Applications 38 (6) (2011) 6957–6966. doi:10.1016/j.eswa.2010.12.019.
URL http://dx.doi.org/10.1016/j.eswa.2010.12.019

[10] D. Kim, I.-B. Lee, Process monitoring based on probabilistic PCA, Chemometrics and Intelligent Laboratory Systems 67 (2) (2003) 109–123. doi:10.1016/S0169-7439(03)00063-7.
URL http://linkinghub.elsevier.com/retrieve/pii/S0169743903000637

[11] N. Mehranbod, M. Soroush, C. Panjapornpon, A method of sensor fault detection and identification, Journal of Process Control 15 (2005) 321–339. doi:10.1016/j.jprocont.2004.06.009.

[12] M. F. D'Angelo, R. M. Palhares, R. H. Takahashi, R. H. Loschi, L. M. Baccarini, W. M. Caminhas, Incipient fault detection in induction machine stator-winding using a fuzzy-Bayesian change point detection approach, Applied Soft Computing 11 (1) (2011) 179–192. doi:10.1016/j.asoc.2009.11.008.
URL http://linkinghub.elsevier.com/retrieve/pii/S156849460900221X

[13] L. Dobos, J. Abonyi, On-line detection of homogeneous operation ranges by dynamic principal component analysis based time-series segmentation, Chemical Engineering Science 75 (2012) 96–105. doi:10.1016/j.ces.2012.02.022.
URL http://linkinghub.elsevier.com/retrieve/pii/S0009250912001182

[14] C. Angeli, A. Chatzinikolaou, On-line fault detection techniques for technical systems: a survey, International Journal of Computer Science & Applications; I (1) (2004) 12 – 30.
URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.100.6189&rep=rep1&type=pdf

[15] S. Joe Qin, Statistical process monitoring: basics and beyond, Journal of Chemometrics 17 (2003) 480–502. doi:10.1002/cem.800.
URL http://onlinelibrary.wiley.com/doi/10.1002/cem.800/

abstract$\delimiter"026E30F$nhttp://onlinelibrary.wiley.com/store/10.1002/cem.800/asset/800_ftp.pdf?v=1&t=hcmg2etn&s=8ad2049be22a258774d1524922e72d11c5425ff0

[16] J. E. Jackson, G. S. Mudholkar, Control Procedures for Residuals Associated with Principal Component Analysis, Technometrics 21 (3) (1979) 341–349. doi:10.1080/00401706.1979.10489779.
URL http://www.jstor.org/stable/1267757

[17] T. Villegas, M. Fuente, M. Rodríguez, Principal component analysis for fault detection and diagnosis. experience with a pilot plant, Proceedings of the 9th WSEAS International Conference on Computational Intelligence, Man-machine Systems and Cybernetics (2010) 147–152.
URL http://www.wseas.us/e-library/conferences/2010/Merida/CIMMACS/CIMMACS-20.pdf

[18] M. E. Tipping, C. M. Bishop, Probabilistic Principal Component Analysis, Journal of the Royal Statistical Society: Series B (Statistical Methodology) 61 (3) (1999) 611–622. doi:10.1111/1467-9868.00196.
URL http://dx.doi.org/10.1111/1467-9868.00196

[19] M. E. Tipping, C. M. Bishop, Mixtures of probabilistic principal component analyzers, Neural computation 11 (1999) 443–482. doi:10.1162/089976699300016728.

[20] P. D. McNicholas, T. B. Murphy, Parsimonious Gaussian Mixture Models, Statistics and Computing 18 (3) (2008) 285–296. doi:10.1007/s11222-008-9056-0.

[21] B. He, X. Yang, T. Chen, J. Zhang, Reconstruction-based multivariate contribution analysis for fault isolation: A branch and bound approach, Journal of Process Control 22 (7) (2012) 1228–1236. doi:10.1016/j.jprocont.2012.05.010.
URL http://linkinghub.elsevier.com/retrieve/pii/S0959152412001254

[22] C. Biernacki, G. Celeux, G. Govaert, F. Langrognet, Model-based cluster and discriminant analysis with the MIXMOD software, Computational Statistics and Data Analysis 51 (2006) 587–600. doi:10.1016/j.csda.2005.12.015.

[23] C. Biernacki, G. Celeux, G. Govaert, Choosing starting values for the EM algorithm for getting the highest likehood in multivariate Gaussian mixture models, Computational Statistics and Data Analysis 41 (2003) 561–575. doi:10.1016/S0167-9473(02)00163-9.

[24] D. Karlis, E. Xekalaki, Choosing initial values for the EM algorithm for finite mixtures, Computational Statistics and Data Analysis 41 (2003) 577–590. doi:10.1016/S0167-9473(02)00177-9.

[25] V. Melnykov, I. Melnykov, Initializing the em algorithm in Gaussian mixture models with an unknown number of components, Computational Statistics and Data Analysis 56 (6) (2012) 1381–1395. doi:10.1016/j.csda.2011.11.002.
URL http://dx.doi.org/10.1016/j.csda.2011.11.002

[26] N. Greggio, A. Bernardino, P. Dario, J. Santos-Victor, Efficient greedy estimation of mixture models through a binary tree search, Robotics and Autonomous Systems 62 (10) (2014) 1440–1452. doi:10.1016/j.robot.2014.05.016.
URL http://dx.doi.org/10.1016/j.robot.2014.05.016

[27] L. Li, J. Ma, A BYY scale-incremental EM algorithm for Gaussian mixture learning, Applied Mathematics and Computation 205 (2) (2008) 832–840. doi:10.1016/j.amc.2008.05.076.
URL http://dx.doi.org/10.1016/j.amc.2008.05.076

[28] M. E. Musa, D. de Ridder, R. P. Duin, V. Atalay, Almost autonomous training of mixtures of principal component analyzers, Pattern Recognition Letters 25 (9) (2004) 1085–1095. doi:10.1016/j.patrec.2004.03.019.

URL http://linkinghub.elsevier.com/retrieve/pii/S0167865504000728

[29] J. J. Verbeek, N. Vlassis, B. Kröse, Efficient greedy learning of gaussian mixture models., Neural computation 15 (2) (2003) 469–85. doi:10.1162/089976603762553004.
URL http://www.ncbi.nlm.nih.gov/pubmed/12590816

[30] Z. Zivkovic, F. Van der Heijden, Recursive unsupervised learning of finite mixture models, IEEE Transactions on Pattern Analysis and Machine Intelligence 26 (3) (2004) 651–656. doi:10.1109/TPAMI.2004.1273970.

[31] A. Fassò, M. Cameletti, The EM algorithm in a distributed computing environment for modelling environmental space-time data, Environmental Modelling Software 24 (9) (2009) 1027–1035. doi:10.1016/j.envsoft.2009.02.009.
URL http://linkinghub.elsevier.com/retrieve/pii/S1364815209000413

[32] C. L. Philip Chen, C. Y. Zhang, Data-intensive applications, challenges, techniques and technologies: A survey on Big Data, Information Sciences 275 (2014) 314–347. doi:10.1016/j.ins.2014.01.015.
URL http://dx.doi.org/10.1016/j.ins.2014.01.015

[33] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, I. Stoica, Spark: Cluster computing with working sets, in: Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing, HotCloud'10, USENIX Association, Berkeley, CA, USA, 2010, pp. 10–10.
URL http://dl.acm.org/citation.cfm?id=1863103.1863113

[34] M. Zaharia, M. Chowdhury, T. Das, A. Dave, Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing, NSDI'12 Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation (2012) 2–2doi:10.1111/j.1095-8649.2005.00662.x.

URL https://www.usenix.org/system/files/conference/nsdi12/nsdi12-final138.pdf

[35] J. Gama, I. Zliobaite, A. Bifet, M. Pechenizkiy, A. Bouchachia, A survey on concept drift adaptation, ACM Computing Surveys 46 (4) (2014) 1–37. doi:10.1145/2523813.
URL http://dl.acm.org/citation.cfm?id=2523813

[36] M. R. Maurya, R. Rengaswamy, V. Venkatasubramanian, Fault diagnosis using dynamic trend analysis: A review and recent developments, Engineering Applications of Artificial Intelligence 20 (2) (2007) 133–146. doi:10.1016/j.engappai.2006.06.020.

[37] H. Hotelling, Analysis of a complex of statistical variables into principal components, Journal of Educational Psychology 24 (1933) 417–441 and 498–520.

[38] D. A. Jackson, Stopping rules in principal components analysis: A comparison of heuristical and statistical approaches (1993). doi:10.2307/1939574.

[39] H. Hotelling, The generalization of Student's ratio, The Annals of Mathematical Statistics 2 (3) (1931) 360–378.

[40] J. Mina, C. Verde, Fault detection using dynamic principal component analysis by average estimation, 2005 2nd International Conference on Electrical and Electronics Engineering (Cie) (2005) 374–377. doi:10.1109/ICEEE.2005.1529647.

[41] M. Guerfel, K. Othman, M. Benrejeb, On the structure determination of a dynamic PCA model using sensitivity of fault detection, Advanced Control of Chemical Processes 7 (2) (2009) 958–963. doi:10.3182/20090712-4-TR-2008.00157.
URL http://www.nt.ntnu.no/users/skoge/prost/proceedings/adchem09/cd/abstract/153.pdf

[42] Z. Bankó, L. Dobos, J. Abonyi, Dynamic Principal Component Analysis in Multivariate Time-Series Segmentation, Conservation, Information, Evolution 1 (1) (2011) 11–24.
URL http://journal.ke.hu/cie/index.php/cie/article/view/25

[43] W. Ku, R. H. Storer, C. Georgakis, Disturbance detection and isolation by dynamic principal component analysis, Chemometrics and Intelligent Laboratory Systems 30 (1) (1995) 179–196. doi:10.1016/0169-7439(95)00076-3.

[44] S. Ghemawat, H. Gobioff, S.-T. Leung, The Google file system, ACM SIGOPS Operating Systems Review 37 (5) (2003) 29. doi:10.1145/1165389.945450.

[45] K. Shvachko, H. Kuang, S. Radia, R. Chansler, The hadoop distributed file system, in: Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST), MSST '10, IEEE Computer Society, Washington, DC, USA, 2010, pp. 1–10. doi:10.1109/MSST.2010.5496972.
URL http://dx.doi.org/10.1109/MSST.2010.5496972

[46] H. Garraway, Parallel Computer Architecture: A Hardware/Software Approach, IEEE Concurrency 7 (2). doi:10.1109/MCC.1999.766975.

[47] M. J. Quinn, Parallel Programming in C with MPI and OpenMP, Vol. 1, McGraw-Hill, 2003.

[48] M. J. Flynn, Some Computer Organizations and Their Effectiveness, IEEE Transactions on Computers C-21 (9) (1972) 948–960. doi:10.1109/TC.1972.5009071.

[49] J. Dean, S. Ghemawat, MapReduce: Simplied Data Processing on Large Clusters, Symposium on Operating Systems Design and Implementation 6 (2004) 137–149. doi:10.1145/1327452.1327492.

[50] C.-T. Chu, S. K. Kim, Y.-A. Lin, Y. Yu, G. Bradski, A. Y. Ng, K. Olukotun, Map-Reduce for Machine Learning on Multicore, Advances in Neural Information Processing Systems 19 (2007) 281–288. doi:10.1234/12345678.

[51] D. Gillick, A. Faria, J. DeNero, MapReduce : Distributed Computing for Machine Learning, Icsiberkeleyedu (2006) 1–12.
URL http://roger-data.googlecode.com/svn-history/r14/trunk/mapred/gillick_cs262a_proj.pdf

[52] X.-L. Meng, D. Van Dyk, The em algorithman old folk-song sung to a fast new tune, Journal of the Royal Statistical Society: Series B (Statistical Methodology) 59 (3) (1997) 511–567. doi:10.1111/1467-9868.00082.
URL http://dx.doi.org/10.1111/1467-9868.00082

[53] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, Scikit-learn: Machine learning in Python, Journal of Machine Learning Research 12 (2011) 2825–2830.

[54] T. Chen, J. Morris, E. Martin, Probability Density Estimation via an Infinite Gaussian Mixture Model: Application to Statistical Process Monitoring, J ROY SOC C-APP 55 (2006) 699–715.

[55] T. A. Nakamura, A. P. Lemos, A batch-incremental process fault detection and diagnosis using mixtures of probabilistic PCA, IEEE Conference on Evolving and Adaptive Intelligent Systems 2014 (2014) 1–8. doi:10.1109/EAIS.2014.6867472.

[56] J. J. Downs, E. F. Vogel, A plant-wide industrial process control problem, Computers & Chemical Engineering 17 (1993) 245–255.

[57] N. L. Ricker, Decentralized control of the Tennessee Eastman challenge process, Journal of Process Control 6 (1996) 205–221.