Parametric and Non-Parametric Statistics for Program Performance Analysis and Comparison

Julien Worms, Sid Touati

Parametric and Non-Parametric Statistics for Program Performance Analysis and Comparison

Julien WORMS, Sid TOUATI

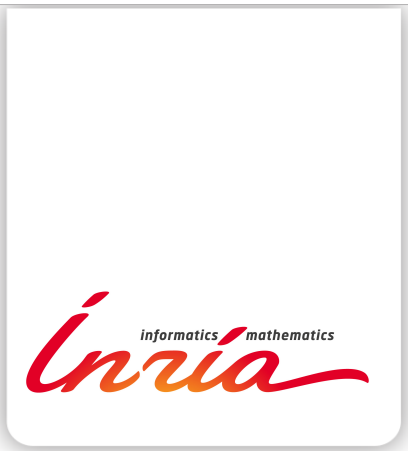
RESEARCH REPORT

N° 8875

Mar 2016

Project-Teams "Probabilités et Statistique" et AOSTE

ISSN 0249-6399 ISRN INRIA/RR--8875--FR+ENG



Parametric and Non-Parametric Statistics for Program Performance Analysis and Comparison

Julien Worms*, Sid Touati†

Project-Teams "Probabilités et Statistique" et AOSTE

Research Report n° 8875 — Mar 2016 — 70 pages

Abstract: This report is a continuation of our previous research effort on statistical program performance analysis and comparison [TWB10, TWB13], in presence of program performance variability. In the previous study, we gave a formal statistical methodology to analyse program speedups based on mean or median performance metrics: execution time, energy consumption, etc. However mean or median observed performances do not always reflect the user's feeling of performance, especially when the performances are really instable. In the current study, we propose additional precise performance metrics, based on performance modelling using Gaussian mixtures. We explore the difference between parametric and non parametric statistics applied on program performance analysis. Our additional statistical metrics for analysing and comparing program performances give the user more precise decision tools to select best code versions, not necessarily based on mean or median numbers. Also, we provide a new metric to estimate performance variability based on Gaussian mixture model. Our statistical methods are implemented with R and distributed as open source code.

Key-words: Gaussian mixture, statistical testing, parametric statistics, non parametric statistics, goodness-of-fit testing, bootstrap methodology, program performance modelling, program performance variability, benchmarking.

Résultat de collaboration interdisciplinaire mathématique-informatique financée par le projet PEPS CNRS 2013-2014: MC-VarExec

* Maître de conférences en mathématiques, Laboratoire de Mathématiques de Versailles, UVSQ, CNRS, Université Paris-Saclay, 78035 Versailles)

† Professeur d'informatique à l'Université Nice Sophia Antipolis, UFR des sciences, laboratoires I3S-INRIA

RESEARCH CENTRE
SOPHIA ANTIPOLIS – MÉDITERRANÉE

2004 route des Lucioles - BP 93
06902 Sophia Antipolis Cedex

Parametric and Non-Parametric Statistics for Program Performance Analysis and Comparison

Résumé : Dans des résultats précédents [TWB10, TWB13], nous avons présenté un protocole statistique rigoureux pour analyser et comparer les performances moyennes et médianes des programmes (temps d'exécution, consommation d'énergie, etc.). Cependant, les métriques de la moyenne ou de la médiane des observations ne sont pas nécessairement celles qui reflètent le sentiment qu'ont les utilisateurs vis à vis de leurs programmes, surtout si les performances sont instables. Dans le rapport présent, nous persévérons dans notre travail d'étude statistique pour proposer de nouvelles métriques précises, basées sur une modélisation des performances avec un mélange de Gaussiennes. Nous explorons la différence entre des statistiques paramétriques et des statistiques non paramétriques. Nos nouvelles métriques de performances apportent aux utilisateurs de meilleurs outils précis pour l'analyse et la comparaison des performances des programmes. Également, nous proposons une métrique pour évaluer la variabilité des performances de codes basée sur la modélisation par mélange de Gaussiennes. Nos méthodes statistiques sont implémentées dans R et diffusées en logiciel source libre.

Mots-clés : mélange de gaussiennes, tests statistiques, test d'adéquation, méthodologie bootstrap, statistiques paramétriques, performances des programmes, statistiques non paramétriques, modélisation statistique des performances des programmes, variabilité des performances des programmes, benchmarking.

Contents

1 Introduction and motivations

When someone reads a books or articles on computer architectures, performance analysis, operating systems (OS) or compilation, he may still think that the execution time of a program P on a fixed machine M with fixed data input I is stable around a value, which can be denoted as a single number `ExecutionTime`(P,I,M). This situation was true for old computers, but nowadays nobody really observe constant execution times, except in rare situations: ideal execution environment, special processor architectures devoted to performance stability, bare mode execution, sudo or root permissions, etc. Even SPEC organisation summarises the execution times of a program as a single number (computed as the sample median of a set of observed values).

In everyday life of computer usage, no-one really observes constant execution times, even with fixed data input and low overhead operating systems workload. The consequence is that the reported values of program performances in the literature are not reproducible, and it becomes more and more difficult to select the most effective program version or OS configuration. As in car industry, a lambda driver can hardly ever reproduce the gas consumption reported by the vendor.

The following section recalls how people use or used computers in practice.

1.1 On the usage of computers in practice

1.1.1 Historical usage of computers: batch mode

Some people may forget that computers were initially managed with batch mode systems. That is, users were submitting their jobs to the system, that used to allocate the whole machine to a single program. A program used to be the unique executing code on a machine, from the first instruction to the last one, no other process was able to execute on the machine. Jobs were executed sequentially.

The view of non shared machine by programs is assumed in many situations dealing with performances : machine benchmarking, code optimisation research projects, compilation technology, etc. In all these situations, people have an ideal view that a program executes lonely on a machine.

Nowadays, machines have multiple executing programs at the same time (concurrent process and threads). High performance computing and intensive computation still use batch systems nowadays to allocate part of a big machine (computing nodes) to single job, giving to a user the illusion that his program does not compete on hardware resources with other programs. As we will explain later, this vision is just an illusion, because micro-architectural hardware resources may be shared between jobs even if neither the OS nor the user knows it.

As an example, let us consider a real-world HPC scientific application called CONVIV [CCL03, CCL06], not a benchmark devoted to research in HPC as SPEC. It is a Fortran OpenMP parallel application devoted to scientific simulation (dynamic molecule in chemistry). This application runs on an HPC machine in batch mode. The batch scheduling system (under Linux) guarantees that a complete computing node (16 cores) is reserved to a single job, and no other process/application will be scheduled on the computing node. The user does not have a physical access to the machine, he has a remote access without root privileges. We compiled the application with two compilers (Intel fortran compiler and gnu fortran compiler, used both with `-O3` compiler optimisation flag), we tested three main thread affinities: none (Linux scheduling), scatter and compact. So we have six distinct configurations. For each configuration, we repeated

the execution of the application 35 times with exactly the same data input. The 35 execution times are visualised with violin plot[1] for each configuration in Figure 1, where the number of thread is to 4 or 8. As can be seen, even if thread affinity is fixed, the execution time of an application varies greatly. The same conclusion is observed for any number of threads, going from 1 to 16, even for sequential code version (which is usually stable). This is the case of many real-world applications executed on real-world computing systems.

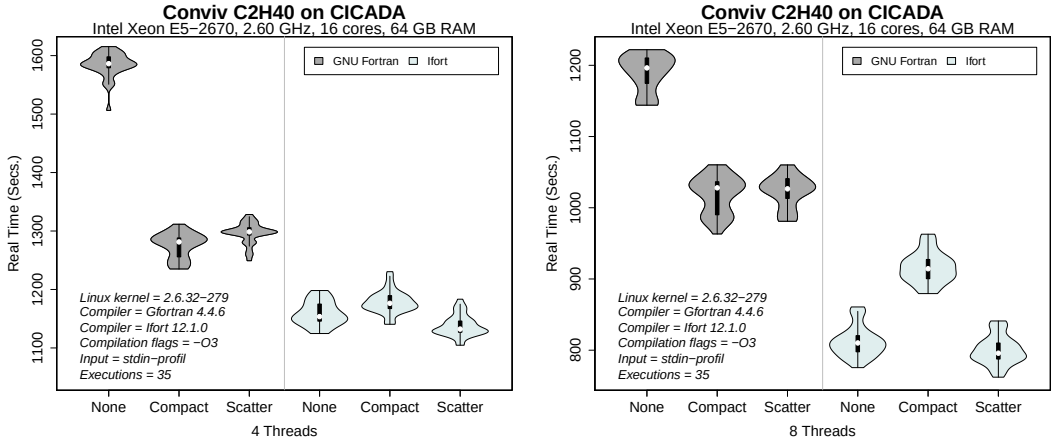


Figure 1: Running a realworld HPC application (CONVIV) on a production computing machine

1.1.2 Usage of computers for doing experiments in research projects in computer science and engineering

In the situations of research projects around compilation technology, parallelism, operating systems, code optimisation, benchmarking, people create ideal practical experimental configurations. The performances observed in these situations can hardly ever be observed outside the research experiment. This is because the researcher may have root access to the machine, may kill all non necessary concurrent process, have access to the physical machine to configure it (deactivate CPU frequency scaling via the BIOS or the OS, hardware prefetching, hyper-threading, etc...), reboot the machine, is allowed to schedule threads and process, to configure memory management, etc... The super user here has full control on a almost all factors that make the program performances vary. In this ideal situation, maybe the observed performances are stable !

As an example, let us consider a benchmark from Spec OMP (called swim). This application is compiled with two compilers (Intel cc and gcc), and is run in ideal execution environment on a local machine. We have a physical access to the machine to modify BIOS options (no hardware prefetching, no hyperthreading, etc), to cool the machine if needed, and we have root access to control the concurrent process and OS services (switch off dynamic voltage scaling for instance). We repeat the execution of the application 35 times, and we test different thread affinities: none (Linux scheduling), scatter and compact. The 35 execution times are visualised with violin plot for each configuration in Figure 2. As can be seen, if we fix thread affinity, and if we have full

[1]The Violin plot is similar to box plot, except that they also show the probability density of the data at different values.

control on the machine, we are able to stabilise the execution times of a parallel application. This is possible because these experiments are done in a research laboratory devoted to HPC and compilation technology, all the machines and OS were fully under our control [MtATB10].

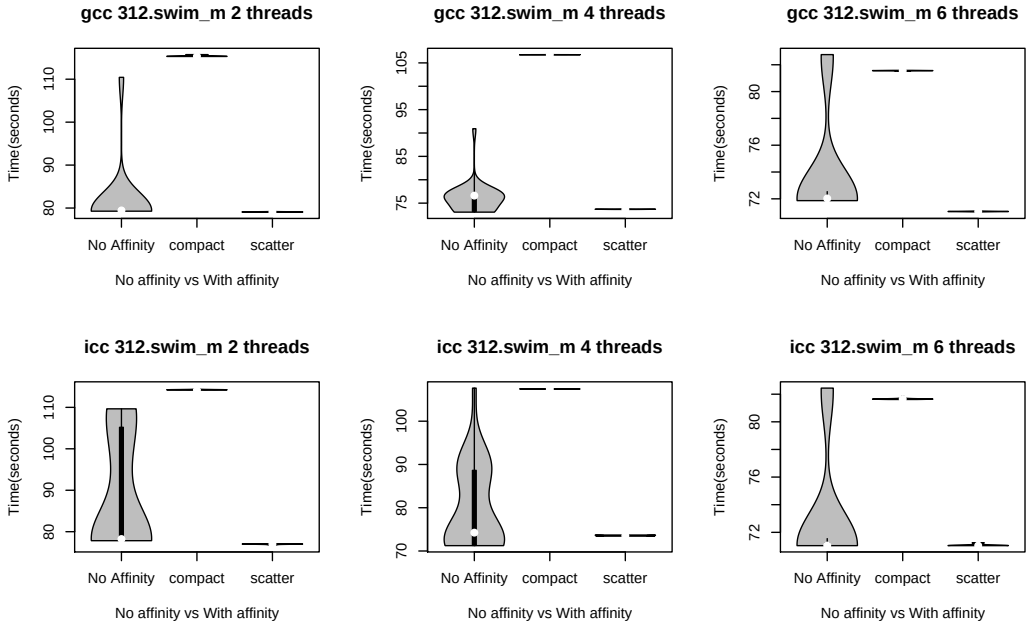


Figure 2: Running a benchmark of HPC application on a fully controlled machine (research project)

1.1.3 Nowadays usage of computers in everyday life devoted to production computing

Nowadays, users execute codes on machines (local or remote) on which they have no control. Concurrent processes, other connected users, operating system maintenance actions, hardware behavior, graphical interactive environment, virtual machines, these features influence the program performances, which become severely unstable. So program performances observed in ideal situations can hardly ever be reproduced by everyday user. Consequently everyday users need rigorous statistics that allow them to take decisions to select their *best* code versions, to make real and not ideal performance analysis, etc.

1.2 Some known factors that make program performances vary

When one considers a fixed binary code and input data, and if such code is executed n times on exactly the same machine with the same OS with the fixed data input, then n distinct performance measurements would be observed. Here, performance may be execution time, energy consumption, memory usage, network traffic, Instructions Per Cycle (IPC), Cycles Per Instruction (CPI), etc... Usually, execution time is the most important and interesting performance metric in practice, it is easily measured by OS commands or hardware performance counters.

With physical experiments (without simulation), performance measurement tools have naturally limited precision and sensitivity, so it is common to observe some variability as in any physical measurement. Such variability may be considered as noise due to non perfect measurement process. However, additional uncontrolled factors induce substantial variation in the performance. Here are below some known categories of such factors.

1.2.1 Technological factors

These factors are due to electronics design and physical phenomena.

- Nowadays, the clock frequency of a chip or a socket is not constant during program execution. The hardware can decide to over-clock the processor or to reduce its frequency, depending on the temperature and on the machine workload. So in practice, a program can speed up ou slow down many times during its execution because of CPU frequency modulation.
- Input/Output execution times depend on the quality of the hardware (mechanical movement, disks, etc.). Some input/output devices do not guarantee constant access time. So from one program execution to another, the time spent for input/output may vary.
- Some peripherals are asynchronous, or communicate asynchronously with the processor. RAM, network, disks, PCI interfaces and hardware cards, do not guarantee constant read/write access time from one execution to another.

1.2.2 Micro-architectural factors

These factors are due to the internal design of processors, not necessarily visible to the software or to the programmer.

- Out of order execution mechanism of superscalar processors re-schedules dynamically the program instructions during execution.
- Hardware branch prediction may lead to variable execution time of branch instructions.
- Hardware data prefetching may pollute the different level of caches.
- Memory hierarchy effects (multiple level of caches, some are shared between cores, some are private).

1.2.3 Software competition to access shared resources

A parallel application may have multiple processes or threads that execute concurrently and compete for resources. From one execution to another, the processes and thread do not necessarily compete for the same resources exactly at the same moment, consequently the scenarios of resource sharing change.

- A computer with a shared memory between multiple cores have to arbitrate between conflicting access. So from one execution to another, there is no guarantee that a thread or a process has access to the shared memory exactly at the same time as in previous executions.
- Network routing protocols do not necessarily guarantee exactly the same network traversal time from one execution to another.

1.2.4 Operating system factors

From one execution to another, the OS state changes. So the decisions taken by the OS may change. For instance, the process and thread scheduler and the memory allocator can take different decisions from one execution to another, producing distinct overall program execution times.

- NUMA memory may exhibit severe performance variability, if the operating system does not take care of it during memory allocation. Indeed, from one program execution to another, program pages may be allocated on distinct NUMA nodes, which dramatically change the duration of memory accesses.

- The exact memory location where a program is loaded has a direct consequence on program execution speed. For instance, the performance memory disambiguation mechanism (micro-architectural feature) is sensitive to the starting adresses of accessed data [JLT06, LJT04]. Also, the exact memory location on which the program stack is allocated has also a direct consequence on program execution speed [MDSH09].

- Thread and process scheduling policies implement coarse grain scheduling algorithms to optimise CPU usage in general. However, such scheduling policies do not consider micro-architectural ressources and interactions to take decisions. Indeed, the memory hierarchy of caches is hidden to the OS scheduler, while some caches are shared between cores. So from one execution to another, distinct scheduling decisions lead to different cache usage between threads, which lead to distinct execution times [MtATB10].

- Operating systems may mount some disk partitions using the network interface (NTF disk for instance). Consequently, accessing the files stored on these network disks imply network traffic, Input/Output execution times become subject to unpredictable performance variations.

1.2.5 Algorithmic factors

Some parallel programs implement algorithms that assign thread workload differently from one execution to another. So some threads have more or less workload from one execution to another, leading the synchronisation barriers (meeting points) to finish early or lately depending on the workload of the critical path thread.

Also, some parallel algorithms are designed to be non deterministic, they behave differently from one execution to another.

When faced to unpredictable behaviour, people try to rely on statistics (based on probability theory) to take decisions. Unfortunately, some people forget important assumptions to know, presented in the following section.

1.3 Fundamental assumptions to know before using statistics that may bring to wrong conclusions

Some people do statistics in a wrong way, and then they become disappointed when they took wrong decisions based on statistics. Statistics and probability are rigorous science, that few users understand in details and make correct usage of them. Below we give three hidden assumptions that people forget when they use statistics.

1.3.1 Independence of the measurements

A hidden assumption that people forget is the independence of the measurement. Usually, program performance data are collected by repeating the execution of a program multiple times on the same machine. But we know that the i^{th} execution of a program may influence the $(i+1)^{th}$ execution on the same machine: OS caches, CPU temperature, etc. are modified by a previous program execution and may influence the performance of a following program execution. We can reduce this dependance by rebooting the machine after each execution, but no one wants to waste its time for doing it, especially if this would not be sufficient to guarantee independence between measurement.

Indeed, in statistics, we cannot prove that measurements are independent, we can just assume it by taking some experimental precautions. For instance, the experimental protocol in biology impose to do n distinct experiments on n distinct white mouses, and all white mouses must be identical (DNA copies). In computer science, this means that executing a program n times would require to use n identical machine with identical software environment, each machine within n would execute the program one time. In practice, nobody does this. We all execute a program multiple times on the same machine, which do not necessarily guarantee the independence between the measurements. If the measurements are not independent, the conclusions made on statistics may be biased: however, it is known in the statistical community that many statistical procedures are robust to a mild amount of dependence between the observations, so we consider that it will not be an issue for our study.

1.3.2 Continuous and discrete random variable models, ties in the data

Random variables in probability and statistics are divided into two categories: continuous variables and discrete variables. Depending on the nature of the random variables, we cannot necessarily do the same statistical analysis. Using statistical tests and protocols proved for continuous variables on discrete variables, or vice-versa, could lead to wrong conclusions. Continuous variables are quantities that may have unbounded precision: for instance, execution time is a continuous quantity (at least when it is not too severely rounded when recorded, see below). Even if measurement tools provide integral values, these integral values are due to sampling of physical continuous values. Inherently, the execution time is continuous. So we can use statistics with continuous models, as we present in this report.

If the nature of the measured performance is discrete, then we cannot use continuous models, and all the statistics presented in this report cannot be used. For instance, if some-one is interested on performances such as "the number of accessed files", or "the number of allocated memory pages", all such performance metrics are discrete. If continuous variables are used to model discrete variables, then the final conclusion may be wrong. In this report, we are interested only in continuous variables.

Finally, another less trivial but certainly very common problem can occur when analysing performance results: ties in the data, *i.e.* identical values. If there are only a very small portion of the data which are tied, then this should not have too much consequences on the subsequent analysis. But if the data contains many ties (this is generally due to a too severe rounding during the data collecting step), this can seriously compromise the validity of the statistical analysis: our opinion is that, without this ties problem, the performance of the statistical procedure we propose in this work will be even better than it already is (as presented in Section 5.3).

After the above recalls on fundamental assumptions ins statistics, the next section present some considerations to take when we use statistics for analysing and comparing program performances.

1.4 Statistical considerations related to program performance evaluation

Until then in this introduction, we pointed out that uncontrollable sources of variability of the program execution time may be fairly numerous in the computer science practice. It is natural to consider that the better we can grasp the program performance variability, the more comfortable we will then be when reporting or comparing program performances.

There exist different work in the literature which address the problem of program performance evaluation, based on rigorous statistical procedures (with varying degrees of sophistication and rigour though), and Section 7 reviews some of them. The purpose of this work being to propose a new approach for the evaluation and the comparison of program performance (focusing on program execution times, but the approach is more general and can be applied to any kind of continuous performances).

1.4.1 Program performance variability is generally not of simple Gaussian type in practice

It is important to remark that many people still think that the variability of program performance is due to a sort of noise, and that the performance can be considered as an average value plus some noise term varying around zero (therefore, the performance value could be modelled as a Gaussian distribution, and reduced to its mean or median value). Indeed, this is not true. The observed data distribution is often multi-modal. This variability is very often not of that simple type, and the use of simple indicators, such as the mean execution time, can be misleading or let the user miss important features about the performance.

Concerning the non-Gaussian nature of most execution times data, we can rely on statistical tests to assert our affirmation. Considering thousands of various samples available to us (see Appendix B for the details about these data), we used the Shapiro-Wilk normality test to check whether they can be considered as non-Gaussian. This normality test computes for every sample a probability, named p-value, which is the risk of mistaking when rejecting the normality assumption. In other words, if the p-value is low (less than a risk budget, for instance 5%), we can reject the normality assumption with a low risk. If the p-value is high (exceeding the risk budget), we should not reject the normality assumption, and hence we accept it.

Figure 3 illustrates all the p-values we computed on many thousands of samples. The p-values are reported with a histogram and with a cumulative distribution. As can be seen, most of the p-values are low, so we can rather safely reject the normality hypothesis most of the time. For instance, if we consider a risk of 5%, we find that 67% of the observed program performances do not follow Gaussian distribution. If the risk is 10%, 71,45% of the observed program performances can be considered as not following a Gaussian distribution.

So what is the consequence if the performances of a program do not follow a Gaussian distribution ? In theory, it means that some statistical tests designed for Gaussians (such as the Student t-test) would compute a wrong risk level. Likewise, the well known formula that com-

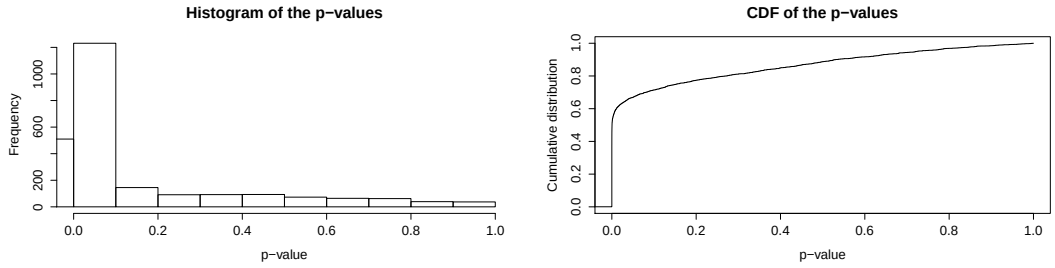


Figure 3: Shapiro-Wilk test applied on program performances: p-values are mostly very low, so program execution times do not follow Gaussian distribution.

putes the confidence interval of an average would correspond to a wrong confidence level. Of course, if the data sample is large enough, and nobody is able to define how *large* it should be, the error must be asymptotically bounded.

1.4.2 What if performances do not vary in practice ?

Under some software configuration on some specific processor architectures, the observed program performances do not vary. For instance, it is possible that the same execution time is observed, even at the processor clock cycle granularity. This situation is indeed possible in practice. If all performance data are identical, we cannot do statistics because the empirical variance is zero. So in this situation, we can naturally admit that the performances are stable and we can summarise them with a single number without doing advanced statistical analysis.

Statistics can be decomposed into two main families. Parametric statistics and non-parametric ones. The next section explains this.

1.5 Parametric versus non-parametric statistics

In this section, we will do our best explaining the differences between the parametric and the non-parametric approaches of the statistical practice. As a matter of fact, one aim of this work is to propose a suitable and flexible *parametric* modelling for program execution times, and to assess whether it is more efficient and reliable than a non-parametric modelling strategy, or not.

In short, rigorous statistical methods fall into two main families: the parametric approach, to which the well-known t-test of Student belongs (at least its small sample version), and the non-parametric approach, to which the also well-known Wilcoxon-Mann-Whitney or chi-square test of independence belong.

Now, more generally, what does the word parametric mean for a statistical model ? This is fairly simple : if $\mathcal{X} = (x_1, \ldots, x_n)$ denotes some execution times data, we suppose that they are independent realisations of some probability distribution having a probability density function (p.d.f.) f_X. The approach will be parametric if we suppose that this p.d.f. belongs to some family of distributions f_θ where θ is a vector parameter composed of d real-valued sub-parameters, which exact values are unknown: d is the dimension of the model. Since the set of all possible dis-

tributions is infinite dimensional, we thus reduced the problem of estimating f_X to the problem of estimating a finite-dimensional parameter θ: this is the main purpose of the parametric approach.

Consequently, the advantage of this parametric approach is that, in general, as soon as a suitable estimator for the parameter θ is available, every possible quantity related to the execution time distribution can be estimated fairly easily, including the uncertainty associated with this estimation (at least approximatively). The drawback of the parametric approach, however, is that the validity of the subsequent statistical analysis may depend on this assumption that the true distribution of the observations is issued from that particular parametric family: when a different family of distributions is considered, the process of designing estimators and tests needs to be performed again, and this can be quite time-consuming and unefficient.

On the other hand, the non-parametric approach supposes no particular shape for the underlying probability distribution f_X of the data, and therefore the subsequent statistical analysis should be valid whatever f_X is, *i.e.* it should be valid for any kind of data. This makes this approach very appealing: however, the statistical results issued from this approach are often only valid for a sufficiently large size of the data sample and, moreover, the estimation of a given quantity of interest needs a specific technique or approach, without a global view. In addition, the non-parametric approach has the reputation of yielding statistical results which are less efficient than those issued from a parametric approach (for a given common sample size); this will be explored in our context of program performance evaluation.

Let us also mention that, despite the appeal of the non-parametric approach, the parametric approach is still very heavily used in the scientific activities involving statistics thanks to numerous reasons:

(i) parameters of a parametric model can often be interpreted (which helps understanding and summarising the data);

(ii) some computations cannot be performed, or some mathematical results cannot be proved, without parametrically specifying some parts of the model (for example, the perturbation part);

(iii) for really large datasets, it is often proved that the parametric techniques are indeed more accurate than their non-parametric alternatives;

(iv) for high dimensional data, non-parametric techniques may suffer from a lack of accuracy which is called the "curse of dimensionality", and which is out of the scope of our present research work.

A final note: it is useful to recall that statistics and probability bring tools to help make decisions (but in the end, the user is the real decision-maker), they bring insights about the underlying and unknown features of the data at hand, along with some probability of error, some risk, which is valid only under a precise set of formal assumptions. If these assumptions are not checked beforehand, or taken into account, the resulting information may be misleading. This note of caution applies to the parametric approach (which is sometimes used without checking the goodness-of-fit of the considered model) as well as to the non-parametric approach (which sometimes needs quite a large number of data values to be efficient).

1.6 Report plan description

The above introduction motivates our research effort. We define and recall the notations of some basic notions in statistics in Section 2. After demonstrating in the introduction that program performances do not follow Gaussian distribution, we study a new distribution modelling based on Gaussian mixtures in Section 3. We chose Gaussian mixture (GM) as a target data distribution model because we observed in practice that sample distributions are multi-modal. Building a Gaussian mixture model from data is called clustering, that we present in Section 4. In Section 5, we describe a statistical method which checks whether Gaussian mixture model fits well some given experimental data. Then based on Gaussian mixture modelling, we propose new program performance metrics based on parametric and non-parametric statistics in Section 6. Some state of the art of program performance analysis using statistics is summarised in Section 7. Limitations and future research plan are presented in Section 8. Finally we conclude this research report with a synthesis and some opinions. In the appendix, the reader may find the description of our experimental data and the software that we implemented using R. This software is called VARCORE, it is free and open source.

2 Basic notations and definitions in statistics and probability theory

Let $X \in \mathbb{R}$ be a continuous random variable. Let $\mathcal{X} = (x_1, \ldots, x_n)$ be a sample of observations issued from the same distribution as X. The following items recall basic notations and definitions used in this document. Below, x denotes some arbitrary real number.

Absolute value is noted $|x|$.

Indicator function is noted $\mathbb{1}$, it is defined by $\mathbb{1}_A = 1$ if the relation A holds true, and $= 0$ otherwise. For example, $\mathbb{1}_{x \leq y} = 1$ if $x \leq y$, and 0 otherwise.

Probability density function (PDF or p.d.f.) of the random variable X is noted f_X. This well-known function describes the relative likelihood that this random variable takes on a given value (it is often approximately presented as $f_X(x) \simeq \mathbb{P}[x \leq X \leq x + dx]/dx$)

Probability is noted $\mathbb{P}[.]$. For instance, for $x \in \mathbb{R}$, the probability that $X \leq x$ is $\mathbb{P}[X \leq x] = \int_{-\infty}^{x} f_X(t)\,dt$.

Probability under hypothesis is noted $\mathbb{P}_{H_0}[.]$. It is equal to the probability $\mathbb{P}[.]$ under the assumption that some hypothesis H_0 (about the theoretical distribution of X) is true.

Cumulative distribution function (CDF or c.d.f.) of the random variable X is the function noted F_X defined by $F_X(x) = \mathbb{P}[X \leq x] = \int_{-\infty}^{x} f_X(t)dt$.

Empirical distribution function (EDF) of the sample $\mathcal{X}$ is the function, noted $\widetilde{F}_{\mathcal{X}}$, built from the observations $\mathcal{X}$ and which estimates the true CDF F_X: for every $x \in \mathbb{R}$, $\widetilde{F}_{\mathcal{X}}(x)$ is defined as the proportion of the observations $x_1, \ldots, x_n$ which are lower or equal to the value x (it is a step function which jumps by $1/n$ every time it reaches one of the observations x_i, $i = 1, .., n$, until it finally equals 1)

The expected value (or theoretical mean) of X is noted $\mu_X = \mathbb{E}[X] = \int_{-\infty}^{+\infty} x f_X(x)dx$.

The sample mean of the sample $\mathcal{X}$ is noted $\overline{\mathcal{X}}$, it is equal to $\frac{1}{n}\sum_{i=1}^{n} x_i$.

The theoretical median of the variable X is noted $\text{med}(X)$, it satisfies $F_X(\text{med}(X)) = \frac{1}{2}$.

The sample median of the sample $\mathcal{X}$ is noted $\overline{\text{med}}(\mathcal{X})$.

The theoretical variance of the random variable X is noted $\sigma_X^2 = \mathbb{E}\left[(X - \mathbb{E}[X])^2\right]$

The sample variance of the sample $\mathcal{X}$ is noted $s_{\mathcal{X}}^2$, it is equal to $\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \overline{\mathcal{X}})^2$

The standard deviation of the random variable X is noted $\sigma_X = \sqrt{\sigma_X^2}$

The sample standard deviation of the sample $\mathcal{X}$ is noted $s_{\mathcal{X}} = \sqrt{s_{\mathcal{X}}^2}$

The Gaussian PDF is the p.d.f. of the Gaussian distribution $\mathcal{N}(\mu, \sigma)$, defined by $\varphi(x; \mu; \sigma) = (2\pi\sigma^2)^{-1/2} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$, which has expectation μ and standard deviation σ.

The standard Gaussian PDF is the p.d.f. of the standard Gaussian distribution $\mathcal{N}(0, 1)$, and it is denoted by the function $\varphi(x) = \varphi(x; 0; 1)$

The Gaussian CDF is the c.d.f. associated to the Gaussian distribution $\mathcal{N}(\mu, \sigma)$, it is denoted by the function $\Phi(x; \mu; \sigma) = \int_{-\infty}^{x} \varphi(t; \mu; \sigma)\, dt$

The standard Gaussian CDF is the c.d.f. of the standard Gaussian distribution $\mathcal{N}(0, 1)$, it is denoted by the function $\Phi(x) = \Phi(x; 0; 1)$

Other notations will be introduced later in the text (in particular notations related to Gaussian mixtures, estimations of their parameters, and the performance metrics).

The next section explains and defines the theoretical probability model based on Gaussian mixtures.

3 Gaussian mixtures

In the previous sections, we explained the importance of adequately modelling execution times and evoked the possibility of finding a parametric family which will suit our needs. This means that we look for a family of probability distributions from which we think that most of the execution times are issued.

The central idea of our work stems from the following remark: in our own experience in parallel and sequential application performance analysis, the variability of such data is not necessarily of the "deviation around a single mean value" kind, it often exhibits a clear "clustering" pattern: in other words, the execution times often vary from each other by clustering around two or more central values. Therefore, we cannot choose for our modelling family classical families of distributions such as the Gaussian, Exponential, Gamma or Weibull family, which are by nature monotonic or unimodal. And summarising the data by a single mean or median value can be misleading for the end-user, when comes the time to compare different code optimisation methods or code versions: he could miss important features of the execution time distribution.

Examples of real execution times data are provided in Figure 4, plotted with histograms and sample density distributions. These data are obtained from three well known SPEC benchmarks executed 35 times with reference input on a dedicated Linux machine (no other running application except the benchmark). The X-axis represent the observed execution times in seconds, and

the Y-axis represent their frequencies (sample density).

In practice, many parallel HPC applications executed on supercomputers or on HPC machines exhibit such multi-modal execution times distributions, even if we keep fixed the input and if we execute the application on a dedicated machine. This observation remains true even if we use different compilers and distinct code optimisation flags, and even if we run the executed application with different numbers of threads and affinity strategies. It is in practice very difficult to obtain stable performances. Let us consider the example of a real-world HPC scientific application, called CONVIV, as presented in Section 1.1.1. Figure 5 plots the observed execution times in many configurations (thread numbers, affinity, compilers). The executions times are plotted with histograms and also with estimated sample density functions: the curves that estimate the sample density functions are automatically plotted using the R software, and may not fit very well the histograms. For instance, look at the leftmost figure, we clearly see that the left tail of the curve does not match the left packet of the histogram. It is possible to modify the plotting of a sample density function by tuning some graphical parameters: however the modelling that we propose in this work will provide better curve fitting than this graphical automatic one.

We propose to model the execution times by mixtures of Gaussian distributions: this family of distributions has proved to be an essential tool in many areas of scientific activities for many years now (biology, engineering, astronomy, among many others), particularly in the image analysis and pattern recognition fields. Mixtures of Gaussian distributions is a highly flexible family of distributions which can fit a great variety of data: it is not only naturally adequate for modelling data which exhibits clusters, but it can also (to some extent) handle the problem of possible skewness in the data, despite the symmetry of the Gaussian components of the mixtures. It can also efficiently model multivariate data, but in this work we will limit ourselves to univariate data.

In the following section, we introduce the notion of Gaussian mixture models.

3.1 Definition of the Gaussian mixtures family

Remind that we consider data $\mathcal{X} = (x_1, \ldots, x_n)$ which are independent realizations of a probability density function (p.d.f.) f_X. We will say that the data are issued from a finite mixture of Gaussian distributions (or simply a Gaussian mixture, which we will abbreviate by GM from now on) if f_X is equal to some p.d.f. $g_{\theta,K}$ (parametrized by θ and K described below) of the form:

$$\boxed{g_{\theta,K}(x) = \pi_1\varphi(x;\mu_1;\sigma_1) + \ldots + \pi_K\varphi(x;\mu_K;\sigma_K) = \sum_{k=1}^{K} \pi_k\varphi(x;\mu_k;\sigma_k)} \quad (1)$$

where:

- $\pi_1, \ldots, \pi_K$ are the mixture weights, which are positive and sum to 1;
- $\mu_1, \ldots, \mu_K$ and $\sigma_1, \ldots, \sigma_K$ are the mean values and standard deviations of the mixture individual components;
- $\varphi(\,\cdot\,;\mu_k;\sigma_k)$ denotes the p.d.f. of the Gaussian/normal distribution $\mathcal{N}(\mu_k, \sigma_k)$;
- K is the (integer) number of components in this mixture;

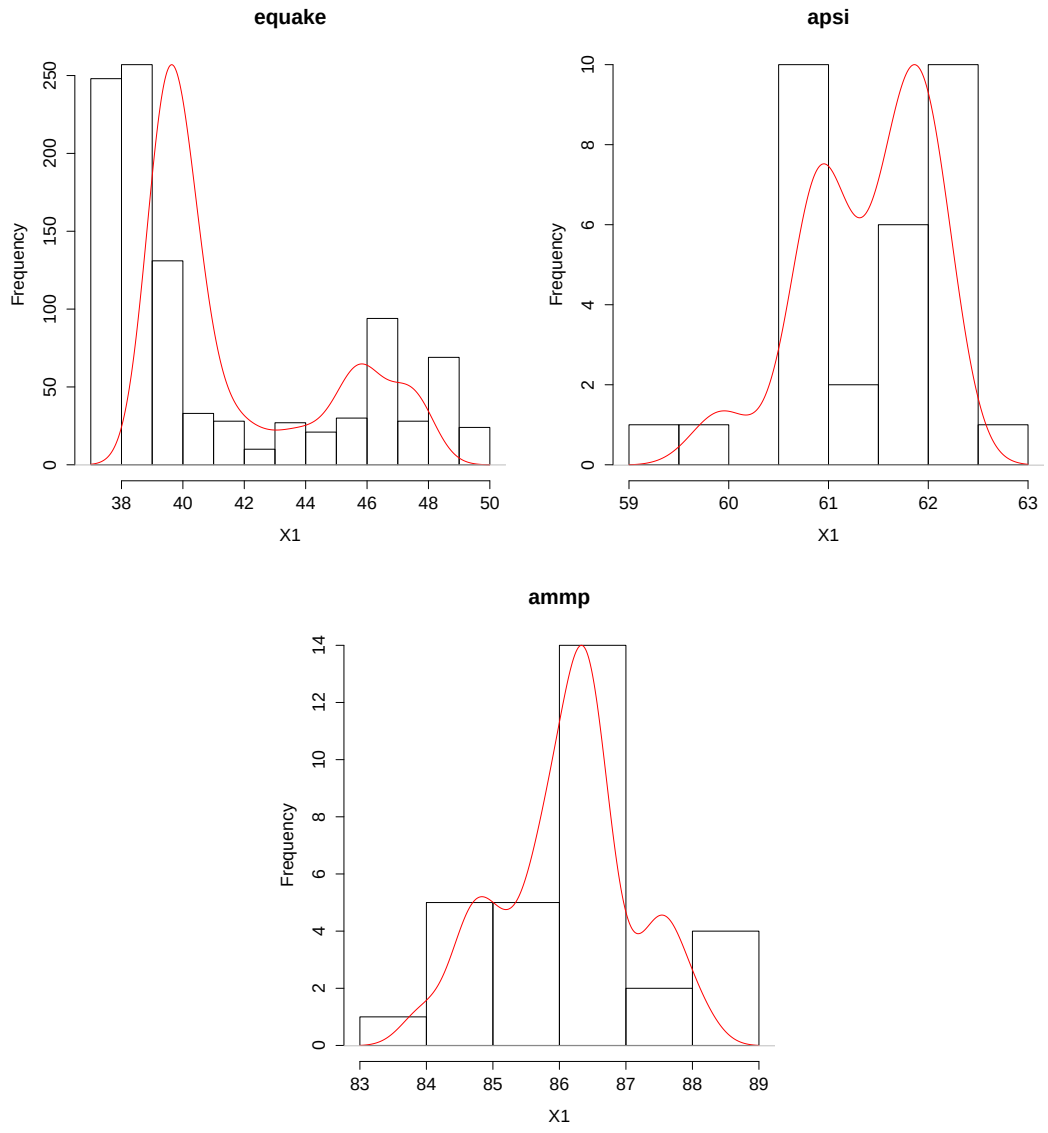


Figure 4: Examples of execution times distributions for three SPEC benchmarks

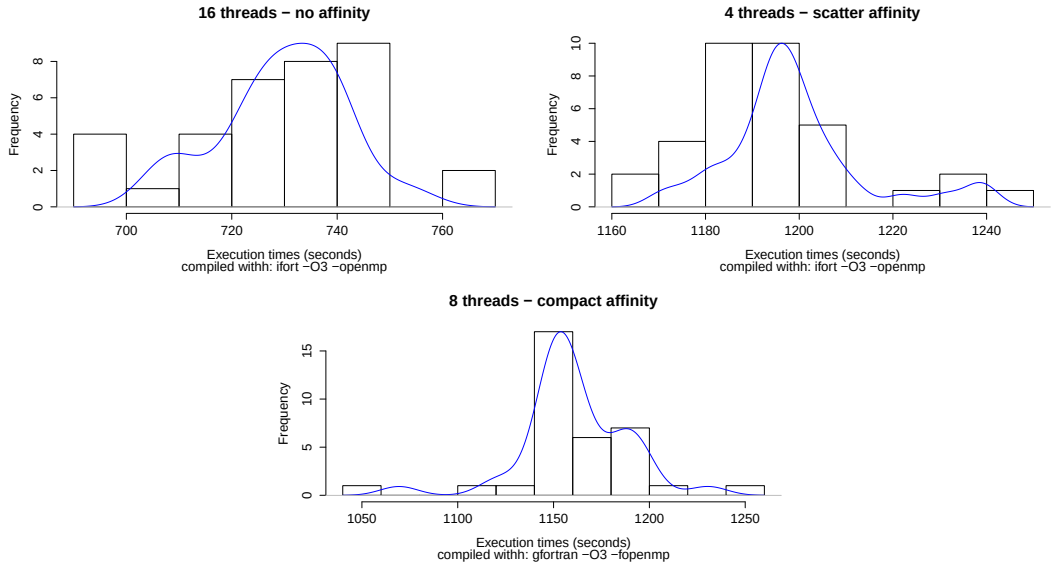


Figure 5: Examples of execution times distributions of CONVIV

– θ is a vector gathering all the parameters (except K) in a single notation,

$$\theta = (\pi_1, \ldots, \pi_K \,;\, \mu_1, \ldots, \mu_K \,;\, \sigma_1, \ldots, \sigma_K)$$

Examples of GM probability distributions are illustrated in Figure 6: the plain line corresponds to $K = 3$ and $\theta = (0.5, 0.35, 0.15\,;\, 2, 8, 25\,;\, \frac{1}{2}, \frac{1}{2}, \frac{1}{2})$, the dashed line to $K = 3$ and $\theta = (\frac{1}{4}, \frac{1}{2}, \frac{1}{4}\,;\, 4, 16, 23\,;\, 1.5, 2, 2)$, and for the dotted line $K = 2$ and $\theta = (0.35, 0.65\,;\, 10, 14\,;\, 2, 4)$.

We will denote by $\mathcal{F}^{\mathrm{GM}}$ the set of all mixtures of Gaussian distributions, and say that X is GM-distributed if its cumulative distribution function (c.d.f.) F_X belongs to $\mathcal{F}^{\mathrm{GM}}$, which means there exists some parameters K and $\theta = (\pi_1, \ldots, \pi_K \,;\, \mu_1, \ldots, \mu_K \,;\, \sigma_1, \ldots, \sigma_K)$ such that

$$\boxed{\forall x \in \mathbb{R},\ F_X(x) \ \text{ equals } \ F_{\theta,K}(x) = \sum_{k=1}^{K} \pi_k \Phi(x; \mu_k; \sigma_k)} \tag{2}$$

which is (of course) itself equivalent to f_X being equal to the density $g_{\theta,K}$ defined in Equation 1. Naturally, the more components the mixture has, the more flexible the shape of the distribution can be (but this has a cost: the model has more parameters to be estimated).

In practice, our modelling problem will be the following. If we consider this model of Gaussian mixtures, the task will be to estimate the parameters θ and the value K from the data $x_1, \ldots, x_n$. In fact, the process will be the following: for each possible value of K (starting from 1, and ending at some reasonable value K_{max}), the parameter θ (which dimension depends on K) will be estimated, and only then an adequate value of K will be chosen, according to some criterion (which will be detailed in Section 5). Note that, although these GM models have been imagined by statisticians for decades, for a long time the process of estimating their parameters was a problem which could be hardly solved in practice. It is only since the breakthrough work by

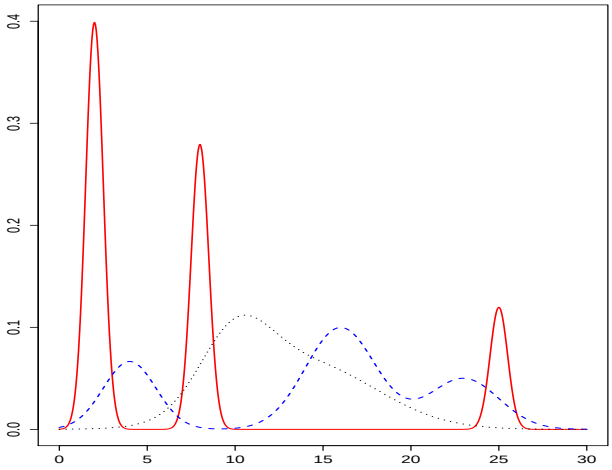


Figure 6: Examples of Gaussian mixtures distributions

Dempster, Laird, and Rubin ([DLR77]) on the so-called EM algorithm (see Section 4) that these models became estimable and began to be used in a huge variety of scientific domains and problems, either for modelisation or for other purposes (for instance, in image analysis, the Gaussian Mixture Models or GMMs are ubiquitous). One could consider mixtures of other families of distributions[2] (Weibull, log-normal, Pareto, Student, etc) but in this work mixtures of Gaussian distributions will prove to be more than enough in order to adequately model execution times data.

3.2 Motivation for Gaussian mixtures modelling in our context

Now that we have defined the Gaussian mixtures distributions, let us take a moment for motivating their appearance in our context, and describe a convenient way of viewing Gaussian mixtures.

The random variable X of interest, which is observed, is the execution time for instance (or any other continuous performance under study). Now suppose that there exists some hidden and unobserved variable C which can take the values $1, \ldots, K$ with respective probabilities $\pi_1, \ldots, \pi_K$, and that the distribution of X is affected by the value this latent variable C takes, in the following way: if C takes the value k, then X follows the Gaussian distribution $\mathcal{N}(\mu_k, \sigma_k)$ (*i.e.* X follows a Gaussian distribution which parameters depend on the value that C takes). It is easy to check that such construction leads to X following the distribution defined in Equation (1), with clusters corresponding to the different values that the hidden variable C may take.

This variable C, unobserved, may correspond to some sort of state of the operating system or processor, which affects the speed of execution of the program: depending on this state, the execution time X is issued from a different Gaussian distribution. This could be a good explanation of why clusters are observed in practice for execution times (as well as a very good

[2]there is even a recent literature on non-parametric mixtures (see for instance the survey [CHL15]), which concern is however more about mixtures of multivariate distributions, since Gaussian modelling is often inadequate in dimension greater than 1.

motivation for considering mixture distributions for modelling these execution times): the main variability between the execution times would be due to the value of the state C that the system had when the time was observed, while the minor variability (around the mean execution time in that particular cluster) observed would be due to other minor factors of the operating system. For example, if we look at the top-left histogram in Figure 4, we see that execution times are clustering around 2 values (roughly 39 and 48), suggesting that when the system is in some state, then the execution times are observed around the value 39, while when the system isn't in that state, then the execution times are more likely to be around the value 48: we could summarise this by defining the latent variable C to be equal to 1 in the first case, and equal to 2 in the second case.

Let us finally make a technical remark. The GM model defined in Equation (1) can be split into two main categories: either the different components share the same standard deviation $\sigma_1 = \cdots = \sigma_K = \sigma$, or the different components are allowed to have different standard deviations. We can refer to these two categories as the "common variance model" and "distinct variances model" respectively. The second possibility sounds like a much more flexible model than the first one, but it has the disadvantage that it can become non-estimable, and provoke instabilities or errors in the clustering algorithm (*i.e.* the algorithm which estimates the underlying Gaussian mixture). It thus turns out that it is an important issue in practice (especially if the sample size is small); the clustering algorithm we will use later on will allow for both possibilities.

After having defined the Gaussian mixture model, we present in the next section a method for estimating ("building") such a model from a data sample.

4 Clustering method

In this section, we explain how the Gaussian mixture model is concretely estimated from the data. A detailed explanation would require many mathematical details and prerequisites, so we decided to provide only an overview of the principle and concepts at stake, so that the reader can grasp the main idea and understand the issues of the crucial step: the choice of an "adequate" number K of components in the Gaussian mixture.

Our aim is thus, for the moment, to estimate the parameters of the distribution described in Equation (1), from which we assume our data are issued: the parameters are the clusters weights $(\pi_k)_{k=1..K}$, the clusters means $(\mu_k)_{k=1..K}$ and the clusters standard deviations $(\sigma_k)_{k=1..K}$, and they are gathered in one single notation, θ (which is $3K$-dimensional). The use of this estimated GM model for providing concrete statistical insight about the data at hand will be the subject of Section 6. For the moment, the most important is to consider the clusters means as the central values around which the performance values (execution times) tend to cluster.

4.1 Estimation of the parameters of the mixture for fixed K

How then is θ estimated ? Remind that the dimension of θ depends on the value of K, which is fixed for the moment. We naturally adopt a parametric approach, and intend to compute the maximum likelihood estimator $\hat{\theta}$ of θ, which is defined as the (global) maximizer of the log-likelihood function. This function, given the observations $x_1, \ldots, x_n$ and Equation 1, is defined

by:

$$L(\theta) = \ln \prod_{i=1}^{n} g_{\theta,K}(x_i) = \sum_{i=1}^{n} \ln \left(\sum_{k=1}^{K} \pi_k \varphi(x; \mu_k; \sigma_k) \right)$$

Maximising it consists in calculating the various partial derivatives of L with respect to the different parameters π_k, μ_k, σ_k $(k = 1, \ldots, K)$, and equalising them to 0 to obtain the so-called score equations. It is rather clear that any attempt to directly solve these score equations will turn out to be an unsolvable problem, due to the presence of a log of a sum.

This major obstacle was overcome thanks to the approach synthesised in the celebrated *EM algorithm* [DLR77]. The acronym EM means a succession of E-steps (E for Expectation) and M-steps (M for Maximisation), which should lead to obtaining (numerically) the value of the maximum likelihood estimator $\hat{\theta}$. Below, we will only sketch the general idea of the method, and then present the simple final form the algorithm takes when dealing with Gaussian mixtures as we do here (as a matter of fact, the EM-algorithm can deal with mixtures of many other families of distributions, and is also used in several other areas of statistical practice, different from clustering; this adaptability is one aspect of its strength).

The idea of the EM algorithm in the mixture framework is to consider that the hidden, unobserved, variable C mentioned in Section 3.2 (being some sort of label for the data, possibly representing the state of the operating system), is part of a so-called *complete model*, for which the observations would be $(x_1, C_1), \ldots, (x_n, C_n)$ (x_i and C_i being the values that the variables X and C take for the i-th execution of the program, x_i being the observed execution time itself, and C_i being the associated label, unknown and unobserved in practice). A so-called *complete likelihood*, which includes the variables C_i, is then considered. For $j \geq 1$, the j-th E-step consists in computing the expected value of this complete likelihood (the expectation is with respect to the unknown random variable C_i) given the observations and the previous temporary estimation $\theta^{(j-1)}$ of θ; then the j-th M-step consists in defining $\theta^{(j)}$ as the value which maximises this expected complete likelihood. It is proved that, proceeding like this, the value $L(\theta^{(j)})$ is necessarily higher than the previous one, $L(\theta^{(j-1)})$. The algorithm then runs until the gain $L(\theta^{(j)}) - L(\theta^{(j-1)})$ is considered as negligible, and $\hat{\theta}$ is then defined as the current value $\theta^{(j)}$.

Of course, an initial guess $\theta^{(0)}$ must be determined, and (more problematic) the obtained maximum might turn out to be only a local maximum of the function L (if this function is multimodal) or, worse, the function L may be unbounded above, and the maximum likelihood estimator would then be undefined. These issues need to be handled in EM clustering, either theoretically or in practice. In our framework though (mixtures of univariate Gaussian distributions), while these issues are indeed present, they are less critical than in more complex use of the EM algorithm (namely mixtures of multivariate distributions). Moreover, the E and M steps greatly simplify when considering Gaussian mixtures: if we introduce the following notations, for $j \geq 0$,

$$\theta^{(j)} = (\pi_1^{(j)}, \ldots, \pi_K^{(j)}, \mu_1^{(j)}, \ldots, \mu_K^{(j)}, \sigma_1^{(j)}, \ldots, \sigma_K^{(j)})$$

and, for $i = 1, \ldots, n$, $k = 1, \ldots, K$, $j \geq 1$,

$$\alpha_{i,k}^{(j)} = \frac{\pi_k^{(j-1)} \varphi(x_i; \mu_k^{(j-1)}; \sigma_k^{(j-1)})}{\sum_{l=1}^{K} \pi_l^{(j-1)} \varphi(x_i; \mu_l^{(j-1)}; \sigma_l^{(j-1)})}$$

then we have the following simple formulas relating $\theta^{(j-1)}$ to $\theta^{(j)}$: for every $k = 1, \ldots, K$

$$\begin{aligned}
\pi_k^{(j)} &= \frac{1}{n} \sum_{i=1}^{n} \alpha_{i,k}^{(j)} \\
\mu_k^{(j)} &= \sum_{i=1}^{n} \frac{\alpha_{i,k}^{(j)}}{\sum_{i'=1}^{n} \alpha_{i',k}^{(j)}} \, x_i \\
\sigma_k^{(j)} &= \left(\sum_{i=1}^{n} \frac{\alpha_{i,k}^{(j)}}{\sum_{i'=1}^{n} \alpha_{i',k}^{(j)}} \left(x_i - \mu_k^{(j)} \right)^2 \right)^{1/2}
\end{aligned}$$

These formulas will not be proved here, see (for instance) [MP00] for justifications (but more self-contained proofs, specific to this Gaussian mixture case, can be found without too much difficulty in academic course notes, available for instance on the worldwide web).

4.2 Determination of the number K of components of the mixture

Now that the estimation of θ has been explained for a given value of K, let us explain how the number K of components of the Gaussian mixture model can be chosen. In practice, several candidate values are considered for K, and one of them, noted $\hat{K}$, is chosen so that the corresponding GM model best fits the data at hand. The determination of $\hat{K}$ is nearly the most important issue in clustering analysis, and in this work we will adopt a simple and widespread strategy: using the BIC criterion (BIC stands for Bayesian Information Criterion).

The principle of the BIC criterion for determining $\hat{K}$ is the following. If, for a given $K \geq 1$, we note L_K the maximum value of the log likelihood for the model with K components (or, more precisely, the maximum value which is issued from the EM algorithm, which is hopefully the actual maximum likelihood), then it should be easy to conceive that the greater K is, the greater the value L_K will be: indeed, for instance, if we consider the model with $K+1$ components which best fits the data, then it will certainly better fit the data than the best model having K components (maybe not far better, but better all the same). Therefore, choosing K which maximises L_K will not work. The likelihood value needs to be penalised by a value which grows with K, in order to counterbalance the fitting gain that more complex models yield. That is the idea of the so-called information criterions, for instance the BIC criterion: to choose the value of K that minimises the value $BIC(K) = -2L_K + K \log(n)$ (this value of the penalisation term $K \log(n)$ has theoretical justifications, which will not be detailed here). Therefore, if $BIC(\hat{K}) = \max_{K \geq 1} BIC(K)$, then the model with $\hat{K}$ components will be a tradeoff between good data fitting and reasonable complexity. Note that, in practice, the maximum is chosen among a finite number of candidate values, for instance $1 \leq K \leq K_{max}$ (where K_{max} does not exceed 10 in general).

From now on, $\widehat{F}_{\mathcal{X}}^{\mathrm{GM}}$ will denote the Gaussian mixture distribution $F_{\hat{\theta},\hat{K}}$ estimated from the observed sample $\mathcal{X}$, following the procedure we just described above. This notation is important, since we will also deal in Section 5 with GM distributions estimated from samples which are different from $\mathcal{X}$.

The entire algorithm (EM and choice of K) has been implemented in a great variety of languages and statistical softwares. Since we intend to use the R statistical software/language, we need to use some of its existing packages. We chose to use the popular `Mclust` package

([FRMS12]). There exists other packages doing the job (like `Rmixmod` or `EMCluster` for instance[3]) , but `Mclust` is pretty widely used, documented, and has the advantage of handling both common variance mixtures and distinct variance mixtures (see end of Section 3): it can therefore always propose an estimated model, even in situations where the distinct variances model causes problems (as a matter of fact, when all the σ_k are supposed equal, the function $L(\theta)$ always has a maximum, which is not always the case when $\sigma_1, \ldots, \sigma_K$ are estimated separately). Note that, in the common variance framework, the formulas provided above change, we will not detail them here. Concerning the choice of the initial value $\theta^{(0)}$ of θ, the `mclust` function in `R` uses a simple hierarchical clustering step (which will not be detailed here), and overall the function yields very rapidly the values of $BIC(K)$ for several values of K (7 by default) and for the common and distinct variances cases.

References about possible other solutions to the problem of choosing K are provided in the bibliography Section 7.4.

A final note now, related to the "data label" hidden variable C mentioned previously (in Sections 3.2 and 4.1): the desire to correctly model the execution times data is our initial aim, but if the different (estimated) clusters are found to be quite distant from each other, another desire would arise: to associate each data to one of the clusters (noting C_i the number k of the cluster from which the data point x_i is the closest), and therefore creating a new variable C which study could be interesting in its own right (C_i would then be the value the variable C takes for the i-th execution time). This second aim is what is called *clustering* in the litterature, and it is often more important than the first one (*i.e.* the estimation of the parameters of the underlying mixture distribution).

4.3 Experiments on clustering

We have implemented our clustering method using R. The software is presented in Appendix A. The performance data have been collected during many years and are presented in Appendix B. Note that, while we consider execution times as example of study, any continuous performance data can be analysed using our statistical methods (energy consumption, network traffic, input/output latency, throughput, etc.).

There are 2438 samples, each sample contain between 30 and 1000 observed execution times. Our clustering method was applied on each sample with success. The computation time of the clustering method using R is fast, in almost all cases it does not exceed few milliseconds per sample.

When a clustering is applied on a sample, it computes a Gaussian mixture model. That is, for each sample, a number $K \in \mathbb{N}$ of clusters is computed (it is the chosen number of Gaussian components selected by the BIC criterion described in the previous section). Figure 7 illustrates the histogram of the obtained numbers of clusters. The median value is equal to 2, which means that half of the samples can be modeled with mixtures of 2 Gaussians, and half of the samples can be modeled with mixtures of more than 2 Gaussians. The third quartile is equal to 3, which means that 75% of the samples are modeled with mixtures of 1, 2 or 3 Gaussians. The maximal observed number of clusters is 9, which means that there are samples which require a model with 9 Gaussians.

[3]see the page *https://cran.r-project.org/web/views/Cluster.html*, which contains an updated list of the various R packages which are related to mixture modelling

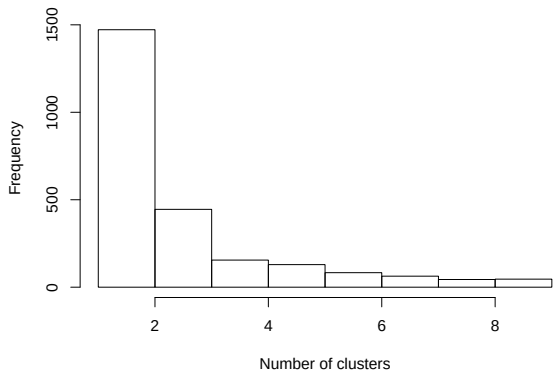


Figure 7: Histogram of the numbers of clusters. They are obtained after applying our clustering method on each sample.

Recall that in very rare situation, all the execution times of a sample may be identical. This means that the sample variance is equal to zero. Since no variability is observed, no need to make a statistical analysis. The program performances are stable enough to be reduced to a single number.

After building a Gaussian mixture model from a sample, we need to test if the model fits the data well or not. The next section is devoted to this issue.

5 Checking the fitting of the data to the Gaussian mixture model

Some statistical studies simply accept a graphical validation by visualising the the CDF of the model and the CDF of the sample: if they are close enough each other, then one could conclude that the modelling is satisfactory. For instance, we draw in Figure 8 the empirical CDF (step staircase function) versus theoretical CDF (continuous curve). For both examples, the fitting seems fairly good graphically. Unfortunately such graphical fitting vlidation is not formally evaluated, and sometimes graphics induce mistakes (the observations by eyes may be misleading).

We address in this section the important problem of validating our method of modelling program performances by a Gaussian distribution by studying an automatic method. This problem, known in the statistical domain as "assessing goodness-of-fit of the model" is here motivated, explained and addressed in detail, not only via simulations (Sections 5.1.2 and 5.2), but also through a great variety of real execution times data. The use of the so-called *bootstrap method* for validating our model is one of the main contribution of our work.

5.1 Description of the method

5.1.1 Preliminaries about goodness-of-fit

In the previous section, we explained how the parameters of the Gaussian mixture are estimated from the sample data via the EM algorithm. However, so far we did not address an important issue: what if the program performances were not issued from a Gaussian mixture distribution, but from another family of distributions ?

Let us be clear from the start: since we will rely upon statistical methods, we cannot be absolutely sure that our data are GM-distributed, we can only build a procedure which assesses, with some given level of confidence, whether the data are not fitting the GM family of distributions. We will return to this later in this section, but it is important to understand that a given dataset may reasonably fit a handful of distributions families: all that we want is to build a reliable statistical test which will warn us when the Gaussian mixture model is a bad model for our data. With such a tool, we will be in a position of assessing that a very large amount of execution times data can be reasonably modelled by a Gaussian mixture. This is one of the main goal of our work.

The statistical test we are about to describe is called, in the statistical language, a *goodness-of-fit test*; the most famous of this kind of statistical tests is the χ^2-goodness-of-fit test, which tests whether some given discrete/integer data fit some given probability distribution. We need here a test which applies to continuous data: our goodness-of-fit test will be based on the computation of a distance between the data and the GM distribution estimated from the data. The distance we chose is the well-known Kolmogorov-Smirnov distance between the empirical distribution function $\widetilde{F}_X$ and the estimated GM distribution function $\widehat{F}_X^{\mathrm{GM}}$:

$$KS_{\mathcal{X}} = ||\widetilde{F}_{\mathcal{X}} - \widehat{F}_{\mathcal{X}}^{\mathrm{GM}}||_\infty = \sup_{x\in\mathbb{R}} \left| \widetilde{F}_{\mathcal{X}}(x) - \widehat{F}_{\mathcal{X}}^{\mathrm{GM}}(x) \right| \tag{3}$$

where[4]

$$\widetilde{F}_{\mathcal{X}}(x) = \frac{1}{n}\sum_{i=1}^{n} \mathbb{1}_{x_i \le x} \quad \text{and} \quad \widehat{F}_{\mathcal{X}}^{\mathrm{GM}}(x) = \sum_{k=1}^{\hat{K}} \hat{\pi}_k \Phi(x; \hat{\mu}_k; \hat{\sigma}_k)$$

The underlying idea is the following: if the data are issued from a distribution which is too different from (or cannot be approached by) a GM distribution, then it is clear that the distance $KS_{\mathcal{X}}$ will be large, *i.e* larger than the distance which would be computed if the data actually came from a GM distribution.

Example 5.1. *In Figure 8, a sample of size $n = 50$, which shows 4 distinct clusters, is considered. On the top line of the figure, a Gaussian mixture of size $K = 4$ has been estimated from the data, showing good fit, the upper left figure showing the data with the estimated GM density, and the upper right figure showing the empirical and GM estimated cumulative distribution functions. On the bottom line, a Gaussian mixture of size $K = 3$ has been estimated from the same data, and the consequence of this (desired, for the purpose of illustration) bad choice of K is that the fit is not good anymore, since the two clusters on the right have been gathered into a single one: this implies a much larger value of $KS_{\mathcal{X}}$ than in the first situation.*

Our goal is then to define a statistical test of the hypothesis:

H_0 : "the underlying distribution F_X of the data is a Gaussian mixture distribution"

[4]the ∞ symbol in subscript of $\|$ in Equation (3) is a common notation in statistics, meaning that the distance is computed by taking the supremum over all $x \in \mathbb{R}$.

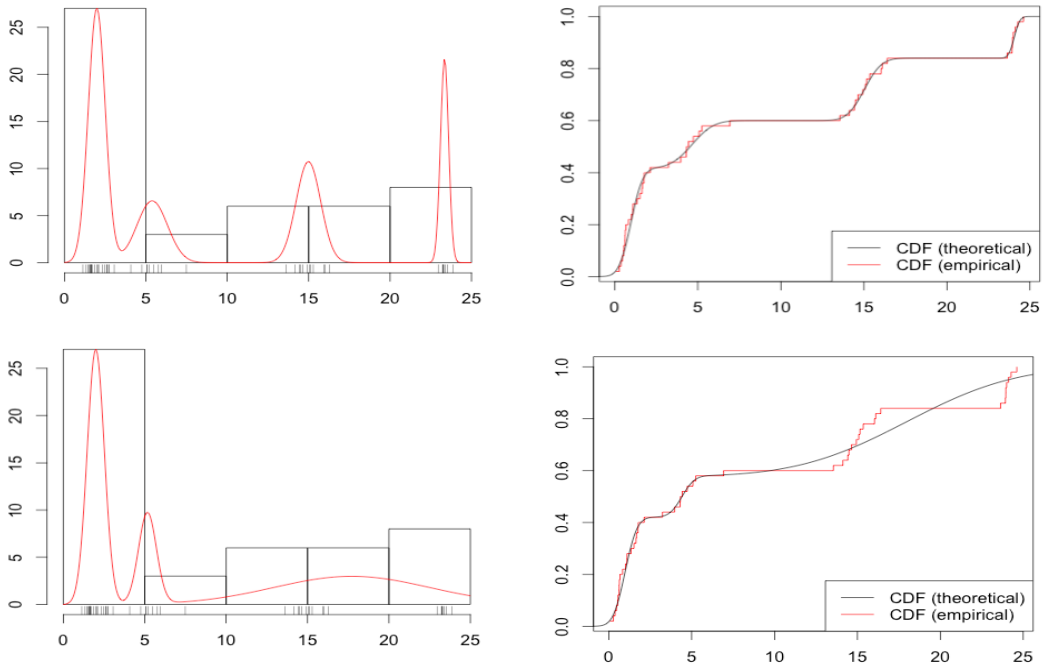


Figure 8: Example of good and bad GM fitting: in the upper part of the figure (raw data, histogram and estimated GM density on the left, estimated empirical and GM cdf on the right), a GM with 4 clusters is fitted to some data, and in the lower part a GM with 3 clusters is fitted to the same data, which leads to bad fitting and a high value of $KS_{\mathcal{X}}$

(which means H_0: "$F_X \in \mathcal{F}^{\text{GM}}$ ") against the alternative hypothesis

H_1 : "the underlying distribution F_X of the data does not belong to the family of Gaussian mixture distributions"

In practice, we will reject H_0 in favor of H_1 (with a given risk α) if the distance $KS_{\mathcal{X}}$ is too large, *i.e.* if $KS_{\mathcal{X}}$ exceeds some given critical value $c = c_\alpha$.

Of course, the crucial point in statistical testing is: what does "large" mean ? what is the risk α associated with a given choice of the value c ? For example, for a sample of size $n = 30$, what is the probability of falsely rejecting H_0 when the chosen value is $c = 0.15$? Answering this question is called *calibrating* the test, *i.e.* to be able to determine a critical value $c = c_\alpha$ associated with a given risk α (which is the risk of rejecting H_0 when the data is actually distributed as a Gaussian mixture). For instance, if for $n = 30$ we have $\mathbb{P}_{H_0}\left[KS_{\mathcal{X}} > 0.189\right] \simeq 0.05$, then a choice of $c = 0.189$ will provide a test of risk 5%, and a choice of $c = 0.15$ will provide a test with risk greater (or much greater) than 5%.

Generally, this calibration is made possible by a mathematical theorem, valid under some conditions (often including conditions on the size of the sample n). For instance, concerning the usual Kolmogorov-Smirnov (KS) test, which tests H_0: "$F_X = F_0$" for some unique and given distribution F_0, a famous theorem[5] states that the sampling distribution of $\sup_{x\in\mathbb{R}} |\widetilde{F}_{\mathcal{X}} - F_0(x)|$ is known (for every n) and does not depend on the choice of F_0. Therefore, for the usual KS test, the value c_α can be determined for any choice of α. However, this theorem is not applicable here, because the hypothesis H_0 is composite, it contains a whole family of distributions and not a single target distribution F_0: therefore we cannot use the usual KS calibration for our test.

This can be better understood by the following remark: when we observe data issued from a given GM distribution $F \in \mathcal{F}^{\text{GM}}$, then the empirical distribution of such data would much better fit the estimated GM distribution $\widehat{F}^{\text{GM}}_{\mathcal{X}}$, than it would fit the original underlying GM distribution F. This is because the estimated distribution is influenced by the particular data we have at hand (this phenomenon is sometimes called *overfitting*). Therefore, the statistic $\|\widetilde{F}_{\mathcal{X}} - \widehat{F}^{\text{GM}}_{\mathcal{X}}\|_\infty$ will tend to take lower values than would do the statistic $\|\widetilde{F}_{\mathcal{X}} - F\|_\infty$; in other words, if we used the usual KS calibration for our test statistic defined in Equation (3), then we would reject the null hypothesis far less often than we would have to, and the risk we would announce would be erroneous (in the sense too low).

5.1.2 Bootstrap as a calibration tool: the KSfit test

This calibration problem of goodness-of-fit tests when parameters need to be estimated beforehand is known in the statistical community[6] (references are provided in Section 7.4), and the solution generally adopted to overcome it is to rely on the *bootstrap methodology*. The idea of the bootstrap is to take advantage of the information contained in the data by resampling it, in order to hopefully estimate properly the sampling distribution of the test statistic. It is somehow a computer-intensive procedure (especially when the sample size is high), but it provides good results, even for small size data: we will investigate the performance of this bootstrap strategy in Section 5.2, let us simply describe it for the moment.

[5] see Theorem 19.3 in [vdV00] for instance

[6] but it is not necessarily the case from casual users of statistical methodology, who are likely to erroneously use in their work the usual KS critical values with a KS distance computed after estimation of parameters...

The goodness-of-fit procedure will be the following:

(*a*) Estimate the presumed underlying GM distribution by $\widehat{F}_{\mathcal{X}}^{\mathrm{GM}}$ from the original sample $\mathcal{X}$;

(*b*) Compute the corresponding test statistic $KS_{\mathcal{X}}$ defined in Equation (3);

(*c*) Repeat for $i = 1, \ldots, N$ (with N at least[7] 200) the following steps:

(*i*) Simulate a sample $\mathcal{X}^{(i)} = x_1^{(i)}, \ldots, x_n^{(i)}$ following the estimated GM distribution $\widehat{F}_{\mathcal{X}}^{\mathrm{GM}}$;

(*ii*) Compute the estimation $\widehat{F}_{\mathcal{X}^{(i)}}^{\mathrm{GM}}$ based on the i-th bootstrap sample $\mathcal{X}^{(i)}$;

(*iii*) Compute the KS distance $KS_{\mathcal{X}^{(i)}}$ (shortened as $KS^{(i)}$) between the empirical distribution $\widetilde{F}_{\mathcal{X}^{(i)}}$ of the i-th bootstrap sample, and the GM distribution $\widehat{F}_{\mathcal{X}^{(i)}}^{\mathrm{GM}}$ estimated from it;

(*d*) Denote by $KS^{(1)} \leq \ldots \leq KS^{(N)}$ the N distances obtained in step (*c*), ordered from the smallest to the highest value. Then:

(*i*) Define the critical value c_α as the value[8] $KS^{([N(1-\alpha)])}$;

(*ii*) Define the p-value $p(\mathcal{X})$ associated with the data as the proportion of the N values $KS^{(1)} \leq \ldots \leq KS^{(N)}$ which exceeds the initial value $KS_{\mathcal{X}}$;

(*e*) Conclude as follows:

$$\text{reject } H_0 \text{ in favour of } H_1 \text{ at risk } \alpha \text{ if } KS_{\mathcal{X}} > c_\alpha$$

or, equivalently,

$$\text{reject } H_0 \text{ in favour of } H_1 \text{ at risk } \alpha \text{ if the } p\text{-value } p(\mathcal{X}) \text{ is smaller than } \alpha.$$

It should be reminded here that, in statistics, the p-value of a statistical test is the minimum risk one can undertake when rejecting H_0 (by risk, we mean the risk of rejecting H_0 while in fact it holds true).

The idea behind this bootstrap methodology is the following. The N bootstrap samples have the same size as the original data, they are GM distributed with distribution as close as possible to that of the original data, and therefore the N distances $KS^{(1)} \leq \ldots \leq KS^{(N)}$ (obtained in step (*c*) above) provide a good idea of the distribution of the test statistic $KS_{\mathcal{X}}$ if the original data were indeed GM distributed. Therefore, if the original data do not fit well a GM distribution, then the observed value $KS_{\mathcal{X}}$ will be high with respect to the values $KS^{(1)} \leq \ldots \leq KS^{(N)}$, and consequently the p-value $p(\mathcal{X})$ will be low. On the other hand, if the original data indeed follows (or can be approached by) a GM distribution, then the initial value $KS_{\mathcal{X}}$ is likely to be not particularly high with respect to the values $KS^{(i)}$, and the p-value $p(\mathcal{X})$ will consequently be moderate or high, which means that we will not be able to reject H_0 (and in practice, we will consider the GM model as adequate for our data).

Example 5.2. *We continue with Example 5.1, considering the model with 4 clusters as illustrated in the top line of Figure 8. The value of the test statistic is* $KS_{\mathcal{X}} = 0.0764$*, and the values* $KS^{(1)} \leq \ldots \leq KS^{(N)}$*, for* $N = 500$*, are illustrated in Figure 9. The corresponding p-value*

[7] $N = 200$ is a reasonable value but a higher value of 500 can be taken for better estimation of the p-value, because we are studying the tail distribution of the test statistic, and not just estimating one of its central parameters

[8] where $[x]$ denotes the integer part of x

equals 7.2% *(which means that 36 out of the 500 values* $KS^{(i)}$ *where greater than 0.0764), and the critical value for* $\alpha = 5\%$ *is* $c_{0.05} = 0.0792$. *Theore, if we wanted to conduct a test with a risk of* 5%, *we would not have rejected* H_0. *But we would have at a risk of* 10%. *The fit seemed very good at first glance, but in fact it is hardly correct...*

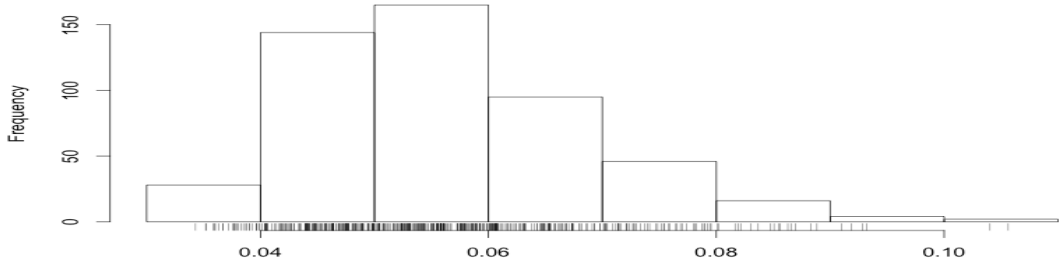


Figure 9: Histogram of the values $KS^{(1)} \leq \ldots \leq KS^{(N)}$ for $N = 500$ for Example 5.1

It is usually considered that this bootstrap methodology provides good estimates of the actual p-values of statistical tests in practice, even for n moderate or small. However, formal proofs are hard to obtain, particularly for complicated models such as the mixture models (because there are no closed formulas for the estimators of the numerous parameters of the model), and the only existing formal proofs found in the literature are only asymptotically valid (*i.e.* when the sample size n is very high), references will be given in Section 7.4. We thus need to validate our method by relying on exhaustive simulations: this is the purpose of the Section 5.2. We will pay particular attention to the performance of the test for small sample sizes n (*e.g.* of the order of 30) and to the power of the test (*i.e.* the capacity of the test of rejecting H_0 when it is indeed false).

5.1.3 Various remarks on the adopted bootstrap methodology

First note that the bootstrap strategy chosen here is called *parametric bootstrap*, since it does not exactly involve resampling of the original data (as the classical, or so-called naive, bootstrap does): we rather take advantage of the fact that we deal with a parametric and flexible family of distributions, and we can therefore produce bootstrap samples which are more diversified than those which would be obtained by simple resampling with replacement from an initial sample $\mathcal{X}$.

A second remark is that we could have considered other distances than the Kolmogorov-Smirnov distance, such as the Cramér-Von-Mises or the Anderson-Darling distance (see [vdV00] or [Tha10] for instance for a definition of these alternative distances), for our goodness-of-fit test: but the KS distance seems to suit our needs, since a perfect fitting of the underlying distribution of the data is not our primary aim.

Now, let us also mention that we could have considered a different strategy for calibrating our test, which is called the *subsampling* approach: we will not describe it here (references are for instance [RBRGTM14]), but we mention that, though it has the advantage of not being computer intensive, this strategy yields very poor power when dealing with small to moderate sample sizes (which is generally the case in practice), and it involves the choice of a tuning parameter (this

turned out, in our own experiences, to be rather delicate and to yield poor results).

Another remark is the following. The careful reader would have noticed that the bootstrap methodology yields a random response, which means that for a given dataset, two executions of the bootstrap calibrated test described above might yield two different p-values. The main tuning parameter in this issue is the number N of bootstrap samples (see step (c) of the bootstrap algorithm): if $N = 200$, then the resulting p-value is the proportion of the $KS_{\mathcal{X}^{(i)}}$ values $(i = 1, .., N)$ which exceed the original $KS_{\mathcal{X}}$, which is a multiple of $1/200 = 0.5\%$. Since the selection of the N subsamples is random, the p-value is therefore random (which adds another part of uncertainty in the decision), but this randomness can be considered as limited, particularly when the "true" p-value is small[9]. For instance, when the "true" p-value actually equals 3% (resp. 1%), then there is only 4% (resp. 6.10^{-6}) of chance that the p-value which results from the bootstrap method will exceed 5%. Moreover, if this randomness is considered as an issue to the user, then he/she can raise the number N of bootstrap resamples (to 500 for instance, or higher if more computing time can be afforded).

Finally, it is important to note that existing results about bootstrap calibration of goodness-of-fit tests (references are given in Section 7.4) would only validate the method we described in the previous section (i) for a fixed number K of components, and (ii) for n large (requiring by the way an asymptotic normality of the estimator $\hat{\theta}$ which we cannot formally guarantee in fact). Therefore, these important work should thus be considered only as very good signals, but simulation experiences are still needed in order to validate our method of Gaussian mixtures fitting testing in our "small n and estimated K" framework: this is the purpose of the next section.

5.2 Validation of the fitting method by simulations

In the previous section, we described the statistical procedure which makes it possible to test the hypothesis that some given dataset is issued from some Gaussian mixture distribution. This goodness-of-fit test is calibrated via bootstrap, and the procedure needs to be validated via some simulations (because formal proofs are not manageable, for finite sample sizes n) in order to assess the following:

- (i) *the bootstrap calibration yields the announced risk for the test*, *i.e.* whatever α may be chosen, if we apply the test to a dataset satisfying H_0 (*i.e.* issued from a GM distribution) with a critical value which is supposed to be associated with the risk α, then there is indeed a probability α (or very close to α) that the test leads to a rejection of H_0.
- (ii) *the constructed test has satisfying power*, *i.e.* if we apply the test to a dataset which is not issued from a GM distribution, then the test will lead to a rejection with sufficient probability. This should be as high as possible when the true underlying distribution is very different from a GM, for instance when the data exhibit a neat/strong asymmetry (in the right tail for instance, but not exclusively), or heavy tails (ie presence of several extreme values).

The first objective (i) is clearly defined and the way it should be investigated via simulation is clear as well; we will deal with it in Section 5.2.1. The second objective (ii) is vaguer, and consequently we will be only partially able to verify it by simulation (because, on one hand,

[9] if it is large, then the problem discussed here is not one anymore, the test will lead to a non-rejection in any case

there is an infinity of distributions which are not GM, and with different intensity, and on the other hand the definition of a satisfying power cannot be clearly defined non-asymptotically); we will address this objective in Section 5.2.2. In Section 5.2.3 we will discuss a modification of the procedure that yields more accuracy to our method.

5.2.1 Validation of the calibration risk

In this section, we are going to comment simulations which were performed in order to check that the following property is satisfied for small to moderate values of n (below, c_α denotes the critical value which is obtained in step (d) of the goodness-of-fit procedure described in the previous Section 5.1.2):

> if a Gaussian mixture distribution is randomly selected, and a random sample $\mathcal{X}$ of size n is drawn from that distribution, then, for every given α, the probability that $KS_{\mathcal{X}}$ exceeds c_α is actually α.

We thus want to check whether the probability of rejecting the null hypothesis, when it is true and when using the critical value c_α, is indeed α. In fact, this property is equivalent to the following one (where $p(\mathcal{X})$ denotes the p-value, associated with $\mathcal{X}$, and issued from step (d) of the goodness-of-fit procedure):

> if $p(\mathcal{X})$ denotes the p-value associated to a sample $\mathcal{X}$ randomly drawn from a randomly selected Gaussian mixture distribution, then the random variable $p(\mathcal{X})$ is uniformly distributed on $[0, 1]$

That these 2 properties are equivalent should sound natural to the reader, if he/she knows the basics of statistical inference: indeed, if we are in a situation where a null hypothesis H_0 is true, then saying that a test of H_0 is valid means that the test will yield p-values which can be anywhere on $[0, 1]$ with equal probability. For instance, when H_0 is true, we should have a 5% chance of observing a p-value which is lower than 0.05, or a 30% chance of observing $p(\mathcal{X})$ lower than 0.3; whereas if H_0 does not hold, then the p-value has more chance of taking values close to 0 (*i.e.* the distribution of the statistic $p(\mathcal{X})$ will put more weight on the small values of $[0, 1]$ than on the values close to 1, and therefore the test has more chance of rejecting H_0 when H_0 is false than when it is true).

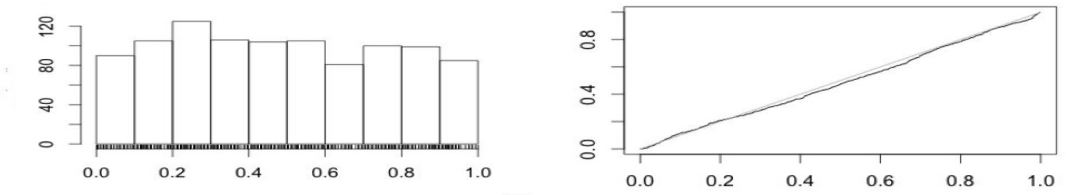
What we have done is thus the following: for a given value of n, we have, a large number N (500 or 1000) of times

- randomly generated a GM model (according to some previously defined strategy), then
- simulated a sample $\mathcal{X}$ of size n from that GM distribution, then
- computed the estimation $\widehat{F}_{\mathcal{X}}^{\mathrm{GM}}$, the test statistic $KS_{\mathcal{X}}$, and the corresponding p-value $p(\mathcal{X})$ (by bootstrap, with 200 bootstrap replications).

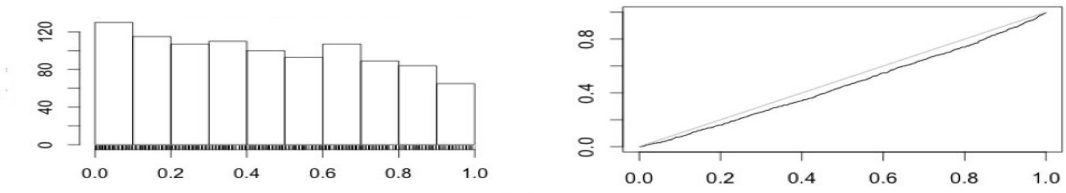
We thus have obtained (for a given size n) a sample of N p-values $p_1, \ldots, p_N$, and we checked whether these values constituted a random sample of the uniform distribution on $[0, 1]$. This check can be made graphically (via an histogram, or a QQ plot), or by using goodness-of-fit tests of uniformity (for instance a classical Kolmogorov-Smirnov test).

Before we examine the experimental results, lets us say a word about the random generation of a GM model. In such an experience, it is impossible to obtain any possible GM distribution,

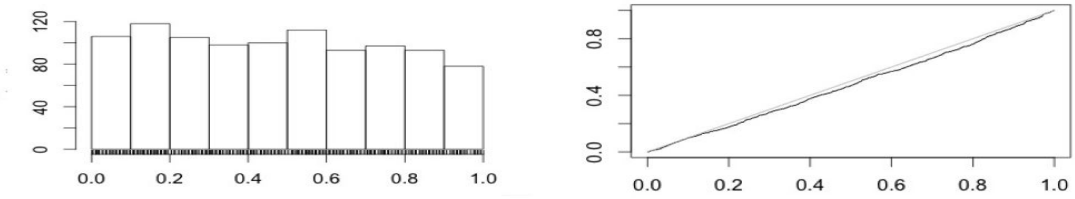
limits must be set in order to adapt our framework to program performances in practice. We did so in the following way, selecting randomly the different parameters of the GM model: we generated $K = 1$ plus a Poisson random variable of parameter 3.5 (K is then ≥ 1 and in general lower than 7 or 8), then we picked up the weights $\pi_1, \ldots, \pi_K$ completely at random (summing to 1 of course), and the means $\mu_1, \ldots, \mu_K$ completely at random between 10 and 60. Finally we generated the variances $\sigma_1^2, \ldots, \sigma_K^2$ randomly, according to 3 different possible settings, leading to small, moderate or high values[10] of the σ_k^2, which means that the clusters are likely to be well separated, mildly separated, or rather overlapping: in fact, in this simulation experience, the exact values of the parameters of the GM model (except K) are not relevant (because of the scaling properties of Gaussian distributions), but the way the different clusters may overlap or not is a characteristic which may (or not) affect the conclusions of the experience (we will see that it is not really the case, though).



(a) $n = 30$, under H_0, well separated clusters



(b) $n = 30$, under H_0, mildly separated clusters

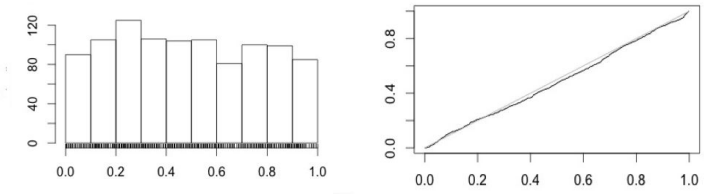


(c) $n = 30$, under H_0, rather overlapping clusters

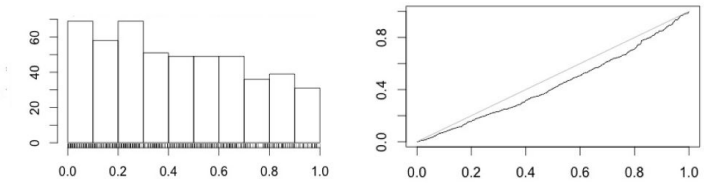
Figure 10: Histogram and QQ-plot of the $N = 1000$ p-values generated under a GM model, for $n = 30$ and under 3 possible levels of overlapping of clusters: a uniform distribution is expected for a good calibration of the test (for the QQ-plot, uniformity means being very close to the diagonal line)

[10] more precisely, these settings correspond to a distribution of the σ_k's which is that of $0.3 + c \times B$ where B follows a $Beta(3, 2)$ distribution (therefore $0 \leq B \leq 1$), and c respectively equals 2, 4 or 6 in the well separated, mildly separated, and rather overlapping cases

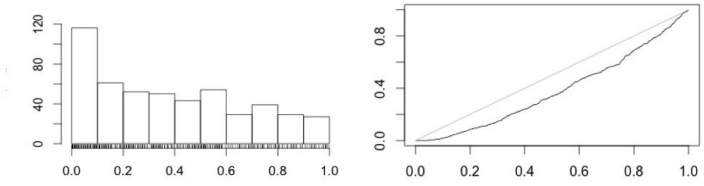
In Figure 10 are presented the distribution of the $N = 1000$ p-values of the goodness-of-fit test obtained for $n = 30$ and the 3 different ways the variances were generated (well separated, mildly separated, and overlapping). Judging whether the p-values are uniformly distributed on $[0, 1]$ can be done graphically by viewing the histogram or the uniform quantile-quantile plot (known as the QQ-plot, which has to be close to the first bisector line for assessing a good fit). One can also rely on a uniformity test (we chose[11] the chi-square test with 10 cells); the p-values of this uniformity test are respectively 26%, 0.1% and 12% for the 3 cases. The overall comment we can then make when studying these results is that, for small n, uniformity of the p-values is globally assessed (except maybe for the "mildly separated groups" setting).



(a) $n = 30$, under H_0, well separated clusters



(b) $n = 100$, under H_0, well separated clusters



(c) $n = 500$, under H_0, well separated clusters

Figure 11: Histogram and QQ-plot of the $N = 1000$ p-values generated under a GM model, for $n = 30$, 100, or 500: there is a little calibration problem when n is not small, p-values tend to be a bit too low, the test is a bit too severe

Let us now observe how the situation evolves when making n vary. We examine the cases $n = 30$, $n = 100$, $n = 500$ in the "well separated clusters" case[12], limiting now the simulation

[11] we did not rely on the Kolmogorov-Smirnov uniformity test, because of the presence of ties, due to the fact that the p-values are multiples of 1/200 since there have been only 200 bootstrap replications. This is not a problem anyway, the chi-square test will conveniently do the job here.

[12] which seems to us to be the more frequent situation in practice, according to the numerous execution times data we

experience to $N = 500$ simulations. The results are presented in Figure 11. We can observe that the method leads to p-values which tend to be a bit too low: this means that the test tends to falsely reject GM fitting a bit too often when n is not small (of the order of some hundreds). In Section 5.2.3, we will though present a modification of the method which will partially fix this problem. Anyway, even without this modification, our test is acceptable as it is, since a little too severe test is to be preferred to a too permissive test (which would lead to accept the GM model in situations where it shouldn't be), and in practice low values of n are more frequent than high ones.

5.2.2 Power of the goodness-of-fit test

In this section, we want to check out whether our goodness-of-fit test has satisfying power. As a matter of fact, a test with correct risk (*i.e.* which has an actual risk corresponding to the announced risk) is useless if it cannot reject the null hypothesis H_0 when it must. We do not want a test which has roughly the same chance of rejecting H_0 when H_0 is true than when it is false[13] ! Unfortunately, it is difficult to planify the assesment of the power of a test (non-asymptotically, *i.e.* for finite small or moderate sample sizes), because there are many ways in which an underlying model cannot be a Gaussian mixture model.

That is why we restricted ourselves, for evaluating the power of our test, to a family of non-GM distributions which seemed to us particularly interesting: the family of *shifted exponential mixtures*. This family generates clusters, but these clusters tend to exhibit some assymetry (unlike the Gaussian clusters, which are symmetric), and we want to see if our test is able to detect this feature, *i.e.* to detect the difference between a mixture of Gaussians and a mixture of (shifted) exponentials. Note that a shifted exponential of parameter λ, shifted by some constant c, is the distribution of $Z = c + E$ where E follows the exponential distribution of parameter λ (thus Z is always greater than c, has expectation $c + 1/\lambda$ and variance $1/\lambda^2$).

What we have done to evaluate the power is then the following: for a given value of n, we have, a large number N (500 or 1000) of times

- randomly generated a mixture of shifted exponentials model, then
- simulated a sample $\mathcal{X}$ of size n from that non-GM distribution, then
- computed the estimation $\widehat{F}_{\mathcal{X}}^{\mathrm{GM}}$, the test statistic $KS_{\mathcal{X}}$, and the corresponding p-value $p(\mathcal{X})$ (by bootstrap, with 200 bootstrap replications).

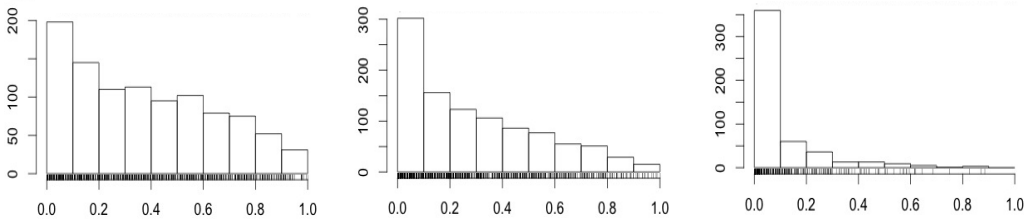
We thus have obtained (for a given size n) a sample of N p-values $p_1, \ldots, p_N$, and we expect these p-values to be not uniformly distributed, but instead to take rather small values[14] on $[0, 1]$. This check is made graphically by viewing the histogram, and for instance we may look at the proportion of these N p-values which are $< 5\%$: this proportion is then equal to the chance that, when applying our goodness-of-fit test with nominal risk $\alpha = 5\%$ to this non-GM data, we actually reject H_0 (this is thus the power of detecting that H_0 does not hold, when α is set to 5%).

In Figure 12, the results are presented one can see that the p-values are closer to 0 as the sample size n increases: for example, there are respectively 12.3%, 18.8% and 62.8% of the p-values

[13] this is the case of the pure subsampling strategy for small or moderate n, according to an extensive simulation study we led, and which will not be presented here.

[14] because the closer the p-value is to 0, the more confident we can be (*i.e.* the less risk we can take) when rejecting H_0

which are < 0.05, when $n = 30$, 100 or 500. The test has therefore more chance of rejecting H_0 when it does not hold, than when it holds. And the test has (fortunately) more chance to detect that H_0 does not hold as n increases. According to our personal experience, these power values are very satisfying, because a mixture of shifted exponentials is different from a GM, but not overly different, so detecting it may be difficult, especially when n is small ($n = 30$ is small when the aim is to test a whole distribution).



(a) $n = 30$, $N = 1000$ (b) $n = 100$, $N = 1000$ (c) $n = 500$, $N = 500$

Figure 12: Histograms of the p-values generated under a non GM model (a mixture model of shifted exponentials), for $n = 30$, 100, or 500: the capacity of the test to detect that H_0 does not hold, called the power, is measured by the fact that the distribution of the p-values is more concentrated on the vicinity of 0.

5.2.3 A pinch of undersampling yields more accurate risks for large sample sizes

We observed in Section 5.2.1 that our test has correct risk when n is small, but tends to falsely reject H_0 a bit too often when n gets large, *i.e.* when n is of the order of some hundreds. This is represented by the fact that the p-values tend to be a bit too close to 0 when n is a bit large. Although such large sample sizes are less likely to be encountered in practice, this is nonetheless problematic, and we decided to include in this report a modification of our procedure which somewhat fixes this problem of severity of our test for large n. This modified procedure will be called *bootstrap undersampling*, for reasons which will be made clearer below. Below, we first describe it, then see if it improves the situation via simulations (and whether it affects the power), and finally give some words of explanation.

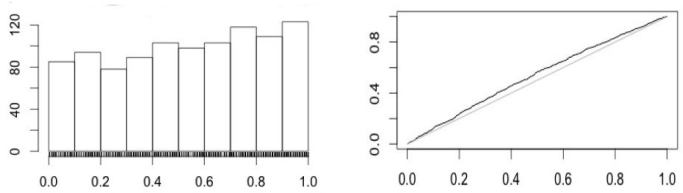
The idea of the modification stems from the following observation: if the p-value tends to be abnormally too small, this means that too high a proportion of the N bootstrap values $KS^{(1)} \leq \ldots \leq KS^{(N)}$ are lower than the original value KS_X. Said differently, the distribution of these N bootstrap values is too shifted to the left. But if the bootstrap values were generated by building bootstrap samples of size $n' < n$ lower than the original sample size, then the KS statistics will tend to take greater values[15], and one can hope that this will balance the situation, and yield correct p-values. This mechanism can be called *bootstrap undersampling*[16], because the size of the bootstrap samples is chosen a bit lower than the original size: this size n' will be

[15] because the KS statistic is more likely to take small values when computed from a large sample than from a small sample, since the empirical distribution function converges (under H_0) to the true distribution as n grows.

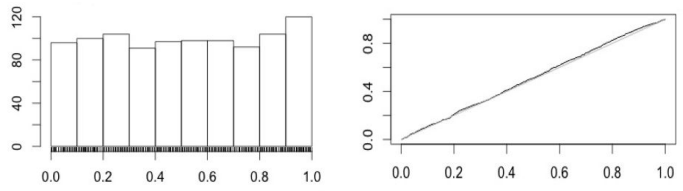
[16] not to be confused with the *subsampling* methodology, which is a known statistical technique, competitor to the bootstrap and not computing-intensive, but which fails completely in our framework, because of extremely low power for small and moderate sample sizes (simulations, which we performed but do not present here, prove this finding).

chosen equal to $n' = cn$, where c is a constant close to 1 (in the sequel the value $c = 0.9$ will yield good results).

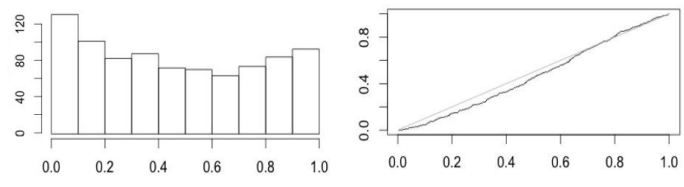
The modification is thus simple: in the goodness-of-fit test procedure, in step (c), simulate bootstrap samples of size $n' = cn$ instead of size n, and that's it. Let us now see whether this fixes the problem of severity of the test for relatively large samples. In the same simulation framework as in section 5.2.1 (well-separated clusters case), we applied the undersampling version of our test to Gaussian mixtures samples of sizes $n = 30$, 100 and 500, and we want to compare the p-value distributions to the situation with no modification: we thus want to see if the non-uniformity of the p-values problem observed in Figure 11 is fixed. The results are illustrated in Figure 13. We observe that the situation is better for $n = 100$ or $n = 500$, the p-values are better uniformly distributed, there are less too small p-values, and therefore there are less false rejections, *i.e.* the actual risk of the test is closer to the announced risk. For $n = 30$, the p-values are not as uniformly distributed as in the no-modification case, and unsurprisingly we advise against the use of undersampling for small values of n, since there is then no calibration problem.



(a) $n = 30$, under H_0, well separated clusters, with undersampling



(b) $n = 100$, under H_0, well separated clusters, with undersampling



(c) $n = 500$, under H_0, well separated clusters, with undersampling

Figure 13: Histogram and QQ-plot of the $N = 1000$ p-values generated under a GM model and using the undersampling bootstrap with proportion $c = 0.9$, for $n = 30$, 100, and 500: the p-values are better uniformly distributed for $n = 100$ and $n = 500$

We also conducted the same simulations as in section 5.2.2, but with the undersampling fea-

ture, in order to see if this modification of our test induces a severe loss in power: this is not the case, there is indeed an effect, the test is a bit less powerful when undersampling is applied, but the loss is rather mild (for the $n = 100$ and $n = 500$ cases, the powers of the tests with nominal risk $\alpha = 5\%$ have been respectively found to be around 14% and 52% instead of 18.8% and 63%; we did not provide the corresponding figures for conciseness).

Before closing this section, let us give our opinion about the fact that the bootstrap procedure needs to be undersampled in the way we have just described in this section. We think that the bootstrap does not converge correctly when n grows, *i.e.* some sort of (unknown) rate of convergence must be included in order to have the bootstrap distribution correctly approximate the true sampling distribution of our test statistic $KS_{\mathcal{X}}$. Since this rate is not known and the p-values tend to be too small experimentally, the idea of slightly undersampling the bootstrap samples looked like a good proposition (for large n), even if it is not supported by some theoretical results (or not yet).

5.3 Experiments on data-model fitting

We have implemented the our goodness-of-fit method using R. The software is presented in Appendix A. The performance data samples are presented in Appendix B.

There are 2438 real data samples which were considered, each sample containing between 30 and 1000 observed execution times. Our clustering method was applied successfully on each sample. In this section, we study the quality of data-model fitting on this quite large and various database. After the computation of an estimated Gaussian mixture distribution as detailed in Section 4, our KSfit method tests the quality of the fitting between the Gaussian mixture model and the data as explained previously in this section. It returns a probability called p-value: remind that this probability is the risk of error when falsely rejecting the hypothesis that the data is not issued from a Gaussian mixture model. Therefore, if the p-value is low, then we will reject the fitting between the data and the Gaussian mixture model; if it is not, then the model is acceptable for the data at hand.

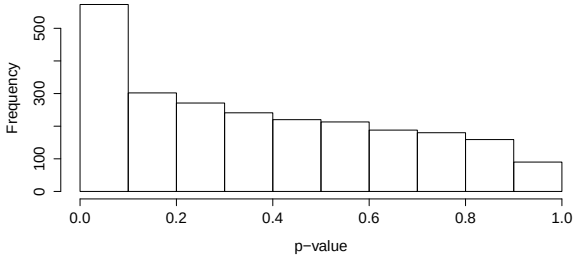


Figure 14: Histogram of the $p$values. They are obtained after applying our test of fitting using Kolmogorov-Smirnov distance with bootstrap calibration.

Figure 14 illustrates the histogram of the obtained p-values: clearly, we observe that the Gaussian mixture is a very good modelling for the majority of the samples. Indeed, if we consider a given risk value $0 < \alpha < 1$ of, say, 5%, we count the proportion of samples having a p-value smaller or greater than 5% (meaning that the Gaussian mixture model is rejected

at this risk), and we respectively find 16.81% and 83.09%. This means that the Gaussian mixture model is an acceptable model for 83% of the real datasets, which is quite high and satisfying.

We investigated the reasons why some datasets did not fit the Gaussian mixture model (16.81% of the samples). We found out that there are basically two main reasons:

1. Some datasets are apparently issued from distributions which cannot easily be approximated by Gaussian mixtures. For instance, heavy-tail distributions, exponential distributions, Pareto distributions, etc. The rejection of the Gaussian mixture model is therefore logical, since this model is flexible, but not that flexible.

2. In the majority of the situations though, the reason is that datasets contain *ties* (identical data values which are repeated inside the same sample). As a matter of fact, our experiments collected execution times with rounding precision: it may thus happen that some execution times are artificially perfectly equal in the sample. And when too high a proportion of data values are tied, our fitting test turns out to be severe, because it is not designed to be applied to non-continuous data, and ties artificially increases the Kolmogorov-Smirnov distance (with "high steps" in the step function $\widetilde{F}_X$). We think that, in practice, we can reduce the severity of our test by increasing the precision of the collected data, and find out that the Gaussian mixture model suits more situations that it seems in Figure 14.

In regard to the great variety of real data samples we have at our disposal, we can conclude that Gaussian mixtures are a good and flexible model for describing program execution times in general.

The next section defines new performance metrics that can be used in practice to select good or better program versions, and the parametric and non-parametric approaches to compute them.

6 New program performance metrics

In the literature, people are mainly focused on the average or median program performance. However in practice, the average or the median value may not be the most interesting summary measure in order to reflect the program performance, or may not be the best performance metric to use for making a decision about the most suitable program version.

For instance, consider the situation of a very long running application that a user executes only a few times. The user has the choice between many code versions, which one should he select ? If he bases his choice on the expected average or median execution time only, he may be disappointed if he executes his application very few times. If an application is rarely executed, the mean or median performances are not felt. So, additional performance metrics can help him making better selection.

Also, consider the situation where a user wants to know if the performances of his application are stable or not. Which metric can he use ? The well known variance is a metric that measures how data is spread out around the average only: knowing how to interpret this metric is not so widespread among the practicians, and can be misleading when the data distribution is multi modal, because the variance is a measure of dispersion around a single value, the average. Therefore it cannot be the unique metric used for evaluating performance stability, additional

metrics can be introduced and used, as we will see later.

This section provides new performance metrics that help the user to select a "good" program version based on performance analysis. For every new performance metric, we provide two ways to define and compute them: a parametric method (based on Gaussian mixture modelling) and a non-parametric method (based on data sample only).

Let us start with a first metric that computes the mean difference between two code versions, in the next section.

6.1 The metric $\mathcal{I}_1$: mean difference between two code versions

Let X and Y two random variables, representing the performances of two code versions. Let $\mathcal{X}$ be a sample of X and $\mathcal{Y}$ be a sample of Y, meaning that $\mathcal{X} = (x_1, \ldots, x_n)$ and $\mathcal{Y} = (y_1, \ldots, y_m)$, with n and m denoting the respective sample sizes. In some situations, we are not only interested in computing the speedup of X compared to Y or vice versa. We may be interested in quantifying the average difference between the performances of the two code versions. That is, we may be interested in computing the expected value $\mathbb{E}[\,|X - Y|\,]$. This defines our first performance metric as

$$\mathcal{I}_1 = \mathbb{E}[\,|X - Y|\,] \tag{4}$$

6.1.1 Non-parametric estimation of $\mathcal{I}_1$

Our non-parametric estimation of $\mathcal{I}_1$ is noted $\widetilde{\mathcal{I}_1}$, and we naturally define it by the following formula:

$$\boxed{\widetilde{\mathcal{I}_1} = \frac{1}{nm} \sum_{i=1}^{n} \sum_{j=1}^{m} |x_i - y_j|} \tag{5}$$

6.1.2 Parametric estimation of $\mathcal{I}_1$

Our parametric estimation $\mathcal{I}_1$, noted $\widehat{\mathcal{I}_1}$, assumes that both X and Y are modelled with Gaussian mixture distributions. As explained in Section 3, this means that the underlying p.d.f.s f_X and f_Y of X and Y equal weighted combinations of Gaussian p.d.f.s

$$f_X(x) = \sum_{i=1}^{K} \pi_i \varphi(x; \mu_i; \sigma_i) \quad \text{and} \quad f_Y(x) = \sum_{j=1}^{K'} \pi'_j \varphi(x; \mu'_j; \sigma'_j)$$

where we recall here that $\varphi(x; \mu; \sigma)$ denotes the p.d.f. of the Gaussian distribution $\mathcal{N}(\mu, \sigma)$, and K and K' are the respective numbers of clusters of these Gaussian mixtures. Under this model, we readily have

$$\begin{aligned} \mathcal{I}_1 &= \iint |x - y|\, f_X(x) f_Y(y)\, dx\, dy \\ &= \sum_{i=1}^{K} \sum_{j=1}^{K'} \pi_i \pi'_j \iint |x - y| \varphi(x; \mu_i; \sigma_i)\, \varphi(y; \mu'_j; \sigma'_j)\, dx\, dy \;=\; \sum_{i=1}^{K} \sum_{j=1}^{K'} \pi_i \pi'_j \mathbb{E}\left[\,|Z_i - Z'_j|\,\right] \end{aligned}$$

where Z_i and Z'_j denote independent Gaussian variables with distributions $\mathcal{N}(\mu_i, \sigma_i)$ and $\mathcal{N}(\mu'_j, \sigma'_j)$. By classical properties of the Gaussian family, $Z_i - Z'_j$ has distribution $\mathcal{N}(\mu_i - \mu'_j, (\sigma_i^2 + (\sigma'_j)^2)^{1/2})$. Therefore, using the following formula (proved later in this section)

$$\text{if } Z \text{ has distribution } \mathcal{N}(\mu,\sigma), \text{ then } \mathbb{E}\,[\,|Z|\,] = (2\Phi(\mu/\sigma) - 1)\mu + 2\sigma\varphi(\mu/\sigma) \tag{6}$$

we obtain the following formula for our theoretical performance metric:

$$\mathcal{I}_1 \;=\; \sum_{i=1}^{K}\sum_{j=1}^{K'} \pi_i\pi'_j \left((\mu_i - \mu'_j)\left(2\Phi\left(\frac{\mu_i - \mu'_j}{(\sigma_i^2 + (\sigma'_j)^2)^{1/2}}\right) - 1\right) + \sqrt{\frac{2(\sigma_i^2 + (\sigma'_j)^2)}{\pi}}e^{-(\mu_i - \mu'_j)^2/(2(\sigma_i^2 + (\sigma'_j)^2))}\right)$$

Consequently, using the estimations $\hat{\theta}, \hat{K}, \hat{\theta}', \hat{K}'$ of the parameters θ, K, θ', K' (*i.e.* the parameters of the estimated Gaussian mixtures distributions $\widehat{F}_{\mathcal{X}}^{\text{GM}}$ and $\widehat{F}_{\mathcal{Y}}^{\text{GM}}$), the parametric estimation $\widehat{\mathcal{I}_1}$ of our first performance metric $\mathcal{I}_1$ comes:

$$\boxed{\widehat{\mathcal{I}_1} = \sum_{i=1}^{\hat{K}}\sum_{j=1}^{\hat{K}'} \hat{\pi}_i\hat{\pi}'_j \left((\hat{\mu}_i - \hat{\mu}'_j)\left(2\Phi\left(\frac{\hat{\mu}_i - \hat{\mu}'_j}{(\hat{\sigma}_i^2 + (\hat{\sigma}'_j)^2)^{1/2}}\right) - 1\right) + \sqrt{\frac{2(\hat{\sigma}_i^2 + (\hat{\sigma}'_j)^2)}{\pi}}e^{-(\hat{\mu}_i - \hat{\mu}'_j)^2/(2(\hat{\sigma}_i^2 + (\hat{\sigma}'_j)^2))}\right)} \tag{7}$$

Let us now quickly prove formula (6). If $Z' = (Z - \mu)/\sigma$, then Z' has a standard gausian distribution, and we can write (below we use the fact that the derivative of $\varphi(z)$ is $-z\varphi(z)$):

$$\begin{aligned}
\mathbb{E}\,[\,|Z|\,] \;&=\; \mathbb{E}\,[\,|\mu + \sigma Z'|\,] \;=\; -\int_{-\infty}^{-\mu/\sigma} (\mu + \sigma z)\varphi(z)\,dz \;+\; \int_{-\mu/\sigma}^{+\infty} (\mu + \sigma z)\varphi(z)\,dz \\
&=\; \mu\,(-\Phi(-\mu/\sigma) + (1 - \Phi(-\mu/\sigma)) \;+\; [\,\varphi(z)\,]_{-\infty}^{-\mu/\sigma} \;+\; [\,-\varphi(z)\,]_{-\mu/\sigma}^{+\infty} \\
&=\; (2\Phi(\mu/\sigma) - 1)\mu + 2\sigma\varphi(\mu/\sigma) \quad (\text{ since } \Phi(-z) = 1 - \Phi(z)\,)
\end{aligned}$$

The next section presents a second performance metric, which is the probability that a single program run is better than another.

6.2 The metric $\mathcal{I}_2$: probability that a single program run is better than another

Consider the same framework as above, with X and Y denoting two random variables representing the performances of two code versions, and $\mathcal{X}$ being a sample of size n of X and $\mathcal{Y}$ a sample of size m of Y. The user needs to select which code version to execute. He may base his selection criteria on the expected average speedup for instance. But if his application is rarely executed, the average performance gain may not be interesting for him. He may be interested in executing a single time his application, and he wishes that a single run has the most chances of being the fastest between two code versions. Formally, to help him to decide, we can compute $\mathbb{P}\,[X < Y]$, the probability that a single run of X would be better than a single run of Y. This defines our second metric of program performances:

$$\mathcal{I}_2 = \mathbb{P}\,[X < Y] \tag{8}$$

6.2.1 Non-parametric estimation of $\mathcal{I}_2$

Our non-parametric estimation of $\mathcal{I}_2$ is noted $\widetilde{\mathcal{I}_2}$. It is based on the identity $\mathbb{P}[X < Y] = \mathbb{E}[\mathbb{1}_{X<Y}]$, and simply consists in counting all pairs of values (x_i, y_j) which satisfy $x_i < y_j$:

$$\boxed{\widetilde{\mathcal{I}_2} = \mathbb{E}[\mathbb{1}_{X<Y}] = \frac{1}{nm}\sum_{i=1}^{n}\sum_{j_1}^{m}\mathbb{1}_{x_i<y_j}} \tag{9}$$

where $\mathbb{1}_{x_i<y_j} = 1$ if $x_i < y_j$, and $= 0$ otherwise. Alternatively, we can also generalise this metric to consider a constant real shift $\Delta \in \mathbb{R}$ to check between X and Y, and thus consider:

$$\boxed{\mathcal{I}_2 = \mathbb{P}[X < Y + \Delta] = \mathbb{E}[\mathbb{1}_{X<Y+\Delta}] \quad \text{and} \quad \widetilde{\mathcal{I}_2} = \frac{1}{nm}\sum_{i=1,n}\sum_{j_1,m}\mathbb{1}_{x_i<y_j+\Delta}}$$

6.2.2 Parametric estimation of $\mathcal{I}_2$

Our parametric estimation of $\mathcal{I}_2$, noted $\widehat{\mathcal{I}_2}$, assumes that both X and Y are modeled with Gaussian mixture distributions. We have:

$$\begin{aligned}\mathcal{I}_2 &= \iint \mathbb{1}_{x<y}\, f_X(x) f_Y(y)\, dx\, dy \\ &= \sum_{i=1}^{K}\sum_{j=1}^{K'}\pi_i\pi'_j \iint \mathbb{1}_{x<y}\, \varphi(x;\mu_i;\sigma_i)\, \varphi(y;\mu'_j;\sigma'_j)\, dx\, dy \;=\; \sum_{i=1}^{K}\sum_{j=1}^{K'}\pi_i\pi'_j\mathbb{P}\left[Z_i - Z'_j < 0\right]\end{aligned}$$

where Z_i and Z'_j denote independent Gaussian variables such that $Z_i - Z'_j$ is Gaussian distributed with expectation $\mu_i - \mu'_j$ and variance $\sigma_i^2 + (\sigma'_j)^2$. Since $\mathbb{P}[Z < 0] = \Phi(-\mu/\sigma)$ whenever Z has distribution $\mathcal{N}(\mu, \sigma)$, we thus have

$$\mathcal{I}_2 = \sum_{i=1}^{K}\sum_{j=1}^{K'}\pi_i\pi'_j\,\Phi\left(\frac{\mu'_j - \mu_i}{\sqrt{\sigma_i^2 + (\sigma'_j)^2}}\right)$$

Consequently, plugging in the estimators of the parameters of the Gaussian mixture distributions leads to the following parametric estimator of $\mathcal{I}_2$:

$$\boxed{\widehat{\mathcal{I}_2} = \sum_{i=1}^{\hat{K}}\sum_{j=1}^{\hat{K}'}\hat{\pi}_i\hat{\pi}'_j\,\Phi\left(\frac{\hat{\mu}'_j - \hat{\mu}_i}{\sqrt{\hat{\sigma}_i^2 + (\hat{\sigma}'_j)^2}}\right)} \tag{10}$$

Alternatively, we can also generalise this metric to consider a constant real shift $\Delta \in \mathbb{R}$ to check between X and Y, and thus consider:.

$$\boxed{\mathcal{I}_2 = \mathbb{P}[X < Y + \Delta] \quad \text{with} \quad \widehat{\mathcal{I}_2} = \sum_{i=1}^{\hat{K}}\sum_{j=1}^{\hat{K}'}\hat{\pi}_i\hat{\pi}'_j\,\Phi\left(\frac{\Delta + \hat{\mu}'_j - \hat{\mu}_i)}{\sqrt{\hat{\sigma}_i^2 + (\hat{\sigma}'_j)^2}}\right)}$$

The next section presents a third performance metric, which is the probability that a single program run is better than many others.

6.3 The metric $\mathcal{I}_3$: probability that a single run is better than all the others

In practice, a user may have more than only two code versions. How can he decide about the best code version among many others, for a single run only ? Comparing code versions two by two is misleading. All code versions must be compared together. Let $X_1, X_2, \cdots, X_r$ denote r random variables corresponding to r distinct code versions[17]. We propose to compute the probability that one code version, say the first one, executes faster than all the others for a single run only (not in average or in median), *i.e.* that $X_1 < \min(X_2, \ldots, X_r)$. This defines the following program performance metric:

$$\mathcal{I}_3 = \mathbb{P}\left[X_1 < \min(X_2, \cdots, X_r)\right] = \mathbb{E}\left[\mathbb{1}_{X_1 < \min(X_2, \cdots, X_r)}\right] \tag{11}$$

6.3.1 Non-parametric estimation of $\mathcal{I}_3$

Our non-parametric estimation of $\mathcal{I}_3$ is noted $\widetilde{\mathcal{I}_3}$. Here we consider, for every $j \in \{1, \ldots, r\}$ a sample $\mathcal{X}_j$ of the random variable X_j, with n_j denoting the size of the j-th sample and $x_{1,j}, x_{2,j}, \ldots, x_{n_j,j}$ the observations. By considering all the r-tuples $(x_1, \cdots, x_r)$, we can count how many tuples verify the condition $x_1 \leq \min(x_2, \cdots, x_r)$ (with an adequate algorithm, this is not too time-consuming). This yields our non-parametric estimation of $\mathbb{P}\left[X_1 < \min(X_2, \cdots, X_r)\right]$ defined as follows:

$$\boxed{\widetilde{\mathcal{I}_3} \;=\; \frac{1}{\prod_{j=1}^{r} n_j} \sum_{i_1=1}^{n_1} \sum_{i_2=1}^{n_2} \cdots \sum_{i_r=1}^{n_r} \mathbb{1}_{x_{i_1,1} < \min(x_{i_2,2}, \cdots, x_{i_r,r})}} \tag{12}$$

where $\mathbb{1}$ is the Dirac function of an event, the same used previously.

6.3.2 Parametric estimation of $\mathcal{I}_3$

The parametric estimation of $\mathcal{I}_3$ is noted $\widehat{\mathcal{I}_3}$. Here we assume that, for every given $j \in \{1, \ldots, r\}$, the random variable X_j is distributed as a Gaussian mixture with parameters $K = K_j$ and $\theta = \theta_j = (\pi_{1,j}, \ldots, \pi_{K_j,j}; \mu_{1,j}, \ldots, \mu_{K_j,j}; \sigma_{1,j}, \ldots, \sigma_{K_j,j})$. In other words, its p.d.f. is $f_{X_j}(x) = \sum_{i=1}^{K_j} \pi_{i,j} \varphi(x; \mu_{i,j}; \sigma_{i,j})$.

As we did for $\mathcal{I}_1$ and $\mathcal{I}_2$, we need to obtain a formula for the metric $\mathcal{I}_3$ in terms of the parameters. Let us note $Y = \min(X_2, \cdots, X_r)$, and let G be the c.d.f. of Y. By the mutual independence of $X_1, X_2, \ldots, X_r$, the variables X_1 and Y are independent and therefore a classical probability property yields, since $\mathbb{P}[x < Y] = (1 - G)(x)$,

$$\mathcal{I}_3 = \mathbb{P}\left[\, X_1 < Y \,\right] = \mathbb{E}\left[\mathbb{1}_{X_1 < Y}\right] = \mathbb{E}\left[(1 - G)(X_1)\right] = \int_{-\infty}^{\infty} (1 - G(x)) f_1(x) dx.$$

The job is then over because, by independence of the variables $X_2, \ldots, X_r$, and relation (2),

$$(1 - G)(x) = \mathbb{P}\left[\, X_2 > x, \ldots, X_r > x \,\right] = \prod_{j=2}^{r} \mathbb{P}\left[X_j > x\right] = \prod_{j=2}^{r} \sum_{i=1}^{K_j} \pi_{i,j} (1 - \Phi(x; \mu_{i,j}; \sigma_{i,j})).$$

[17] These r distinct random variables must not be confused with a sample $(X_1, X_2, \cdots, X_n)$ of a single random variable X)

Our parametric estimator $\widehat{\mathcal{I}_3}$ of the metric $\mathcal{I}_3$ is then equal to the following integral (which we be compute numerically, by using R for instance)

$$\boxed{\widehat{\mathcal{I}_3} = \int_{-\infty}^{\infty} (1 - \hat{G}(x)) \hat{f}_1(x) dx} \tag{13}$$

where

$$\hat{G}(x) = \prod_{j=2}^{r} \sum_{i=1}^{\hat{K}_j} \hat{\pi}_{i,j} (1 - \Phi(x; \hat{\mu}_{i,j}; \hat{\sigma}_{i,j})) \quad \text{and} \quad \hat{f}_1(x) = \sum_{i=1}^{\hat{K}_1} \hat{\pi}_{i,1} \varphi(x; \hat{\mu}_{i,1}; \hat{\sigma}_{i,1})$$

The next section introduces a fourth and last performance metric which evaluate the variability level program performances.

6.4 The metric $\mathcal{I}_4$: the variability level

People do not always know how to quantify the variability of program performances. By default, they use variance, but they may not know how to interpret it. The variance measures how the data spread out around the average: but if the data is multi-modal or presents clusters, then the average is not necessarily a good measure of the variability, especially when the modes or clusters are particularly distant from each other, and therefore the variance loses its attractiveness.

We propose to consider an alternative or complementary measure of the variability of the program performances: the number of modes of the underlying p.d.f. of the data, which we will note $\mathcal{I}_4$. It is simply equal to the number of *local maxima*[18] of the pdf, which is supposed to represent the different values around which the data are spreading or clustering. A local maxima is also called a *mode* in statistics. For instance, look at Figure 8 in page 26: the upper left part shows a density function with 4 modes, while the lower left part shows a density function with 3 modes.

Thanks to the Gaussian mixture modelling, which yields an explicit formula for the estimated pdf, we can compute a parametric estimation $\widehat{\mathcal{I}_4}$ of $\mathcal{I}_4$, which is simply equal to the number of local maxima of the Gaussian mixture p.d.f. $f_{\hat{\theta},\hat{K}}$ estimated from the data. Often, this estimation $\widehat{\mathcal{I}_4}$ turns out to be equal to $\hat{K}$, the estimated number of clusters of the fitted Gaussian mixture distribution. But it is not necessarily always the case, since sometimes the Gaussian mixture fitting algorithm proposes a higher value of $\hat{K}$ than the actual number of groups in the data, in order to flexibly account for assymetry.

As an example, if a program produces execution times data which presents 3 clusters (components) in its fitted Gaussian mixture model, then we say that its variability level is equal to 3, meaning that the program performances are spread out around 3 distinct values. Note that we do not propose a non-parametric version $\widetilde{\mathcal{I}_4}$ of $\mathcal{I}_4$, because it would require using a non-parametric density estimator, which needs a tuning parameter (the bandwidth) which is rather delicate to choose: a basic automatic choice of this parameter could often lead to inadequate estimations of $\mathcal{I}_4$, and a more sophisticated choice of this parameter would certainly lead to the same estimation

[18] Formally, a local (strict) maximum of a real function f is a value m such that there exists $h > 0$ such that $f(x) < f(m)$ for every $x \neq m$ with $m - h \leq x \leq m + h$

as $\widehat{\mathcal{I}_4}$ (which is an integer) in most cases.

$$\boxed{\widehat{\mathcal{I}_4} = \text{number of local maxima of the estimated Gaussian mixture pdf, which is often equal to } \hat{K}\text{, the number of components in this model}} \quad (14)$$

6.5 Experiments: analysing parametric versus non-parametric program performance metrics by simulation

This section presents our empirical study, based on extensive simulations, to compare the quality of the parametric and non-parametric estimators of our performance metrics. Note that we perform a simulation study because we cannot rely to theory for such a comparison, because (for finite samples) no such theory exists, nor is feasible (particularly due to the complexity of the estimation process).

In the first part of this section, we explain how the quality of an estimator can be quantitatively measured, and present our methodology for performing this comparison by simulation. The second part contains additional remarks and informations concerning the simulation protocol and the choices we made. In the rest of the section, we present the comparison results and our conclusions.

6.5.1 Estimation quality measurement and simulation methodology

In statistics, approaching a theoretical unknown parameter with an estimator introduces an error. If $\mathcal{I}$ denotes the unknown parameter[19] and $\hat{\mathcal{I}}$ one of its estimator, the estimator quality is usually measured via its Theoretical Mean Square Error (TMSE), which definition is the following:

$$\texttt{TMSE}(\hat{\mathcal{I}}|\mathcal{I}) = \mathbb{E}\left[(\hat{\mathcal{I}} - \mathcal{I})^2\right]$$

It is important to understand that the value of this TMSE not only depends on the sizes of the data samples which were used to define the estimator $\hat{\mathcal{I}}$, but it also depends on the underlying distributions of those data samples[20]. Saying that a given estimator is of good quality means that, whatever the underlying distributions, the $\texttt{TMSE}(\hat{\mathcal{I}}|\mathcal{I})$ is small, smaller than that of competitor estimators, and becomes even smaller as the sample sizes increase. However, except in very rare occasions, there is no explicit formula for the theoretical MSE (usable for allowing comparison between competitive estimators), and if there are, these are in most cases only valid for n large or very large.

Therefore, in order to check and compare the precision of our performance metric estimators (parametric and non-parametric ones), especially for ordinary (*i.e.* not large) sample sizes, we most often need to rely on *simulations* to evaluate the values of the differents theoretical MSEs. These evaluations will be empirical versions of these theoretical MSEs, computed on the basis of repeated simulations of the data. And, in order to obtain a broad opinion on the quality of the estimator, we must repeat this process for various choices of the underlying distributions (and not only on one or two situations, as it is unfortunately done in many statistical papers).

Concretely, in our particular situation, the simulation methodology is the following, where $\mathcal{I}$ denotes any fixed metric among $\mathcal{I}_1$, $\mathcal{I}_2$, $\mathcal{I}_3$, $\mathcal{I}_4$.

[19] such an unknown parameter is usually denoted by θ, but this notation is already used elsewhere in this work, and here the estimated objects are the theoretical performance metrics $\mathcal{I}_1, \ldots, \mathcal{I}_4$, so we rather call the generic parameter $\mathcal{I}$

[20] in order to lighten the notations though, we did not make this dependence explicit

(1) We consider a number N_F of various underlying frameworks $\mathcal{F}_1, \ldots, \mathcal{F}_{N_F}$ for the data. By framework, we mean a choice of underlying distributions of the data, and a choice of the data samples sizes. In the sequel, we will generate these frameworks randomly, and explain how we do so.

For instance, if $\mathcal{I}_1$ is the metric of interest, each framework consists in specifying an underlying Gaussian mixture distribution for X and an underlying Gaussian mixture distribution for Y, and sample sizes n and m. If it is $\mathcal{I}_3$ which is studied, then the framework must specify a Gaussian mixture distribution for every random variable $X_1, \ldots, X_r$ representing the execution times of the different versions of the code, and of course a value for the number r of those different versions and values $n_1, \ldots, n_r$ for the sample sizes.

(2) For each framework $\mathcal{F}_j$ $(1 \leq j \leq N_F)$:

(i) we compute the true value of the metric $\mathcal{I}$, which we note $\mathcal{I}^{(j)}$ (remind that this true value changes every time we change the underlying distributions, *i.e.* the framework).

For instance, if if $\mathcal{I}_1$ is at stake, since the underlying distributions of X and Y are Gaussian mixtures, of parameters determined in step (1), we compute ${\mathcal{I}_1}^{(j)}$ by using the formula between relation (6) and equation (7) of this chapter.

(ii) we then simulate, N times, random data samples according to the framework $\mathcal{F}_j$, each time computing the corresponding value of the parametric and non-parametric estimators of the current value $\mathcal{I}^{(j)}$ of the metric. We note these estimations $(\widehat{\mathcal{I}})_1^{(j)}, \ldots, (\widehat{\mathcal{I}})_N^{(j)}$ and $(\widetilde{\mathcal{I}})_1^{(j)}, \ldots, (\widetilde{\mathcal{I}})_N^{(j)}$.

For instance, if $\mathcal{I}_1$ is at stake, for every $i = 1$ to N we do the following: we simulate a data sample $\mathcal{X}$ of size n for X and a data sample $\mathcal{Y}$ of size m for Y, compute the non-parametric estimation $(\widetilde{\mathcal{I}_1})_i^{(j)}$ of ${\mathcal{I}_1}^{(j)}$ according to formula (5), then we fit estimated Gaussian mixtures distributions to the samples $\mathcal{X}$ and $\mathcal{Y}$ and compute the corresponding parametric estimation $(\widehat{\mathcal{I}_1})_i^{(j)}$ according to formula (7).

(iii) since we now have $2 \times N$ estimations of the current value $\mathcal{I}^{(j)}$ of the metric $\mathcal{I}$ for the framework $\mathcal{F}_j$, we can then evaluate the true mean squared errors of the estimators for this j-th framework via the so-called *empirical mean square errors*, defined as follows:

$$
\begin{aligned}
MSE^{(j)}(\widetilde{\mathcal{I}}|\mathcal{I}) &= \frac{1}{N}\sum_{i=1}^{N}\left((\widetilde{\mathcal{I}})_i^{(j)} - \mathcal{I}^{(j)}\right)^2 \\
MSE^{(j)}(\widehat{\mathcal{I}}|\mathcal{I}) &= \frac{1}{N}\sum_{i=1}^{N}\left((\widehat{\mathcal{I}})_i^{(j)} - \mathcal{I}^{(j)}\right)^2
\end{aligned}
$$

(3) At this point, we have evaluations of the TMSE of the non parametric estimators $\widetilde{\mathcal{I}}$ and the parametric ones $\widehat{\mathcal{I}}$ of $\mathcal{I}$ in N_F different frameworks, making it possible to have an overview of the overall superiority of one of these estimators over the other. The estimator that has

the lowest MSE in more precise. In this regard, we decided to consider the following N_F ratios

$$(R_1, \ldots, R_{N_F}) \;=\; \left(\frac{MSE^{(1)}(\widehat{\mathcal{I}}|\mathcal{I})}{MSE^{(1)}(\widetilde{\mathcal{I}}|\mathcal{I})} \;,\; \cdots \;,\; \frac{MSE^{(N_F)}(\widehat{\mathcal{I}}|\mathcal{I})}{MSE^{(N_F)}(\widetilde{\mathcal{I}}|\mathcal{I})} \right)$$

which values should permit us to decide which type of estimator is the best: for instance, if most of these ratios are lower than 1, this would mean that the parametric estimation method is generally more precise than the non-parametric method, while if these ratios are overall close to 1, that would mean that the two approaches have similar statistical accuracy. A boxplot of these ratio values will yield a visualisation of this issue.

We can (and will) repeat this procedure for various sample sizes, in order to assess that the quality of the estimators improves as the data samples get larger.

Note that, on one hand, while we are free to choose the sample sizes as we want, on the other hand, the described simulation procedure requires a sufficiently large value of N in order to have good evaluations of the unknown theoretical MSEs (and therefore trustable conclusions). Concerning now the choice of the number N_F of frameworks, it just depends on how broad we want our simulation study to be, and how general we want our conclusions to be. See the next section for the details about the chosen values of N_F, N, and the sample sizes, and details on the random generator of frameworks in step (1) of the simulation procedure.

Finally, note that the whole process can be performed with an alternative measure of estimation quality, which we call the Theoretical Mean Absolute Percentage Error (TMAPE): it measures a relative error instead of an absolute (squared) error like the TMSE, and is therefore sometimes more interpretable. Its definition is:

$$\texttt{TMAPE}(\mathcal{I}|\hat{\mathcal{I}}) = \mathbb{E}\left[\left| \frac{\hat{\mathcal{I}} - \mathcal{I}}{\mathcal{I}} \right| \right]$$

The whole process described above can be adapted to this alternative measure by replacing all occurences of empirical MSEs by empirical mean absolute percentage errors defined by

$$\begin{aligned} MAPE^{(j)}(\widetilde{\mathcal{I}}|\mathcal{I}) &= \frac{1}{N}\sum_{i=1}^{N} \left| \frac{(\widetilde{\mathcal{I}})_i^{(j)} - \mathcal{I}^{(j)}}{\mathcal{I}^{(j)}} \right| \\ MAPE^{(j)}(\widehat{\mathcal{I}}|\mathcal{I}) &= \frac{1}{N}\sum_{i=1}^{N} \left| \frac{(\widehat{\mathcal{I}})_i^{(j)} - \mathcal{I}^{(j)}}{\mathcal{I}^{(j)}} \right| \end{aligned}$$

6.5.2 Additional precisions on the simulation methodology

In this section, we provide some additional information about choices we made for the simulation protocol.

1. Concerning the framework random generation, depending on the performance metric which is considered, there are 1, 2, or $r \geq 3$ theoretical Gaussian mixture distributions which must be generated for every framework. We explain here how we conducted the random generation of one GM distribution.

Such a distribution has a p.d.f. of the form

$$g_{\theta,K}(x) = \sum_{k=1}^{K} \pi_k \varphi(x; \mu_k; \sigma_k).$$

These parameters are generated as follows (remember that θ is the notation for the vector containing all of these parameters):

(a) the value $K \in \mathbb{N}$ ($K \geq 1$) represents the number of clusters/components of the mixture: according to a preliminary empirical study on some real datasets, we decided to have K being equal to 1 plus a Poisson random variable of mean 2, which seemed rather realistic.

(b) for each cluster (there are K distinct clusters), the parameters μ_k and σ_k of the corresponding Gaussian p.d.f. $\varphi(x; \mu_k; \sigma_k)$ have been generated uniformly on some bounded subset of the positive real line. These subsets were chosen so that a fair variety of situations could arise, while nonetheless avoiding to have mixtures being too separated or too "flat". Note that the boundedness reflects real program performances in practice.

(c) then random weight values π_k were generated completely at random between 0 and 1, with the constraint that they must sum to 1.

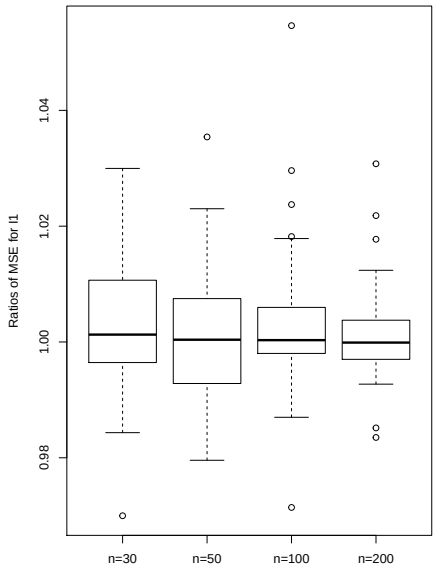
2. Concerning the choice of the number N of simulations performed for each framework, we must be careful about the fact that the empirical MSEs should approximate relatively well the real value of the theoretical MSE in the framework: it must therefore not be too small. A minimum value of $N = 200$ is advised, and we chose $N = 1000$ for our simulations. With such values, we will be confident when comparing the empirical MSEs (and MAPEs) of the parametric and non-parametric estimators.

3. Concerning the step (2)(ii) of the simulation protocol described above, there is implicitely Gaussian mixtures fitting which must be performed in order to provide parametric estimation of the performance metric. This, of course, involves following the fitting method described in Section 4, and provides the estimated parameters which appear in Equations (7), (10), (13) and (14) defining the parametric estimations of the performance metrics.

4. Concerning now the number N_F of frameworks considered in this simulation study, in order to be able to conclude as generally as possible regarding the comparison of the parametric and non-parametric approaches of performance estimation precision, we decided to consider no less than $N_F = 40$ random frameworks.

6.5.3 Simulation results for $\mathcal{I}_1$: mean difference between two code versions

We did extensive simulations that computed the ratios between the MSE of parametric versus non parametric estimations of $\mathcal{I}_1$. For each sample size n (which equals 30, 50, 100 or 200) we generated 40 random couples of Gaussian mixture models, and random samples from those distributions, and computed the ratios of the parametric and non-parametric empirical MSEs, and then represented the boxplots of those ratios. These are contained in Figure 15. We observe that those ratios are very close to 1, for each value of n, and even closer to 1 as the sample size n increases. We did not report the actual values of the different MSEs, because they are not particularly interesting on their own, and because they vary in function of the (random) choice of distribution couples (they are nevertheless small, the estimation is of good quality). This



n	Min	Q_1	med	mean	Q_3	Max
30	0.97	0.99	1	1	1.01	1.03
50	0.97	0.99	1	1.00	1	1.03
100	0.97	0.99	1	1	1.00	1.05
200	0.98	0.99	0.99	1	1	1.03

Figure 15: Ratios between the MSE of parametric vs. non parametric estimators of $\mathcal{I}_1$

	Non parametric estimator $\widetilde{\mathcal{I}_1}$	Parametric estimator $\widehat{\mathcal{I}_1}$
Min	3.24%	3.27 %
Q1	5.67%	5.68%
Median	6.75%	6.79%
Q3	8.04%	8.06%
Max	9.84%	9.81%

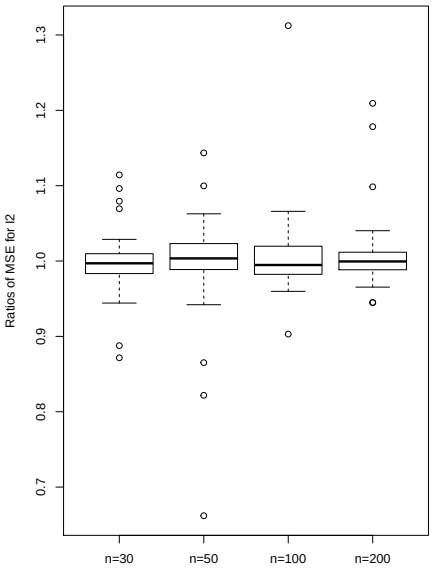
Table 1: Mean absolute percentage errors of non parametric and parametric estimation of $\mathcal{I}_1$, case when $n = 30$ and $N = 1000$

closeness to 1 of the ratios means that none of the two approaches is preferable to the other for accurately estimating the underlying value of $\mathcal{I}_1$.

Regarding mean absolute percentage error (MAPE), Table 1 summarises the obtained values for these errors measurements for the case when $n = 30$ and $N = 1000$ (other cases show same conclusions). As can be seen, these MAPE values are very small for both estimators, which mean that both estimators are precise enough.

6.5.4 Simulation results for $\mathcal{I}_2$: probability that a single program run is better than another

The simulation methodology is the same as in the previous section, and the conclusion as well: the observed MSE ratios are close to 1 (a bit less than for $\mathcal{I}_1$ though). See Figure 16.



n	Min	Q_1	med	mean	Q_3	Max
30	0.87	0.98	0.99	0.99	1	1.11
50	0.66	0.98	1.	0.99	1.02	1.14
100	0.90	0.98	0.99	1	1.019	1.31
200	0.94	0.98	0.99	1.01	1.01	1.20

Figure 16: Ratios between the MSE of parametric vs. non parametric estimators of $\mathcal{I}_2$

Regarding mean absolute percentage error (MAPE), Table 2 summarises the obtained values for these errors measurements, when $n = 200$ and $N = 1000$. This table clearly shows that the two estimators are precise enough. Note that for lower values of n, the observed MAPE are higher.

6.5.5 Simulation results for $\mathcal{I}_3$: probability that a single run is better than all the others

For each value of $r = 3$, 4 or 5, and for each sample size n (which equals 30 or 100) we generated 30 times r different random Gaussian mixture distributions, then drew random samples of (same) size n from those distributions, and computed the ratios of the parametric and non-parametric

	Non parametric estimator $\widetilde{\mathcal{I}_2}$	Parametric estimator $\widehat{\mathcal{I}_2}$
Min	0.63%	0.22%
Q1	3.13%	6.03%
Median	4.81%	8.46%
Q3	7.32%	14.75%
Max	26.54%	66.24%

Table 2: Mean absolute percentage errors of non parametric and parametric estimation of $\mathcal{I}_2$, case when $n = 200$ and $N = 1000$

empirical MSEs for estimating $\mathcal{I}_3$, and then represented the boxplots of those ratios. These are contained in Figure 17.

Once again, we observe that the ratios are spreading around the value 1 , with a little bit less dispersion when $n = 100$ instead of $n = 30$ (and with some particular cases where the performances were found to be quite different). No influence of the value of r was found. A possible conclusion could be that, for a given set of performance datasets, it is not possible to predict which of the two approaches will bring the smaller MSE.

A note of caution when interpreting the boxplots of the ratios $\widehat{\mathcal{I}_1}/\widetilde{\mathcal{I}_1}$, $\widehat{\mathcal{I}_2}/\widetilde{\mathcal{I}_2}$ or $\widehat{\mathcal{I}_3}/\widetilde{\mathcal{I}_3}$: when the ratio equals 1.5, this means that the MSE of the parametric estimator is 1.5 higher than the the MSE of the non-parametric estimator, but when the ratio equals 0.5, then the MSE of the non-parametric estimator is $1/0.5 = 2$ times higher than the MSE of the parametric estimator, therefore the ratio boxplots should be read and interpreted with caution (*i.e.* not symmetrically).

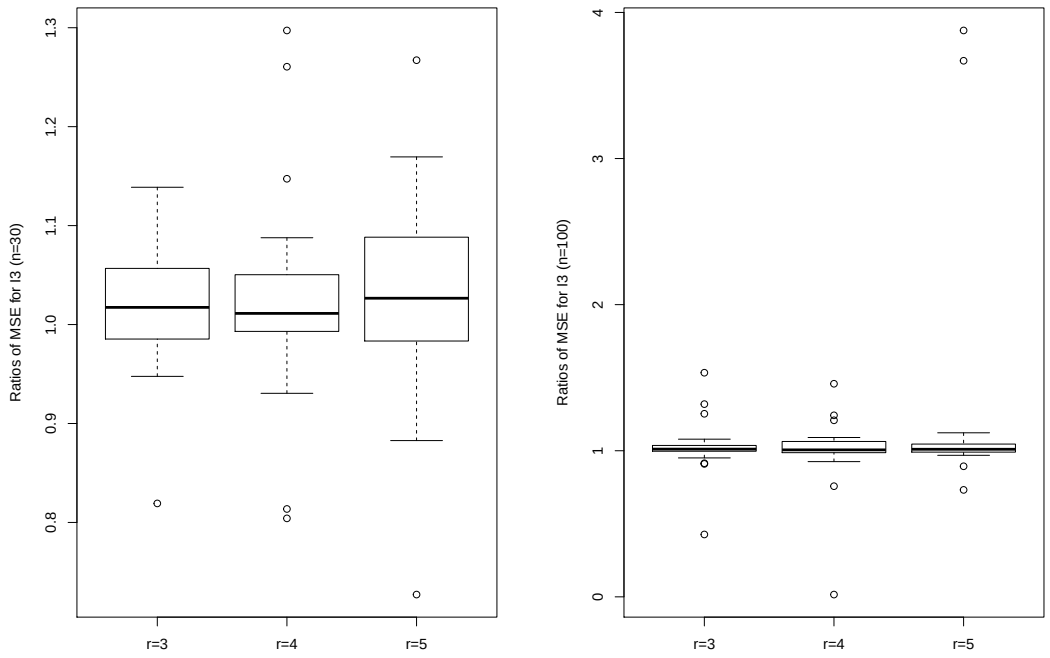
Regarding mean absolute percentage error (MAPE), Table 3 summarises the obtained values for these errors measurements, for the case of $r = 5$ and $n = 100$ and $N = 200$. We can see here that the MAPE are reasonably low, which means that the two estimators are precise enough.

	Non parametric estimator $\widetilde{\mathcal{I}_3}$	Parametric estimator $\widehat{\mathcal{I}_3}$
Min	5.40%	5.48%
Q1	7.10%	7.75%
Median	10.09%	10.28%
Q3	13.28%	13.10%
Max	41.61%	43.35%

Table 3: Mean absolute percentage errors of non parametric and parametric estimation of $\mathcal{I}_3$, when $r = 5$ and $n = 200$ and $N = 200$

6.5.6 Simulation results for $\mathcal{I}_4$: the variability level

First note that $\mathcal{I}_4$ is a discrete (integer) value, so we will not base our simulation study on its MSE or its MAPE, which are preferably devoted to continuous valued indicators. Therefore, in order to have an idea on the quality of our parametric estimator $\widehat{\mathcal{I}_4}$, we will rely on frequency tables. We considered two values for n, $n = 30$ (small sample size) and $n = 200$ (big sample size), and simulated 2000 random Gaussian mixture models: in this section, the simulation plan



(a) $n = 30$ (b) $n = 100$

Figure 17: Ratios between the MSE of parametric vs. non parametric estimators of $\mathcal{I}_3$

is a bit different from the previous sections, since here we simulated only one random sample of size n for each generated GM model, and then computed the value of $\mathcal{I}_4$ for this model, and its parametric estimation $\widehat{\mathcal{I}_4}$. We thus obtained 2000 couples of values $(\mathcal{I}_4, \widehat{\mathcal{I}_4})$ corresponding to a different GM model each time.

Table 4 shows the number (and proportion) of cases where the estimation was just right, the number of case where the error is lower than 1, as well as the number of cases where the estimated number of modes was lower than the true number of modes. The second table (Figure 5) contains more details: it presents, for each number k of modes (the columns), the repartition of the models generated with $\mathcal{I}_4 = k$ according to their estimated number $\widehat{\mathcal{I}_4}$ of modes (the rows).

A first comment on these results is that the performance of $\widehat{\mathcal{I}_4}$ is better when $n = 200$ than when $n = 30$, and is better when $\mathcal{I}_4$ is small than when it is not. A second comment is that the estimation $\widehat{\mathcal{I}_4}$, when it is not equal to the true number $\mathcal{I}_4$ of modes, strongly tends to be lower than it: on Table 5, one can see that for each column, between 80% and 97% of the models have an estimated number of modes which is lower than or equal to the underlying number of modes.

These results and this moderate performance can be explained by a number of arguments. The first one is that the method for choosing the number K of components of the Gaussian

	n=30	n=200
Number and proportion of exact estimations ($\mathcal{I}_4 = \widehat{\mathcal{I}}_4$)	910 (45.5%)	1277 (63.85%)
Number and proportion of correct estimations with error ≤ 1, *i.e.* $\left\|\mathcal{I}_4 - \widehat{\mathcal{I}}_4\right\| \leq 1$	1624 (81.20%)	1834 (91.35%)
Number and proportion of estimations with $\widehat{\mathcal{I}}_4 \leq \mathcal{I}_4$	1816 (90.8%)	1867 (93.75%)

Table 4: Number and proportion of good estimations of variability level metric ($\mathcal{I}_4$) in function of sample sizes.

$\widehat{\mathcal{I}}_4$ \ $\mathcal{I}_4$	1	2	3	4	5	6
1	126 (80.77%)	107 (15.81 %)	96 (11.96 %)	39 (13.00%)	14 (22.58 %)	0
2	24 (15.38 %)	493 (72.82 %)	389 (48.44%)	134 (44.67%)	23 (37.10%)	2 (100%)
3	3 (1.92%)	59 (8.71%)	254 (31.63 %)	78 (26.00%)	22 (35.48%)	0
4	2 (1.28 %)	12 (1.77 %)	46 (5.73%)	36 (12.00%)	2 (3.23%)	0
5	0	4 (0.59 %)	12 (1.49%)	9 (3.00%)	1 (1.61%)	0
6	1	2 (0.30%)	3 (0.37%)	2 (0.67%)	0	0
7	0	0	1 (0.12 %)	2 (0.67%)	0	0
8	0	0	2 (0.25%)	0	0	0

(a) $n = 30$

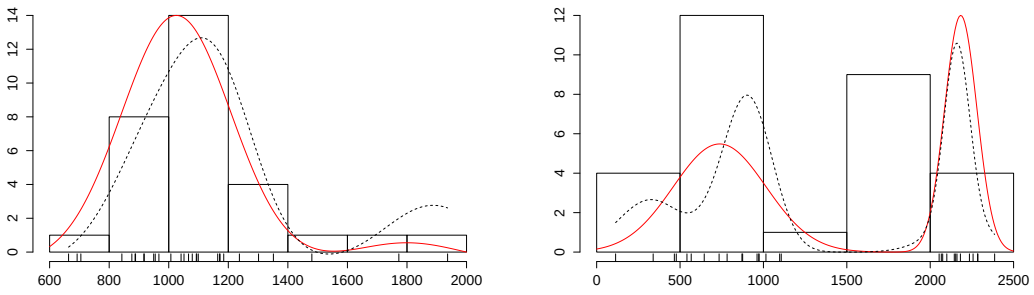
$\widehat{\mathcal{I}}_4$ \ $\mathcal{I}_4$	1	2	3	4	5	6
1	144 (92.90%)	49 (6.90%)	30 (3.94%)	5 (1.59%)	2 (3.57%)	0
2	10 (6.45 %)	611 (86.06 %)	249 (32.68%)	84 (26.75%)	16 (28.57%)	0
3	1 (0.65%)	47 (6.62 %)	437 (57.35%)	128 (40.76%)	17 (30.36%)	0
4	0	3 (0.42%)	37 (4.86%)	81 (25.80%)	15 (26.79%)	1 (33.33%)
5	0	0	8 (1.05%)	11 (3.50%)	4 (7.14%)	2 (66.67%)
6	0	0	1 (0.13 %)	5 (1.59%)	2 (3.57%)	0

(b) $n = 100$

Table 5: Tables of the column relative frequencies that count the number of times when $\mathcal{I}_4 = \widehat{\mathcal{I}}_4$

mixture, is the use of the BIC criterion (see Section 4): this criterion is not devoted to the quality of estimation of the number of local modes of the underlying distribution, but to a good quality of overall fitting of the data to the GM distribution. It is well known that the BIC criterion tends to "prefer" a simpler model to a slightly more complicated one (i.e. with a higher number of clusters) if the gain of quality fitting is not sufficient. It is therefore not surprising that the estimated number of local modes is slightly biased downwards. A second argument is that, in practice, and particularly when the clusters are overlapping, it is not surprising that the method does not perfectly detect the correct number of local modes, simply because of sampling randomness, (for small n, but also for n as large as 200). For example, in Figure 18, in the left sub-figure there are two underlying clusters, and the smaller one (on the right) is well detected

due to the presence of the two data points on the right (which are indeed certainly issued from this rightmost cluster). On the contrary, on the right sub-figure, we see that there are 3 underlying clusters, but the data values issued from the two leftmost clusters are so close that the clustering method only detects one cluster on the left, yielding $\widehat{\mathcal{I}_4} = 2$ instead of the awaited 3. We thus understand that a good estimation of the variability level is not as easy as one might expect.



(a) dataset example 1

(b) dataset example 2

Figure 18: Examples of two datasets of size $n = 30$, represented with their histogram, their true underlying GM distribution p.d.f. (in dotted line), and their estimated GM distribution p.d.f. (in solid line) , the individual data values being indicated by ticks on the horizontal axis

6.6 Empirical study of variability levels of programs execution times

Based on our metric for the estimation of variability level ($\mathcal{I}_4$ presented in Section 6.4), we computed the estimated value of this metric for all program samples presented in Appendix B. Figure 19 plots the histogram of the obtained variability level. More precisely:

- $\approx 37\%$ of the samples have a variability level equal to one, which means that the execution times are spread around a single value.
- $\approx 32\%$ of the samples have a variability level equal to 2, which means that the execution times are spread around two values.
- $\approx 12\%$ of the samples have a variability level equal to 3, which means that the execution times are spread around three values.
- $\approx 19\%$ of the samples have a variability level ≥ 4, which means that the execution times are spread around more than three values.

This histogram clearly demonstrate that summarising the execution times of a program with a single number (mean or median) is often inadequate.

The next section present some state of the art in code performance analysis using statistics. It is a recall of related work already presented in [TWB13]. It also lists some references concerning the statistical concepts dealt with in the present work.

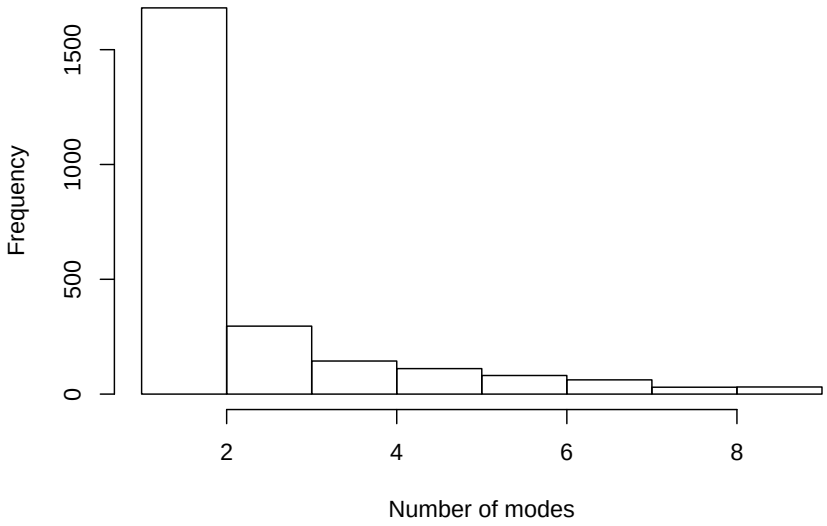


Figure 19: Variability levels of programs executions times

7 Related work on code performance analysis and evaluation using statistics

7.1 Observing execution times variability

The literature contains some experimental research highlighting that program execution times are sometimes increasingly variable or unstable. In the article of raced profiles [LOW09], the performance optimisation system is based on observing the execution times of code fractions (functions, and so on). The mean execution time of such code fraction is analysed thanks to the Student's t-test, aiming to compute a confidence interval for the mean. This previous article does not fix the data input of each code fraction: indeed, the variability of execution times when the data input varies cannot be analysed with the Student's t-test. Simply because when data input varies, the execution time varies inherently based on the algorithmic complexity, and not on the structural hazards. So assuming that execution times vary in this situation according exclusively to randomness is an obviously wrong model.

Program execution times variability has been shown to lead to wrong conclusions if some execution environment parameters are not kept under control [MDSH09]. For instance, the experiments on sequential applications reported in [MDSH09] show that the size of Unix shell variables and the linking order of object codes both may influence the execution times. However, it should be noted that one of the experimented benchmark (`perlbench`) has a hidden input which is an environment variable. So if the environment variable size varies, this means that the program input varies, so it is expected that executions times vary as consequence. Thus, the

variability here cannot be considered as randomness !

We published in [MtATB10] an empirical study of performance variation in real world benchmarks with fixed data input. Our study concludes three points: 1) The variability of execution times of long running sequential applications (SPEC CPU 2000 and 2006) can be marginal if we fully control the hardware machine. 2) The variability of execution times of long running parallel applications such as SPEC OMP 2001 is important on multicore processors, such variability cannot be neglected. 3) Almost all the samples of execution times do not follow a Gaussian distribution.

In the current research report, the variability of code performance is not related to varying data input, but is related to external factors of the binary code that the user cannot control.

7.2 Program performance evaluation in presence of variability

In the field of code optimisation and high performance computing, most of the published articles declare observed speedups or other performance metrics. Unfortunately, few studies based on rigorous statistics are really conducted to check whether the observations of the code performance improvements are statistically significant or not.

Program performance analysis and optimisation may rely on two well known books that explain digest statistics to our community [Jai91, Lil00] in an accessible way. These two books are good introductions for doing fair statistics for analysing performance data. Based on these two books, previous work on statistical program performance evaluation have been published [GBE07]. In the latter article, the authors rely on the Student's t-test to compare between two average execution times (the two sided version of the student t-test) in order to test whether two theoretical means are equal or not. We improved this previous work in [MtATB10]: first, we showed how to conduct a one-sided Student's t-test to validate that $\mu_X > \mu_Y$. Second, we showed how to check the normality in small samples and the equivalence of their variances (using the Fisher's F-test) in order to use the classical Student's t-test instead of the Welch's variant.

In addition, we must note that [Lil00, GBE07, Jai91] focus on comparing between the mean execution times only. When the program performances have some extrema values (outliers), the mean is not a good performance measure (since the mean is sensitive to outliers). Consequently the median is usually advised for reporting performance indicators (such as for SPEC scores). Consequently, in [MtATB10], we relied on known academic books on statistics [BD02, HW73, Sap90] for comparing between two medians.

7.3 The Speedup-Test: analysing and comparing the average and median execution times

In [TWB10, TWB13], we presented a rigorous statistical methodology regarding program performance analysis. We rely on well known statistical tests (Shapiro-wilk's test, Fisher's F-test, Student's t-test, Kolmogorov-Smirnov's test, Wilcoxon-Mann-Whitney's test) to study if the observed speedups are statistically significant or not. By fixing $0 < \alpha < 1$ a desired risk level, we are able to analyse the statistical significance of the average execution time as well as the median. We can also test if $\mathbb{P}[X > Y] > \frac{1}{2}$, the probability that an individual execution of the optimised code is faster than the individual execution of the initial code. In addition, we can compute the confidence interval of the probability to get a speedup on a randomly selected benchmark that

does not belong to the initial set of tested benchmarks. Our methodology defined a consistent improvement compared to the usual performance analysis method in high performance computing based on mean and median performance metrics.

7.4 References on Gaussian mixtures, goodness-of-fit and bootstrap

References about mixture models and particularly Gaussian mixture models are particularly numerous in the statistics literature, machine-learning literature, as well as in many statistics-using fields (in particular image analysis, bioinformatics, biology, medecine, etc); we will therefore only cite the reference book on mixture models, [MP00], which is an excellent starting point for the theory and a great source of applications of the subject. Other possible general textbooks on mixtures are also [TSM85] or [MRT11]. Note that in many settings, the concern is about mixtures of multivariate Gaussian distributions: our univariate framework is a simpler subcase, and therefore several issues evoked in the clustering literature are therefore not relevant to our framework.

The use of the EM algorithm as a solution to finite mixtures fitting is a classical subject in the statistics and pattern recognition literature, since the release of the breakthrough paper [DLR77] (note that the strength of the EM algorithm is that it is not restricted to the estimation of Gaussian mixtures, but extend to other mixtures, often with a computational cost though). In the present work, we decided to address the important issue of choosing the appropriate number K of components by relying on the BIC criterion: there are however several popular alternatives, for which some references are [TWH01] (the gap criterion), [FTP14] (the slope criterion; this paper is recent and contains a good review and comparison with other methods). However, in our univariate and often small sample size framework, we did not explore these alternatives (which did not yield very different results in a preliminary study which we will not present here)

Well known references about goodness-of-fit tests are [dS86] or, more recently, [Tha10] for instance. However, this topic is also addressed in standard textbooks of advanced statistical analysis, Chapter 19 of [vdV00] is also one reference concerning the problem of calibrating the Kolmogorov-Smirnov test when the target is a whole family of distributions (but it stays on the theoretical ground).

General references about the bootstrap and its use in applied statistics are the following books, for instance [DH97], [Che07] or [Man97]. Some specific references about the use of the bootstrap principle for addressing the problem of calibration in goodness-of-fit tests are the papers [SMQ93], [BR04] and [Bab11] (see also [KY12] for a variation of the bootstrap in this framework). We point out that these existing work only provide theoretical results which guarantee the asymptotic (*i.e.* for large n) validity of the bootstrap principle, and would be applicable in our framework only for a fixed number K of components (they also require asymptotic normality results about the estimator $\hat{\theta}$, which we cannot formally guarantee). These important work should thus be considered as very good signals, but simulation experiences were quite necessary in the present work to validate our method of Gaussian mixtures fitting test.

8 Perspectives and discussions

8.1 Multi-dimensional performance modelling

Nowadays, people are not only interested in analysing and optimising a single type of program performance. Users can be interested in studying multiple types of performances conjointly: execution time, energy consumption, power consumption, memory consumption, network traffic, memory-CPU bandwidth, input/output latency, etc. This means that we must be able to collect and measure performance data conjointly, saved in multi-dimensional data. For instance, we may be interested in studying the pairs of values (execution time, energy consumption), or the triplets (execution time, energy consumption, network traffic), etc. Each performance type constitutes an independent dimension.

As far as we know, no statistical study on multi-dimensional performance has been done. The aim here would not be to write a model of a single performance dimension as a function of the other performance types, as is done with linear and non-linear regression models. The objective would be to analyse the correlation and the relationship between performances of different natures. This could be a good starting point for investigating the possible software or hardware reasons that lead to a variability of multimodal type. Regression models are not suitable because they make a projection of a performance dimension, and do not model all performance dimensions conjointly. Fortunately, Gaussian mixtures naturally and easily adapt to the multidimensional framework (see [MP00] for examples of applications of mixtures of multivariate Gaussians): we could thus be in a position of modelling, for instance, the bi-dimensional observed performances {execution time, energy consumption} as a mixture of bi-dimensional Gaussian distributions, which would result in exploring the possible relationship between the time performance and the consumption performance, more precisely than by just computing the coefficient of correlation between these two indicators (which is a global, and sometimes misleading, indicator of statistical relationship). For instance, if we find out that these bi-dimensional performance data present a clustered nature, we could then be in a position of finding out the reasons of such between-cluster variability, by comparing the number of the clusters with other data measured during the execution. Moreover, bi-dimensional versions of some of the performance indicators described in Section 6 could be introduced.

8.2 Considering mixtures of other distributions

Gaussian mixture modelling allows to fit quite well multi-modal distributions. According to our experiments, Gaussian mixtures fit most of the cases (execution times in our situation). However, some cases cannot be modeled by GM. If the observed data are issued from heavy tail distributions, exponential distributions, Pareto distributions, or mixtures of theses distributions, then the Gaussian mixture modelling may be disappointing: in that case, mixtures of other families of distributions (other than the Gaussian family) could be considered. This should be considered as necessary only in particular situations though (and only when estimation of worst or best possible performance is at stake).

Another disadvantage of GM model is that it considers theoretical performance values from $-\infty$ to $-\infty$. In practice, if we assume that a program executes and terminates correctly, the theoretical performance values are bounded. So GM mixture are not necessarily well suited for studying extreme values such as minimal execution times and worst case execution times. For extreme values statistics, other data distributions should be used.

8.3 Discussion: how to decide about the best code version ?

When dealing with efficient programming, an application may have different code versions: multiple source code versions may exist, also with multiple binary code versions (the compiler may generate distinct binary codes depending of the compilation flags). The question of selecting the most efficient code version on a given machine, for a given data input set, under a given software environment, is not easy. Statistics provide a tool to help decision, statistics do not provide strict guarantees, since the conclusions issued from statistical methods always come along with a degree of error, a risk which is often only proved asymptotically (although in the present work, we did our best assessing the validity of our method for reasonable and finite data sample sizes).

For helping decision making, suitable performance metrics must be considered:

- If the code is devoted to be run a high number of times, it is better to chose a code version with the best mean or median execution time. The median execution time is more suitable for codes that exhibit outliers, *i.e.* some extreme performance values.
- If a code is not executed a high number of times, the mean or the median execution time are not adequate as a performance metric. It would better to study the probability that a single run of a code version would be better than a single run of another code version, or better than multiple other code versions. We proposed in Section 6 such performance metric.

The user of statistics must remember that this science is based on probability theory. This means that if the underlying physical phenomena is not hasard, or if the collected sample data are not independent, conclusions may be biased.

9 Conclusion

When executing a binary code on a physical machine, one could be interested in analysing and optimising the performances. The performance considered in our work is any continuous value, such as execution time, energy consumption, power consumption, network traffic, etc. In ideal execution environment, when a user has a full control on the executing machine and operating systems, it may be possible to stabilise program performances. But in practice, users do not have full control on their machine and operating systems. On realworld executing machines, the sharing of resources and the modern processor micro-architectures makes it hard to observe stable performances. And the observed performances are quite rarely normally distributed, although many people consider or expect they are. Indeed, we observed that the collected performances are multi-modal distributions.

In the presence of performance variability, we must rely on formal statistics to decide about the best code version. In the past, we presented such statistical protocol called the Speedup-Test [TWB13]. It analyses the mean and median execution time only. In the current research report, we extend our previous study as follows:

1. We build a statistical modelling based on Gaussian mixtures to fit multi-modal datasets.
2. We build a statistical test to quantify the quality of data-model fitting.
3. We define new code performance metrics that go beyond mean and median performances.

4. For each new performance metric, we propose a non-parametric estimator and a parametric one based on Gaussian mixtures modelling.
5. We implemented all our statistical methods using R, and we demonstrate its practical efficiency.
6. Our software, called `VARCORE`, is publicly distributed as a free open source code.

Gaussian mixtures seem to be good model for most of the performances that we observed in practice. Also, such data distributions provide interesting perspective regarding multi-dimensional performance data. Indeed, an interesting performance analysis must consider the performance as multi-dimensional data, each dimension corresponds to a specific performance nature. For instance we can consider the triplet {Execution time, energy consumption, network traffic}. Fortunately Gaussian mixtures are a good solution for modelling multi-dimensional data in order to analyse the relationship between multiple dimensions.

Concerning the performance of the goodness-of-fit test we introduced in this work, we found out that it was very satisfying. We nonetheless observed that, in the presence of too much a proportion of equal data in the sample, our fitting test tends to artificially reject the Gaussian mixture model too often: a simple solution to this problem would be to increase the precision of the measurement method so that the risk of observing tied values (exæquo) is severely reduced.

Some people are interested in extreme values statistics (Worst Case Execution Times, Best Case Execution Times). In that case, Gaussian mixture modelling is not necessarily the adequate way of addressing this issue: either mixtures of other data distributions must be considered, or alternative methods should be considered (extreme value analysis techniques in particular, although they require quite a large number of data values to be truly reliable).

Our Gaussian mixture modelling provides a new and interesting metric to evaluate the variability level of performances. Indeed, instead of only considering means and variances (the variances being, by the way, difficult to interpret in practice), we propose to consider as well the *modes* of the data distributions. Thus, the variability level of performances can be measured by the number of these modes, which we can compute with a parametric method based on Gaussian mixtures. The number of modes is a natural way of giving an idea of the performance variability: a data distribution with a single mode means that the performances are quite stable around a single value; with two modes, it means that the performances are varying around two values, etc. In addition, if the number of modes is greater than one, this can be a good indication of being careful with the interpretation (and comparison) of the variance (since the usual interpretation generally assumes that the data are issued from a unimodal distribution). Moreover, the existence of these modes can be further investigated by trying to explain them with auxiliary measurements made during the execution of the program, which is certainly the most fruitful perspective of this research work.

A The VARCORE software for parametric and non parametric statistics

This section presents our software implemented with R. Our software is called VARCORE and requires the following R packages:

```
mclust
R.utils
```

Our software has been tested with R version 3.2.2 and mclust package version 5.1. The source code is public, and the functions are commented with enough information to describe their parameters in details. The most important functions are summarised in Table 6 for data modelling, in Table 7 for plotting and in Tables 8-9 for performance metrics. The parameters of these functions are documented in the source code, where other interesting functions can be found.

Function name	Description
`VARCORE_Clustering`	It computes an estimated Gaussian mixture model based on a data sample $\mathcal{X}$
`VARCORE_calculKSFit`	It measures, using a Kolmogorov-Smirnov distance, the quality of fitting between data and the estimated Gaussian mixture model.

Table 6: Clustering functions (building and checking the data model)

Function name	Description
`VARCORE_plot_clustering_result`	It plots conjointly the histogram of the data and the density function of a previously computed clustering result
`VARCORE_extract_cluster_information`	It extracts the parameters of a Gaussian mixture model, to store them in a data frame.
`VARCORE_extract_cluster_list`	It extracts the list of clusters: in other words, it assigns to each data value one of the clusters of the Gaussian mixture.
`VARCORE_plotCDF_fit`	It plots conjointly the empirical (cumulative) distribution function of the data and the (cumulative) distribution function of a previously computed Gaussian mixture model.

Table 7: Plotting functions

Function name	Description
`VARCORE_meandiff_Mclust`	It computes $\widehat{\mathcal{I}_1}$, a parametric estimator of the mean difference $\mathbb{E}[\,\lvert X-Y\rvert\,] = \mathcal{I}_1$
`VARCORE_probXY_Mclust`	It computes $\widehat{\mathcal{I}_2}$, a parametric estimator of the probability $\mathbb{P}[X < Y] = \mathcal{I}_2$.
`VARCORE_probX1min_Mclust`	It computes $\widehat{\mathcal{I}_3}$, a parametric estimator of the probability that a random variable X_1 is the minimum of a set of r random variables: $\mathbb{P}[X_1 < \min(X_2, \cdots, X_r)] = \mathcal{I}_3$.
`VARCORE_nbmodes_estimation`	It computes $\widehat{\mathcal{I}_4}$, i.e. it estimates the number of modes of a Gaussian mixture model. It is our proposed metric to estimate the variability level.
`VARCORE_quantile_Mclust`	It computes the α-quantile of a Gaussian mixture model.
`VARCORE_probXa`	It computes the probability that a random variable X is below a given constant a: $\mathbb{P}[X \leq a]$.

Table 8: Parametric performance metrics (require Gaussian mixture modelling of the data)

Function name	Description
`VARCORE_meandiff_nonparam`	It computes $\widetilde{\mathcal{I}_1}$, an non-parametric estimator of the mean difference $\mathbb{E}[\,\lvert X-Y\rvert\,] = \mathcal{I}_1$.
`VARCORE_probXY_nonparam`	It computes $\widetilde{\mathcal{I}_2}$, a non-parametric estimator of the probability $\mathbb{P}[X < Y] = \mathcal{I}_2$.
`VARCORE_probX1min_nonparam`	It computes $\widetilde{\mathcal{I}_3}$, a non-parametric the probability that a random variable $X1$ is the minimum of a set of random variables: $\mathbb{P}[X_1 < \min(X_2, \cdots, X_r)] = \mathcal{I}_3$.

Table 9: Non-parametric performance metrics (do not require modelling of the data)

Example (done with R) 1. *Below, we give an example of using our software to model a dataset by a Gaussian mixture, and to plot and print the result.*

```
# Filename of this example: example1-VARCORE.R
# First, load the VARCORE software in R
> source('VARCORE.R')

# Load a sample of data from a file, named C1, execution times in seconds
> C1 <- read.csv("ammp-C2", header=F)$V1

# Clustering: building a Gaussian mixture model for the dataset C1
# Be careful, data inside a sample must not be identical otherwise no clustering is possible
> clusC1 <- VARCORE_Clustering(C1)
```

```
# Plot the figure
> VARCORE_plot_clustering_result(clusC1,C1, maintitle="ammp", subtitle="Config 1",
           plot_legend=T, plot_rug=T, xlabel="Execution Times", ylabel="Probability")
> VARCORE_plot_clustering_result(clusC1,C1) # it also works, but with less details

# Extract cluster information and print them
> clsinfo <- VARCORE_extract_cluster_information(clusC1)
> clsinfo
# Every line corresponds to a cluster (Gaussian) with its weight, mean and variance
# Here we have 5 clusters
     weights    means      stdevs
1 0.09677359 92.21333 0.163372026
2 0.15280620 93.26964 0.146702827
3 0.45830339 93.54552 0.227167773
4 0.16059283 94.21802 0.003986046
5 0.13152400 94.99607 0.392762363

# Let us compute the variability level of C1:
> nbmodes = VARCORE_nbmodes_estimation(clusC1)
> nbmodes
[1] 4
# So we say that the variability level of C1 is equal to 4, even if its number of
                                                              clusters is equal to 5.

# Extract the data that belong to every cluster (here we have 5 clusters)
> VARCORE_extract_cluster_list(clusC1,C1)
[[1]]
[1] 92.41 92.01 92.22

[[2]]
[1] 93.22 93.21 93.21 93.02 93.21

[[3]]
 [1] 93.61 93.62 94.01 93.42 93.82 93.41 93.61 93.41 93.42 93.42 93.61 93.62 93.42 93.81

[[4]]
[1] 94.22 94.22 94.22 94.22 94.21

[[5]]
[1] 95.61 95.02 94.62 94.81

# Check the quality of the data fitting to the model
# p-val gives the risk error: the higher it is, the better the fitting is
> ksfit = VARCORE_calculKSFit(clusC1, C1)
> ksfit$pval
[1] 0.205
# This means that the risk of rejecting (with error) the hypothesis that the data
# is issued from a  Gaussian mixture model is 20.5%: 20.5% is not a small risk,
# so we do not reject  the assumption that the data C1 fits well
```

```
# the computed Gaussian mixture model clusC1.


# Plot fitting check
> VARCORE_plotCDF_fit(clusC1,C1,maintitle="Estimated GM and empirical CDF",
                      subtitle="Kolmogorov-Smirnov Fitting Test",
                      xlabel="Data Values",ylabel="Probabilities")
```

Example (done with R) 2. *Below, we give an example of using our software to compare the performance between two or multiple program versions.*

```
# Filename of this example: example2-VARCORE.R
# First, load the VARCORE software in R
> source('VARCORE.R')

# Load three sets of data, corresponding to three program versions: C1, C2 and C3
> C1 <- read.csv("ammp-C1", header=F)$V1
> C2 <- read.csv("ammp-C2", header=F)$V1
> C3 <- read.csv("ammp-C3", header=F)$V1

# Non-parametric performance comparison does not require clustering
# Estimate the probability that C2 < C3
> p=VARCORE_probXY_nonparam(C2,C3)
> p
0.4989594

# Estimate the probability that a single run of C1 is lower than a single run of C3
> p=VARCORE_probXYmannwhitney(C1,C3)
> p
1

# Estimate the probability that C2=min(C1,C2,C3)
> p=VARCORE_probX1min_nonparam(list(C2,C1,C3))
> p
[1] 0

# Estimate the probability that C1=min(C1,C2,C3)
> p=VARCORE_probX1min_nonparam(list(C1,C2, C3))
> p
[1] 1

# Compute the mean difference between C1 and C2
> VARCORE_meandiff_nonparam(C1,C2)
[1] 7.507419

# Parametric performance comparison
# Start by building Gaussian mixture model for each of C1, C2 and C3
> clusC1 <- VARCORE_Clustering(C1)
> clusC2 <- VARCORE_Clustering(C2)
```

```
> clusC3 <- VARCORE_Clustering(C3)

# Estimate the probability that a single run of C2 is lower than a single run of C3
> p=VARCORE_probXY_Mclust(clusC2,clusC3)
> p
0.5092883

# Estimate the probability that a single run of C1 is lower than a single run of C3
> p=VARCORE_probXY_Mclust(clusC1,clusC3)
> p
0.9999989

# Estimate the probability that C2 is the lowest execution time (single run),
# i.e. C2=min(C1,C2,C3)
> p=VARCORE_probX1min_Mclust(list(clusC2, clusC1, clusC3))
> p
[1]  2.891931e-07

# Estimate the probability that  is the lowest execution time (single run),
# i.e. C1=min(C1,C2,C3)
> p=VARCORE_probX1min_Mclust(list(clusC1, clusC2, clusC3))
> p
[1] 0.9999668

# Compute the mean difference between C1 and C2
> val=VARCORE_meandiff_Mclust(clusC1,clusC2)
> val
[1] 7.507419

# Compute the alpha-quantile of C1, for instance alpha=0.33
> val=VARCORE_quantile_Mclust(0.33, clusC1)
> val
[1] 85.58795

# Compute the probability that C1 < 85
> p=VARCORE_probXa(clusC1, 85)
> p
[1] 0.1874038

# In this example, the first version of the code is certainly always faster than
# the other two versions (and each of these two versions has about the same
# chance of being lower than the other), and it is estimated that about 33% of
# the time the first version has an execution time lower than 85.59 (and the
# probability that the execution time is lower than 85 drops to about 19%)
```

B Experimental data presentation

In this section, we present the collected performance data used in our experimental study. Here we collected real program execution times (no simulation). While we did not have opportunities to collect other kind of data (energy consumption for instance), we insist that any kind of data can be analysed by our statistics, under the condition that they are continuous (*i.e.* they do not contain tied values, or just a very little proportion). The process of collecting the data and its nature is beyond the scope of this report.

The performance data are execution times collected during many years of experimental research (around 15 years). Some experiments were very time consuming, and required many months of full time computations before finishing. Below we give the synthesis of the experimental environment and configurations used to collect our performance data.

- Total number of data samples: 2438 files, every file contains n execution times resulted for the repetition of a program execution in the same hardware and software configuration, with fixed data input.
- Number of repetitive executions per experiments (sample sizes n): from 30 up to 1000 (but the majority of samples have a size around to 30).
- Used benchmarks: all SPEC CPU applications (2001, 2006), all SPEC OMP applications, NAS Parallel Benchmark, own micro-benchmarks, parallel applications not belonging to official benchmarks (such as [CCL06, CCL03]), etc.
- The chosen program data inputs are the reference ones, which are representative of the usage of the application.
- Machines: many different Linux workstations and grid computers, going from desktop machines to HPC machines.
- OS: multiples Linux kernel versions (more than 10 years of Linux generations).
- Compilers: GNU (gcc and gfortran) and Intel (icc and ifortran), with multiple versions and compilation flags.
- Various experimental conditions: dedicated machine, shared machine, isolated machine, remote machine (batch mode, interactive mode). Mostly on low overhead fully dedicated machines.
- Measurement methods: hardware performance counters, software instrumentation, Linux `time` command.
- Time units: real time (seconds and so on), number of clock cycles (maybe very high integral numbers).

We think that we have collected enough performance data to make credible performance analysis of everyday life users.

List of Figures

List of Tables

References

[Bab11] G. J. Babu. Resampling methods for model fitting and model selection. *Journal of Biopharmaceutical Statistics*, 21:1177–1186, 2011.

[BD02] Peter J. Brockwell and Richard A. Davis. *Introduction to Time Series and Forecasting*. Springer, 2002. ISBN-13: 978-0387953519.

[BR04] G. Jogesh Babu and C. R. Rao. Goodness-of-fit tests when parameters are estimated. *Sankhyā : The Indian Journal of Statistics*, 66(1):63–74, 2004.

[CCL03] P. Cassam-Chenaï and J. Liévin. Alternative perturbation method for the molecular vibration–rotation problem. *International Journal of Quantum Chemistry*, 93(3):245–264, 2003.

[CCL06] P. Cassam-Chenaï and J. Liévin. The vmfci method: A flexible tool for solving the molecular vibration problem. *Journal of Computational Chemistry*, 27(5):627–640, 2006.

[Che07] Michael R. Chernick. *Bootstrap Methods: A Guide for Practitioners and Researchers (2d edition)*. Wiley series in Probability and Statistics, 2007.

[CHL15] Didier Chauveau, David R. Hunter, and Michael Levine. Semi-parametric estimation for conditional independence multivariate finite mixture models. *Statistics Surveys*, 9:1–31, 2015.

[DH97] A.C. Davison and D.V. Hinkley. *Bootstrap Methods and their Application*. Cambridge Series in Statistical and Probabilistic Mathematics, 1997.

[DLR77] A.P. Dempster, N.M. Laird, and D.B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society, Series B*, 39(1):1–38, 1977.

[dS86] R. B. d'Agostino and M. A. Stephens. *Goodness-of-fit techniques*. CRC Press, 1986.

[FRMS12] Chris Fraley, Adrian Raftery, Brendan Murphy, and Luca Scrucca. mclust version 4 for r : Normal mixture modeling for model-based clustering, classification, and density estimation. Technical Report 597, Department of Statistics, University of Washington, 2012.

[FTP14] André Fujita, Daniel Y. Takahashi, and Alexandre G. Patriota. A non-parametric method to estimate the number of clusters. *Computational Statistics and Data Analysis*, 73:27–39, 2014.

[GBE07] Andy Georges, Dries Buytaert, and Lieven Eeckhout. Statistically rigorous Java performance evaluation. In *Proceedings of the Twenty-Second ACM SIGPLAN Conference on OOPSLA*, ACM SIGPLAN Notices 42(10), pages 57–76, Montréal, Canada, October 2007.

[HW73] Myles Hollander and Douglas A. Wolfe. *Nonparametric Statistical Methods*. Wiley-Interscience, 1973. ISBN-13 978-0-471-190455.

[Jai91] Raj Jain. *The Art of Computer Systems Performance Analysis : Techniques for Experimental Design, Measurement, Simulation, and Modelling*. John Wiley and Sons, inc., New York, 1991. ISBN-13: 978-0-471-503361.

[JLT06] William Jalby, Christophe Lemuet, and Sid-Ahmed-Ali Touati. An Efficient Memory Operations Optimization Technique for Vector Loops on Itanium 2 Processors. *Concurrency and Computation: Practice and Experience*, 11(11):1485–1508, 2006.

[KY12] Ivan Kojadinovic and Jun Yan. Goodness-of-fit testing based on a weighted bootstrap : A fast large-sample alternative to the parametric bootstrap. *Canadian Journal of Statistics*, 40(3):480–500, 2012.

[Lil00] David J. Lilja. *Measuring Computer Performance: A Practitioner's Guide*. Cambridge University Press, 2000. ISBN-13: 978-0521641050.

[LJT04] Christophe Lemuet, William Jalby, and Sid-Ahmed-Ali Touati. Improving Load/Store Queues Usage in Scientific Computing. In *the International Conference on Parallel Processing (ICPP)*, Montreal, Canada, August 2004. IEEE.

[LOW09] Hugh Leather, Michael O'Boyle, and Bruce Worton. Raced Profiles: Efficient Selection of Competing Compiler Optimizations. In *Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES '09)*. ACM SIGPLAN/SIGBED, June 2009.

[Man97] Bryan F.J. Manly. *Randomization, Bootstrap and Monte Carlo Methods in Biology (2d edition)*. Chapman & Hall/CRC Press, 1997.

[MDSH09] Todd Mytkowicz, Amer Diwan, Peter F. Sweeney, and Mathias Hauswirth. Producing wrong data without doing anything obviously wrong! In *ASPLOS*, 2009.

[MP00] Geoffrey MacLachlan and David Peel. *Finite Mixture Models*. Wiley series in Probability and Statistics, New York, 2000.

[MRT11] K. L. Mengersen, C. P. Robert, and D. M. Titterington. *Mixture Estimation and Applications*. Wiley, 2011.

[MtATB10] Abdelhafid Mazouz, the Ali Touati, and Denis Barthou. Study of Variations of Native Program Execution Times on Multi-Core Architectures. In *International Workshop on Multi-Core Computing Systems (MuCoCoS)*. IEEE, Krakow, Poland, February 2010.

[RBRGTM14] Patricia Reynaud-Bouret, Vincent Rivoirard, Franck Grammont, and Christine Tuleau-Malot. Goodness-of-fit tests and nonparametric adaptive estimation for spike train analysis. *The Journal of Mathematical Neuroscience*, 4, 2014.

[Sap90] Gilbert Saporta. *Probabilités, analyse des données et statistique*. Editions Technip, Paris, France, 1990. ISBN 978-2-7108-0814-5.

[SMQ93] W. Stute, W. Gonzáles Manteiga, and M. Presedo Quindimil. Bootstrap based goodness-of-fit-tests. *Metrika*, 40:242–256, 1993.

[Tha10] O. Thas. *Comparing Distributions*. Springer, 2010.

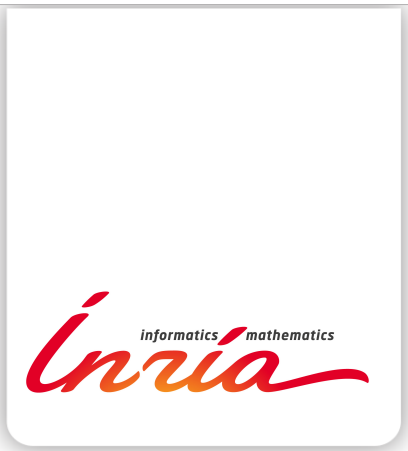
[TSM85] D. M. Titterington, A. F. Smith, and U. E. Makov. *Statistical Analysis of Finite Mixture Distributions*. Wiley, 1985.

[TWB10] Sid Touati, Julien Worms, and Sébastien Briais. The Speedup-Test. Technical report, University of Versailles Saint-Quentin en Yvelines, January 2010. Research report number HAL-inria-00443839, http://hal.archives-ouvertes.fr/inria-00443839.

[TWB13] Sid Touati, Julien Worms, and Sébastien Briais. The Speedup-Test: A Statistical Methodology for Program Speedup Analysis and Computation. *Concurrency and Computation: Practice and Experience*, 25(10):1410–1426, 2013.

[TWH01] Robert Tibshirani, Guenther Walther, and Trevor Hastie. Estimating the number of clusters in a data set via the gap statistic. *Journal of the Royal Statistical Society, Series B*, 63(2):411–423, 2001.

[vdV00] A. W. van der Vaart. *Asymptotic statistics*. Cambridge University Press, 2000. ISBN-13: 978-0-521-784504.



RESEARCH CENTRE
SOPHIA ANTIPOLIS – MÉDITERRANÉE

2004 route des Lucioles - BP 93
06902 Sophia Antipolis Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399