Probability Distributions as Program Variables

Dimitrios Milios

Master of Science

School of Informatics

University of Edinburgh

2009

Abstract

In this work we introduce a new method for performing computations on arbitrarily distributed random variables. Concisely, the probability distributions of input random variables are approximated by mixture models. Computations are then applied to each one of their mixture components. Thus, the final results are also random variables that can be either queried for their density and distribution functions, or even used in future computations. Two alternative types of mixture model approximations have been implemented: mixture of uniforms and mixture of Gaussians. It is remarkable that in some cases, our approach outperformed the equivalent numerical approaches from the literature, in terms of accuracy.

The greatest amount of work in this project was spent on the efficiency and accuracy of computations of independent random variables. However, issues of dependencies that are arise in the computations have been studied as well. A way of tracking these dependencies has been developed, although it was not incorporated into the main approach. Instead, a Monte Carlo version was implemented, as a demonstration and proof of concept.

Eventually, a C++ library named *Stochastic* is produced, offering the users a datatype for random variables and a set of operations. The source code is available at `git://github.com/dmilios/stochastic.git`.

Acknowledgements

I would like to thank my supervisor Conrad Hughes for his help and support throughout the project.

Declaration

Contents

Chapter 1

Introduction

In this introductory chapter, we will have an overview of the project's goals and the methods used to achieve them. Moreover, the most significant related works are presented, so as to have an overview of the advances in the field and highlight the differences regarding our approach.

The second chapter is an introduction to the concepts of random variables. Moreover, we present some theoretical results that are actually exploited in the implementation. The third chapter outlines the main elements of the *Stochastic* library, which is the piece of software produced. Some implementation issues are also discussed. Eventually, the fourth chapter includes a number of experiments and examples that are used for the evaluation of our approach.

1.1 Problem Definition

The main goal of this research is to develop the beginnings of a framework that enables the manipulation of probability distributions as program variables. In other words, the purpose is to implement an internal representation of probability distributions of continuous random variables that supports numerical operations on them. The result eventually will be a new datatype that implements the notion of random variable. A set of operations on this type will be implemented including addition, subtraction, multiplication, division, minimum and maximum of two random variables. These operations will also be defined between random variables and scalar values. We shall refer to these kinds of operation as functions of one random variable, or unary operations. The binary operations are essentially functions of two random variables. Moreover, since these binary operations are defined for arbitrarily distributed random variables, the generalisation to n random variables is straightforward, since it is equivalent to successive application of the binary operators.

For the moment, let us consider continuous random variables just as a generalisation of the scalar real-valued ones. The latter can be assigned with specific values, while the former are associated with a whole set of values described by a probability distribution. In fact, a random variable can be degenerated to scalar if its variance is equal to zero. A formal definition for the random variables can be found in Chapter 2.

The results of these operations are well defined in theory, and their definitions can be found in several textbooks [44, 47]. In Chapter 2, there is a detailed presentation of these results and their limitations. Most of these are defined as integrals over expressions of the probability density and the cumulative distribution functions. However, there are more tractable expressions for certain kinds of distributions, such as the sum of uniforms for example described in Section 2.3.2. Even in this case though, these expressions can not be generalised for n arbitrarily distributed random variables.

It is also noted that we are mainly interested in independent random variables, in this project. Performing computations between random variables with known dependency, is subject for future work. However, even if we assume that the initial variables of the program are independent, that is not valid for those that are derived thoughout the computations. Let us consider the example of the following pseudocode, where A, B, C and D are random variables:

```
1. Compute the sum
   of the minimum and the maximum of $A$ and $B$
2. $C \leftarrow min(A, B)$
3. $D \leftarrow max(A, B)$
4. return $D + C$
```

Even if we accept that the input variables A and B are independent, this assumption does not apply for their minimum and maximum. In addition, it is very difficult to characterise the dependency of the example, since it is non-linear. These kinds of dependency, which arise throughout the computations, will actually be examined in the current work.

1.2 Outline of the Solution

In order to address the problems described in Section 1.1, we have implemented two completely different approaches, each one aiming at a different objective. Eventually, the users will be able to choose the computational method according to their needs.

Approximation with Mixture Models: It has been already reported that for certain kinds of distributions the results of binary operations can be written as closed-form expressions of a few parameters. In this first approach, we take advantage of this property by approximating the densities of the original distributions with mixture models. The idea that any probability distribution can be approximated arbitrarily well by a mixture model was first encountered in [2].

Concisely, a mixture distribution is a linear combination of other distributions, called mixture components. Since each component is a valid probability distribution as well, its area is equal to 1. However, each one of them is assigned a non-negative weight, where the weights sum up to 1. In other words, the weights scale down the area of its components, and since the original area was 1, the new area equals to the weight. The idea is that the distributions of the input random variables are going to be approximated by mixtures of N simpler distributions, whose results are either known or easy to approximate. The application of an operator on mixture models involves:

- **In case of unary operators**
  Applying the operator on each component independently. While the area (or weight) of each components will remain unchanged, its shape and position will be altered though, resulting in a translated mixture distribution.

- **In case of binary operators**
  The operator will be applied on each pair of components of the input distributions. The result will be N^2 components, where the weight of each one of them will be the product of the original components' weights. Eventually, we re-approximate the result with a mixture of N components, otherwise the component number would grow exponentially with the number of computations we perform.

A proof for the validity of the method of applying operations on mixture models can be found in Section 3.5. Two alternative probability distributions were chosen as mixture components. The user will be able to choose between uniform and Gaussian components. The operations between pairs of Gaussian and uniform random variables are described in Chapter 2.

Dependency Tracking Monte Carlo: Monte Carlo simulation is a powerful alternative for estimating the results of operations on random variables. It involves sampling from the input distributions and applying the operator on each pair of samples. If we repeat the process an infinite number of times, we can hope that the density of the samples approximates the true density of the resulting random variable. It is evident that a Monte Carlo approach can prove rather computationally expensive, however it can always be useful as an evaluation measure.

So in this project, the Monte Carlo method adopted does not essentially aims at an optimum combination between accuracy and efficiency, in contrast with the mixture model method. Instead, we propose a Monte Carlo approach that takes into account the dependencies that arise thoughout the computations. More specifically, the relations between the random variables are recorded in a form of dependency graph that affects the sampling procedure. A similar approach is used in [21], where the objective was to describe a large family of probabilistic models, while we are interested in computations. The method is further described in Section 3.6.

1.3 Related Work

The most obvious motivation for research in this area is the need for performing computations introducing uncertainty. A solution to the problem given in [38], involves the use of intervals in order to describe the upper and the lower values to be expected for a variable. One pretty obvious drawback of the interval approach however, is the fact that all the values that lie within an interval are considered as equally probable, even if that is not the case.

The use of random variables implies that our expectation is described by a probability distribution, instead of just intervals. Actually, the use of probability distributions was quite natural, since by definition the probability is a way of quantifying the uncertainty. The most straightforward approach to the problem of performing operations on random variables is the analytical one. However, the theoretical results of operations between random variables turn out to be difficult to compute, as most of the time there is no closed-form expression for them. This subject and the relevant bibliography are further discussed in Chapter 2, since the currently proposed method exploits the results of the analytical methods.

Given these difficulties, numerical methods were used for the task. The term “Monte Carlo Method” was first introduced in [36], although methods that rely on random sampling had been in use long ago for solving problems. In fact, Monte Carlo can be proven a powerful tool, provided that we simulate adequately the process that we want to estimate. It has been traditionally used as a point of reference in numerous works, including [33, 12]. However, the high computational cost of the method led past researchers towards more efficient approaches.

Most of the numerical non-Monte Carlo approaches make use of discretisation of probability distributions. The earliest works [9, 29] focus on operations between independent variables only. Dependency issues are addressed in some notable approaches based on the theory of copulas [39]. Starting from the work in [37], it gives an answer to Kolmogorov’s question: what is the distribution of the sum of two random variables $Z = X + Y$

when the dependency between them is unknown? This is the case where the marginal distributions $F_X(x)$ and $F_Y(y)$ are known and there is no information about the joint distribution of X and Y. The solution is given in the form of upper and lower bounds for the resulting distribution $F_Z(z)$. This approach was expanded in [51], where numerical algorithms had been developed for the basic arithmetic operations. The problem of unknown dependencies was also a subject in the approach described in [4, 12, 5], which uses a discretisation method. In [26], it was shown that both discretisation-based and copula-based methods are equivalent in the way they produce bounds for the resulting distributions. It is noted however that producing dependency bounds is beyond the goals of this project. What we are examining is whether the discretisation approach can be evolved in a way that produces more accurate results when the dependency structure is known.

It is worth outlining the discretisation approach as described in [4], so as to identify the differences from the mixture model approximation currently proposed. A discretisation of a probability density function is defined as a histogram, as shown in Figure 1.1, adapted from [26]. This implies that the density function of a random variable is mapped to a number of intervals, each one of them assigned a probability. The concept of histograms is generalised so as to allow overlapping intervals as well.
Thus, binary operations between random variables are expressed as operations on sets of intervals and their associated probabilities. For instance, suppose that we have two variables, A and B. For any of the four basic arithmetic operations, each interval of the N_A intervals of A, will interact with each of the N_B intervals of B, as interval arithmetic requires [30]. Finally, the histogram corresponding to the result distribution will consist of $N_A \times N_B$ intervals. The probability assigned to each one of the intervals produced, will be equal to the product of the probabilities of the original intervals, and the sum of these probabilities will be 1.

If we think of a histogram as a mixture of uniform distributions, it is evident that each interval of the histogram corresponds to a uniformly distributed mixture component. In the same way, the probability associated with an interval is equal to the weight of the component, since any distribution integrates up to 1. Thus, the result of a pair of uniform components is approximated by a uniform distribution as well. However for the case of the basic binary arithmetic operations, this result is not consistent with the algebra of random variables [47]. We will see in Chapter 2 that we can use distributions that better approximate the theoretical results. The result will essentially be a mixture of another kind of distributions that are more complex that uniforms, but they will still have known functional forms.

Another interesting work is [33], where a way of tracking the dependencies that arise throughout the computations was introduced. In fact, this work deals with the second of our objectives as presented in the previous section. It uses an interval arithmetic-based approach, similar to that of [4] in

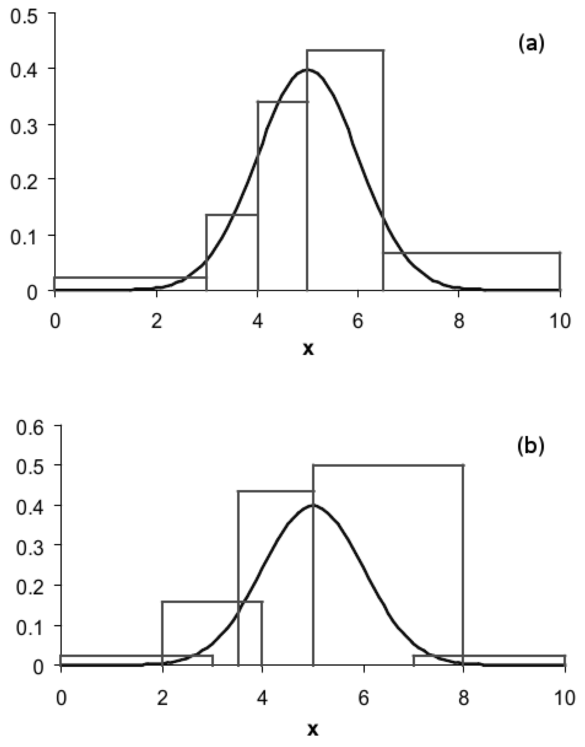

Figure 1.1: Discretisation (a) and Generalised Discretisation (b) of a probability density function

order to perform operations between random variables. The main difference is that it allows interaction between specific pairs of intervals, according to the dependency status between the random variables. A modification of that method so as to be incorporated into the current implementation, could be subject of future work. The Monte Carlo counterpart that is implemented in the current project, could serve as a valid alternative.

1.4 Hypotheses to Test

Given the state of the art in the field as recorded in the literature, the tool discussed in this work will enable us to experiment with a number of hypotheses that concern performing operations on random variables. The issues that we are interested in are summarised as follows:

- Is the currently proposed mixture model approach more accurate than the discretisation approach adopted in a number of previous works? In this work, we have been experimenting with using the actual intermediate results between components, instead of the interval arithmetic-based results, according to the discretisation approach in [33, 4]. Does this choice lead to better approximations of the final results?

- What is the relationship in terms of both accuracy and efficiency between the two different types of components, uniforms and Gaussians?

- Does actually the proposed dependency tracking Monte Carlo method effectively capture dependencies that arise as a result of common subexpressions in computations?

The evidence that can be used to address these questions can be found in Chapter 4, where we also discuss the evaluation criteria used. The following chapter focuses on the theoretical backround of this project, while in Chapter 3, we present the most significant implementation issues.

Chapter 2

The Algebra of Random Variables

The major task of this project is to perform as efficiently as possible a computational algorithm for performing operations on random variables. In brief, the approach adopted involved approximation with mixture models and performing operations between the approximating components, taking advantage of the fact that the theoretical results of these operations are easily tractable for certain kinds of probability distributions. The purpose of this chapter is to elaborate on the random variable concept, and to give an overview of the theoretical results of performing operations on them. This knowledge is also essential for evaluating the results of the approximations.

In this chapter, we are presenting the theoretical results of the sum, the difference, the product, the ratio, the minimum and the maximum of two random variables, covering the entire range of operators we are interested in for this project. Proofs for most of these can be found in *"The Algebra of Random Variables"* [47], whose title was borrowed for this chapter. It is important to note that the results presented in the current section are only valid under the assumption that the variables are independent.

2.1 Formal Definition of Random Variables

More formally, a random variable $X(\zeta)$ is a function that associates the events ζ of a sample space S, with real numbers. In other words, the domain of a random variable X is the sample space S, while the range of X is the collection of all the values of $X(\zeta)$.

According to an intuitive interpretation given in [22], a random variable is an expression whose value is the outcome of an experiment. The outcomes of an experiment such as flipping of a coin, are the events ζ of the sample space S. In the case of coin flipping, the outcome can be either heads or tails, which are the possible values for ζ. Thus, a random variable is essentially a mapping of the possible outcomes to real numbers, say 0 for tails and 1 for heads. The coin flipping example is illustrated in Figure 2.1 adapted from [27].

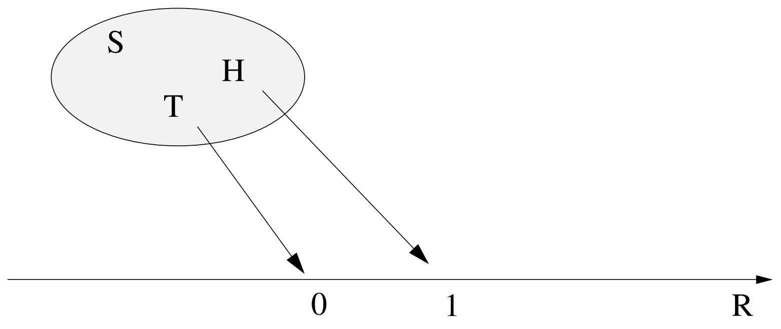

Figure 2.1: Illustration of how a random variable X associates outcomes of an experiment with real numbers

That was an example of discrete random variables, however, continuous ones are mainly of interest in this project. An experiment with no discrete outcomes could be the measuring of the duration of a task. Provided that there is no limit in accuracy, any real positive number could be the outcome.

There is an interesting discussion in Chapter 9 of [17], suggesting that the term "random variable" is confusing, as it is more like a "random function". However, we should not confuse these two terms, as the author was only trying to emphasise the function-like character of a random variable. After all, a random function denotes a random mapping of the domain values onto the values of the range. On the contrary, the mapping between the outcomes of an experiment and the values of a random variable is not random at all.

Probability Distributions

As already stated in Chapter 1, random variables are associated with probability distributions, instead of single values. Probability distributions assign probabilities to each one of the possible values of a random variable, and they are completely described by the cumulative distribution function (CDF), or simply "distribution function". The cumulative distribution function of a random variable X, denoted by $F_X(x)$, is a non-decreasing function with domain $(-\infty, \infty)$ and range $[0, 1]$. It is defined as the probability of X being less than or equal to some given value x.

$$F_X(x) = P(X \leq x) \tag{2.1}$$

Continuous random variables can also be described by the probability density function (PDF), or simply "density function". It is essentially a non-zero function, denoted by $f_X(x)$ and defined as the derivative of the distribution function.

$$f_X(x) = \frac{\mathrm{d}F_X(x)}{\mathrm{d}x} \tag{2.2}$$

An interesting property of continuous distributions is that any event on its own has probability equal to zero, despite the value of the density function. The density function at a point x should be interpreted as the probability that the outcome lies in the nearby area. More formally, the probability of an outcome lying within the interval $[a, b]$ will be:

$$P(a \leq X \leq b) = \int_a^b f_X(x)\mathrm{d}x \tag{2.3}$$

On the other hand, each event of a discrete random variable is assigned with a specific probability. If a probability of an event is zero, then it is just impossible to happen. The probability mass function $f_X(x)$ is defined as the mapping of values x of a random variable X onto probabilities.

$$f_X(x) = P(X = x) \tag{2.4}$$

Eventually, it is also possible a random variable is both discrete and continuous. This means that its probability distribution possesses properties of a continuous distribution for certain areas of its domain, while for others possesses properties of a discrete one. The way such mixed distributions are handled in terms of the project is described in Section 2.2.3.

2.2 Functions of One Random Variable

In the general case, a function $g(X)$ of a random variable X is a composite function

$$Y = g(X) = g[X(\zeta)] \tag{2.5}$$

that has as domain the sample space S, since ζ are events of S. It is evident that the function $g(X)$ maps events of S onto real values, which is consistent with the definition of random variables given in Section 2.1. Thus, the result Y of $g(X)$ is a random variable as well.

Examples of functions of one random variable are expressions such as $X + 5$ or $min(X, 2)$. We will focus on the ones that are related to the four basic arithmetic operations, plus the minimum and the maximum.

2.2.1 Linear Functions

Most of the cases we are concerned with fall in the case of linear functions. More specifically, the cases that involve addition, subtraction, multiplication and division of a random variable by a scalar value, are covered in this section. A linear function $g(X)$ of a random variable X is of the form:

$$Y = aX + b \tag{2.6}$$

where a and b are scalar values. In Chapter 5 of [41], it is shown that the distribution function of random variable Y can be expressed in terms of the distribution function of X:

$$F_Y(y) = P\left(X \leq \frac{y-b}{a}\right) = F_X\left(\frac{y-b}{a}\right), \quad a > 0 \tag{2.7}$$

$$F_Y(y) = P\left(X \geq \frac{y-b}{a}\right) = 1 - F_X\left(\frac{y-b}{a}\right), \quad a < 0 \tag{2.8}$$

We are also attaching the density $f_Y(y)$ of the linear function of the random variable X. The proof can be found in [41] as well.

$$f_Y(y) = \frac{1}{|a|} f_X\left(\frac{y-b}{a}\right) \tag{2.9}$$

The Uniform Case We can see that a linear transformation, such as the one above, does not alter the form of the distribution function. Parameter b shifts the distribution function, while coefficient a scales it along the horizontal axis. For example, if X follows a uniform distribution, its distribution function will be of the form:

$$F_X(x) = \begin{cases} 0, & x < c_1 \\ \frac{x-c_1}{c_2-c_1}, & c_1 \leq x < c_2 \\ 1, & c_2 \leq x \end{cases} \tag{2.10}$$

where c_1, c_2 are the endpoints of the distribution. In case a is positive, according to Equation (2.7) the second case of (2.10) becomes $\frac{y-b-ac_1}{ac_2-ac_1}$. Since the distribution function of X was right continuous, so must be its linear transformation as well. Thus, the following equations should hold:

$$F_Y(c_1') = 0 \Rightarrow \frac{c_1' - b - ac_1}{ac_2 - ac_1} = 0 \tag{2.11}$$

$$\lim_{x \to c_2'^-} F_Y = 1 \Rightarrow \frac{c_2' - b - ac_1}{ac_2 - ac_1} = 1 \tag{2.12}$$

By solving (2.11) and (2.12) we can find that Y follows a uniform distribution with endpoints $c_1' = ac_1 + b$ and $c_2' = ac_2 + b$. We can also use Equation (2.8) in a similar way to obtain the endpoints, $c_1' = ac_2 + b$ and $c_2' = ac_1 + b$ of the uniform variable Y for negative values of a. Summarising, a linear function of a uniform random variable X will be:

$$F_Y(y) = \begin{cases} 0, & y < ac_1 + b \\ \frac{y-ac_1+b}{ac_2-ac_1}, & ac_1 + b \leq y < ac_2 + b, \quad a > 0 \\ 1, & y \geq ac_2 + b \end{cases} \tag{2.13}$$

$$F_Y(y) = \begin{cases} 0, & y < ac_2 + b \\ \frac{y-ac_2+b}{ac_1-ac_2}, & ac_2 + b \leq y < ac_1 + b, \quad a < 0 \\ 1, & y \geq ac_1 + b \end{cases} \tag{2.14}$$

The Gaussian Case The distribution function of a random variable X following a Gaussian distribution with mean μ_X and variance σ_X^2 will be:

$$F_X(x) = \frac{1}{2} + \frac{1}{2}\text{erf}\left(\frac{x-\mu_X}{\sqrt{2\sigma_X^2}}\right) \tag{2.15}$$

where erf(x) is the error function, which according to [1] is defined as:

$$\text{erf}(x) = \frac{2}{\sqrt{\pi}}\int_0^x e^{-t^2}\mathrm{d}t \tag{2.16}$$

By using Equation (2.7) again, considering positive a, we obtain:

$$F_Y(y) = \frac{1}{2} + \frac{1}{2}\text{erf}\left(\frac{y-a\mu_X-b}{\sqrt{2a^2\sigma_X^2}}\right), \quad a>0 \tag{2.17}$$

So, we can see that Y is a Gaussian variable with mean $\mu_Y = a\mu_X + b$ and variance $\sigma_Y^2 = a^2\sigma_X^2$. Now for the negative values of a, if we use (2.8) we obtain:

$$F_Y(y) = \frac{1}{2} - \frac{1}{2}\text{erf}\left(\frac{y-a\mu_X-b}{\sqrt{2a^2\sigma_X^2}}\right), \quad a<0 \tag{2.18}$$

As reported in [1], error function is an odd function, which entails that $-\text{erf}(x) = \text{erf}(-x)$. Thus, (2.18) will become:

$$F_Y(y) = \frac{1}{2} + \frac{1}{2}\text{erf}\left(\frac{-y+a\mu_X+b}{\sqrt{2a^2\sigma_X^2}}\right), \quad a<0 \tag{2.19}$$

Eventually, (2.19) implies that the distribution of the result will be a Gaussian with mean $\mu_Y = a\mu_X + b$ and variance $\sigma_Y^2 = a^2\sigma_X^2$, for $a < 0$ as well.

We can see that result for both uniform and Gaussian distributions are tractable and very easy to be estimated. Such findings seem to justify the choice of the author to use this kind of distributions as components, in order to describe more complex ones. So, we can apply any linear function to each one of the components independently. The final result does not need to be re-approximated, as only the parameters of the mixture distributions will change, not their form.

2.2.2 Division by Random Variables

Given that we already know the results of linear functions of random variables, dividing by a random variable simply demands estimating the expression $1/X$, which does not fall in the linear case. In spite of this, it is still within the range of operators we wish to implement in terms of the project. Again in Chapter 5 of [41], there is a proof that the probability density function $f_Y(y)$ of the random variable $Y = 1/X$ will be:

$$f_Y(y) = \frac{1}{y^2} f_X\left(\frac{1}{y}\right) \tag{2.20}$$

One way to obtain the distribution function would be to compute the indefinite integral of (2.20). Indeed, by applying the chain rule we can find that the distribution function will be of the form $-F_X(\frac{1}{y}) + c$. We can not find the constant c for any kind of distribution though, as we do not know its support. Instead, we can think of the definition of the distribution function $F_Y(y)$, which is $P(Y \leq y)$ or $P(1/X \leq y)$. So we can discriminate the following cases:

- If $y > 0$, then we have $1/x \leq y$
  - for $x \geq 1/y$, if $x > 0$
  - always, if $x < 0$

  Hence:

  $$F_Y(y) = P(X \geq 1/y) + P(X \leq 0) = 1 - F_X\left(\frac{1}{y}\right) + F_X(0) \tag{2.21}$$

- If $y < 0$, then we have $1/x \leq y$
  - never, if $x > 0$
  - for $x \geq 1/y$, if $x < 0$

  Hence:

  $$F_Y(y) = P(X \geq 1/y) - P(X \geq 0) = -F_X\left(\frac{1}{y}\right) + F_X(0) \tag{2.22}$$

The equations above make clear that the density and the distribution functions of the division by a random variable can be expressed in terms of the original density and distribution functions repsectively. These functions however, are known for any kind of distribution in terms of the project, since the distributions are supposed to be approximated by mixture models. So, it is possible to estimate the quotient $1/X$ for any distribution of X by applying Equations (2.20), (2.21) and (2.22) directly, instead of on each one of the components individually. Eventually, we can re-approximate this intermediate result with a new mixture model.

Together with the linear functions in the previous section, we have covered all the cases of the four basic arithmetic operations between random and scalar variables.

2.2.3 Minimum and Maximum between a Random Variable and a Constant

The list of functions to be implemented also includes the minimum and the maximum of random variables, as we have seen in the introduction. In this section, we will have a look on the expressions $min(X, a)$ and $max(X, a)$, where X is a random variable and a is a scalar value. Starting form the minimum, we can distinguish three cases, also illustrated in Figure 2.2:

- If the value a lies after the whole support of the distribution of X, then the result of $min(X, a)$ will be the random variable X itself.
- If the value a lies before the whole support of the distribution of X, then the result of $min(X, a)$ will always be the scalar value a. In terms of continuous probability distributions, this result can be thought of as a Gaussian distribution with mean a and zero variance (or really close to zero). In the current project, such results are approximated with low variance Gaussian spikes.
- If the value a lies within the support of the distribution of X, the result will be a mixed random variable. More specifically, the continuous part of the result will be in the range $(-\infty, a)$, and the density function in that range will be the same as that of X, while its area will be integrated up to $F(a)$. The remaining $1 - F(a)$ area is actually "vaporised" to the probability of the value a. As we did in the previous case, the probability of a will be expressed as an extremely tight Gaussian. So, the final result will essentially be a mixture model consisting of these two components.

It is straightforward to produce similar results for the maximum, just by taking into account the complementary areas of the support of X.

2.3 Sum and Difference of Random Variables

The density function of the sum of two independent random variables X and Y, is the convolution of the density functions of the operant distributions [47], $f_X x$ and $f_Y(y)$ respectively. The random variable $Z = X + Y$ will have density function:

$$(f_X * f_Y)(z) = \int_{-\infty}^{\infty} f_X(z-y) f_Y(y) \mathrm{d}y = \int_{-\infty}^{\infty} f_Y(z-x) f_X(x) \mathrm{d}x \quad (2.23)$$

The difference of two random variables $Z = X - Y$ can be regarded as the sum $Z = X + Y'$, whose second argument is $Y' = -Y$. The use of a negative sign on a random variable falls in the linear function case explained in Section 2.2.1. More specifically, the density function of Y will

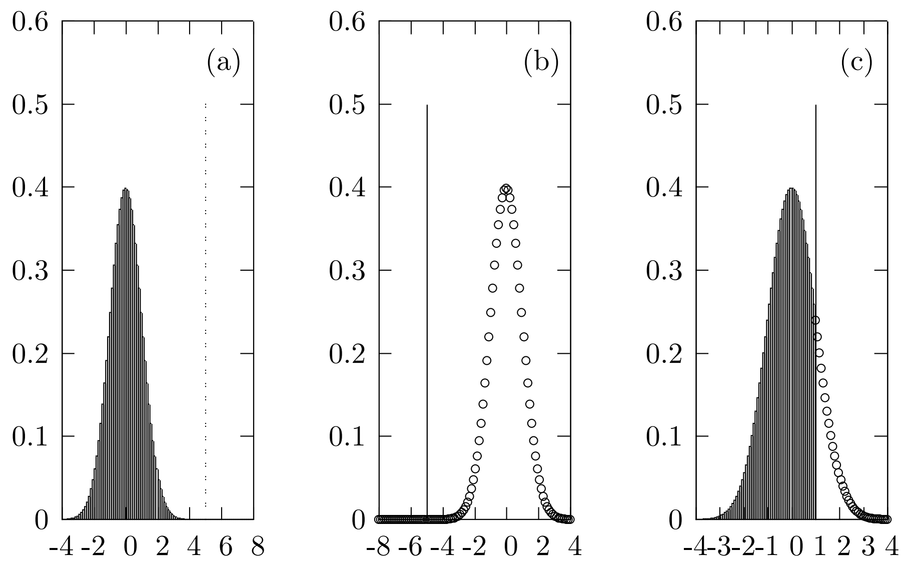

Figure 2.2: Minimum of a random variable and a constant

be $f_{Y'} = f_Y(-y')$, since $a = -1$ and $b = 0$, according to Equation (2.9). This simple linear transformation entails that the density function is only "reversed", with no change in its shape or its variance.

Summarising, provided that we know the probability density functions of the input variables, we can theoretically compute both sum and difference. The density function is not always known though, a fact that led to research towards both parametric and non-parametric approaches to density estimation. This subject is further discussed in Section 3.3.1. Nevertheless, even if the densities are well known, there is no guarantee that the integral in Equation (2.23) is always tractable, a fact that motivated the approximation with mixture models in the first place. In the following paragraphs we show the results for the candidate mixture components.

2.3.1 Sum and Difference of Gaussians

The results for the sum and the difference of independent Gaussian random variables can be found in several textbooks, such as [20]. In brief, the sum $Z = X + Y$ will follow a Gaussian distribution as well, with mean $\mu_Z = \mu_X + \mu_Y$ and variance $\sigma_Z^2 = \sigma_X^2 + \sigma_Y^2$. In the same way, the difference between two normally distributed variables $Z = X - Y$ is also normally distributed with mean $\mu_Z = \mu_X - \mu_Y$ and variance again $\sigma_Z^2 = \sigma_X^2 + \sigma_Y^2$. This result can be easily obtained if we invert the sign of the variable Y.

2.3.2 Sum and Difference of Uniforms

Although both convolution and cross-correlation of uniform densities are relatively easy to be computed, as we will see the results do not fall into the known types of densities, in contrast with the Gaussian example. The authors of [8] provide a form for the convolution of N independent uniformly distributes random variables. Since we are interested in binary operations only, we present the formula for the sum of two random variables that follow uniform distributions $U(a_1, b_1)$ and $U(a_2, b_2)$.

- If $a_1 + b_2 < a_2 + b_1$

$$f_Z(z) = \begin{cases} 0, & z < a_1 + a_2 \\ \frac{z-(a_1+a_2)}{(b_1-a_1)(b_2-a_2)}, & a_1 + a_2 \leq z < a_1 + b_2 \\ \frac{1}{b_1-a_1}, & a_1 + b_2 \leq z < a_2 + b_1 \\ \frac{-z+(b_1+b_2)}{(b_1-a_1)(b_2-a_2)}, & a_2 + b_1 \leq z < b_1 + b_2 \\ 0, & b_1 + b_2 \leq z \end{cases} \tag{2.24}$$

$$F_Z(z) = \begin{cases} 0, & z < a_1 + a_2 \\ \frac{z^2-2z(a_1+a_2)}{2(b_1-a_1)(b_2-a_2)} + c_1, & a_1 + a_2 \leq z < a_1 + b_2 \\ \frac{z}{b_1-a_1} + c_2, & a_1 + b_2 \leq z < a_2 + b_1 \\ \frac{-z^2+2z(b_1+b_2)}{2(b_1-a_1)(b_2-a_2)} + c_3, & a_2 + b_1 \leq z < b_1 + b_2 \\ 1, & b_1 + b_2 \leq z \end{cases} \tag{2.25}$$

- If $a_1 + b_2 > a_2 + b_1$

$$f_Z(z) = \begin{cases} 0, & z < a_1 + a_2 \\ \frac{z-(a_1+a_2)}{(b_1-a_1)(b_2-a_2)}, & a_1 + a_2 \leq z < a_2 + b_1 \\ \frac{1}{b_2-a_2}, & a_2 + b_1 \leq z < a_1 + b_2 \\ \frac{-z+(b_1+b_2)}{(b_1-a_1)(b_2-a_2)}, & a_1 + b_2 \leq z < b_1 + b_2 \\ 0, & b_1 + b_2 \leq z \end{cases} \tag{2.26}$$

$$F_Z(z) = \begin{cases} 0, & z < a_1 + a_2 \\ \frac{z^2-2z(a_1+a_2)}{2(b_1-a_1)(b_2-a_2)} + c_1, & a_1 + a_2 \leq z < a_2 + b_1 \\ \frac{1}{b_2-a_2} + c_2, & a_2 + b_1 \leq z < a_1 + b_2 \\ \frac{-z^2+2z(b_1+b_2)}{2(b_1-a_1)(b_2-a_2)} + c_3, & a_1 + b_2 \leq z < b_1 + b_2 \\ 1, & b_1 + b_2 \leq z \end{cases} \tag{2.27}$$

- If $a_1 + b_2 = a_2 + b_1$

$$f_Z(z) = \begin{cases} 0, & z < a_1 + a_2 \\ \frac{z-(a_1+a_2)}{(b_1-a_1)(b_2-a_2)}, & a_1 + a_2 \leq z < a_1 + b_2 = a_2 + b_1 \\ \frac{-z+(b_1+b_2)}{(b_1-a_1)(b_2-a_2)}, & a_1 + b_2 = a_2 + b_1 \leq z < b_1 + b_2 \\ 0, & b_1 + b_2 \leq z \end{cases} \tag{2.28}$$

$$F_Z(z) = \begin{cases} 0, & z < a_1 + a_2 \\ \frac{z^2-2z(a_1+a_2)}{2(b_1-a_1)(b_2-a_2)} + c_1, & a_1 + a_2 \leq z < a_1 + b_2 = a_2 + b_1 \\ \frac{-z^2+2z(b_1+b_2)}{2(b_1-a_1)(b_2-a_2)} + c_3, & a_1 + b_2 = a_2 + b_1 \leq z < b_1 + b_2 \\ 1, & b_1 + b_2 \leq z \end{cases} \tag{2.29}$$

It is straightforward to compute the constants c_1 and c_3 by solving the equations $F_Z(a_1 + a_2) = 0$ and $F_Z(b_1 + b_2) = 1$ respectively. By using these values, we can also find the c_2. For example in the first case, the CDF is supposed to be right continuous at the point $a_1 + b_2$. Thus, the value $F_Z(a_1 + b_2)$ should be equal to the limit $\lim\limits_{z \to a_1+b_2^-} F_Z(z)$. Eventually, the values for the constants are:

$$c_1 = \frac{(a_1 + a_2)^2}{2(b_1 - a_1)(b_2 - a_2)} \tag{2.30}$$

If $a_1 + b_2 < a_2 + b_1$

$$c_2 = \frac{(a_1 + b_2)^2 - 2(a_1 + a_2)(a_1 + b_2)}{2(b_1 - a_1)(b_2 - a_2)} + \frac{a_1 + b_2}{(b_1 - a_1)} + c1 \tag{2.31}$$

If $a_2 + b_1 < a_1 + b_2$

$$c_2 = \frac{(a_2 + b_1)^2 - 2(a_1 + a_2)(a_2 + b_1)}{2(b_1 - a_1)(b_2 - a_2)} + \frac{a_2 + b_1}{(b_2 - a_2)} + c1 \tag{2.32}$$

$$c_3 = \frac{-(b_1 + b_2)^2}{2(b_1 - a_1)(b_2 - a_2)} + 1 \tag{2.33}$$

Keep in mind that the equations above, are not applicable for the sum of more that two uniforms densities, since their result does not follow a uniform distribution at all. Actually, the resulting density function has either a triangular or a trapezoid form, depending on the the ranges of the input uniforms.

What we have seen so far can also be used for the computation of the difference of between two uniform random variables. In the beginning of the section we have verified that a difference of the kind $Z = X - Y$ can be considered as sum $Z = X + (-Y)$. The effect of a negative sign in a uniform is well known, since it is a simple linear function such as the ones we have seen in Section 2.2.1.

Alternative Uniform Sum and Difference As we have seen in the introductory chapter, previous works [4, 33] use an interval arithmetic-based approach, in order to obtain the intermediate results between the approximating intervals. This approach is implemented in the current project as well, as an alternative of what we have seen about the sum and difference of uniform distributions. This alternative implementation will be mainly used for evaluation purposes, trying to quantify the improvement, if any, resulting from the adoption of the exact results described before.

Concisely, the sum of two uniform components $Z' = X + Y$ will be approximated by a uniform random variable Z, whose endpoints are determined by interval computations as following:

$$\begin{aligned} a_Z &= a_X + a_Y \\ b_Z &= b_X + b_Y \end{aligned} \tag{2.34}$$

For the difference $Z' = X - Y$, the endpoints of the uniform approximation Z will be:

$$\begin{aligned} a_Z &= a_X - b_Y \\ b_Z &= b_X - a_Y \end{aligned} \tag{2.35}$$

Operations between intervals are described in textbooks such as [30].

2.4 Product and Ratio of Random Variables

The first attempt at devising of a generally applicable form for the product and the ratio of two arbitrary random variables is attributed to [11]. The density $f_Z(z)$ of the product $Z = XY$ of two independent random variables X and Y will be:

$$f_Z(z) = \int_{-\infty}^{\infty} f_X(x) f_Y\left(\frac{z}{x}\right) \frac{1}{|x|} \mathrm{d}x \tag{2.36}$$

According to the same work, the density $f_Z(z)$ of the ratio $Z = X/Y$ of two random variables X and Y will be:

$$f_Z(z) = \int_{-\infty}^{\infty} |y| f_X(zy) f_Y(y) \mathrm{d}y \tag{2.37}$$

Despite the generality of these results, it is evident that no closed-form expression can be derived, even for the simplest of the distributions. The use of the Mellin integral transform was proposed in [16] as a tool for studying the products and the ratios of random variables. Springer and Thompson produced a generalisation of that method for the case of the product of n independent random variables [40]. The same method was also used in [34] in order to produce simpler results for certain families of distributions; this is not feasible however for all kinds of distribution.

2.4.1 Approximation for the Product of Gaussians

A form for the product of two arbitrary independent normally distributed variables is given in [10], however it is not considered as a closed-form expression, since it involves integrals. The difficulties in the computation of integrals gave rise to approximate methods. The normality of the product of two normally distributed variables was first investigated in [3], and later experimentally verified in [23] for a number of cases. More specifically, it was proven that the product of two Gaussian X and Y variables approaches a normal curve when:

$$\rho_X = \frac{\mu_X}{\sigma_X} \to \infty, \quad \rho_Y = \frac{\mu_Y}{\sigma_Y} \to \infty \tag{2.38}$$

In this case, the mean and the variance of the Gaussian variable Z that approximates the product $Z' = XY$ will be:

$$\mu_Z = \mu_X \mu_Y \tag{2.39}$$

$$\sigma_Z^2 = \sigma_X^2 \sigma_Y^2 + \mu_Y^2 \sigma_X^2 + \mu_X^2 \sigma_Y^2 \tag{2.40}$$

Equations (2.39) and (2.40) were adapted from [23], with zero correlation coefficient though, as we are interested in independent variables only.

So, it seems acceptable to approximate products of Gaussian components, instead of using exact analytic solutions. The preconditions under which this approximation is valid are summarised in (2.38), and they will almost always be true if the variances of the input distributions are close to zero. In fact, that is the case for our Gaussian approximation components, since the idea is to approximate the original density with a large number of low variance overlapping Gaussian mixture components.

2.4.2 Approximation for the Ratio of Gaussians

A general form for the ratio of two correlated Gaussian variables was first introduced by D.V. Hinkley in [25]. The formula invented by Hinkley includes a correlation variable ρ, which captures the dependency between the input variables. We present this formula based on zero correlation though, as we are mainly interested in the ratio of independent X and Y. Thus, the density function of the ratio $Z = X/Y$ will be:

$$f_Z(z) = \frac{b(z)c(z)}{a^3(z)} \frac{1}{\sqrt{2\pi}\sigma_X\sigma_Y} \left[2\Phi\left(\frac{b(z)}{a(z)}\right) - 1 \right] + \frac{1}{a^2(z)\pi\sigma_X\sigma_Y} e^{-1/2\left(\frac{\mu_X^2}{\sigma_X^2} + \frac{\mu_X^2}{\sigma_X^2}\right)} \tag{2.41}$$

where

$$\begin{aligned} a(z) &= \sqrt{\frac{1}{\sigma_X^2} z^2 + \frac{1}{\sigma_Y^2}} \\ b(z) &= \frac{\mu_X}{\sigma_X^2} z + \frac{\mu_Y}{\sigma_Y^2} \\ c(z) &= e^{1/2 \frac{b^2(z)}{a^2(z)} - 1/2 \left(\frac{\mu_X^2}{\sigma_X^2} + \frac{\mu_X^2}{\sigma_X^2} \right)} \\ \Phi(z) &= \int_{-\infty}^{z} \frac{1}{\sqrt{2\pi}} e^{-1/2 u^2} \mathrm{d}u \end{aligned} \tag{2.42}$$

It is evident that the the ratio of two normally distributed variables has no simple form at all, since the integral for the $\Phi(z)$ expression in Equation (2.42) cannot be computed analytically. A numerical computation could be an acceptable solution in terms of accuracy, but not in terms of efficiency. After all, any operation between individual components is an action that is going to be repeated very many times. Nevertheless, it could be worth performing some experiments using the exact result of the ratio, in some future work.

Instead, we can use a trick that will enable us computing the ratio of Gaussian mixtures without bothering about the ratios of their components. Actually, we can write the expression $Z = X/Y$ as a product of X and $1/Y$. As we have seen in Section 2.2.2, it is straightforward compute both the density and the distribution function of $1/Y$ for any Y. If this intermediate result is approximated by a mixture of Gaussians, then all it remains is to compute the product of Gaussian mixtures, as outlined in Section 2.4.1.

2.4.3 Approximation for the Product and the Ratio of Uniforms

It appears that there is no closed-form expression for either the product or the ratio of two arbitrary independent uniformly distributed variables in the literature. So, we have resorted to using an approach based on interval arithmetic, which has been previously used in applications similar to the current one [4, 33]. This means that we are approximating the result of the product of two random variables $Z' = XY$ with a uniform distribution Z, whose endpoints a_Z and b_Z are expressed in terms of the endpoints of the input distributions. According to the interval arithmetic as described in relative textbooks such as [30], these will be:

$$\begin{aligned} a_Z &= min(a_X a_Y, a_X b_Y, b_X a_Y, b_X b_Y) \\ b_Z &= max(a_X a_Y, a_X b_Y, b_X a_Y, b_X b_Y) \end{aligned} \tag{2.43}$$

For the ratio $Z' = X/Y$, the endpoints of the uniform approximation Z will be:

$$\begin{aligned} a_Z &= min(a_X/a_Y, a_X/b_Y, b_X/a_Y, b_X/b_Y) \\ b_Z &= max(a_X/a_Y, a_X/b_Y, b_X/a_Y, b_X/b_Y) \end{aligned} \tag{2.44}$$

Experimentation with a better approximation of the actual results could be subject of future work.

2.5 Minimum and Maximum of Random Variables

Proofs for the results of the minimum and the maximum between two independent random variables can be found in several textbooks [41, 46]. The probability density function and the cumulative distribution function of $Z = min(X, Y)$ for X and Y independent will be:

$$f_Z(z) = f_X(z)(1 - F_Y(z)) + f_Y(z)(1 - F_X(z)) \tag{2.45}$$

$$F_Z(z) = F_X(z) + F_Y(z) - F_X(z)F_Y(z) \tag{2.46}$$

The probability density function and the cumulative distribution function of $Z = max(X, Y)$ for X and Y independent will be:

$$f_Z(z) = f_X(z)F_Y(z) + f_Y(z)F_X(z) \tag{2.47}$$

$$F_Z(z) = F_X(z)F_Y(z) \tag{2.48}$$

It is evident that the minimum and the maximum of two random variables is written as a closed-form expression of the density and the distribution functions of the inputs. Actually, these findings are more than convenient, as these functions are always known in this project. More precisely, all the distributions are going to be approximated by mixture models with well known forms for their density and distribution functions. Thus, there is no need to perform these operations between the mixture components individually. Instead, we can store either the minimum or the maximum in an intermediate result, whose density and distribution functions can be found using either (2.45) and (2.45), or (2.47) and (2.47). Eventually, we can re-approximate this intermediate result with a mixture model, so as to be used in future computations. This direct approach saves us from the $O(N^2)$ complexity of performing operations on components.

Chapter 3

The *Stochastic* Library

The *Stochastic* library is the result of the current research. As its name implies, it provides a framework for manipulating stochastic variables. The library is written in standard C++, in order to ensure the portability of the source code. C was probably the most obvious option for maximising efficiency, however, C++ was preferred so as to incorporate the Object-Oriented principles into the project. So, the choice of the programming language is an attempt to combine efficiency, modularity and portability.

The main idea is that the user will have access to a new data type that represents continuous random variables. The user should be able to create instances of the random variable data type and define the probability distribution that these instances follow. The framework should also allow the user to ask questions about the probability and cumulative probability of specific values, and produce samples from the distributions defined. Finally, the random variable type should be used in arithmetic operations so that the results are consistent with the theoretical ones presented in Chapter 2. By consistent, we mean that the resulting random variables have distributions that are good approximations to the real ones. The quality of this approximation will be evaluated in Chapter 4. All this functionality is implemented in the *RandomVariable* class.

Apart from the *RandomVariable* class, the user will have access to a number of "concrete" distribution classes, which represent various probability distributions of the real world. We emphasise "concrete" since all of them are subclasses of abstract data types, as we will see on the next section. The full list of the concrete distributions can be found in Appendix A. Eventually, random variables can be initialised by either using known distributions or fitting distributions to real data. The following code illustrates a simple example of creating *RandomVariable* instances and using them in operations.

```
#include <stochastic.h>

using namespace stochastic;

int main()
{
  RandomVariable a = new Gaussian(0,1);
  RandomVariable b = new EmpiricalDistribution("data");
  RandomVariable c;
  c = a + b;
  return c.getDistribution().nextSample();
}
```

RandomVariable `a` follows a standard normal distribution, while instance `b` has the distribution of the data contained in the file `data`. So, the result of the sum of these variables is stored in *RandomVariable* `c`. Finally, the program returns a sample from the sum distribution that is stored in `c`. In the sections that follow, we will see more details of the implementation.

3.1 Class Structure

As we have seen so far, the main functionality involves assigning distributions to random variables before performing useful tasks with them. In this section, we will see how the *RandomVariable* class is associated with the various distributions and the other modules of the system. These associations are visible in the class diagram depicted in Figure 3.1.

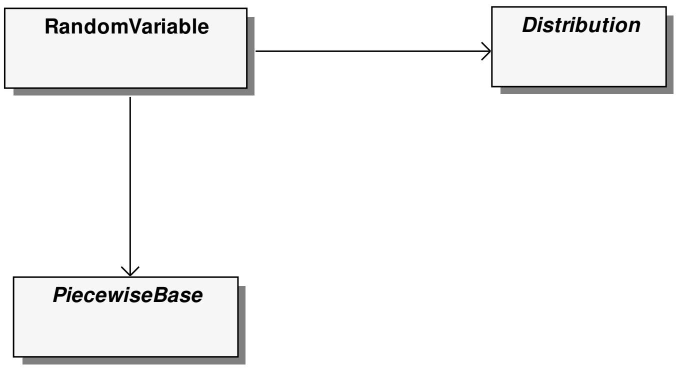

Figure 3.1: Class diagram of *Stochastic* library

The *RandomVariable* class includes methods and overloaded operators that correspond to functions of random variables. The way that distributions are assigned to random variables is captured by an one-to-one association with the *Distribution* class. No instances of *Distribution* can be created since it is abstract — however, it is the base of all the "concrete" distributions.

In other words, it features a number of pure virtual methods that define the interface for any kind of probability distribution. Furthermore, the *RandomVariable* class is also associated with the *PiecewiseBase* class, which is an abstraction of the approximating distributions.

Inheritance Hierarchy for *Distribution* classes All classes derived from the abstract *Distribution* are forced to comply with the specified interface. However, concrete probability distributions are further distinguished into groups. This distinction is defined by the inheritance hierarchy of classes, as depicted in Figure 3.2.

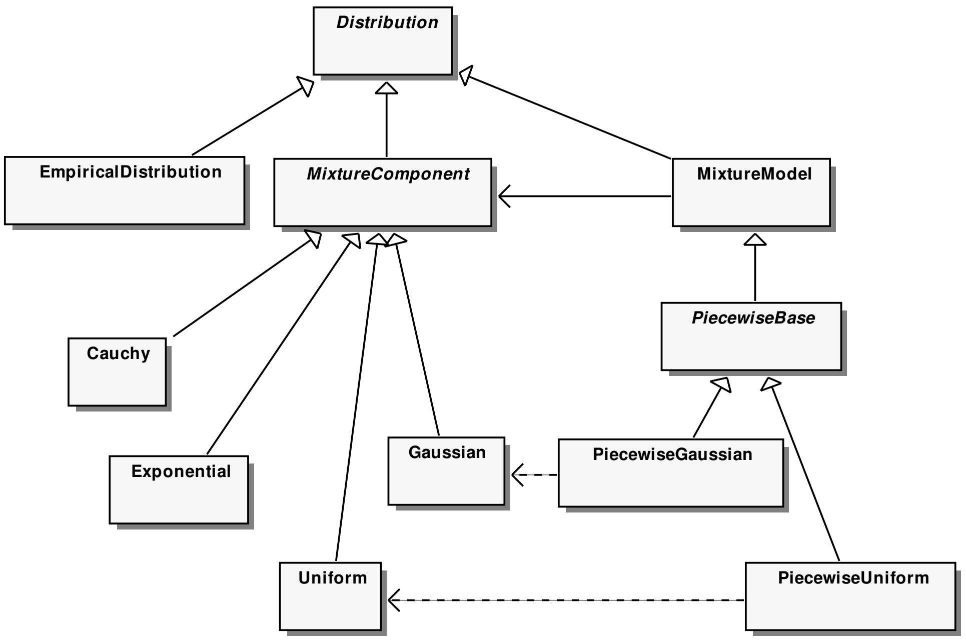

Figure 3.2: The inheritance diagram for *Distribution* classes

One of the principal goals of the architecture design was to support a wide range of probability distributions. The classes that directly derived from the *Distribution* represent the three main types of input distributions:

- **Distributions with known density functions**
  Such distributions can also be components of a mixture model, so they implement the abstract *MixtureComponent* class.
- **Mixture models**
  In this case, the probability density function is expressed as a combination of the probability densities of mixture components.
- **Distributions of real data**
  The *EmpiricalDistribution* class approximates the distribution of a

given dataset using a non-parametric approach. A more detailed discussion can be found in Section 3.3.1.

One could argue that any kind of distribution could be used as a mixture component, instead of only the ones with known density functions. However, only the latter are considered eligible by the author, so as to ensure that mixtures will have tractable forms as well. The knowledge of density functions is essential in order to perform useful operations on random variables. Thus, mixture models are used to estimate the density of non-tractable distributions. For example, an *EmpiricalDistribution* object could capture the exact distribution of a dataset, in some non-parametric form. Even so, such a distribution can be approximated by a mixture model, as discussed in Section 3.4.

The class that represents the special family of mixture models used for approximating other distributions is the abstract *PiecewiseBase*. It is noted that these approximations will also be used to perform arithmetic operations on random variables, which is among the primary goals of this research. In brief, whilst mixture models are allowed to have any number of components that can even be of different type, the classes derived from the *PiecewiseBase* are more restricted. Actually, the number and the type of components affect the approximation procedure, which has an effect on both accuracy and efficiency. So, the choice of these two parameters should be the users'. Approximation with either mixtures of uniforms or mixtures of Gaussian is available. A set of pure virtual methods in the base class imply that the implementations should be able to approximate any input distribution, in terms of the project, and carry out operations between single components. In fact, the implementation of the specific module was a major issue of this research, hence design options are discussed in detail in sections 3.4.1 and 3.4.2.

3.2 Common Distribution Operations

As we have seen, the *Distribution* class is an abstraction for all the kinds of distributions featured in the current project. What really distinguishes one distribution from another are its probability density function and its cumulative distribution function[1]. So, the derived classes are forced to implement the pure virtual methods that correspond to these functions.

Since these two functions are supposed to be known for any distribution (in some cases they are approximated, as we see in Section 3.3), we can use them to perform operations implemented in common for all the distributions.

[1]Actually, only one of them is needed to define a probability distribution, but both are used for convenience

3.2.1 Sampling

Most of the sampling methods demand that we sample from a uniform distribution over the interval $(0, 1)$. Actually, standard C++ does provide a function that generates pseudo-random integers uniformly distributed over $[0, \texttt{RAND\_MAX}]$, which can be easily modified to fit our needs, by simply dividing by $\texttt{RAND\_MAX} + 1$. The pseudo-random property is actually desirable, since any experiment can be repeatable in this way.

It is noted that many authors [6, 35] prefer Markov Chain Monte Carlo methods, especially in the case of multi-dimensional variables. Nevertheless, since we only have one-dimensional continuous random variables, the following methods are both efficient and convenient.

Inverse Transform Sampling

Inverse transform sampling is based upon Theorem 1, adapted from [15], where also a proof of which can be found.

Theorem 1 *If F is a continuous cumulative distribution function on $\Re$ and F^{-1} is its inverse function defined by:*

$$F^{-1}(u) = \inf\{x|F(x) = u, \quad 0 < u < 1\}$$

Then $F^{-1}(u)$ has distribution F, where u is a uniform random variable on $[0, 1]$.

Thus, once the inverse of the cumulative function of a random variable is explictly known, we can sample from it by using an algorithm summarised in the following steps:

```
1. Draw a sample $u$ from $U(0,1)$
2. Compute and return $F^{-1}(u)$
```

The algorithm above is not always applicable though, as an explicit form of the inverse cumulative distribution function might not exist. In fact, this problem is resolved in Section 3.2.2, where an numerical inversion method is presented. The use of various numerical methods is also thouroughly discussed in [15]. Even so, it might be more efficient to use alternative methods for sampling.

Rejection Sampling

Rejection sampling, proposed by [49], allows us to sample from a complex distribution by only using its probability density function $f(x)$. In order to apply this method we need a simpler distribution $g(x)$ that we can easily sample from. This is usually referred as the proposal distribution. The next step is to find a constant k such that $kg(x) \geq f(x)$ for all values of x. In brief, we sample from the proposal distribution and we occasionally reject samples so that the remaining ones are consistent with the original distribution. A formal definition of the algorithm is summarised in the following steps:

```
1. Draw a sample $x$ from the proposal distribution $g(x)$
2. Draw a sample $u$ from $U(0,1)$
3. Accept $x$, if $u < \frac{f(x)}{kg(x)}$
4. Else go to step 1
```

Figure 3.3 depicts an illustration of the rejection sampling method. The proposal distribution, scaled by a suitable k, results in an envelope distribution for the original. Samples are drawn within the area defined by the envelope, while the shaded area corresponds to the samples that are actually accepted.

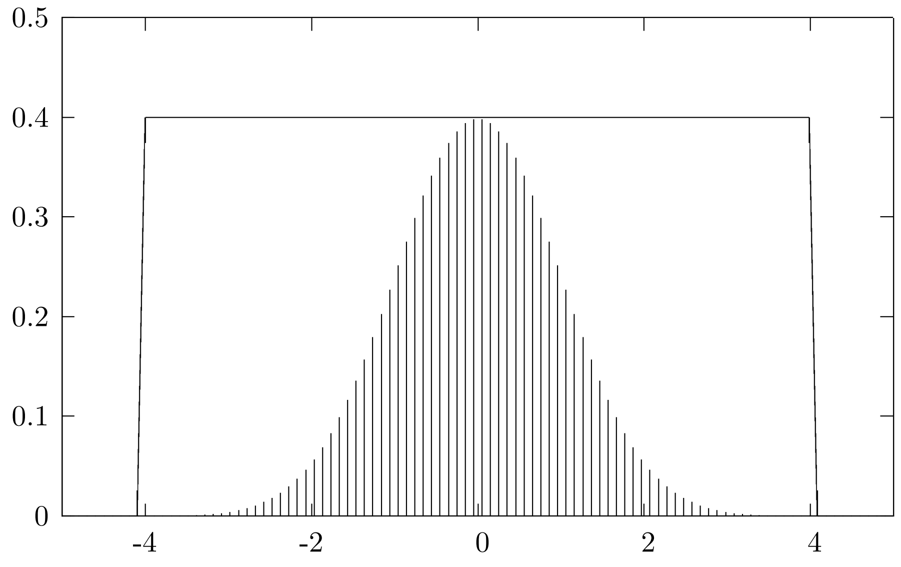

Figure 3.3: Rejection sampling using uniform as proposal distribution

Note that uniform is used as the proposal distribution in all cases, in spite of the fact that it may result in high rejection rates. The reason it is preferred is because it is more straightforward to implement, as a very simple inverse transform method is needed for sampling. In addition, all we need to construct a uniform envelope, is a left and a right margin, and the point of highest probability. One could argue that there are continuous distributions with infinite support, however, we consider it reasonable to truncate any long tails.

3.2.2 The Quantile Function

In this section we discuss the numerical computation of the inverse cumulative distribution function, which is also known as “quantile function”, a term first introduced in [43]. Its importance has been already made clear trying to sample from a distribution, in Section 3.2.1. Since explicit forms only exist for a limited number of distributions, a numerical method has been adopted as a standard implementation in the abstract *Distribution* class. Some distributions, such as the Gaussian or the uniform, overload the original function, taking advantage of more efficient known quantile functions. The classes that make use of a formal definition of the quantile function, instead of the numerical solution, are noted in Appendix A.

The computation of the quantile function, which is denoted by $Q(p)$ with $0 \leq p \leq 1$, can be thought of as the solution of the following equation with respect to x, where $F(x)$ is the cumulative distribution function:

$$F(x) - p = 0 \tag{3.1}$$

A numerical solution involves the use of a root-finding algorithm, such as the bisection method outlined below:

```
1. Let $[a, b]$ interval that contains the solution
2. $x \leftarrow \frac{a+b}{2}$
3. if $(F(a) - p) * (F(x) - p) \geq 0$
       $a \leftarrow x$
4. else
       $b \leftarrow x$
5. if $b - a \geq 2\delta$
       go to 2
6. else
       return $x$
```

The use of any numerical algorithm implies that the true solution is approximated. The quantity δ in the stopping criterion corresponds to the accuracy of the estimated solution.

According to [15], bisection exhibits slow convergence, in comparison with other algorithms such the secant method, or the Newton-Raphson method. Nevertheless, in contrast with other methods, bisection is bound to converge if the initial interval $[a, b]$ does contain the solution. In fact, it is pretty safe to assume that we can always find an initial interval $[a, b]$ such that contains the solution, in the context of probability distributions. The initial interval will be the one that contains the support of the distribution. In the case of distributions with infinite support, we can safely truncate them so as to consider the 99.9% of the support.

3.3 Exceptional Distributions

The majority of the probability distributions implemented have simple parametric forms. The parameters that govern each distribution are stored as private members in the classes that implement them. Therefore, there are methods that make use of these parameters to carry out tasks such as computing the density and distribution functions. For example, the computation of probability density function involves applying the corresponding equation for each distribution. The full list can be found in Appendix A. However, in this section, we will have a closer look at how sampling is performed and density and distribution functions are computed, for distributions with no simple parametric form.

3.3.1 The Empirical Distribution

In order to make the system useful, it is essential that there is a way of initialising the random variables using real data. The distribution of a dataset can be expressed either in a specific functional form or not. The first case is known as the parametric approach, as we assume that the data is drawn from a probability distribution that can be described by a finite set of parameters. For example, assuming that the underlying distribution for a given a dataset is a Gaussian, we can compute the mean and the variance that best explain the data. The computation can be performed by using various methods, including "Maximum Likelihood" and "Bayesian Inference" [6]. Despite the fact that Bayesian inference is the the method of preference of many authors [6, 35], the strong assumptions that it relies on, sometimes result in poor approximation of a real distribution. For example, it is impossible that any single Gaussian approximates well a multimodal distribution.

On the contrary, non-parametric statistics make few assumptions about the form of the underlying distribution. The distinction between the parametric and the non-parametric case was first reported in [52]. Typical examples of non-parametric density estimation are histograms and Parzen window method [42], also known as kernel density estimation. One of the disadvantages of the histograms is that they have discontinuities that may do not reflect the underlying distribution's real nature. The kernel density estimation is the approach chosen by the author, since it produces continuous density estimations, in contrast with histograms. However, a drawback of the method is the fact that it uses the whole dataset. Even so, it has been considered as more important to represent the distribution of input data as accurately as possible. After all, the user has always the choice of discarding a large dataset, by using an approximation — a mixture model, for example.

So, *EmpiricalDistribution* class is the one responsible for representing distributions of input datasets, and is named after the empirical distribution function. The latter is the cumulative distribution function of the data, denoted by $F_N(x)$, where N is the total number of instances in the dataset. According to [28], it is equal to the proportion of sample values that are less or equal to the value x.

$$F_N(x) = \frac{\text{number of samples} \leq x}{N} \tag{3.2}$$

Equation (3.2) is actually used to estimate the cumulative distribution at any datapoint. In order to estimate the probability density function, we make use of the Parzen window method. Given $x_1, x_2, \ldots, x_N$ independent and identically distributed datapoints, the probability density function at a point x is approximated by:

$$f_h(x) = \frac{1}{Nh} \sum_{i=1}^{N} K\left(\frac{x - x_i}{h}\right) \tag{3.3}$$

where kernel K was chosen to be the standard Gaussian for convenience. After all a continuous kernel is required to ensure that the estimated density will have no artificial discontinuities.

$$K\left(\frac{x - x_i}{h}\right) = \frac{1}{\sqrt{2\pi}} e^{-\frac{(x-x_i)^2}{2h^2}} \tag{3.4}$$

Parameter h, which is called bandwidth, plays the role of a smoothing parameter. As, shown in Figure 3.4, too large a bandwidth can result in models that are too smooth to capture multimodal distributions. On the other hand, too small a bandwidth can result in an over-sensitive model with a lot of structure that does not exist.

As reported in [50], the choice of banwidth is more crucial than the choice of the kernel. In order to determine the bandwidth, we have adopted the following rule, as defined in [18]:

$$h = 2\frac{Q(0.75) - Q(0.25)}{\sqrt[3]{N}} \tag{3.5}$$

where $Q(x)$ is the quantile function. An example kernel density estimation using (3.5) is illustrated in Figure 3.5.

3.3.2 Mixture Models

The term mixture model refers to random variables that have mixture densities. More specifically, the probability density function is expressed as

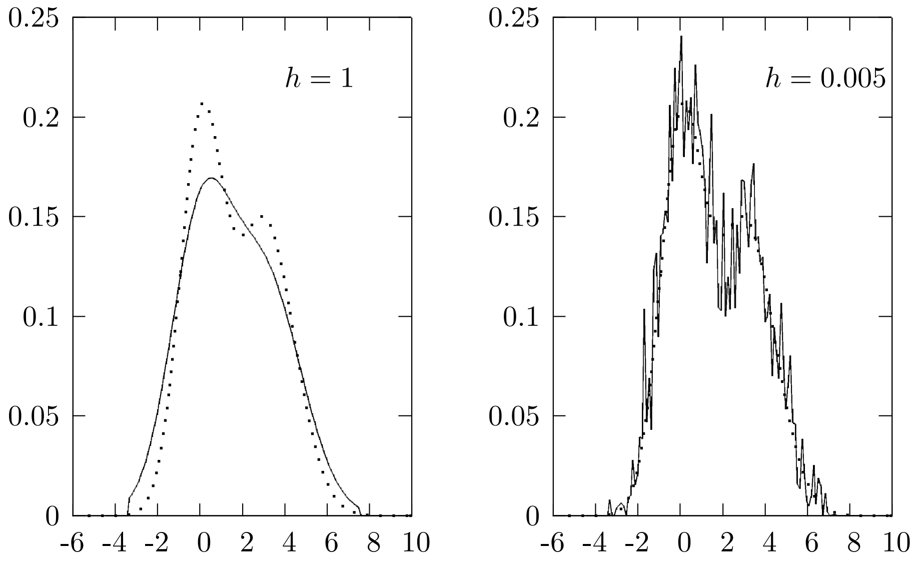

Figure 3.4: Kernel density estimation with standard Gaussian kernel using varying bandwidth h, where the dotted line is the true density

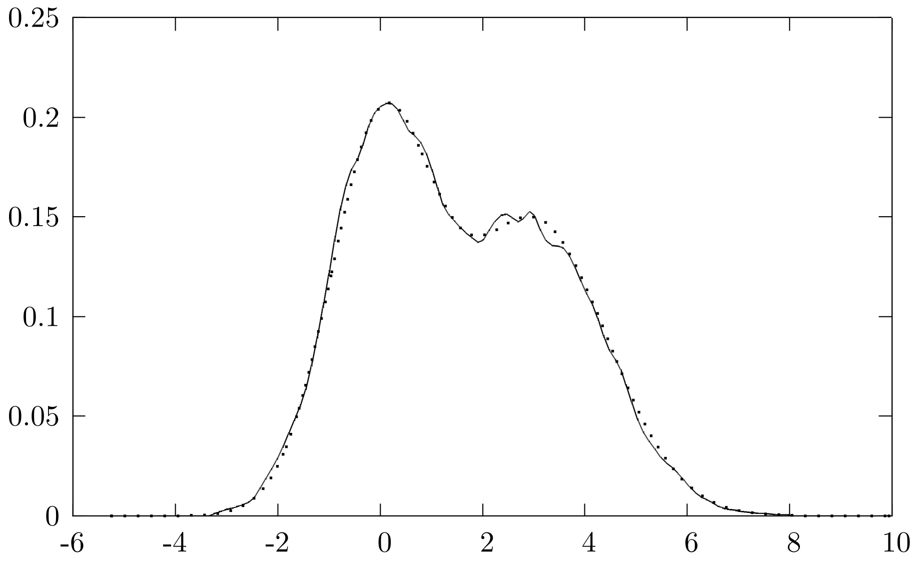

Figure 3.5: Kernel density estimation with standard Gaussian kernel using optimum bandwidth h, where the dotted line is the true density

a linear combination of other probability density functions, where all the coefficients are non-negative and sum up to 1. The generic form for N mixture components with probability density functions $f_1(x), f_2(x), \ldots, f_N(x)$ is summarised in the following equation:

$$f(x) = \sum_{i=1}^{N} w_i f_i(x) \tag{3.6}$$

The coefficients w_k are usually referred as mixture coefficients, or simply weights, and they are subject to the restrictions below:

$$\sum_{i=1}^{N} w_i = 1, \quad w_i \geq 0 \tag{3.7}$$

As we have seen in Section 3.1, only distributions with known parametric form are allowed to be mixture components in terms of this project. In this case, it is very straightforward to compute the density at any given point, since the component densities are well known. The value of the mixture's probability density function at a point x will be the weighted sum of the density values of all the components at that point.

The same approach is also applicable for the estimation of the cumulative distribution function $F(x)$. We can easily show that the cumulative distribution of a mixture model is equal to the weighted sum of the cumulative distribution functions of its components, which are well known. First of all, we know that the cumulative distribution function is defined as

$$F(x) = \int_{-\infty}^{x} f(u)\mathrm{d}u \tag{3.8}$$

If we plug Equation (3.6) into (3.8), then we get the cumulative distribution of a mixture model:

$$F(x) = \int_{-\infty}^{x} \sum_{i=1}^{N} w_i f_i(u)\mathrm{d}u \tag{3.9}$$

However, the sum can be pulled out of the integral, since the sum of integrals is equal to the integral of sums. Moreover, we can do the same with the weights w_k, since they are constant with respect to the integral.

$$F(x) = \sum_{i=1}^{N} w_i \int_{-\infty}^{x} f_i(u)\mathrm{d}u \tag{3.10}$$

Eventually, we see that the integral in Equation (3.10) is consistent with the definition in (3.8). Hence, the cumulative distribution function of a mixture model is the weighted sum of the cumulative distributions of its components:

$$F(x) = \sum_{i=1}^{N} w_i F_i(x) \tag{3.11}$$

In order to perform sampling, we could either use rejection sampling, or the inverse transform sampling method reported in Section 3.2.1, since the former uses the probability density function, and the latter the cumulative distribution function, which are both known. However, none of them is efficient enough to produce large numbers of samples, since we would have to go through all the components each time. Instead, we can take advantage of an alternative view of the mixture models. In [6], they are interpreted as models featuring discrete latent variables. The latent variable in this case corresponds to the component that is responsible for a given datapoint. The distribution of this discrete hidden variable is multinomial with parameters equal to the mixture coefficients. Hence, in order to draw a sample from a mixture model, we carry out the following two steps. First, we choose a mixture component by sampling from the multinomial distribution defined by the weights. Then we can draw a sample from the component chosen, using any of the methods discussed before.

3.4 Approximation with Mixture Models

It is already reported that the *PiecewiseBase* class is an abstraction of the special case of mixture models that are going to be used as approximations of other distributions. Approximations with mixtures of uniforms and with mixtures of Gaussians are implemented as different subclasses of this abstract class, namely *PiecewiseUniform* and *PiecewiseGaussian*. The results of the mixture components used in each case are compliant with the ones discussed in Chapter 2. In this section, we will have a closer insight into the approximation algorithms used.

3.4.1 Piecewise Uniform Approximation

The approximation using mixtures of uniforms is a process that accepts as input a *Distribution* object, and produces a mixture model with uniformly distributed non-overlapping components. Actually, it resembles the discretisation of the probability density function, as described in [4, 33]. In this work however, each interval is assigned a uniform distribution, whose total probability mass is 1. A mixture coefficient is assigned to each component, so as to scale it down. Eventually, each componet will have probability mass equal to the one of the original distribution at that interval. Moreover, the fact that the components do not overlap, allow quick approximation of each area of the support of the original distribution, without worrying about the effects of neighbouring components. A pseudocode for the approximation algorithm can be seen below, where $F(x)$ is the distribution function of the original distribution, N is the number of the components.

```
1. For each $[a_i, a_i+step]$ of the $N$ non-overlapping intervals of
```

```
   the support
2.      $weight \leftarrow F(a_i + step) - F(a_i)$
3.      add $U(a_i, a_i + step)$ and $weight$ to the PiecewiseUniform
   result
4. return PiecewiseUniform result
```

A more sophisticated algorithm could involve variable step size, or variable number of components. However, the current implementation is preferred, since it has linear complexity. In fact, efficiency is a priority at this point, as the computation of the quantity $F(a_i)$ can prove rather expensive. More specifically, we have seen that the result of a binary operation will have N^2 components. This raw result should be re-approximated by a mixture of N uniforms, in order to be used in future computations. We have seen in Equation (3.11) that the distribution function of a mixture model is a combination of the distribution functions of its components. Given that the components are N^2 in this case, it certainly makes sense to prefer efficient approximation algorithms. Figure 3.6 depicts a simple example of approximating the probability density function of a standard Gaussian distribution.

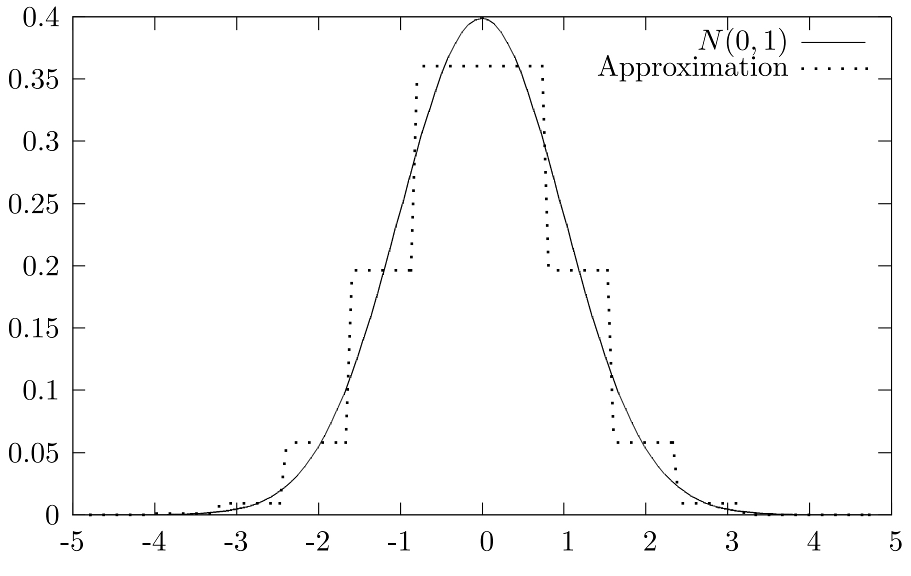

Figure 3.6: Approximation of $N(0, 1)$ using a mixture of 10 uniforms

3.4.2 Piecewise Gaussian Approximation

According to [2], a Gaussian mixture density can be used to approximate another density function with arbitrary accuracy. The approximation using mixtures of Gaussians is more complicated though, since the components should overlap up to a certain extent, in order to successfully approximate smooth density functions. Actually, there are several machine learning methods for the task, the most popular of which is the Expectation-Maximisation (EM) algorithm [14]. The problem with most of the existing algorithms though, is that they are data oriented. In other words, they fit Gaussian mixture models to data, but what we need is to fit a mixture model to a given density function. A very straightforward solution, and perhaps the most correct one, would be to produce a large dataset by sampling from the target distribution. Then we can apply an algorithm such as the EM to construct a mixture distribution that best explains the data produced. Nevertheless, given the fact that we want to perform computations in real time, this is not an applicable solution at all.

An example of fitting mixtures of Gaussians directly to density functions can be found in [19]. However, this solution is not efficient either, as it invloves searching over the $3N$ parameters of the Gaussian mixture (N means, N variances and N weights[2]). More specifically, a hill-climbing method is applied, where each combination of parameters is tested by measuring the distance from the original density, which is an expensive operation itself. In any case, such an approach is not suitable for real time computations.

Instead, an algorithm based on the empirical rule was used so as to produce approximations in linear time. According to the empirical rule for the normal distribution, approximately the 68.2% of its probability mass lies within 1 standard deviation of its mean, the 95.4% lies within two standard deviations, and the 99.6% within three standard deviations. An illustration of this rule can be found in Figure 3.7. The idea is that we choose means and variances for the mixture components, such that the effects of component overlapping are easily predictable.

In brief, N Gaussian components are distributed across the support of the target density. The distances between the means of neighbouring components should be the same; from now on, we will refer to this quantity as "step". If we set the standard deviation σ equal to the step for all of the components, then the probability mass assigned at each interval $[a, b]$ defined within one step will be:

- $0.341 + 0.341$ from the components $N(a, \sigma^2)$ and $N(b, \sigma^2)$
- $0.136 + 0.136$ from the components $N(a - \sigma, \sigma^2)$ and $N(b + \sigma, \sigma^2)$

[2]Actually, the weights are $N - 1$, since the N-th weight is determined by the previous ones, as they all sum up to 1

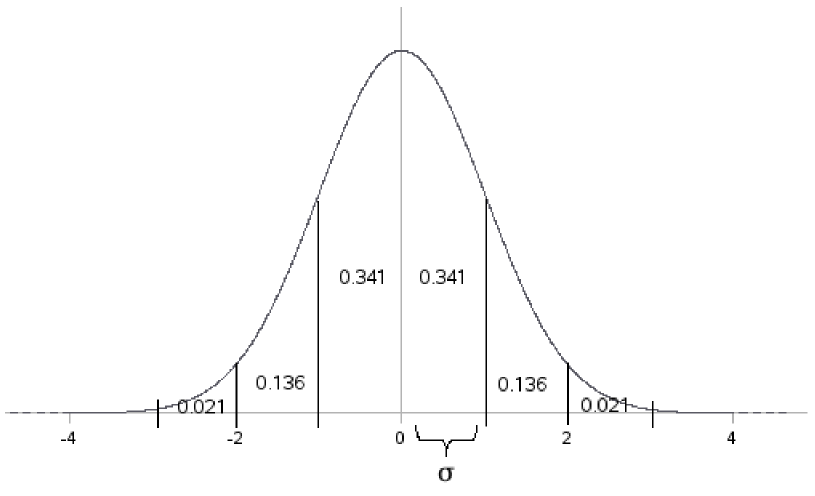

Figure 3.7: Illustration of the empirical rule for the standard Gaussian distribution

- $0.021 + 0.021$ from the components $N(a - 2\sigma, \sigma^2)$ and $N(b + 2\sigma, \sigma^2)$

So, the unnormalised probability mass assigned at each interval is approximately 1, as happened in the piecewise uniform case. Now, by choosing the appropriate mixture coefficients, we can be sure that the resulting mixture model is a valid probability distribution, in other words the areas of its component densities sum up to 1. Of course, this probability is smaller for the marginal intervals, a fact that may cause a loss of information. The components and the corresponding weights are constructed in linear time by the following algorithm:

```
1. Set $N$ centres, each one at every $step$ throughout the
   support of the target distribution
2. For each centre $\mu_i$
3.       $weight \leftarrow F(\mu_i + step) - F(\mu_i - step)$
4.       add $N(\mu_i, step^2)$ and $weight$ to the PiecewiseGaussian
   result
5. re-normalise the weights
6. return PiecewiseGaussian result
```

Again, more sophisticated but less efficient algorithms could have been applied to learn the weights. However, the fact that the distribution function can be rather computationally expensive, as discussed in the previous section, imposed the use of an efficient solution. Figure 3.8 is just an illustration of an approximation result. The accuracy is being improved, as we increase the number of components, which is also verified in the evaluation section.

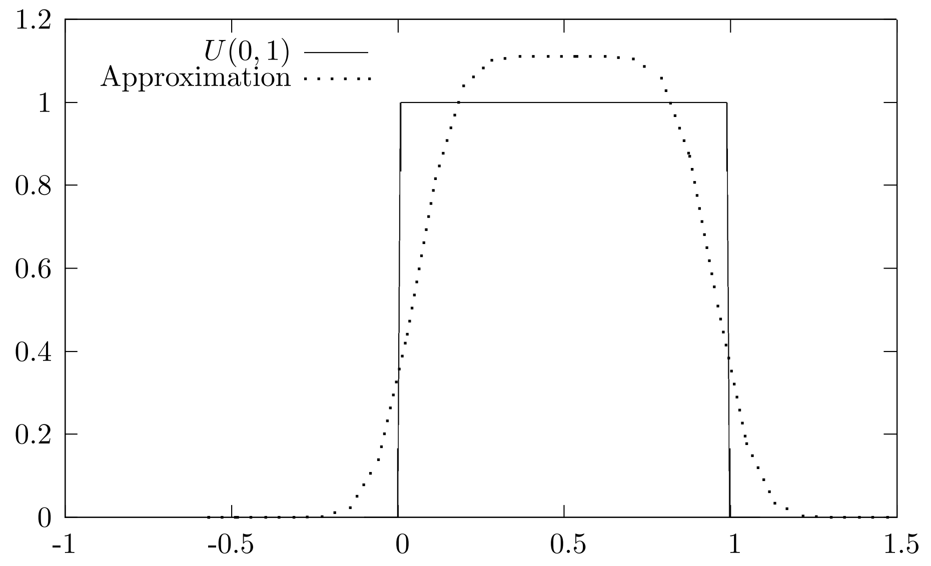

Figure 3.8: Approximation of $U(0,1)$ using a mixture of 10 Gaussians

3.5 Computations Using Mixture Models

Computations on random variables are implemented as overloaded C++ operators in the *RandomVariable* class. An operation on two random variables involves the following steps:

i. Approximate the distributions of the input variables with mixtures of N either uniform or Gaussian components.

ii. Apply the relevant computation at each pair of components. Each one of these intermediate results will be weighted by the product of the weights of the components that is derived from.

iii. The resulting mixture density of N^2 components will be approximated as well, so as to be reduced to N mixture components.

It is straightforward to prove the validity of the second step for any kind of binary operator. For example, the resulting density of the sum $X+Y$ will be the convolution of the input densities, as defined in Equation (2.23). Since X and Y have mixture densities, we can plug (3.6) into (2.23) and then we obtain:

$$(f_X * f_Y)(z) = \int_{-\infty}^{\infty} \sum_{i=1}^{N} w_{X_i} f_{X_i}(z-y) \sum_{j=1}^{N} w_{Y_j} f_{Y_j}(y)\mathrm{d}y \tag{3.12}$$

And if we pull the sums and the weights out of the integral, we obtain:

$$(f_X * f_Y)(z) = \sum_{i=1}^{N} \sum_{j=1}^{N} w_{X_i} w_{Y_j} \int_{-\infty}^{\infty} f_{X_i}(z-y) f_{Y_j}(y)\mathrm{d}y \tag{3.13}$$

which is completely consistent with the description given in Step (ii) above, as the integrals in (3.13) are the densities of the sums of pairs of components. Since we are able to compute these quantities exactly, we can be sure that we have no information loss at this step. The proofs for the remaining binary operations are similar. The intermediate results in each case of operation are computed as described in the relevant sections of Chapter 2.

3.6 Dependency Tracking Monte Carlo

In this section we discuss the implementation of the dependency tracking mechanism, and the way it is used in a Monte Carlo algorithm. It is noted that the incorporation of this feature into the mixture model approximation approach is a subject of future work.

The dependency information is recorded in the form of a directed graph of random variables. The edges in this graph are interpreted as parent-to-child relationships, where a parent is one of the inputs of a binary operation, while a child is the resulting random variable. To put it differently, the arrows indicate the flow of computations. For convenience, the unary operations are considered as binary ones, whose second argument follows a delta distribution, a distribution whose density is described by the Dirac's delta function $\delta(x)$. In the same way, n–ary operations are treated as a successions of binary ones as well. For example, the expression: `c = a + b + 2`, will have the dependency graph depicted in Figure 3.9. It is noted that the nodes that hold constants and intermediate results, are handled as random variables in terms of the graph only. They do not correspond to any *RandomVariable* object visible to the user.

A case that should also be examined is re-assignment of a distribution to a random variable, using an expression such as `a = a + b` or even `a = b`. While the user should not have access to the old distribution of the object `a` anymore, its node in any existing dependency graph should remain unchanged, as there are other variables that depend on it. Instead, a new node should be created that will point to the new value from now on. This policy also ensures that if we change the distribution of a random variable, this will affect future computations only, not the past ones. In Figure 3.10, the

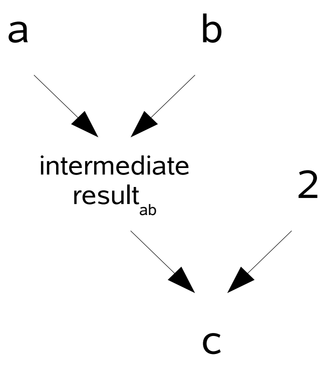

Figure 3.9: Dependency graph for `c = a + b + 2`

object `a_old` is hidden from the user, but it still determines the distribution of `c`.

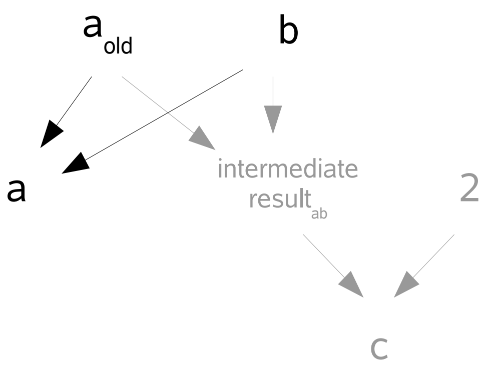

Figure 3.10: Dependency graph for `a = a + b`

Thus, any node of the dependency graph essentially falls into one of the following categories:

Terminal nodes These are the ones with no parents, in other words, they are the initial *RandomVariable* whose distributions have been directly defined by the user. We assume that there is no dependency between these initial variables. Moreover, scalar variables correspond to terminal nodes also. The only difference is that a scalar will not be a parent more that once. For example, if we use `double x` twice in our computations, then two independent terminal nodes will be created.

Non-terminal nodes These are *RandomVariable* instances whose distributions came as results of computations on other random variables. They can be either visible to the user, such as `c` was in the previous example, or intermediate results.

What we have managed so far, is to express the dependencies as directed paths in the graph. Each non-terminal node corresponds to a number of computational paths, let us refer to them as "personal computational graph". Hence, a random variable will be dependent on all the elements of its personal computational graph. Moreover, two non-terminal nodes that are not in the same path, will be also dependent if they have common elements in their personal computational graphs. This structure is essentially a directed acyclic graph [48]. Moreover, all the non-terminal nodes have exactly two edges pointing at them, since they are results of a specific binary operation. Thus, it is very straightforward to find the ancestors of a non-terminal node using a simple depth-first-search algorithm, and consequently describe the dependency between any pair of random variables.

The Monte Carlo Implementation The information captured in the graph can be used in order to develop a Monte Carlo algorithm that respects the dependencies that have been created by the computations. According to the naïve Monte Carlo approach, a sample of the resulting distribution would be drawn by just sampling from the distributions of the input variables, and then apply the operator at this pair of samples. Eventually, the resulting distribution is described by the empirical distribution function of the dataset produced. In this implementation, the results are actually instances of the *EmpiricalDistribution* class discussed in Section 3.3.1.

Now, considering the example `b = a + a`, it is intuitive that this expression should be equivalent to `b = 2 * a`, where `a` and `b` are *RandomVariable* objects. In terms of the Monte Carlo approach, this implies that if the object `a` is already sampled, then it should not be sampled again. Instead, we should use the same sample at each sampling step, so as to simulate the operation `2 * a`.

Thus, the dependency tracking Monte Carlo involves a sampling process that has access to the personal computational graph of each random variable. More accurately, each sampling step results in a recursive traversal of the personal computational graph. Coonsisely, if a random variable has been already sampled at a specific step, the same sample will be used in order to produce the current sample of the resulting distribution. The algorithm is presented in more detail in the following pseudocode.

```
1. Compute the binary operation between
   the random variables $X$ and $Y$
2. for $N$ sampling steps
3.     $sample \leftarrow$ recursive_sampling($operator$, $X$, $Y$)
4.     add $sample$ to the EmpiricalDistribution result
5. return EmpiricalDistribution result
```

The function `recursive_sampling` returns one sample for the resulting random variable of the expression $operator(X, Y)$.

```
1. recursive_sampling($operator$, $X$, $Y$)
2. if $X$ is sampled
3.     $sample_X \leftarrow$ retrieve the sample of $X$
4. else if $X$ is a terminal node
5.     $sample_X \leftarrow$ sample($X$)
6. else if $X$ is a non-terminal node
7.     $X_1, X_2 \leftarrow$ parrents of $X$
8.     $sample_X \leftarrow$ recursive_sampling($operation_X$, $X_1$, $X_2$)
9. if $Y$ is sampled
10.     $sample_Y \leftarrow$ retrieve the sample of $Y$
11. else if $Y$ is a terminal node
12.     $sample_Y \leftarrow$ sample($Y$)
13. else if $Y$ is a non-terminal node
14.     $Y_1, Y_2 \leftarrow$ parrents of $Y$
15.     $sample_Y \leftarrow$ recursive_sampling($operation_Y$, $Y_1$, $Y_2$)
16. return $operation(sample_X, sample_Y)$
```

3.7 General Remarks on the Implementation

The implementation so far includes 20 classes in 3500 source lines of code approximately. The classes that the users should be aware of, have been already outlined in Section 3.1. Consisely, the users are supposed to initialise objects of the *RandomVariable* class, using instances of distribution classes. Once a *RandomVariable* object is assigned a distribution, it can be either queried on its distribution or used in computations. However, there is also a number of distribution classes that are not accessible by the user. Those are used to hold intermediate results during a computation. Such an example is the sum of two uniforms, as defined in Section 2.3.2.

Informatics Research Proposal Timeline Eventually, it is worth reviewing the project's timeline, as it was defined in terms of the research proposal. In Table 3.1, we can see the tasks that the project was divided into, and their associated starting and completion dates. Although the project was on schedule, we have to note that the implementation sub-tasks as initially recorded do not reflect the reality. Originally, the implementation sub-tasks were supposed to correspond to specific C modules. However, it has been decided that the implementation language would change from C to C++, so as to adopt the principles of object-oriented programming. This decision not only improved the code resuability and maintainability, but also made the use of the library more intuitive. This change of plans had a major effect in the architecture of the system though, as many of the modules were spread across a number of classes.

Task	Start	Finish
Researching	18 May	18 May
Specification	25 May	30 May
Implementation	1 Jun	8 Aug
Monte Carlo simulation	1 Jun	13 Jun
Skeleton component	1 Jun	13 Jun
Approximating components	15 Jun	4 Jul
Operators	6 Jul	28 Jul
Dependency tracking	20 Jul	8 Aug
Testing	8 Jun	8 Aug
Perform evaluation experiments	27 Jul	15 Aug
Writing	22 Jun	20 Aug

Table 3.1: Project tasks, starting and completion dates

Chapter 4

Evaluation

The evaluation of our approach mainly involves comparisons between probability distributions. In fact, the resulting distributions given by our implementation for each kind of operator are compared with the theoretical ones, when these are available. In case the theoretical results cannot be analytically computed, we use a Monte Carlo approach featuring 1 million samples as reference. After all, is is reasonable to expect that the more we increase the number of samples, the more accurate the Monte Carlo results will be. In order to completely define the notion of comparison between probability distributions, we introduce a number of similarity measures in Section 4.1.

In this project, we have actually implemented two different approaches that deal with the problem of performing arithmetic operations on random variables, including approximation with mixtures of models and dependency tracking Monte Carlo. As we have seen, the first method involves two alternatives as components for the approximating mixture models. So, the purpose of the evaluation of this method is not only to have a picture of the quality of the computations for each one of them independently, but also to make comparisons between them. Both accuracy and efficiency are issues for the mixture model approximation, so both the distances between the distributions and the running times will be displayed. Of course, the running times are just indicative. It is noted though that the experiments have been conducted in an Intel® Core$^{\text{TM}}$2 Duo T8300 @ 2.40GHz PC running Linux.

On the other hand, efficiency is not of great concern for dependency tracking Monte Carlo. The evaluation of this method should focus on its ability to capture the dependencies that arise through out the computations. Thus the method will be compared against series of computations that can be easily tracked to verify the correct result. The purpose will be to verify that the results given are consistent with the dependencies involved.

4.1 Similarity Measures

Most of the popular measures of similarity between probability distributions are expressed as distances or divergences between them. The term dissimilarity might fit better, since the smaller the distance the greater the similarity will be. While there are numerous measures in the literature, some examples can be found in [13, 45], KL divergence is perhaps the one most widely used in statistics. However, we will see that in some cases its computation may introduce serious error. So, we have also used Kolmogorov distance and CDF distance as alternatives.

4.1.1 Kullback-Leibler Divergence

The Kullback-Leibler divergence [31] (or simply KL divergence) between two probability distributions with density functions $p(x)$ and $q(x)$ is defined as:

$$KL(p||q) = -\int_{-\infty}^{\infty} p(x) \ln \frac{q(x)}{p(x)} \mathrm{d}x \tag{4.1}$$

If we think of $p(x)$ as the original distribution and qx as the approximating one, then the equation above is interpreted as the additional amount of information required to specify the value at the point x, as a result of using the approximating function $q(x)$ instead of the original $p(x)$. Hence, by definition the KL divergence is ideal for comparing the two alternative approximations, piecewise uniform and piecewise Gaussian. The comparison between them essentially involves comparison of their KL divergences of the true distributions given in the analytical way. Moreover, the KL divergence has the following properties:

- $KL(p||q) \geq 0$
- $KL(p||q) = 0, \quad \text{in and only if} \quad p = q$
- $KL(p||q) \neq KL(q||p)$

It is noted that KL divergence is not a distance metric, since it is not symmetric as the third property implies. However it is clear from the properties above that large similarity results in a value close to zero, as would happen for any distance metric.

The computation of the quantity in Equation (4.1) is also an issue, since there is no closed form expression for arbitrary probability distributions. Various methods of approximating the KL divergence can be found in [24], no one however outperformed the Monte Carlo with 100K samples. Since efficiency is not of concern in this evaluation task, we have used a Monte Carlo algorithm with $x_1, x_2, \ldots, x_N$ samples, that can be summarised in the following formula [24]:

$$KL_{MC}(p||q) = \frac{1}{N}\sum_{i=1}^{N} \ln \frac{p(x_i)}{q(x_i)} \rightarrow KL(p||q) \tag{4.2}$$

One important note is that KL divergence makes use of the probability density functions of the distributions to be compared. However, the exact densities are not known for the results of Monte Carlo method, as they are given in form of sets of samples. Of course, in Section 3.3.1 we have discussed about ways of approximating probability densities of datasets, and we have adopted a non-parametric approach based on kernel density estimation. Even so, what we actually have is an estimation of the actual density function, hence the accuracy of the KL divergence computed essentially depends on the quality of the density approximation. On the other hand, the distribution function of a set of samples is exactly described by the empirical distribution function, as we have seen in Section 3.3.1. So, in order to quantify the accuracy of Monte Carlo generated results will we use measures based on the cumulative distribution function.

4.1.2 Kolmogorov Distance

This distance has been originally used in terms of the Kolmogorov-Smirnov test, in order to quantify the difference of a dataset's distribution from a reference distribution [32]. If $P(x)$ and $Q(x)$ are cumulative distribution functions, then the Kolmogorov distance between them will be the supremum of their absolute difference:

$$D_K(P||Q) = \sup_x |P(x) - Q(x)| \tag{4.3}$$

This quantity can be easily computed using a Monte Carlo method. In brief, it can be approximated by finding the maximum of N independent identically distributed samples. Again, a large number of samples is required for a good approximation, but efficiency is not an issue.

4.1.3 CDF Distance

For reasons of convenience, we have also used a rather arbitrary metric, noted as CDF distance. As its name implies, it is actually the Manhattan distance of the cumulative distribution functions.

$$D_{CDF}(P||Q) = \int_{-\infty}^{\infty} |P(x) - Q(x)| \mathrm{d}x \tag{4.4}$$

Its concept is pretty similar to the Kolmogorov distance, however, the difference of the two distribution function is integrated over $(-\infty, \infty)$. Thus, we expect less penalty when two distributions differ only on a small interval of the support. An example of use of the CDF distance in the literature can be found in [19]. In order to compute the integral in (4.4), it is very straightforward to use a numerical method such as the trapezium rule [7].

4.2 Experiments on Mixture Model Approximation

In the experiments that follow, we investigate the performance of our mixture model approximations, namely piecewise uniform and piecewise Gaussian. Typically, our methods are compared against a Monte Carlo simulation that has approximately the same running time. We have also implemented an approach based on interval arithmetic, as adopted in various works [29, 4, 33]. So, for the cases of the sum, the difference, the product and the ratio of random variables, we also present the performance this alternative approach, identified as "discretisation" approach in the rest of this chapter.

4.2.1 Accuracy and Efficiency Issues

The purpose of this experiment is to identify the optimum number of approximating components. On the one hand, it certainly makes sense to use as many mixture components as possible, however it is noted that the computational complexity of a binary operation is $O(N^2)$, where N is the number of components. So, there is actually a tradeoff between accuracy and efficiency, which the user should be aware of.

Thus, we have approximated a standard Gaussian distribution $N(0, 1)$ with number of components varying from 10 to 300, using both mixtures of uniforms and mixtures of Gaussians. In order to quantify the accuracy at each step, we have computed the KL divergence $KL(p||q)$, where $p(x)$ is the density of the original distribution and $q(x)$ the approximation density. The efficiency of the approximation itself is not important, since it is only $O(N)$. Instead, we have measured the running times of the a binary operation at each step. The sum was selected as representative of the binary operations, and the running times presented are just indicative. It is noted that the accuracy and the efficiency of the sum will be examined in more detail in Section 4.2.3. The values for KL divergence and running times for different values of N are presented in Table 4.1. The evolution of accuracy and efficiency are also displayed in a logarithmic scale, in Figures 4.1 and 4.2.

We will not comment on the performance of the alternative approximations, piecewise uniform and piecewise Gaussian, for the moment. It is notable that the piecewise Gaussian approximation seems much more accurate though. In any case, the important point is that for both of them, the divergences given in Table 4.1 verify the expectation that a large number

Number of Components	Piecewise Uniform		Piecewise Gaussian	
	KL Divergence	Time (ms)	KL Divergence	Time (ms)
10	0.02509870	0.2	0.05845930	0.2
30	0.00292771	2.9	0.00119073	2.6
60	0.00069225	20	0.00004625	15
80	0.00030220	50	0.00002190	30
100	0.00004645	100	0.00001575	60
120	0.00001380	200	0.00000862	110
150	0.00001045	600	0.00000789	220
180	0.00001022	1610	0.00000643	440
200	0.00000894	2510	0.00000552	800
220	0.00000514	3420	0.00000316	1300
250	0.00000483	4980	0.00000172	2280
300	0.00000344	8900	0.00000059	4050

Table 4.1: KL divergence and running times for varying number of components

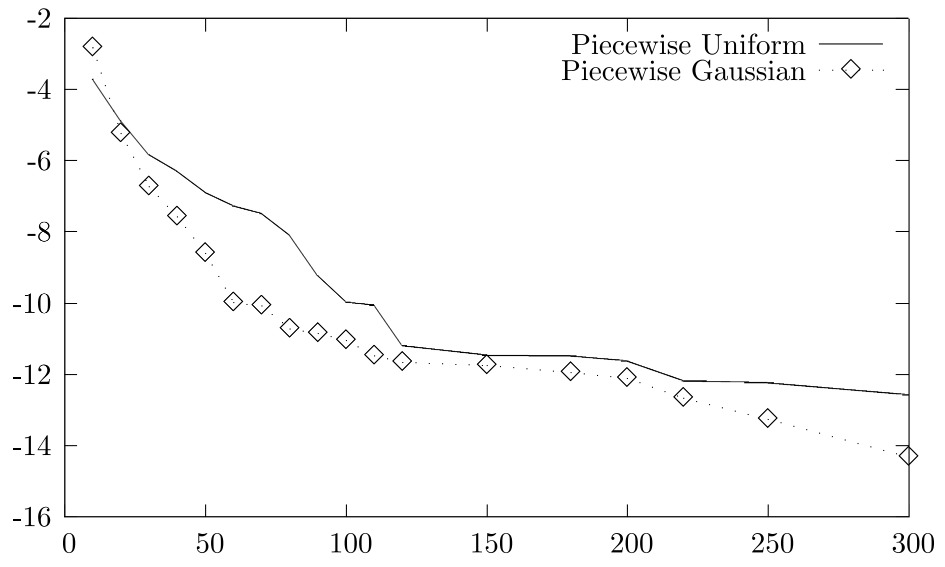

Figure 4.1: KL divergence in a logarithmic scale, for different number of components

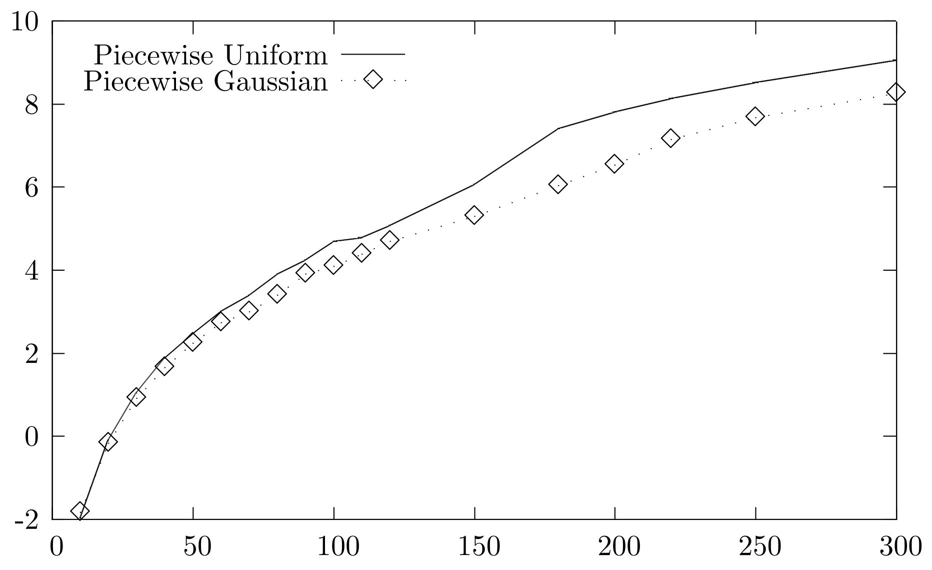

Figure 4.2: Running times (ms) in a logarithmic scale, for different number of components

of componets results in better accuracy. It is also noted that the Monte Carlo method used to approximate the KL divergence (Section 4.1.1), was quite unstable for more than 200 components. In fact, these values were so close to zero, that we had to use more than 1 million samples to obtain a good approximation at each case. On the other hand, we can see that the running times seem to be increased in a quadratic manner. This is actually consistent with the $O(N^2)$ complexity of the algorithm used.

Eventually, the accuracy granted for 100 components seems acceptable, while the running time usually remains below 100 milliseconds, which means that it is feasible to be used for a series of computations. For the rest of this chapter, the number of components is set to 100, unless stated otherwise.

4.2.2 Performance for Unary Operations

The unary operations are actually functions of one random variable, as defined in Section 2.2. They are certainly less interesting than the binary ones, since their complexity is only $O(N)$, however we still need to validate the assumptions we have made so far. It is also noted that from this point and on, we will make use of all the measures of performance at our disposal, including Kolmogorov and CDF distance and running times. The KL divergence will be computed for non-Monte Carlo derived distributions only.

Method	Similarity to True Distribution			Time (ms)
	KL Divergence	Kolmogorov Distance	CDF Distance	
Monte Carlo (2K)	N/A	0.0126734	0.0133106	120
Piecewise Uniform	0.0002347	0.0001985	0.0002928	1.4
Piecewise Gaussian	0.0001788	0.0028174	0.0065224	1.4

Table 4.2: Performance for $-1/2X + 4$, where $X \sim N(-4, 2)$

Linear Function of a Random Variable

The current example involves estimating the expression $Y = aX + b$, where $a = -1/2$, $b = 4$ and $X \sim N(-4, 2)$. According to what we have seen in Section 2.2.1, we know that the true resulting distribution will be of the form $N(a\mu_X + b, \sigma^2 a^2)$. In this case, the distribution of the result will be $N(6, 0.5)$.

In Table 4.2 we can see that both piecewise uniform and piecewise Gaussian approach outperformed the simple Monte Carlo featuring 2000 samples. Moreover, the operation was carried out in about 1 millisecond in both cases. Of course, that was expected form an operation with linear complexity.

Maximum of a Random Variable and a Constant

We will now try to approximate the result of the expression $max(X, a)$, where $a = 3$ and $X \sim N(4, 1)$. Since the constant a falls within the support of the distribution of X, the result will be a distribution that is both continuous and discrete, according to the discussion in Section 2.2.3. The reference result in this case will be produced by Monte Carlo using 100K samples. The performance measures for the different approaches are listed in Table 4.3.

Surprisingly enough, the Kolmogorov distances imply that the Monte Carlo with 2000 samples is more accurate than both of our mixture models approximations. On the other hand, the CDF distances indicate that mixture model approaches are much more accurate. This disagreement should be attributed to the tendency that the Kolmogorov distance has to penalise too much any large divergence, even if it only exists in a very small area of the support. In our case, we can see in Figure 4.3 that this happens in the area close to 3, as this is the discrete component of the mixed distribution, which has been approximated by a Gaussian spike. Thus, the information loss should be attributed to the re-approximation of the resulting distribution, as our approximating algorithms described in Section 3.4 tend to smooth the sharp areas of a density function.

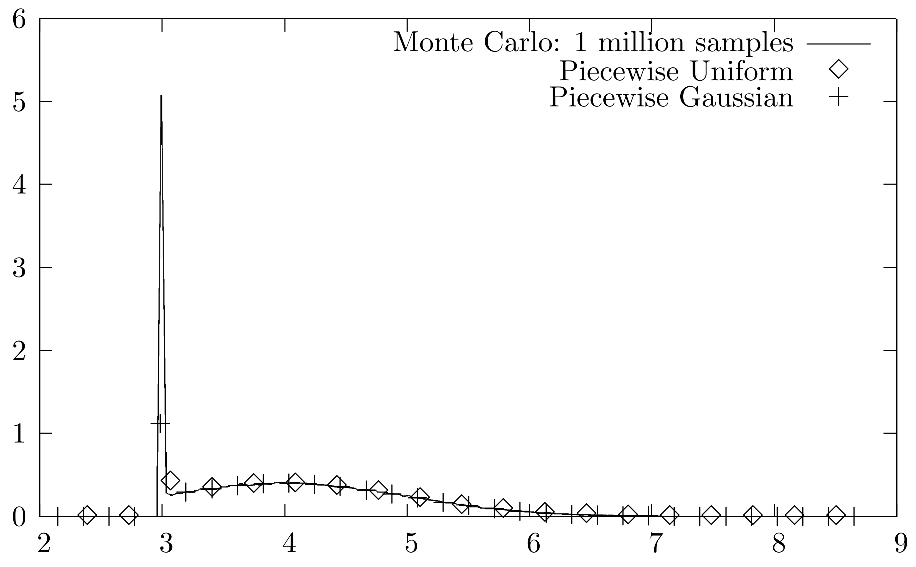

Figure 4.3: Probability density function of $max(X, 3)$, where $X \sim N(4, 1)$

Method	Similarity to True Distribution		Time (ms)
	Kolmogorov Distance	CDF Distance	
Monte Carlo (2K)	0.013207	0.0149956	30
Piecewise Uniform	0.076091	0.0058368	0.08
Piecewise Gaussian	0.078294	0.0099758	0.07

Table 4.3: Performance for $max(X, 3)$, where $X \sim N(4, 1)$

Method	Similarity to True Distribution			Time (ms)
	KL Divergence	Kolmogorov Distance	CDF Distance	
Monte Carlo (2K)	N/A	0.01989180	0.00036718	30
Piecewise Uniform	0.000271939	0.00051964	0.00001326	0.7
Piecewise Gaussian	0.000245397	0.00283481	0.00008361	0.7

Table 4.4: Performance for $1/X$, where $X \sim N(8, 0.5)$

Division by a Random Variable

As we have seen in Section 2.2.2, it is possible to exactly compute the expression $1/X$. Then, our approach approximates this intermediate result with a mixture model, so as to be used in future computations. In spite of the fact that the intermediate result is accurate, sometimes the approximation can be problematic.

What we actually test in this experiment, is the approximation of the well known distribution of $1/X$. Table 4.4 shows the results for $X \sim (8, 0.5)$. These results seem to confirm the validity of our approach, as both piecewise uniform and piecewise Gaussian approaches outperform the Monte Carlo by far.

In the second experiment however, where $X \sim N(0, 1)$, both of our approaches resulted in poor approximation of the result, as shown in Table 4.5. These rather awkward results can be explained if we observe the density of the true resulting distribution, as depicted in Figure 4.4. In fact, its probability density function features extremely sharp sections and discontinuities around them. We recall that the approximating algorithms described in Section 3.4.1 and 3.4.2, used a fixed width for each component, no matter the quality of the approximation at each step. Actually, these algorithms have proved to be quite efficient, however, it seems that this efficiency came at the price of poor approximations for sharp densities. We note this problem is also encountered in the cases of product and ratio of random variables, in Section 4.2.4.

4.2.3 Performance for the Sum and the Difference of Random Variables

In this experiment we will investigate if the mixture model approach currently proposed can actually outperform the discretisation method used in a number of previous works [29, 4, 33]. We will actually evaluate one of our main hypotheses; that is if the use of mixture models in order to perform computations of random variables, does actually improve the accuracy of the existing approximating methods based on discretisation. Table 4.6 shows

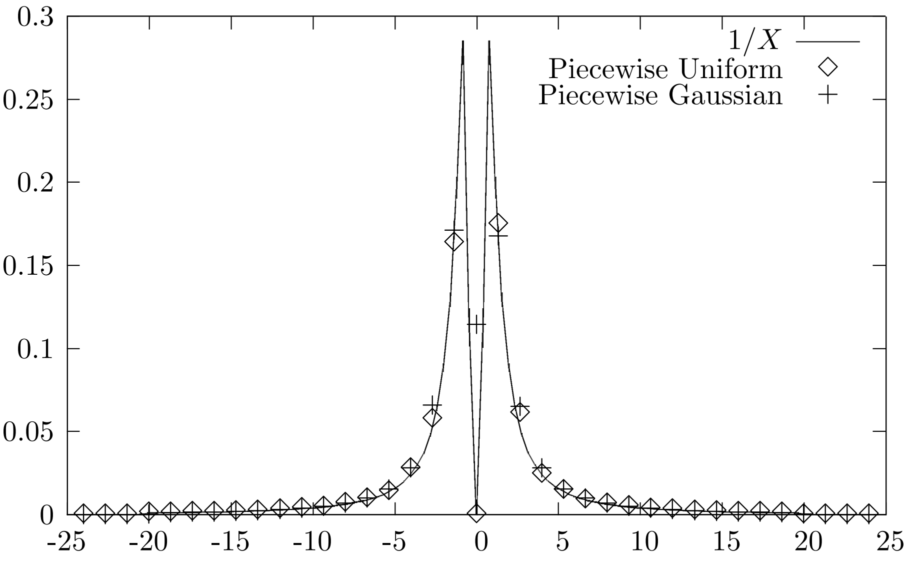

Figure 4.4: Probability density function of $1/X$, where $X \sim N(0, 1)$

Method	Similarity to True Distribution			Time (ms)
	KL Divergence	Kolmogorov Distance	CDF Distance	
Monte Carlo (2K)	N/A	0.017782	2.20295	30
Piecewise Uniform	0.038805	0.020362	0.68778	0.7
Piecewise Gaussian	0.204265	0.048501	0.68955	0.7

Table 4.5: Performance for $1/X$, where $X \sim N(0, 1)$

Method	Similarity to True Distribution			Time (ms)
	KL Divergence	Kolmogorov Distance	CDF Distance	
Monte Carlo (10K)	N/A	0.0122061	0.0361947	100
Discretisation	0.000387272	0.0005758	0.0022914	60
Piecewise Uniform	0.000354792	0.0005086	0.0019316	100
Piecewise Gaussian	0.000180433	0.0025144	0.0141960	60

Table 4.6: Performance for $X + Y$, where $X \sim N(2, 1)$ and $Y \sim N(3, 2)$

the results for two normally distributed random variables, $X \sim N(2, 1)$ and $Y \sim N(3, 2)$. The true resulting distribution will be $N \sim (5, 3)$.

Again the Monte Carlo exhibits the poorest performance, according to all of the accuracy measures. The most remarkable result though, is that the mixture of uniforms approach appears to be more accurate than the discretisation approach discussed in the literature. However, this improvement in terms of accuracy is not for free at all. The running times of for the sum using piecewise uniform approximation is approximately 30 or 40 milliseconds slower that the discretisation method. This was more or less expected, as the computation of the actual sum of uniforms is essentially an expensive operation, which is repeated N^2 times. On the other hand, the computation of the intermediate result for the piecewise Gaussian method, involves just the creation of the resulting normal distribution, which makes the whole process more efficient.

Another interesting finding is the fact that not all accuracy measures agree on the approximation quality of piecewise Gaussian. The KL divergence indicates that the sum is more accurately approximated using a Gaussian mixture, while on the other hand, distance measures based on the cumulative distribution function seem to favour the piecewise uniform approach. This issue is further investigated in Section 4.2.6, where we examine the error propagation.

The difference is also covered in this section. The implementation of the sum and the difference of random variables is equivalent, since the difference $X - Y$ is defined as the sum $X + (-Y)$, as discussed in Section 2.3.

4.2.4 Performance for the Product and the Ratio of Random Variables

As we have discussed in Section 2.4, there is no closed form expression for the product and the ration of random variables, even for simple distributions. For the piecewise uniform approach, the intermediate results for both product and ratio were approximated by uniforms, as an efficient substi-

Method	Similarity to True Distribution		Time (ms)
	Kolmogorov Distance	CDF Distance	
Monte Carlo (5K)	0.017146	0.110625	60
Discretisation	0.005952	0.064949	60
Piecewise Gaussian	0.001606	0.013265	60

Table 4.7: Performance for XY, where $X \sim U(2, 6)$ and $Y \sim N(3, 0.5)$

Method	Similarity to True Distribution		Time (ms)
	Kolmogorov Distance	CDF Distance	
Monte Carlo (5K)	0.010158	0.036066	60
Discretisation	0.018202	0.025796	60
Piecewise Gaussian	0.059393	0.158304	60

Table 4.8: Performance for XY, where $X \sim N(-1, 2)$ and $Y \sim N(0, 1)$

tute of the true product and ratio of uniform distributions. In fact, these are the only cases of operations implemented according to the discretisation approach. On the other hand, the product and the ratio for the piecewise Gaussian approach (sections 2.4.1 and 2.4.2), are supposed to approximate better the intermediate and hopefully the final results. That remains to be seen by comparing these alternative approaches using a Monte Carlo approach of 1 million samples as reference.

Tables 4.7 and 4.8 list the product results for different examples. Given the running times observed, the two alternative methods have almost identical efficiency in both examples. This was pretty much expected, if we think of the way the intermediate results are produced. Now, in the first example, where $X \sim U(2, 6)$ and $Y \sim N(3, 0.5)$, piecewise Gaussian approximation seems to achieve considerably better accuracy than the piecewise uniform approach. Our assumption that a better approximation of the intermediate result would improve the accuracy is actually confirmed in this experiment.

However, the second experiment, is actually an example of poor performance of the current implementation. For $X \sim N(-1, 2)$ and $Y \sim N(0, 1)$, the resulting distribution has a high probability spike in the area close to zero. This area is poorly approximated by both approaches, as we can even see in Figure 4.5. It is also notable that the simple Monte Carlo method exhibits higher accuracy in this case.

As happened with the product, the performance for the ratio of random

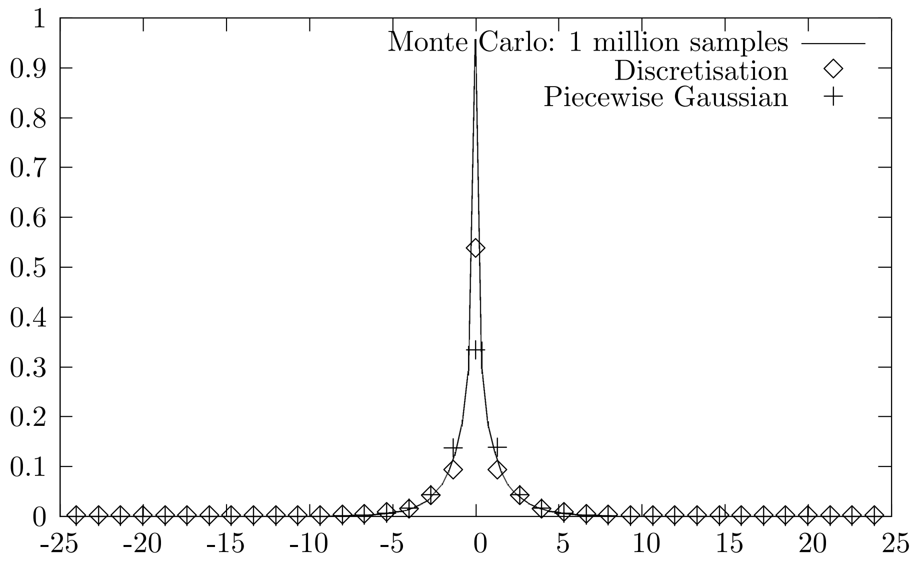

Figure 4.5: Probability density function of XY, where $X \sim N(-1, 2)$ and $Y \sim N(0, 1)$

Method	Similarity to True Distribution		Time (ms)
	Kolmogorov Distance	CDF Distance	
Monte Carlo (5K)	0.014614	0.00217576	60
Discretisation	0.008510	0.00194967	60
Piecewise Gaussian	0.005199	0.00097181	60

Table 4.9: Performance for X/Y, where $X \sim U(2, 6)$ and $Y \sim U(9, 11)$

variables tends to be poorer if the true resulting has discontinuous high probability areas. Again, we present examples that illustrate both cases. The first example, whose results are outlined in Table 4.9, clearly indicates that the piecewise Gaussian approach is more accurate.

In the second example, where $X \sim N(1, 1)$ and $Y \sim N(0, 1)$, the probability density function of the ratio is shown in Figure 4.6. Given that the currently used algorithms do not produce accurate enough approximations of such sharp density functions, the poor performance of the piecewise Gaussian shown in Table 4.10 was rather expected. It is important though, that the discretisation approach performed badly as well. Judging by its shape, it is obvious that the most of the 100 components approximate the low probability tails. The probability peak near zero, which is the most in-

Method	Similarity to True Distribution		Time (ms)
	Kolmogorov Distance	CDF Distance	
Monte Carlo (5K)	0.006029	0.4415	60
Discretisation	0.022808	3.6360	60
Piecewise Gaussian	0.083779	3.6593	60

Table 4.10: Performance for X/Y, where $X \sim N(1,1)$ and $Y \sim N(0,1)$

teresting part, is poorly approximated. Thus, Monte Carlo exhibits greater accuracy, a fact that seems to support the suggestion that the approximating algorithms should be revised to cope better with discontinuous or spiky distributions.

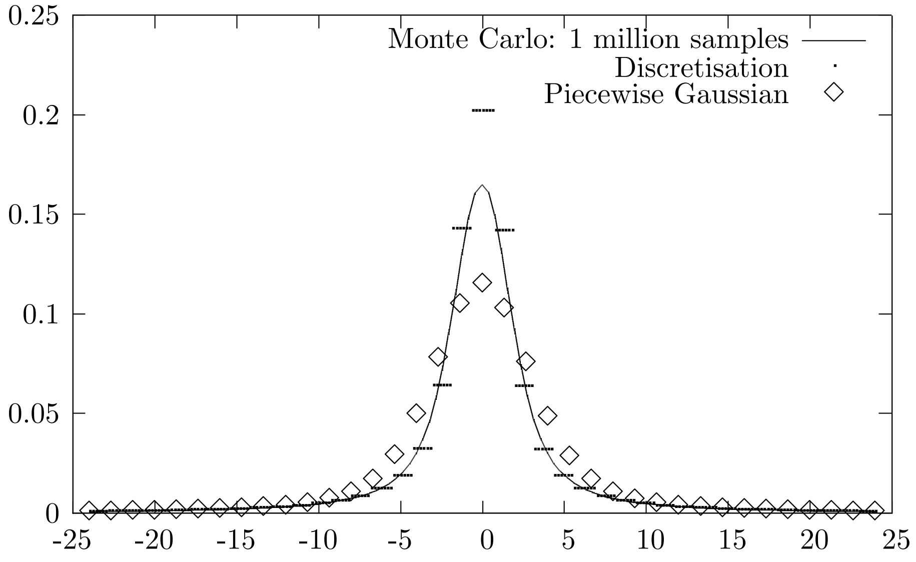

Figure 4.6: Probability density function of X/Y, where $X \sim N(1,1)$ and $Y \sim N(0,1)$

After all, we have observed that under certain circumstances, the mixture model approach does actually outperform its discretisation equivalent, for both product and ratio. In fact, in the cases that our method exhibits poor performance, the discretisation approach does not perform much better either. In fact, both approaches make use of efficient but rather naïve approximating algorithms, as we have seen that their accuracy is significantly decreased when the target distribution has a sharp density. In any case, the current results are encouraging, and it is certainly worth experimenting with more sophisticated approximating algorithms.

4.2.5 Performance for the Minimum and the Maximum of Random Variables

The implementations of both minimum and maximum of random variables make use of the exact results, as described in Section 2.5. Eventually, the theoretical results are approximated with mixture models. So, we have avoided the N^2 steps of the other binary operations, a fact that explains the high efficiency achieved for the current ones. In the first example, where $x \sim N(0, 1)$ and $Y \sim N(-1, 8)$, we have also achieved high levels of accuracy for both mixture model approximations. It is noted that the similarity measures, which are presented in Table 4.11, are with respect to the theoretical result.

In the second example summarised in Table 4.12, we have computed the minimum of two exponentially distributed variables, $X \sim \text{Exp}(2)$ and $Y \sim \text{Exp}(3)$. It is known that their minimum will follow exponential distribution $\text{Exp}(\lambda)$ with parameter $\lambda = \lambda_X + \lambda_Y$. This falls in the case of densities that have discontinuities at high probability areas, which has been already identified as problematic for our system. In Figure 4.7, we can see that the result is poorly approximated by the Gaussian mixture near the area of discontinuity. On the other hand, the piecewise uniform approximation seems less affected, a fact also verified by the similarity measures recorded.

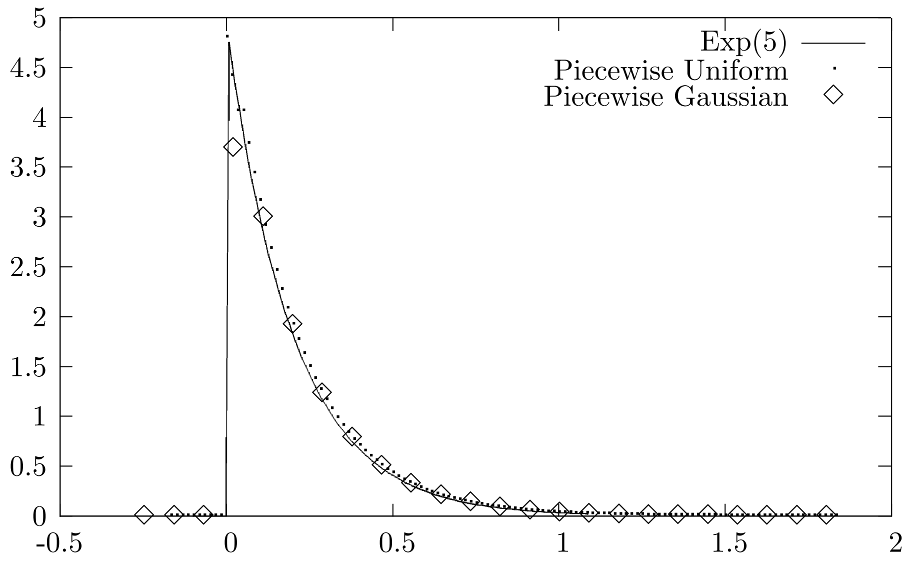

Figure 4.7: Probability density function of $min(X, Y)$, where $X \sim \text{Exp}(2)$ and $Y \sim \text{Exp}(3)$

Method	Similarity to True Distribution			Time (ms)
	KL Divergence	Kolmogorov Distance	CDF Distance	
Monte Carlo (2K)	N/A	0.0216223	0.0526379	20
Piecewise Uniform	0.00049898	0.0005259	0.0011792	0.06
Piecewise Gaussian	0.00016757	0.0022473	0.0072830	0.06

Table 4.11: Performance for $max(X, Y)$, where $x \sim N(0, 1)$ and $Y \sim N(-1, 8)$

Method	Similarity to True Distribution			Time (ms)
	KL Divergence	Kolmogorov Distance	CDF Distance	
Monte Carlo (2K)	N/A	0.0203852	0.00474036	20
Piecewise Uniform	0.0001100	0.0008231	0.00035788	0.06
Piecewise Gaussian	0.0349908	0.0373825	0.00860947	0.06

Table 4.12: Performance for $min(X, Y)$, where $X \sim \text{Exp}(2)$ and $Y \sim \text{Exp}(3)$

4.2.6 Error Propagation

Since our method involves approximation of the actual result, this essentially entails some error. From what we have seen so far, at least in the case of smooth resulting densities, our approach tends to produce smaller error than the simple Monte Carlo method, or the discretisation method used in the literature so far [29, 4, 33]. However, it is worth investigating how this error evolves in a series of computations. The experiment conducted in order to estimate the error propagation, is outlined by the following steps:

i. Pick a normally distributed random variable $X \sim N(\mu_X, \sigma_X^2)$, with random parameters

ii. Pick a normally distributed random variable $Y \sim N(\mu_Y, \sigma_Y^2)$, with random parameters

iii. Keep the result $Y \leftarrow X + Y$ and go to Step (ii).

This procedure has been applied to the piecewise uniform, the piecewise Gaussian and the discretisation approach. Figure 4.8 depicts the evolution of the error as captured by the KL divergence between the original result and the results given by the different methods.

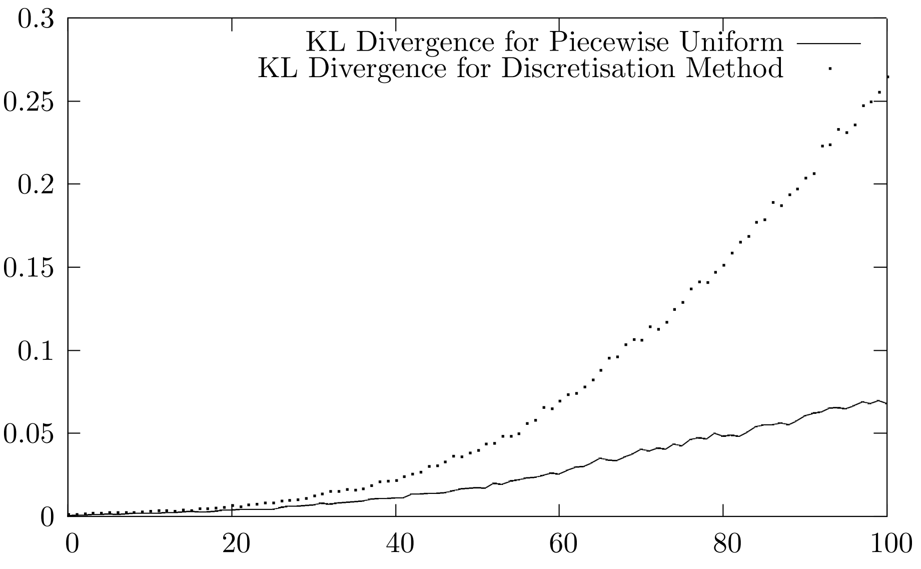

Figure 4.8: Error evolution the piecewise uniform and the discretisation approach, with N varying from 1 to 100

In the first place, we will compare the graphs of error propagation for the piecewise uniform and the discretisation method. In fact, it is confirmed that the use of the actual intermediate results can reduce the error in the long term as well. After all, the discretisation method involves information loss at two steps: the approximation of the input densities, and the approximation of the intermediate results. On the other hand, our approach implies loss of information at the first step only.

A less expected result though, was the error evolution for the piecewise Gaussian, whose error seems to have a substantially larger growth, as depicted in Figure 4.9. This result does not comply with what we have seen in Section 4.2.1, where the direct approximation with mixtures of Gaussians outperformed the one of uniforms. In both cases, the results for the sum between the components are accurate, as specified in sections 2.3.2 and 2.3.1. Their only difference is in the way they re-approximate these intermediate results. Once again, it is obvious that we need a smarter approximating algorithm, especially in the case of Gaussian mixtures.

4.3 Experiments on Dependency Tracking

The purpose of this set of experiments is to investigate the ability of the mechanism proposed in Section 3.6 to capture the comptationally arisen dependencies. The experiments involve examining the results for a number

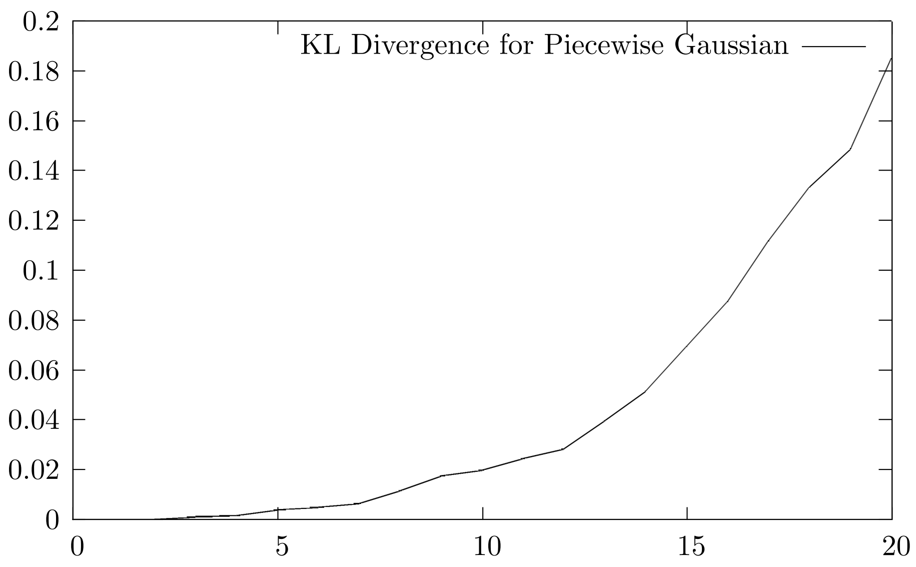

Figure 4.9: Error evolution the piecewise Gaussian approach, with N varying from 1 to 20

of simple expressions that imply dependency of the input variables.

In the algebra of random variables, variables that follow the same distribution are not essentially supposed to be equal. For example, if $X \sim N(0,1)$ and $Y \sim N(0,1)$, the result of $X + Y$ would follow a normal distribution $N(0,2)$, as we have seen in Section 2.3. On the other hand, the distribution of the resulting distribution of $2X$ would be $N(0,4)$, since we apply a linear function (Section 2.2.1). Moreover, it would be reasonable that the expression $2X$ decomposed into $X+X$. The inequality between $X+Y$ and $X+X$ can be explained if we observe that in the second case, the variables are not independent at all. That is the kind of dependency the tracking mechanism is supposed to follow. In Figure 4.10, we present the resulting densities of these expressions, as given by our Monte Carlo implementation.

Another very intuitive example is the computation of the expressions $X - X$ and $X - Y$, where X and Y are independent random variables. For convenience, let be $X \sim N(0,1)$ and $Y \sim N(0,1)$. The distribution of the difference of these independent Gaussians will be $N(0,2)$, while it is quite obvious that the in the case of $X - X$ the result should be zero in any case. These are exactly the densities derived using the dependency tracking Monte Carlo approach, as depicted in Figure 4.11.

Finally, the expression XY/X should give Y as result, but it is not the same in the case of XY/Z for independent X and Z that follow the same distribution. The results are displayed in Figure 4.12. Yet again, it is confirmed

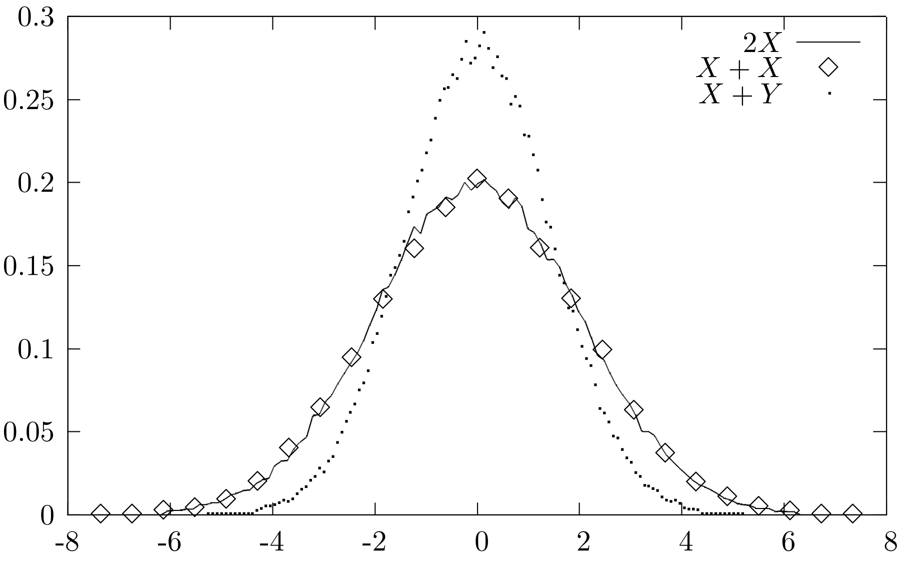

Figure 4.10: Probability density functions of $2X$, $X + X$ and $X + Y$, where $X \sim N(0, 1)$ and $Y \sim N(0, 1)$

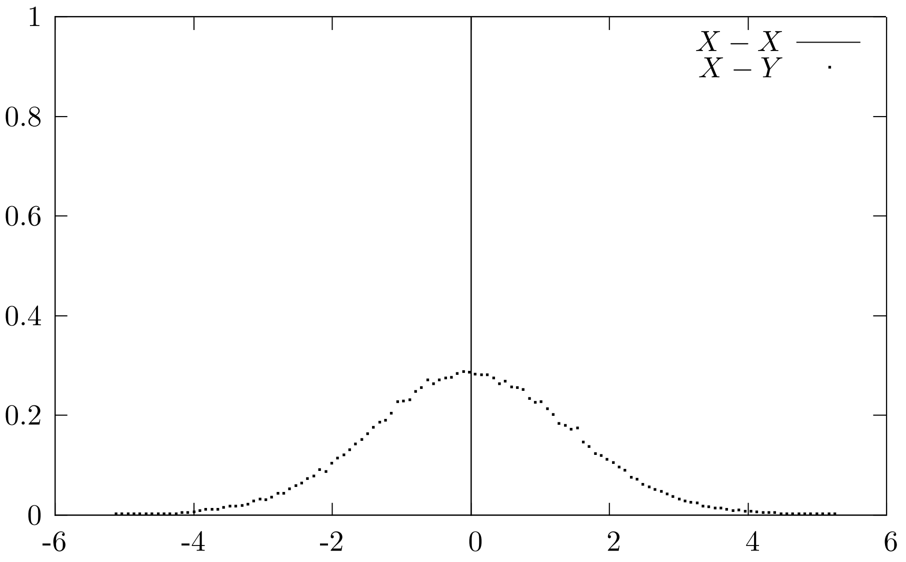

Figure 4.11: Probability density functions of $X - X$ and $X - Y$, where $X \sim N(0, 1)$ and $Y \sim N(0, 1)$

that our approach captures such computationally arisen dependencies.

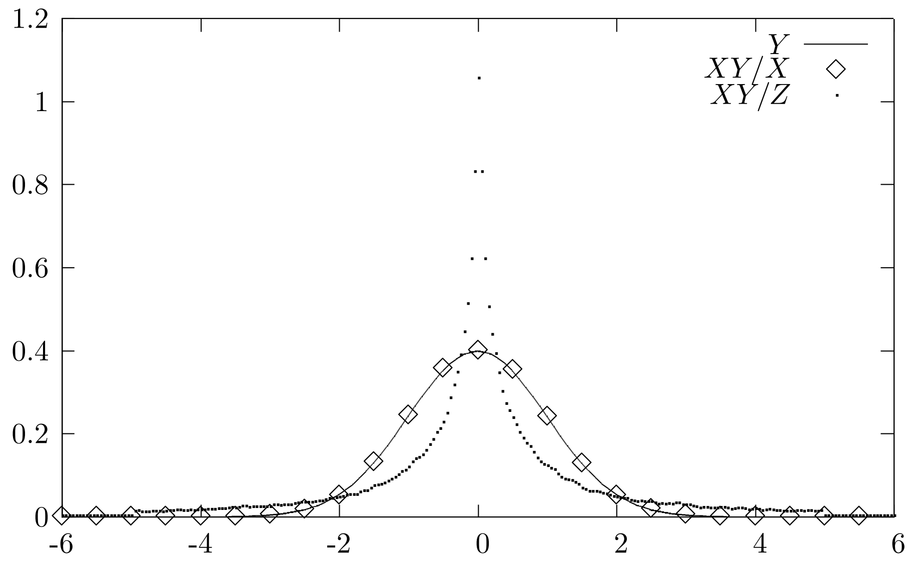

Figure 4.12: Probability density functions of XY/X and XY/Z, where $X \sim N(0,1)$, $Y \sim N(0,1)$ and $Z \sim N(0,1)$

4.4 General Remarks on the Experiments

The experiments we have performed so far should produce evidence enough to evaluate the hypotheses outlined in chapter 1. In the first place, we have seen that in most of the cases both piecewise uniform and piecewise Gaussian tend to exhibit higher accuracy than the discretisation approach used in the literature so far. However, we have also identified some problematic cases. More specifically, probability distributions that have sharp densities are usually poorly approximated, resulting in low accuracy for our method. This behaviour is mainly attributed to the approximating algorithms used (Section 3.4), as the components are uniformly distributed across the support of the target density.

The next question to be answered is whether there is significant difference between the performance of the two alternative mixture approximations proposed. The evidence collected is rather confusing, as KL divergence tends to favour the piecewise Gaussian, while the other distance metrics that are based on the cumulative distribution function seem to favour the piecewise uniform. We have to note though, that the accuracy of the piecewise uniform approach seems to be less sensitive to the problematic cases identified above. The efficiency has proved to be identical for both methods in most of the cases. However, we have to note that the running times are higher for piecewise uniform, in the cases of binary sum and difference. This was pretty mush expected, since it is more expensive to determine the exact sum and difference of uniform components, as we have seen in Section 2.3.2. In any case, the use of more sophisticated approximating algorithms, which is in the plans of future work, may change the entire picture.

Furthermore, the experiments on the dependency tracking verified that our Monte Carlo implementation does actually cature the dependencies that arise throughout the computations.

Chapter 5

Conclusion and Future Work

Summarising, the objective of this project was the development of a tool for performing computations on random variables, both accurately and efficiently. While the results for the computations discussed are well defined in terms of the probability theory, closed form expressions only exist for a limited number of distribution families and operations. Thus, we have introduced an approach that makes use of mixture model approximations in order to perform computations on arbitrarily distributed random variables. In this way, a binary operation on random variables will be implemented as a series of operations on their approximating mixture components. We have developed two alternative versions of this approach, using either mixtures of uniforms or mixtures of Gaussians.

Trying to estimate the performance of our system, we have outlined the relationship between efficiency and accuracy. The running times achieved using 100 components are usually acceptable for real time computations. A further increase in the component number can actually improve the accuracy, but also has a significantly negative effect on the efficiency. The running times for binary operations seemed to increase in a quadratic manner, which is rather expected as the computational complexity with respect to the components is $O(N^2)$. So, the use of 100 approximating components is actually a trade-off between accuracy and efficiency.

The mixture model approach can be thought of as a generalisation of the discretisation approach seen in previous works [29, 4, 33]. However, the currently proposed method produces the resulting distributions by computing exactly the intermediate results of their components. So, we would expect to see an improvement in accuracy, in comparison with the discretisation-based methods. The experiments performed supply a good amount of evidence that seems to confirm this expectation. In fact, our system exhibits higher accuracy in a number of cases, including the sum and the difference of random variables, and some cases of the product and the ratio.

We have to note however that the accuracy achieved was not satisfactory in some cases. In particular, we have discovered that high probability discontinuous areas are poorly approximated by any of our approaches, especially the piecewise Gaussian. This is actually because of the approximating algorithms we have used. In fact, the high efficiency of the approximation algorithms chosen, came at the price of poor approximation accuracy for sharp densities. In both cases, we use a fixed number of components, where the neighbouring components have the same distance between each other. This entails that too many components are spent to approximate smooth areas of the target density, while sharper areas are approximated with less than adequate components. In terms of future work, it is certainly worth experimenting on alternative algorithms that support variable number of components and variable distance between them.

Another important question that was supposed to be answered by the evaluation process, was whether any of our alternatives, uniform mixture and Gaussian mixture, significantly outperforms the other. Well, the evidence is rather confusing regarding this question, as the accuracy seems to be dependent on the type of operation and the densities of the input distributions. However, if we try more sophisticated approximators, the situation may not be the same, so further investigation is needed.

We have also seen that series of computations on random variables are possible to generate dependencies among them. The tracking of this kind of dependencies was an issue discussed in this work as well. More specifically, we have suggested that the dependencies can be adequately captured by the graph of flow of computations. However, this feature was not incorporated to the main component that makes use of mixture models. Instead, we have developed a Monte Carlo approach that takes advantage of that feature, as a proof of concept prototype. After having performed a number of experiments, we have gathered evidence that confirm our suggestion. Moreover, the dependency tracking Monte Carlo implementation can be used as a future point of reference.

All these findings discussed so far are rather encouraging for the further development of this project. The next task should be the improvement of the approximating process, as it seems to be responsible for the poor results given for distributions with sharp densities. In this case, further experimentation could be much more enlightening about the potential of the proposed method. Furthermore, since the use of computational graphs proved to adequately track the dependencies that are formed, it certainly makes sense to be used in terms of the mixture model approximation.

Appendix A

Concrete Distribution Classes

The following classes represent the continuous distributions with known parameters implemented so far. They can be components in a mixture model, so they implement the abstract *MixtureComponent* class. Along with the *EmpiricalDistribution* and the *MixtureModel*, they are the only distribution classes visible to the users.

A.1 The *Uniform* Distribution Class

Parameters	: endpoints a and b, where $a < b$
PDF	: $f(x) = \begin{cases} \frac{1}{b-a}, & x \in [a, b] \\ 0, & \text{otherwise} \end{cases}$
CDF	: $F(x) = \begin{cases} 0, & x < a \\ \frac{x-a}{b-a}, & a \leq x < b \\ 1, & b \leq x \end{cases}$
Constructor	: `Uniform(double a, double b)`

A.2 The *Gaussian* Distribution Class

Parameters	: mean μ and variance σ^2, where $\sigma^2 > 0$
PDF	: $f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$
CDF	: $F(x) = \frac{1}{2} + \frac{1}{2}\text{erf}\left(\frac{x-\mu}{\sqrt{2\sigma^2}}\right)$
Constructor	: `Gaussian(double mean, double variance)`

The expressions $exp(x)$ and $erf(x)$ are computed using the corresponding functions of `cmath` library.

A.3 The *Exponential* Distribution Class

Parameters	: rate λ, where $\lambda > 0$
PDF	: $f(x) = \lambda e^{-\lambda x}$
CDF	: $F(x) = 1 - e^{-\lambda x}$
Constructor	: `Exponential(double rate)`

The expression $exp(x)$ is computed using `exp(double)` function of `cmath` library.

A.4 The *Cauchy* Distribution Class

Parameters	: location x_0 and scale γ, where $\gamma > 0$
PDF	: $f(x) = \frac{1}{\pi}\left[\frac{\gamma}{(x-x_0)^2+\gamma^2}\right]$
CDF	: $F(x) = \frac{1}{\pi}\arctan\left(\frac{x-x_0}{\gamma}\right) + \frac{1}{2}$
Constructor	: `Cauchy(double location, double scale)`

The expression $arctan(x)$ is computed using `atan(double)` function of `cmath` library.

A.5 The *ChiSquare* Distribution Class

Parameters	: degrees of freedom k, where $k \in \mathbb{N}^*$
PDF	: $f(x) = \frac{(1/2)^{k/2}}{\Gamma(k/2)} x^{k/2-1} e^{-x/2}$
CDF	: $F(x) = \frac{\gamma(k/2,x/2)}{\Gamma(k/2)}$
Constructor	: `ChiSquare(double degrees_of_freedom)`

The expressions $\Gamma(x)$ and $\gamma(s,x)$ are computed using the corresponding functions of `boost` library. Their definitions can be found in [1].

Appendix B

Code Examples of *Stochastic* Library

B.1 Initialisation of Random Variables

```
#include <stochastic.h>

using namespace stochastic;

int main()
{
  // Any of the following is valid

  RandomVariable a(new Gaussian(0, 1));

  RandomVariable b = new Gaussian(0, 1);

  Gaussian normal_distribution(0, 1);
  RandomVariable c = & normal_distribution;
  RandomVariable d(& normal_distribution);

  return 0;
}
```

B.1.1 Initialisation Using a Dataset

```
#include <stochastic.h>

using namespace stochastic;

int main()
{
  // The distribution of the "raw" object is
  // directly defined by the dataset stored in
  // "dataDistribution" object.

  EmpiricalDistribution dataDistribution("data");
  RandomVariable raw = & dataDistribution;

  // The "refined" object is assigned an approximation
  // of the dataset's distribution.
  // The dataset may now be discarded.

  PiecewiseGaussian approximation(& dataDistribution);
  RandomVariable refined = & approximation;

  // It is reminded that PiecewiseUniform
  // and PiecewiseGaussian
  // are special cases of mixture models
  // that are used to approximate other distributions.

  return 0;
}
```

B.1.2 Initialisation Using a Mixture Model

```
#include <stochastic.h>
#include <vector>

using namespace stochastic;

int main()
{
  std::vector <MixtureComponent *> c; // components
  std::vector <double> w;             // weights

  c.push_back(new Gaussian(0, 1));
  c.push_back(new Gaussian(3, 2));
  w.push_back(2);
  w.push_back(1);
  // the weights will be normalised
  // Tass the components and the weights defined
  // to the MixtureModel constructor.

  // Then, pass the MixtureModel created
  // to a RandomVariable

  RandomVariable mixture_model = new MixtureModel(c, w);

  return 0;
}
```

B.2 Switch among Methods of Computation

```
#include <stochastic.h>

using namespace stochastic;

int main()
{
  RandomVariable a = new Gaussian(0, 1);
  RandomVariable b = new Gaussian(3, 2);

  // Each time, the exression "a + b"
  // will be computed in a different way

  // Monte Carlo with 1000 samples
  RandomVariable::setMonteCarlo(1000);
  a + b;

  // Piecewise uniform with 100 components
  RandomVariable::setPiecewiseUniform(100);
  a + b;

  // Piecewise Gaussian with 100 components
  RandomVariable::setPiecewiseGaussian(100);
  a + b;

  return 0;
}
```

Bibliography

[1] Milton Abramowitz and Irene A. Stegun. *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables.* Dover, New York, ninth dover printing, tenth gpo printing edition, 1964.

[2] D. Alspach and H. Sorenson. Nonlinear bayesian estimation using gaussian sum approximations. *IEEE Trans. on Automatic Control*, (17):438–448, 1972.

[3] Leo A. Aroian. The probability function of the product of two normally distributed variables. In *The Annals of Mathematical Statistics*, volume 18, pages 265–271, 1947.

[4] D. Berleant and C. Goodman-Strauss. Bounding the results of arithmetic operations on random variables of unknown dependency using intervals. In *Reliable Computing*, volume 4, pages 147–165, May 1998.

[5] D. Berleant and J. Zhang. Arithmetic on random variables: Squeezing the envelopes with new joint distribution constraints. In *4th International Symposium on Imprecise Probabilities and Their Applications*, 2005.

[6] Christopher M. Bishop. *Pattern Recognition and Machine Learning.* Springer Science+Business Media, 233 Spring Street, New York, NY 10013, USA, first edition, 2006.

[7] R.L. Burden and J.D. Faires. *Numerical Analysis.* Brooks/Cole, seventh edition, 2000.

[8] E.v. Collani and F. Killmann. A note on the convolution of uniform and related distributions and their use in quality control. In *Economic Quality Control*, volume 16, pages 17–41, 2001.

[9] A.G. Colombo and R.J. Jaarsma. A powerful numerical method to combine random variables. In *IEEE Transactions on Reliability*, pages 126–129, June 1980.

[10] C.C. Craig. On the frequency function of *xy*. In *The Annals of Mathematical Statistics*, volume 7, pages 1–15, 1936.

[11] J.H. Curtiss. On the distribution of the quotient of two chance variables. In *The Annals of Mathematical Statistics*, volume 12, pages 409–421. Institute of Mathematical Statistics, December 1941.

[12] L. Xie D. Berleant and J. Zhang. Statool: A tool for distribution envelope determination (denv), an interval-based algorithm for arithmetic on random variables. In *Reliable Computing*, volume 9, pages 91–108, April 2003.

[13] A. DasGupta. *Asymptotic Theory of Statistics and Probability.* Springer Texts in Statistics, first edition, 2008.

[14] 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 (Methodological)*, 39(1):1–38, 1977.

[15] Luc Devroye. *Non-Uniform Random Variate Generation.* Springer, 1986.

[16] Benjamin Epstein. Some applications of the mellin transform in statistics. In *The Annals of Mathematical Statistics*, volume 19, pages 370–379, Sep 1948.

[17] William Feller. *An Introduction to Probability Theory and Its Applications, Volume 1.* Wiley, January 1968.

[18] D. Freedman and P. Diaconis. On the histogram as a density estimator: L_2 theory. 57(4):453–476, December 1981.

[19] R. Frühwirth. A gaussian-mixture approximation of the betheheitler model of electron energy loss by bremsstrahlung. In *Computer Physics Communications*, volume 154, pages 131–142, 2003.

[20] J. Galambos. *Advanced Probability Theory.* CRC Press, second edition, 1995.

[21] Noah D. Goodman, Vikash K. Mansinghka, Daniel Roy, Keith Bonawitz, and Joshua B. Tenenbaum. Church: a language for generative models. In *Uncertainty in Artificial Intelligence*, 2008.

[22] Charles M. Grinstead and James L. Snell. *Introduction to Probability.* AMS Bookstore, second edition, 1997.

[23] J.C. Hayya and W.L. Ferrara. On normal approximations of the frequency functions of standard forms where the main variables are normally distributed. In *Management Science*, volume 19, pages 173–186, 1972.

[24] J.R. Hershey and P.A. Olsen. Approximating the kullback leibler divergence between gaussian mixture models. In *IEEE International Conference on Acoustics, Speech and Signal Processing, 2007. ICASSP 2007.*, volume 4, pages IV–317–IV–320, 2007.

[25] D.V. Hinkley. On the ratio of two correlated normal random variables. In *Biometrika*, volume 56, pages 635–639, 1969.

[26] S. Ferson H.M. Regan and D. Berleant. Equivalence of methods for uncertainty propagation of real-valued random variables. In *International Journal of Approximate Reasoning*, volume 36, pages 1–30, April 2004.

[27] Hwei P. Hsu. *Probability, Random Variables, and Random Processes.* McGraw-Hill, 1996.

[28] S. Chakraborti J.D. Gibbons. *Nonparametric Statistical Inference.* Marcel Dekker Inc., 270 Madison Avenue, New York, NY 10016, USA, fourth edition, 2003.

[29] S. Kaplan. On the method of discrete probability distributions in risk and reliability calculations—application to seismic risk assessment. In *Risk Analysis*, volume 1, pages 189–196, August 1981.

[30] R.B. Kearfott and V. Kreinovich. *Applications of interval computations.* Kluwer Academic Publishers, first edition, 1995.

[31] Solomon Kullback and Richard A. Leibler. On information and sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86, 1951.

[32] Laha and J. Roy Chakravarti. *Handbook of Methods of Applied Statistics Volume I.* John Wiley and Sons, 1967.

[33] W. Li and J.M. Hyman. Computer arithmetic for probability distribution variables. In *Reliability Engineering and System Safety*, volume 85, pages 191–209, April 2004.

[34] Z.A. Lomnicki. On the distribution of products of random variables. 29(3):513–524, 1967.

[35] David J.C. MacKay. *Information Theory, Inference, and Learning Algorithms.* Cambridge University Press, first edition, 2003.

[36] N. Metropolis and S. Ulam. The monte carlo method. 44(247):335341, September 1949.

[37] R.B. Nelsen M.J. Frank and B. Schweizer. Best-possible bounds for the distribution of a sum—a problem of kolmogorov. In *Probability Theory and Related Fields*, volume 74, pages 199–211, June 1987.

[38] R.E. Moore. Risk analysis without monte carlo methods. pages 1–48, 1984.

[39] R.B. Nelsen. *An Introduction to Copulas (Springer Series in Statistics).* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[40] The Distribution of Products of Independent Random Variables. M. d. springer and w. e. thompson. In *SIAM Journal on Applied Mathematics*, volume 14, pages 511–526, May 1966.

[41] Athanasios Papoulis. *Probability, Random Variables and Stochastic Processes.* McGraw-Hill Companies, February 1991.

[42] E. Parzen. On the estimation of a probability density function and the mode. In *Annals of Mathematical Statistics*, volume 33, pages 1065–1076, September 1962.

[43] E. Parzen. Nonparametric statistical data modeling. In *Journal of the American Statistical Association*, volume 74, pages 105–121, March 1979.

[44] V.K. Rohatgi. *An Introduction to Probability Theory Mathematical Statistics.* Wiley, New York, 1976.

[45] R.Y. Rubinstein and D.P. Kroese. *Simulation and the Monte Carlo Method.* John Wiley & Sons, Inc., second edition, 2008.

[46] Marvin K. Simon. *Probability Distributions Involving Gaussian Random Variables: A Handbook for Engineers, Scientists and Mathematicians.* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[47] M.D. Springer. *The Algebra of Random Variables.* Wiley, first edition, 1979.

[48] K. Thulasiraman and M. N. S. Swamy. *Graphs: Theory and Algorithms.* John Wiley and Son, 1992.

[49] J. von Neumann. Various techniques used in connection with random digits. monte carlo methods. In *National Bureau of Standards*, volume 12, pages 36–38, 1951.

[50] L. Wasserman. *All of Statistics: A Concise Course in Statistical Inference.* Springer Texts in Statistics, 2005.

[51] R. C. Williamson and T. Downs. Probabilistic arithmetic. i. numerical methods for calculating convolutions and dependency bounds. *Int. J. Approx. Reasoning*, 4(2):89–158, 1990.

[52] J. Wolfowitz. Additive partition functions and a class of statistical hypotheses. In *Annals of Mathematical Statistics*, volume 13, pages 247–279, September 1942.