UNIVERSITÀ DI PISA

Department of Information Engineering (DII)
M.Sc in Computer Engineering

Symbolic And Evolutionary Artificial Intelligence Project

Pietrangelo Manco, Valerio Secondulfo

Academic Year 2022/2023

Contents

1 Introduction

The goal of this report is to expose and analyze the work carried out for the Symbolic and Evolutionary Artificial Intelligence course. The topic of the project is "New Non-Archimedean Probability Distribution Functions for handling rare events". In this introduction section, a brief theory overview will be presented, as well as a look at the previous work performed by Professor Cococcioni and his team. After that, there will be a section in which the preliminary operations needed to the five experiments performed are explained. Finally, the five experiments are described.

1.1 Brief Theory Overview

Standard Analysis works on the field of Real Numbers $\mathbb{R}$, which doesn't recognize the existence of infinity and infinitesimal quantities. Classical statistics has always worked using standard analysis, and probability distributions have always been defined over $\mathbb{R}$. This approach has proven to be limiting sometimes: there are many heavy tailed phenomena (described by probability distributions with infinite variance) which are extensively studied for their engineering applications, and that would be described almost perfectly by a log normal distribution. The problem is that such distribution is characterized by a finite variance, a feature which isn't ideal in the proposed scenario, so it must be discarded. This problem can be solved using the Robinson's Non Standard Analysis, which works on the field of the Hyper Real Numbers $\mathbb{R}^*$, generated by the union of $\mathbb{R}$, infinite and infinitesimal quantities (the former are larger than any real number, the latter are smaller than any real number, but greater than zero). To be more precise, the theory used in this work is Benci's Alpha Theory, which is a particular version of Non Standard Analysis, and the associated field is $\mathbb{E}$, which stands for Euclidean Numbers, a non standard extension of $\mathbb{R}$ that is axiomatically defined starting from it. Algorithmic Numbers (AN) are a subset of $\mathbb{E}$ that can be better standardized, and therefore easily manipulated via computer. They're based on the concept of monosemium, which is a number in the form $r\alpha^p$, with $r \in \mathbb{R}$, $p \in \mathbb{Q}$ and α is the infinite of the Alpha Theory. An Algorithmic Number $\xi \in \mathbb{E}$ is a finite sum of monosemia:

$$\xi = \sum_{k=0}^{l} r_k \alpha_k^{s_k}; \quad r_k \in \mathbb{R}, \quad s_k \in \mathbb{Q}; \quad s_k > s_{k+1} \tag{1}$$

It's always possible to express an AN in the following normal form:

$$\xi = \alpha^p P(\eta^{\frac{1}{m}}); \quad p \in \mathbb{Q}, \quad m \in \mathbb{N} \tag{2}$$

Where η is the reciprocal of α (and therefore an infinitesimal number), and $P(x)$ is a polynomial with real coefficients such that $P(0) \neq 0$.

Finally, there is a subset of AN called Bounded Algorithmic Numbers (BAN), that's the one which is actually exploited in this work. A BAN is an AN with $m = 1$:

Funzione Log normale è una gaussiana

$$\xi = \alpha^p P(\eta); \quad p \in \mathbb{Q} \tag{3}$$

La distribuzione log-normale può avere code pesanti, il che significa che la probabilità di osservare valori estremi è maggiore rispetto a una distribuzione normale.

1.2 Previous Work

As mentioned before, this work was heavily based on a previous research of Professor Cococcioni and his team. In this section, a brief overview of the results previously obtained is presented. The goal was to construct a Log Normal probability distribution with finite mean and infinite variance, which would be ideal to model the engineering applications aforementioned. It was achieved by generating a sample of random BANs according to a Log Normal distribution, whose chosen mean of the underlying Gaussian was $\mu = -\frac{1}{2}\alpha^2 + 9.54618$, and the relative variance $\sigma = \alpha$, thus obtaining a Log Normal finite mean $E(X) = e^{9.54618}$ and infinite variance $VAR(X) = e^{(1+19.0936\eta^2)\alpha^2} - e^{19.0936}$. Those expected results were verified experimentally, by generating random samples of 10^N BANs, with $N \in [3, 7]$. As shown in the Figure 1, there were some numerical issues for the smaller samples, but with $N = 6, 7$, the results are extremely accurate (with the constraint that real coefficients below $2.5 \cdot 10^{-3}$ are set to 0).

scopo della ricerca di partenza è stato quello di usare i ban per ottenere una funzione log normale a partire da un set di ban randomici avente media finita e varianza infinita

#	$E\{X\}$ (Sample Mean)	$VAR\{X\}$ (Sample Variance)
10^3	$e^{(\mathbf{0.0260943}+9.55455\eta^2)\alpha^2}$	$e^{(1.10438+19.1091\eta^2)\alpha^2} - e^{(\mathbf{0.0521886}+19.1091\eta^2)\alpha^2}$
10^4	$e^{(-\mathbf{0.0126293}+\mathbf{0.0190467}\eta+9.56013\eta^2)\alpha^2}$	$e^{(0.949483+\mathbf{0.0380934}\eta+19.1203\eta^2)\alpha^2} - e^{(-\mathbf{0.0252587}+\mathbf{0.0380934}\eta+19.1203\eta^2)\alpha^2}$
10^5	$e^{(-\mathbf{0.00351095}+9.54501\eta)\alpha}$	$e^{(1.00631-\mathbf{0.0070219}\eta+19.09\eta^2)\alpha^2} - e^{(\mathbf{0.00315714}-\mathbf{0.0070219}\eta+19.09\eta^2)\alpha^2}$
10^6	$e^{9.54617}$	$e^{(0.995845+19.0923\eta^2)\alpha^2} - e^{19.0923}$
10^7	$e^{9.54666}$	$e^{(1.00065+19.0933\eta^2)\alpha^2} - e^{19.0933}$

(a)

Figure 1: Results of the simulation for various sample sizes.

The results reported could help to numerically verify phenomena which require heavy tailed Log Normal PDFs to be correctly modelled, such as multiplicative stochastic processes.

1.3 Project Goal

The goal of the project is to make use of the work reported in the previous section in order to generate a Log Normal probability distribution which is the product of two distributions of the same type, with the same (finite) mean; the two variances have to be infinite, with the same real part and different infinitesimal ones; in addition to that, the two factor distributions have to be completely anti correlated, so that the predicted variance for their product will be finite. To achieve that, the tools received by the professor had to be modified accordingly (Section 2), and some preliminary experiments had to be performed. In order, the experiments designed and performed were the following:

- Product of two random samples of real numbers distributed according to a Log Normal PDF, with the same means and the same real, finite variances, completely anti correlated ($\rho = -1$);
- Sum of two random samples of BANs distributed according to a Gaussian PDF, with the same means and the same real, finite variances, completely anti correlated ($\rho = -1$);

- Sum of two random samples of BANs distributed according to a Gaussian PDF, with the same means and variances composed of the same real, finite part and different first order infinitesimal part (the term in η), completely anti correlated ($\rho = -1$);
- Product of two random samples of BANs distributed according to a Log Normal PDF, with the same means and variances composed of the same real, finite part and different first order infinitesimal part, completely anti correlated ($\rho = -1$).
- Sum of two random samples of BANs distributed according to a Gaussian PDF, with the same means and variances composed of the same real, infinite part and different first order infinitesimal part (in the case of first order α, this term is real, as $\eta = \frac{1}{\alpha}$), completely anti correlated ($\rho = -1$);

In case of success, another evidence of the effectiveness of the Non Standard approach towards statistics would be obtained, and these results may also have some applications in the near future.

1.3.1 Expected Variance in the last three cases

The initial idea was that, having two Log Normal PDFs with the same mean and standard deviations:

$$\sigma_1 = a + b\eta \tag{4}$$

$$\sigma_2 = a + c\eta \tag{5}$$

The product distributions had a standard deviation with a 0 finite part and an infinitesimal part different from 0:

$$\sigma_{product} = 0 + d\eta \tag{6}$$

This idea proved to be wrong during the tests, as the first order term in η appeared to be 0. Upon further calculations, it was realized that the first non 0 term is the second order one, thus:

$$\sigma_{product} = 0 + 0\eta + d\eta^2 \tag{7}$$

As shown in the following calculation:

$$\sigma_1 = a + b\eta + 0\eta^2, \quad \sigma_2 = a + c\eta + 0\eta^2; \tag{8}$$

$$\sigma_1^2 = a^2 + 2ab\eta + b^2\eta^2, \quad \sigma_2^2 = a^2 + 2ac\eta + c^2\eta^2, \quad \sigma_1\sigma_2 = a^2 + ab\eta + ac\eta + bc\eta^2; \tag{9}$$

$$\sigma_{product} = \sigma_1^2 + \sigma_2^2 + 2\sigma_1\sigma_2\rho = \sigma_1^2 + \sigma_2^2 - 2\sigma_1\sigma_2 \quad \Rightarrow$$

$\sigma_{product} = a^2 + 2ab\eta + b^2\eta^2 + a^2 + 2ac\eta + c^2\eta^2 - 2(a^2 + ab\eta + ac\eta + bc\eta^2) = 2ab\eta + b^2\eta^2 + 2ac\eta + c^2\eta^2 - 2(ab\eta + ac\eta + bc\eta^2) = b^2\eta^2 + 2ac\eta + c^2\eta^2 - 2(ac\eta + bc\eta^2) = b^2\eta^2 + c^2\eta^2 - 2bc\eta^2 = (b\eta - c\eta)^2 = (b - c)^2\eta^2$. (10)

This isn't a problematic result: the only implication for our project goal is that it had to be shown that, in the finite variance cases, the first 2 terms go to 0 instead of the first, and that the second order term is finite, and that the resulting variance is finite instead of infinitesimal in the infinite variances case. From a purely theoretical point of view, nothing at all changed, but it became slightly more computing intensive to calculate the second order terms. To conclude this section, the calculation for the infinite variances case is shown:

$$\sigma_1 = a\alpha + b + 0\eta, \quad \sigma_2 = a\alpha + c + 0\eta; \tag{11}$$

$$\sigma_1^2 = a^2\alpha^2 + 2ab\alpha + b^2, \quad \sigma_2^2 = a^2\alpha^2 + 2ac\alpha + c^2, \quad \sigma_1\sigma_2 = a^2\alpha^2 + ab\alpha + ac\alpha + bc; \tag{12}$$

$$\sigma_{product} = \sigma_1^2 + \sigma_2^2 + 2\sigma_1\sigma_2\rho = \sigma_1^2 + \sigma_2^2 - 2\sigma_1\sigma_2 \quad \Rightarrow$$

$\sigma_{product} = a^2\alpha^2 + 2ab\alpha + b^2 + a^2\alpha^2 + 2ac\alpha + c^2 - 2(a^2\alpha^2 + ab\alpha + ac\alpha + bc) = 2ab\alpha + b^2 + 2ac\alpha + c^2 - 2(ab\alpha + ac\alpha + bc) = b^2 + 2ac\alpha + c^2 - 2(ac\alpha + bc) = b^2 + c^2 - 2bc = (b-c)^2$.(13)

As mentioned, the resulting variance is just a finite term in this case.

2 Preliminary Operations

In order to be able to perform the tasks previously listed, the Matlab libraries used for the construction of the sample of BANs distributed according to a Log Normal PDF were exploited; such libraries are called 'Ban' and 'BanArray', and contain several methods and constructors to instantiate BANs, arrays of BANs and to perform simple and complex operations involving them. That said, in order to use those libraries to carry out the proposed work some modifications were needed:

- The BanArray constructor had to be modified;
- A sorting method for the BanArray class had to be implemented;
- The method for the exponentiation of a BAN had to be modified.

2.1 BanArray Constructor

The BanArray constructor needed to be modified: it generated a M x N matrix of BANs with just the first coefficient different from 0, but it was necessary to make it generate an array of M BANs with N coefficients each, in order to be able to generate a sample of random BANs with the desired characteristics. The new constructor code is the following:

```
function obj = BanArray(mat, e)
        if nargin ≠ 0
            % BanArray constructor for BanArray
            if nargin == 1
                e = 0;
            end
                nR = size(mat,1);
                if(size(e) == 1)

                    a = e*ones(nR,1);
                else
                    if(length(e) == nR)
                        a = e';
                    else
                        error("bad alpha's exponent size.");
                    end
                end
                obj = repmat(obj, nR, 1);  % Preallocate obj array
                for r = 1:nR
                    coef = mat(r, :);
                    obj(r).bArr = Ban(coef, a(r));
                end
        end
    end % constructor
```

In addition to that, the variants of the constructor which allow to instantiate an all zeros or all ones vector of BANs were accordingly modified:

```
function obj = zerosLike(obj,varargin)
        if nargin == 1
            error('Please provide dimensions.');
        end
        if nargin == 2
            error('Please provide second dimension.');
        end
        if nargin == 3
            nR = varargin{1};
            nC = varargin{2};

            obj = repmat(obj, nR, 1);  % Preallocate obj array
            coef = zeros(nC,1);
            for r = 1:nR
                obj(r).bArr = Ban(coef, 0);
            end
        end
    end % END zeros
```

```
function obj = onesLike(obj,varargin)
        if nargin == 1
```

```
3                 error('Please provide dimensions.');
4             end
5             if nargin == 2
6                 error('Please provide second dimension.');
7             end
8             if nargin == 3
9                 nR = varargin{1};
10                nC = varargin{2};
11
12                obj = repmat(obj, nR, 1);  % Preallocate obj array
13                coef = ones(nC,1);
14                for r = 1:nR
15                    obj(r).bArr = Ban(coef, 0);
16                end
17            end
18            if nargin == 4
19                nR = varargin{1};
20                nC = varargin{2};
21                e = varargin{3};
22                if(size(e) == 1)
23                    a = e*ones(nR,1);
24                else
25                    if(length(e) == nR)
26                        a = e';
27                    else
28                        error("bad alpha's exponent size.");
29                    end
30                end
31                obj = repmat(obj, nR, 1);  % Preallocate obj array
32                coef = ones(nC,1);
33                for r = 1:nR
34                    obj(r).bArr = Ban(coef, a(r));
35                end
36            end
37        end % END ones
```

2.2 BanArray Sorting Method

The BanArray sorting method wasn't present in the class yet and it was fully implemented. There was the need to order the two samples generated in every experiment in opposite ways, in order to simulate a perfect anti correlation ($\rho = -1$), so its input arguments are a BanArray and a string, which can be 'ascend' or 'descend' (otherwise it returns an error). The method is a quick sort, as it's the fastest possible (if N is the length of the array, the algorithm complexity is $O(N \log N)$). The sorting block is made of 3 methods:

```
    function sortedArray = sort(banArray, order)
            n = numel(banArray);
            sortedArray = banArray;

            % Call the Quicksort function
            sortedArray = quicksort(sortedArray, 1, n, order);
        end

        function sortedArray = quicksort(arr, low, high, order)
            if low < high
                % Partition the array and get the pivot index
                [arr, pivotIndex] = partition(arr, low, high, order);

                % Recursively call quicksort on the left and right partitions
                arr = quicksort(arr, low, pivotIndex - 1, order);
                arr = quicksort(arr, pivotIndex + 1, high, order);
            end

            sortedArray = arr;
        end

        function [arr, pivotIndex] = partition(arr, low, high, order)
            % Select the last element as the pivot
            pivot = arr(high).bArr;

            % Initialize the pivot index
            pivotIndex = low;

            for i = low:high-1
                obj_i = arr(i).bArr;

                % Compare elements based on the specified order
                if (order == "descend" && ¬le(obj_i, pivot)) || (order == ...
                    "ascend" && lt(obj_i, pivot))
                    % Swap elements
                    temp = arr(pivotIndex).bArr;
                    arr(pivotIndex).bArr = arr(i).bArr;
                    arr(i).bArr = temp;

                    % Move the pivot index forward
                    pivotIndex = pivotIndex + 1;
                end
            end

            % Swap the pivot element with the element at the pivot index
            temp = arr(pivotIndex).bArr;
            arr(pivotIndex).bArr = arr(high).bArr;
            arr(high).bArr = temp;
        end
```

2.3 Exponentiation of a Ban

The function for the exponentiation of a BAN was already in the Ban class, but it was probably thought for BANs with the finite part equal to 0, as well as the exponent of α. It was necessary to make it general, so it was modified as follows:

```
function eb = exp(b) % exponentiation
            c = b.coeff(1);
            b = b - Ban(b.coeff(1));
            b_square = b*b;
            b_cube = b_square*b;
            eb = (b + b_square/2.0 +  b_cube/6.0 + Ban(1.0)) * exp(c);
        end
```

This method extracts the finite part and calculates its exponential by hand. Then, the exponential property of the product allows to compute the product of it and the Taylor expansion of the exponential of the remaining BAN, which is justified because it's very close to 0 (the remaining BAN has the form $\xi = 0 + a\eta + b\eta^2 + ...$). This approximation works well in the case of α^0 as infinite factor, but it fails when the power of α is greater than 0. Anyways, it still works better than the previous one in the latter case.

3 Experiments Discussion

In this section, the performed experiments are presented. The general workflow applied for all of them was:

- Choose a size for the generated samples;
- Define the parameters of the underlying Gaussians;
- Construct the Gaussians with the defined parameters;
- Where the LogNormals were considered, construct the LogNormals via exponentiation of the underlying Gaussians and compute their theoretical values;

- Order the two samples in opposite ways, then execute the product of the two to generate the final distribution;
- Confront the computed values of the final distribution with the theoretical ones.

3.1 Finite Numbers Log Normals

The first experiment was performed over finite, real numbers, just to be sure that the proposed approach worked in the simplest case. The code of the Matlab script used, properly commented, is reported below:

```
% Number of elements of the two samples of random generated numbers
N = 1e7;

% Means of the two samples
mu1 = 1;
mu2 = 1;

% Standard Deviations of the two samples
sig1 = 0.2;
sig2 = 1;

% Correlation coefficient of the two samples
rho = -1;

% Generation of the Gaussian sample with the chosen mean and standard ...
    deviation
X1 = randn(N,1)*sig1 + mu1;

% Generation of a Log Normal sample based on the Gaussian one
Z1 = exp(X1);

% Confront theoretical and computed means
TheoreticalSampleMeanZ1 = exp(mu1 + sig1^2/2)
ComputedSampleMeanZ1 = mean(Z1)

% Confront theoritcal and computed variances
TheoreticalSampleVarianceZ1 = exp(2*mu1 + 2*sig1^2) - exp(2*mu1 + sig1^2)
ComputedSampleVarianceZ1 = var(Z1)

% Sorting the first sample by ascending order
SortedZ1 = sort(Z1,"ascend");

% Generation of the Gaussian sample with the chosen mean and standard ...
    deviation
X2 = randn(N,1)*sig2 + mu2;
```

```
34
35 % Generation of a Log Normal sample based on the Gaussian one
36 Z2 = exp(X2);
37
38 % Confront theoretical and computed means
39 TheoreticalSampleMeanZ2 = exp(mu2 + sig2^2/2)
40 ComputedSampleMeanZ2 = mean(Z2)
41
42 % Confront theoritcal and computed variances
43 TheoreticalSampleVarianceZ2 = exp(2*mu2 + 2*sig2^2) - exp(2*mu2 + sig2^2)
44 ComputedSampleVarianceZ2 = var(Z2)
45
46 % Sorting the second sample by descending order
47 SortedZ2 = sort(Z2,"descend");
48
49 % Computing the product of the two samples
50 Z = SortedZ1.*SortedZ2;
51
52 % Computing theoretical mean and variance of the product of the underlying
53 % Gaussians
54 VarGauss = sig1^2 + sig2^2 + 2*sig1*sig2*rho;
55 MeanGauss = mu1 + mu2;
56
57 % Confront theoretical and computed means
58 TheoreticalSampleMeanZ = exp(MeanGauss + VarGauss/2)
59 ComputedSampleMeanZ = mean(Z)
60
61 % Confront theoretical and computed variances
62 TheoreticalSampleVarianceZ = exp(2*MeanGauss + 2*VarGauss) - ...
       exp(2*MeanGauss + VarGauss)
63 ComputedSampleVarianceZ = var(Z)
64
65 % Fit of the Produt LogNormal
66 parfit = lognfit(Z);
67
68 % Further verification: computation of the LogNormal generated with the
69 % lognfit function parameters
70 Xfit = randn(N,1)*parfit(2) + parfit(1);
71 Zfit = exp(Xfit);
72 FitComputedMean = mean(Zfit)
73 FitComputedVar = var(Zfit)
```

In addition to that, the results obtained with the presented script are reported in the following table, with 4 different values for the numerosity of the sample and initial means $\mu_1 = \mu_2 = 1$ and variances $\sigma_1 = 0.2$, $\sigma_2 = 1$, and thus having:

$$\mu_{product,theoretical} = 10.1757 \tag{14}$$

$$\sigma_{product,theoretical} = 92.8255 \tag{15}$$

Table 1: Table of results for the simple case of the product of finite samples distributed according to LogNormals PDFs.

NumerosityOfSample	ComputedMean	ComputedVariance
10^5	10.1883	92.5199
10^6	10.1895	92.7258
10^7	10.1721	92.7159
10^8	10.1762	92.7228

As it's easy to see, in this simple case the results are very accurate. It's also worth noting that this was the only case in which a fit of the generated distribution was possible, in order to verify that its parameters were the ones theoretically expected. The parameters obtained with 10^8 sample size are $\mu = 1.9999$ and $\sigma = 0.8000$, matching the theory perfectly.

3.2 BANs Gaussians (finite, equal σ)

This time, BANs are used to define means, variances and to generate the samples. Only the first 2 terms are taken. The goal is to verify that, keeping the variances finite (the term in η is 0), the sum of two BANs Gaussians with the same mean and variances, but completely anti correlated ($\rho = -1$) has $\sigma = 0 + 0\eta$. The script goes as follows:

```
% Number of elements of the two samples of random generated numbers
NumerosityOfSample = 1e5;

% Means of the two samples
mu1 = Ban([100 100], 0);
mu2 = Ban([100 100], 0);

% Standard Deviations of the two samples
sig1 = Ban([10 0], 0);
sig2 = Ban([10 0], 0);

% Correlation coefficient of the two samples
rho = -1;

```

```
% Generation of the Gaussian sample with the chosen mean and standard ...
    deviation
X1 = BanArray(randn(NumerosityOfSample,2)) * sig1 + mu1;

% Sorting the first sample by ascending order
SortedX1 = sort(X1,"ascend");

% Generation of the Gaussian sample with the chosen mean and standard ...
    deviation
X2 = BanArray(randn(NumerosityOfSample,2)) * sig2 + mu2;

% Sorting the second sample by descending order
SortedX2 = sort(X2,"descend");

% Computing the sum of the two samples (this time the Gaussians are
% considered, so the sum is the one to look at)
X = SortedX1 + SortedX2;

% Confront theoretical and computed means
TheoreticalSampleMeanX = mu1 + mu2
ComputedSampleMeanX = mean(X)

% Confront theoretical and computed variances
TheoreticalSampleVarianceX = sig1*sig1 + sig2*sig2 + sig1*sig2*rho*2
ComputedSampleVarianceX = var(X)
```

Notice that, this time, the sum is taken into consideration instead of the product: that's because the PDFs considered are Gaussians, which are summed when working with the respective LogNormals product. It's also worth noticing that, this time, just 3 different sample numerosity were considered, because the computation past 10^5 values becomes too slow, and 10^2 was considered to be too low. The chosen means are $\mu_1 = \mu_2 = 100 + 100\eta$, the chosen variances are $\sigma_1 = \sigma 2 = 10 + 0\eta$. The associated theoretical values of the sum of the two are:

$$\mu_{sum,theoretical} = 200 + 200\eta \tag{16}$$

$$\sigma_{sum,theoretical} = 0 + 0\eta \tag{17}$$

It's worth noticing that, while in the previous case the smallest samples was already good enough to see an almost perfect correspondence between computation and theory, this time it's true only for the mean, while the

Table 2: Table of results for the case of the sum BANs samples distributed according to Gaussian PDFs, with the same mean and equal, finite variances.

NumerosityOfSample	ComputedMean	ComputedVariance
10^3	199.933 $+200.136\eta$	0.720222 $+0.244348\eta$
10^4	199.869 $+199.906\eta$	0.0413276 -0.106428η
10^5	199.98 $+200.056\eta$	0.00416872 $+0.00112645\eta$

variance goes close to the theoretical value enough only with the largest sample size (10^5).

3.3 BANs Gaussians (equal finite part of σ, different η coefficient)

This case makes a little step further with respect to the last one: variances with the same finite (small) part are considered, but with different first order coefficients (the term in η):

$$\sigma_1 = a + b\eta \tag{18}$$

$$\sigma_2 = a + c\eta \tag{19}$$

As discussed in Section 1.3.1, the variance of the sum of the two perfectly anti correlated BANs Gaussians is expected to be $(b-c)^2\eta^2$ (Equation 10). Since the second order term is expected to be very small, and the computation is far from perfect, it's necessary to choose a small finite term, in order to avoid the noise generated from the non perfect cancellation between the terms in a; thus, again the means chosen were $\mu_1 = \mu_2 = 100 + 100\eta$, and the variances: $\sigma_1 = 10^{-3} + 80\eta$, $\sigma_2 = 10^{-3} + 20\eta$. The relative script is:

```
1 % Number of elements of the two samples of random generated numbers
2 NumerosityOfSample = 1e4;
3
4 % Means of the two samples
```

```
mu1 = Ban([100 100], 0);
mu2 = Ban([100 100], 0);

% Standard Deviations of the two samples
sig1 = Ban([1e-3 80], 0);
sig2 = Ban([1e-3 20], 0);

% Correlation coefficient of the two samples
rho = -1;

% Generation of the Gaussian sample with the chosen mean and standard ...
    deviation
X1 = BanArray(randn(NumerosityOfSample,3)) * sig1 + mu1;

% Sorting the first sample by ascending order
SortedX1 = sort(X1,"ascend");

% Generation of the Gaussian sample with the chosen mean and standard ...
    deviation
X2 = BanArray(randn(NumerosityOfSample,3)) * sig2 + mu2;

% Sorting the second sample by descending order
SortedX2 = sort(X2,"descend");

% Computing the sum of the two samples (this time the Gaussians are
% considered, so the sum is the one to look at)
X = SortedX1 + SortedX2;

% Confront theoretical and computed means
TheoreticalSampleMeanX = mu1 + mu2
ComputedSampleMeanX = mean(X)

% Confront theoretical and computed variances
TheoreticalSampleVarianceX = sig1*sig1 + sig2*sig2 + sig1*sig2*rho*2
ComputedSampleVarianceX = var(X)
```

The theoretical values for mean and variance of the sum are:

$$\mu_{sum,theoretical} = 200 + 200\eta + 0\eta^2 \tag{20}$$

$$\sigma_{sum,theoretical} = 0 + 0\eta + 3600\eta^2 \tag{21}$$

The same considerations made in the previous section for the numerosity of samples holds true here.

Table 3: Table of results for the case of the sum BANs samples distributed according to Gaussian PDFs, with the same mean and equal finite part of the variance, but different η coefficient.

NumerosityOfSample	ComputedMean	ComputedVariance
10^3	200 +200.183η -3.66413η^2	4.40355 $10^{-9} - 0.00375672\eta$ $+3298.9\eta^2$
10^4	200 +199.625η +0.906362η^2	4.86454 $10^{-10} - 0.00150953\eta$ $+3504.77\eta^2$
10^5	200 +199.908η -0.114771η^2	4.03919 $10^{-11} - 0.000433037\eta$ $+3586.44\eta^2$

As in the previous case, the match between computation and theory converges with the growth of the numerosity of the sample used, becoming acceptable for the first time at 10^5 BANs samples.

3.4 BANs Log Normals (equal finite part of σ, different η coefficient)

In this section, the exponentiation of the samples generated in the previous ones is performed, in order to work with the Log Normal PDFs, which are the final goal of the experiment. In this case, smaller coefficients were chosen, due to the degenerative behavior of the exponential function: $\mu_1 = \mu_2 = 1 + 1\eta$, and the same goes for the variances, where the finite coefficients stay close to 0 for the same reasons exposed above: $\sigma_1 = 10^{-3} + 0.8\eta$, $\sigma_2 = 10^{-3} + 0.2\eta$. The script is shown below:

```
% Number of elements of the two samples of random generated numbers
NumerosityOfSample = 1e5;

% Means of the two samples
mu1 = Ban([1 1], 0);
mu2 = Ban([1 1], 0);

% Standard Deviations of the two samples
```

```
sig1 = Ban([1e-3 0.8], 0);
sig2 = Ban([1e-3 0.2], 0);

% Correlation coefficient of the two samples
rho = -1;

% Generation of the Gaussian sample with the chosen mean and standard ...
    deviation
X1 = BanArray(randn(NumerosityOfSample,3)) * sig1 + mu1;

% Generation of a Log Normal sample based on the Gaussian one
Z1 = exp(X1);

% Confront theoretical and computed means
TheoreticalSampleMeanZ1 = exp(mu1 + sig1*sig1/2)
ComputedSampleMeanZ1 = mean(Z1)

% Confront theoritcal and computed variances
TheoreticalSampleVarianceZ1 = exp(mu1*2 + sig1*sig1*2) - exp(mu1*2 + ...
    sig1*sig1)
ComputedSampleVarianceZ1 = var(Z1)

% Sorting the first sample by ascending order
SortedZ1 = sort(Z1,"ascend");

% Generation of the Gaussian sample with the chosen mean and standard ...
    deviation
X2 = BanArray(randn(NumerosityOfSample,3)) * sig2 + mu2;

% Generation of a Log Normal sample based on the Gaussian one
Z2 = exp(X2);

% Confront theoretical and computed means
TheoreticalSampleMeanZ2 = exp(mu2 + sig2*sig2/2)
ComputedSampleMeanZ2 = mean(Z2)

% Confront theoritcal and computed variances
TheoreticalSampleVarianceZ2 = exp(mu2*2 + sig2*sig2*2) - exp(mu2*2 + ...
    sig2*sig2)
ComputedSampleVarianceZ2 = var(Z2)

% Sorting the second sample by descending order
SortedZ2 = sort(Z2,"descend");

% Computing the product of the two samples
Z = SortedZ1*SortedZ2;

% Computing theoretical mean and variance of the product of the underlying
% Gaussians
VarGauss = sig1*sig1 + sig2*sig2 + sig1*sig2*rho*2;
MeanGauss = mu1 + mu2;

% Confront theoretical and computed means
TheoreticalSampleMeanZ = exp(MeanGauss + VarGauss/2)
```

```
59 ComputedSampleMeanZ = mean(Z)
60
61 % Confront theoretical and computed variances
62 TheoreticalSampleVarianceZ = exp(MeanGauss*2 + VarGauss*2) - ...
       exp(MeanGauss*2 + VarGauss)
63 ComputedSampleVarianceZ = var(Z)
```

Given the input parameters, the theoretical expected values for mean and variance of the product log normal PDF are:

$$\mu_{product,theoretical} = 7.3890 + 14.7781\eta + 16.1081\eta^2 \tag{22}$$

$$\sigma_{product,theoretical} = 0 + 0\eta + 19.6553\eta^2 \tag{23}$$

Again, as always with BANs, the numerosity of the samples must be limited.

Table 4: Table of results for the case of the product BANs samples distributed according to Log Normal PDFs, with the same mean and equal finite part of the variance, but different η coefficient.

NumerosityOfSample	ComputedMean	ComputedVariance
10^3	$7.38915 + 14.8773\eta + 16.6294\eta^2$	$2.12328\ 10^{-9} + 0.00259586\eta + 20.6137\eta^2$
10^4	$7.38905 + 14.8353\eta + 16.2424\eta^2$	$1.0798810^{-8} - 7.0673610^{-5}\eta + 19.4986\eta^2$
10^5	$7.38902 + 14.7417\eta + 16.0663\eta^2$	$1.62759\ 10^{-9} - 6.4263410^{-5}\eta + 19.7008\eta^2$

This time the results are incredibly close to the theoretical ones, and that displays the effectiveness of the BAN exponentiation function discussed in Section 2.3.

3.5 BANs Gaussians (equal infinite part of σ, different η coefficient)

Finally, let's consider an infinite mean and variance case (α's exponent = 1). As already mentioned, only the Gaussian case returned meaningful results, as the exponentiation function fails to correctly implement the infinite case. For this last experiment, the chosen parameters were $\mu_1 = \mu_2 = (100 + 100\eta)\alpha$, and the variances: $\sigma_1 = (1 + 80\eta)\alpha$, $\sigma_2 = (1 + 20\eta)\alpha$. The considerations on the size of the finite term don't hold for this case, because there isn't an infinitesimal part (the first term is infinite of the first order in α, the second one is finite). The script is the following:

```
% Number of elements of the two samples of random generated numbers
NumerosityOfSample = 1e3;

% Means of the two samples
mu1 = Ban([100 100], 1);
mu2 = Ban([100 100], 1);

% Standard Deviations of the two samples
sig1 = Ban([1 80], 1);
sig2 = Ban([1 20], 1);

% Correlation coefficient of the two samples
rho = -1;

% Generation of the Gaussian sample with the chosen mean and standard ...
    deviation
X1 = BanArray(randn(NumerosityOfSample,3)) * sig1 + mu1;

% Sorting the first sample by ascending order
SortedX1 = sort(X1,"ascend");

% Generation of the Gaussian sample with the chosen mean and standard ...
    deviation
X2 = BanArray(randn(NumerosityOfSample,3)) * sig2 + mu2;

% Sorting the second sample by descending order
SortedX2 = sort(X2,"descend");

% Computing the sum of the two samples (this time the Gaussians are
% considered, so the sum is the one to look at)
X = SortedX1 + SortedX2;

% Confront theoretical and computed means
TheoreticalSampleMeanX = mu1 + mu2
ComputedSampleMeanX = mean(X)

```

```
% Confront theoretical and computed variances
TheoreticalSampleVarianceX = sig1*sig1 + sig2*sig2 + sig1*sig2*rho*2
ComputedSampleVarianceX = var(X)
```

The theoretical values for mean and variance of the sum are:

$$\mu_{sum,theoretical} = (200 + 200\eta + 0\eta^2)\alpha \tag{24}$$

$$\sigma_{sum,theoretical} = 3600 + 0\eta + 0\eta^2 \tag{25}$$

As anticipated in 13, this time the expected variance is finite.

Table 5: Table of results for the case of the sum BANs samples distributed according to Gaussian PDFs, with the same mean and equal infinite part of the variance, but different η coefficient.

NumerosityOfSample	ComputedMean	ComputedVariance
10^3	$(200.053 + 201.899\eta - 2.48293\eta^2)\alpha$	$(0.00447257 + 1.32653\eta + 3798.93\eta^2)\alpha^2$
10^4	$(200.013 + 200.406\eta + 0.035365\eta^2)\alpha$	$(0.000229686 + 0.821994\eta + 3672.7\eta^2)\alpha^2$
10^5	$(200.009 + 200.233\eta + 0.218476\eta^2)\alpha$	$(5.72684 \; 10^{-5} - 0.42596\eta + 3572.24\eta^2)\alpha^2$

Again, a good match between theory and simulation is displayed, even in the infinite case.

4 Conclusions

All of the proposed experiments were successful, in the sense that simulations and theory manifested a good accord across all of the cases exposed. These results are a further confirmation of the effectiveness of the non standard approach toward statistical analysis, since everything turned out to work exactly as expected, and the hope is that they will help to better treat the rare events (out liars). Talking about the infinite variance Log Normal case, explicit simulations weren't possible for the reasons reported, but the Gaussian simulations in the infinite variance case implicitly prove the fact that constructing such a log normal would be possible. In the future, it would be interesting to try and implement a BAN exponentiation function that better approximates the infinite case.