Gran Sasso Science Institute
XXVI Cycle - A.Y. 2020-2021

LE-6 Exam Report

Monte Carlo Techniques

Stefano Ghislandi

Preface

This report presents the exam exercises solutions of the course "LE-6 Monte Carlo Techniques" for the academic year 2020/2021. C++ programming language together with `ROOT` framework classes have been employed. The codes, written to obtain the results presented throughout the document, are reported in a public GitHub repository at the link `https://github.com/sghislandi/GSSI_LE-6_Exam`.

1 Uniform Random Sampling

I implemented a simple pseudo-random number generator as follows:

$$n_i = n_{i-1} \cdot 663608941 \qquad \text{where} \qquad n_0 = 987654321. \tag{1}$$

It is a Linear Congruential Generator (LCG) with only the multiplicative term. The modulus operation is automatically done, since the stored number has a fixed type with limited memory space. In this case the variable n_i has been defined as an `unsigned int`, thus it occupies 32 bits in memory. This also means that the maximum period of this LCG is $2^{32} - 1$[1].

Chosen the seed and the multiplicative number as in Equation (1), the total length of the sequence is $1073741824 = 2^{30}$. The whole cycle requires $\sim$0.92 s of CPU time to return back to the seed value.

Now, the output has been normalized so that the LCG generates pseudo-random numbers between 0 and 1. The obtained distribution, before the sequence returns to the initial seed, have been plotted in Figure 1.

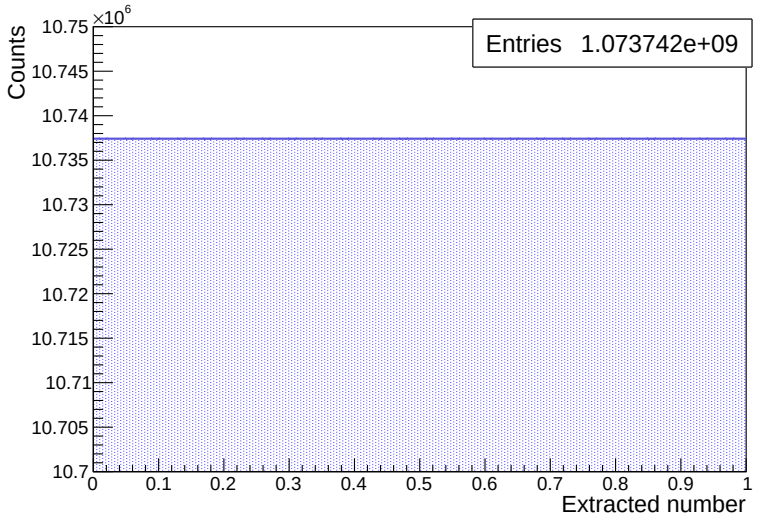

Figure 1: Distribution obtained with the whole sequence.

On the contrary, in Figure 2, I reported the histograms filled respectively with the first 10^3 and 10^6 extracted numbers of the LCG.

In order to verify the uniformity of these extractions I applied a χ^2 test; the results are reported in Table 1. The uniformity over a whole cycle of the specific pseudo-random

Table 1: χ^2 test results.

Histogram	$\tilde{\chi}^2$	p-value
All entries	$1.4 \cdot 10^{-6}$	1
10^3 entries	0.69	0.95
10^6 entries	0.91	0.69

generator defined in Equation (1) is confirmed by the test, whose p-value is practically 1.

[1] This is equivalent to an extraction of every single number of the sequence without repetitions.

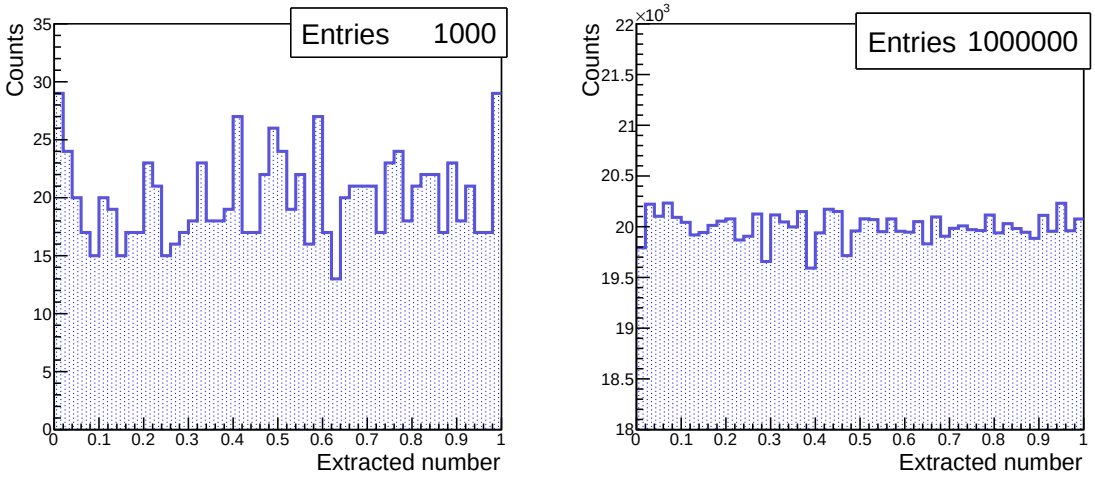

Figure 2: Distributions obtained with 10^3 events (*left*) and 10^6 events (*right*).

For the 10^3 and 10^6 extraction the $\tilde{\chi}^2$ is close to 1 and so, also in this case, the uniformity is checked.

1.1 MINSTD algorithm

I implemented the MINSTD generator defined as:

$$n_i = 7^5 \cdot n_{i-1} \mod (2^{31} - 1). \tag{2}$$

In this case the sequence length coincide with the maximum allowed one which is $2147483646 = 2^{31} - 2$. It can also be written as $m - 1$ where m is the divisor of the modulo operation. The obtained period doesn't depend on the chosen seed. The output sequence is reported in the histogram in Figure 3. A χ^2 test has been performed in this case and it confirms that the MINSTD extraction is uniform.

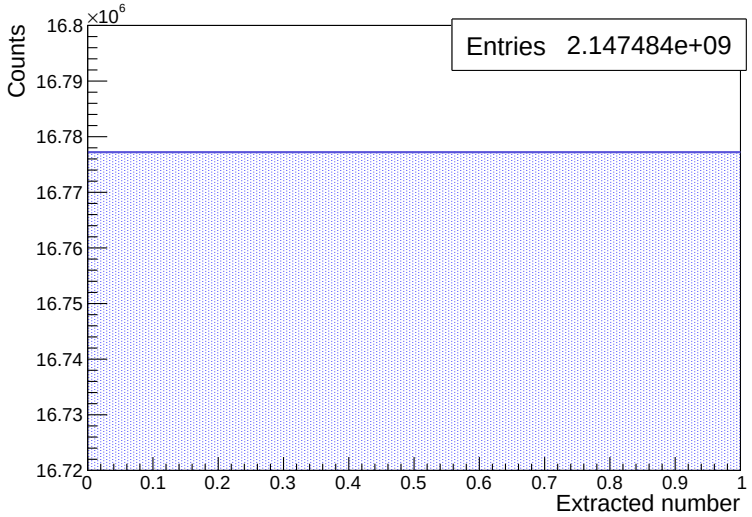

Figure 3: Distribution obtained with the whole sequence.

2 Random Sampling from other distributions

2.1 Inverse method

I want to extract pseudo-random numbers coming from the small-angle Rutherford distribution

$$p(x) = \frac{2x}{(1+x^2)^2} \qquad 0 \leq x < \infty. \tag{3}$$

In order to do that, I have to extract pseudo-random numbers in the interval $[0, 1]$ and apply to them the inverse cumulative function of $p(x)$ which I call $P^{-1}(x)$. The cumulative function is:

$$P(x) = \int_0^x \mathrm{d}x' p(x') = \frac{x^2}{1+x^2}. \tag{4}$$

The inverse cumulative function is:

$$P^{-1}(x) = \sqrt{\frac{x}{1-x}}. \tag{5}$$

In order to check the method, I extracted 10^6 numbers $x_i \in [0, 1]$ and I computed $P^{-1}(x_i)$. In the end, I fitted the distribution with the expected Rutherford formula; the result is reported in Figure 4.

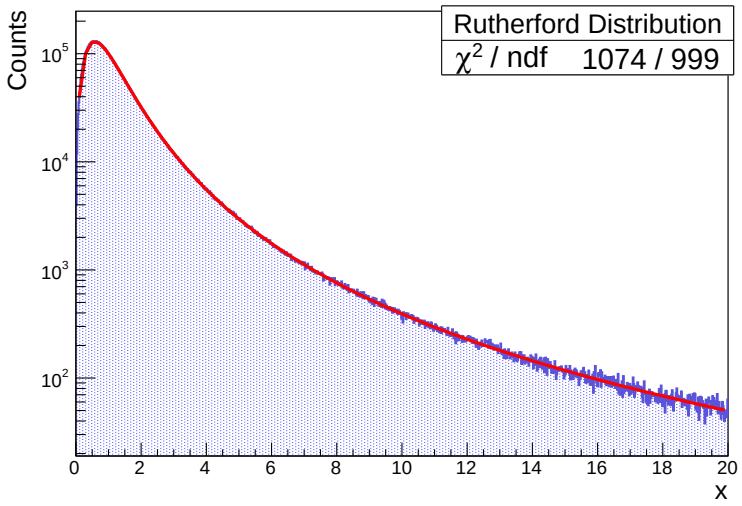

Figure 4: Pseudo-random values extracted from the Rutherford distribution.

2.2 Inversion and rejection

I perform a uniform sampling (x, y) from a unit circle implementing both the analytical and the rejection methods. For the former approach, I just extract pseudo-random numbers R_1 and R_2 and I directly compute the 2D point (x, y) as:

$$\begin{cases} x &= \sqrt{R_1}\cos(2\pi R_2) \\ y &= \sqrt{R_1}\sin(2\pi R_2). \end{cases} \tag{6}$$

For the rejection method I extract the pseudo-random numbers x and y in the square $[0, 1] \times [0, 1]$ and I keep the 2D point (x, y) if $x^2 + y^2 \leq 1$. In both cases I require the result to be composed of 10^8 samples into the unit circle.

The inversion method is surely more efficient than the rejection one for which a fraction of the extracted points are loss because they fall out of the circle. However, the first method makes use of transcendental functions (sine and cosine) which are very slow to be computed if the compilation is not optimized. In this specific case the `c++` compiler has been used. In a first compilation, the "*default*" one, no specific options have been added in the command line. For the "*optimized*" case the option `-Ofast` has been used and it selects a combination of compilation characteristics for optimum execution time. This is confirmed by the CPU time needed for the extraction, reported in Table 2. When the compiler can manage with fast sine and cosine computation the inversion method is faster, exploiting its full efficiency.

Table 2: CPU time required to uniformly extract 10^8 points in a unit circle using the inversion and the rejection methods.

Compiler option	Inversion CPU time	Rejection CPU time
Default	$\sim$5.3 s	$\sim$2.7 s
Optimized	$\sim$1.3 s	$\sim$1.9 s

3 Numerical estimate of π

In order to estimate the value of π with an Hit or Miss Monte Carlo I shot points (x, y) uniformly distributed in $[-1, 1] \times [-1, 1]$, counting the fraction of them falling into the unit circle and dividing the sum by the square area. As an example I extracted 100 couples showing them in Figure 5.

Fixing at 10^5 the total number of point extractions[2] I obtain the value $\pi_{MC} = 3.1416$. Computing for every step the absolute difference with the expected value I obtain the graph in Figure 6. The deviation from the real π goes to zero in a stable way, so the convergence is ok.

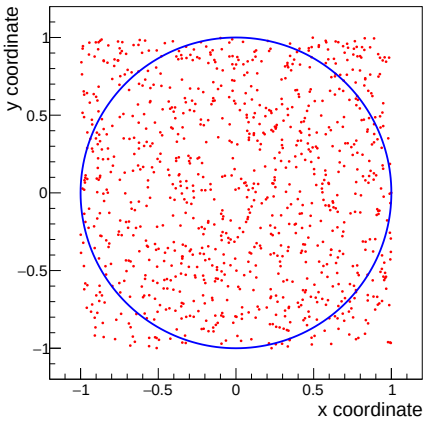

Figure 5: Example of extraction of (x, y) for the unit circle Hit or Miss.

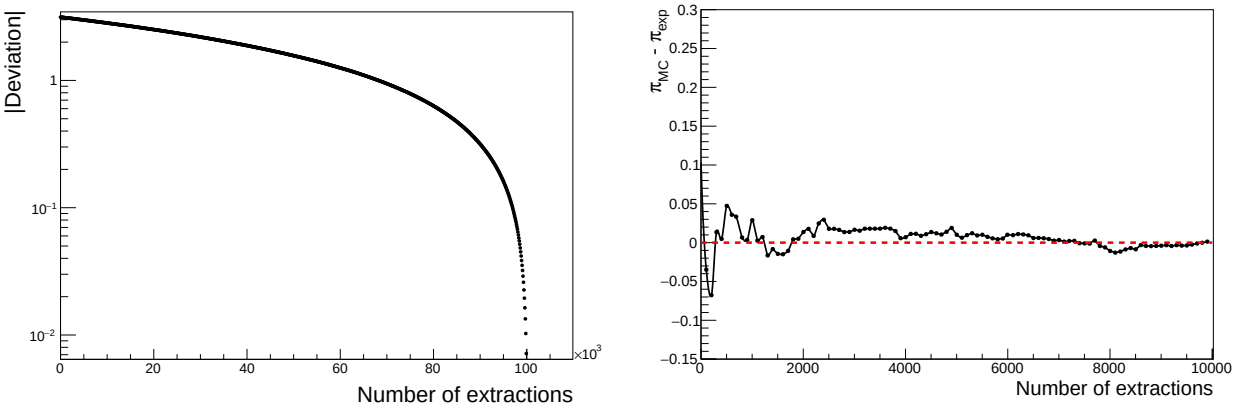

Figure 6: On the *left* the absolute value of the difference between the expected π value and the estimated one. On the *right* the same deviation step by step without the absolute value.

3.1 Uncertainty evaluation

In order to compute the uncertainty on the estimated π, I repeat $N_{evaluation} = 10^5$ times the simulation extracting every time $N = 100$ (x, y) couples. At each evaluation the sequence

[2] 10^5 is the number of generated couples (x, y) and not the total number of points fell into the circle.

seed is extracted with a random generator in order to maintain the independence among the π Monte Carlo values. Thanks to this feature, for $N_{evaluation}$ large, the distribution, shown in Figure 7, becomes "gaussian".

The standard deviation of the distribution scales as:

$$\sigma = \frac{k}{\sqrt{N}} \qquad \longrightarrow \qquad k = \sqrt{N} \cdot \sigma; \tag{7}$$

thus it is possible to extract the standard deviation characteristic parameter k. In the case of Figure 7, $\sigma = 0.1648$ for $N = 100$ and so $k = 1.648$.

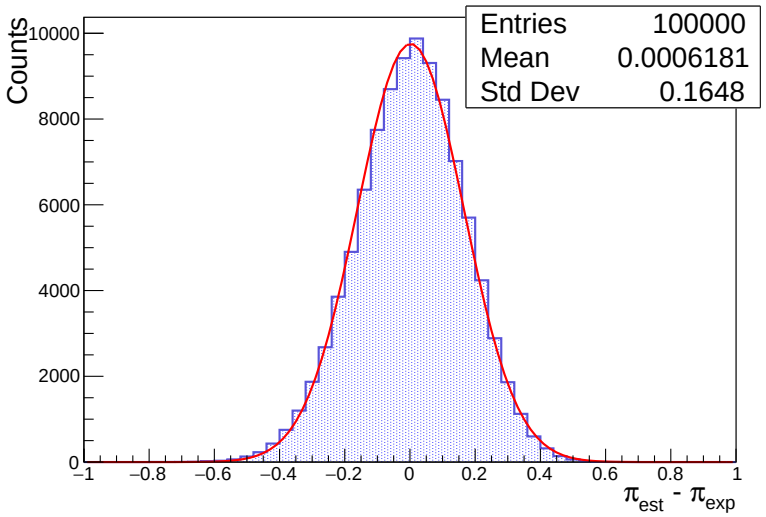

Figure 7: Distribution of the π value simulated through the Hit or Miss Monte Carlo. In order to compare it with the exact value the latter has been subtracted.

Now it is possible to check the dependence foreseen in Equation 7 by repeating the 10^5 evaluations of π but extracting $N = 10^3$ and $N = 5 \cdot 10^3$ (x, y) couples. The result is shown in Figure 8 together with a red line[3] representing the expected behavior of Equation 7.

Eventually, it is also possible to predict the number of extractions N for which I would have a π value estimation with an error $\sigma \sim 10^{-4}$. Using $k = 1.648$ I can find:

$$N = (\frac{k}{\sigma})^2 = (\frac{1.648}{10^{-4}})^2 \sim 2.5 \cdot 10^8 \tag{8}$$

[3]In this case the parameter k has been fixed to the value reported before.

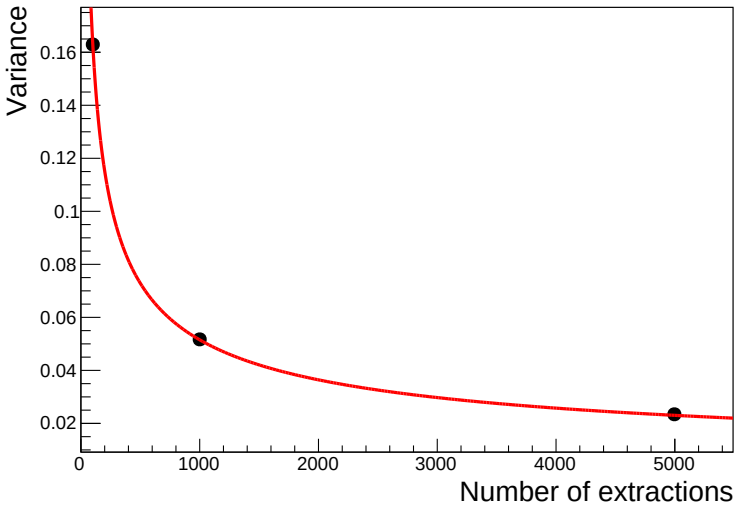

Figure 8: The three black points report the obtained standard deviation with $N = 100$, 1000 and 5000. The red line is obtained by plotting the σ of Equation 7 with $k = 1.648$ found for the estimaton with $N = 100$.

4 A toy Monte Carlo, RisiKo!

The main aim is to implement the basic rules of the RisiKo game and exploit them to produce Monte Carlo simulation.

4.1 Basic probability

First of all I define the functions `rollDice` and `round`. The former extracts a random number from 1 to 6, simulating a rolling dice. The latter makes the attacker and the defender roll 3 times the dice, it compares their outcomes, registers the results and returns the number of armies won by the attacker. I simulate 10^5 rounds and the obtained distribution of armies won by the attacker is show in Figure 9.

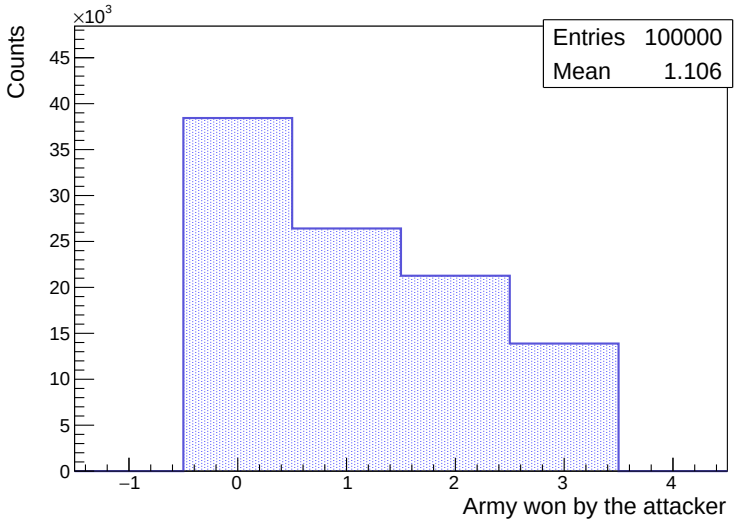

Figure 9: Histogram of the armies won by the attacker per round. The y axis values are scaled such that it's easy to read a probability from them.

The average number of armies won by the attacker each round is A_{won}^{round} 1.106. The attacker rolls 3 dices every round so the average number per single dice is:

$$\langle A_{won}^{dice} \rangle = \frac{\langle A_{won}^{round} \rangle}{3} \sim 0.369. \tag{9}$$

Since the defender also wins when there is a draw, the $\langle A_{won}^{round} \rangle < 0.5$.

4.2 Planning an attack

Now I add the function `conquerAttempt`. It simulates a conquer attempt of the attacker who goes on attacking until the defender loses all his armies (successful conquer) or the attacker do the same (unsuccessful conquer). In this case the `round` function has been modified with additive rules[4]. I fix the initial number of defender armies $N_D^i = 3$ and I launch 10^5 conquer attempts in order to compute the attacker conquer probability. This has been done for every number of initial attacker armies $N_A^i \in [1, 25]$. The results are shown in Figure 10. The minimum number of armies the attacker needs to have a probability 80% to get the territory is $N_A^i = 8$. This number is just slightly lower than $N_D^i / \langle A_{won}^{dice} \rangle \sim 8.1$.

[4]For example if the defender has $N_D < 3$ armies, he will roll the dice only N_D times.

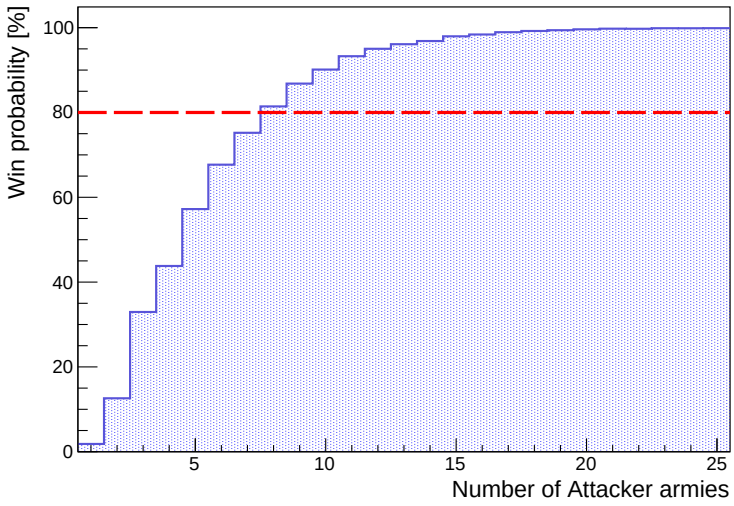

Figure 10: Probability to conquer a territory with $N_D^i = 3$ defender armies while varying the number of attacker armies.

Now I fix $N_A^i = 8$ and I plot in Figure 11 the distribution of remaining attacker armies after the conquer attempt; I call it N_A^f.

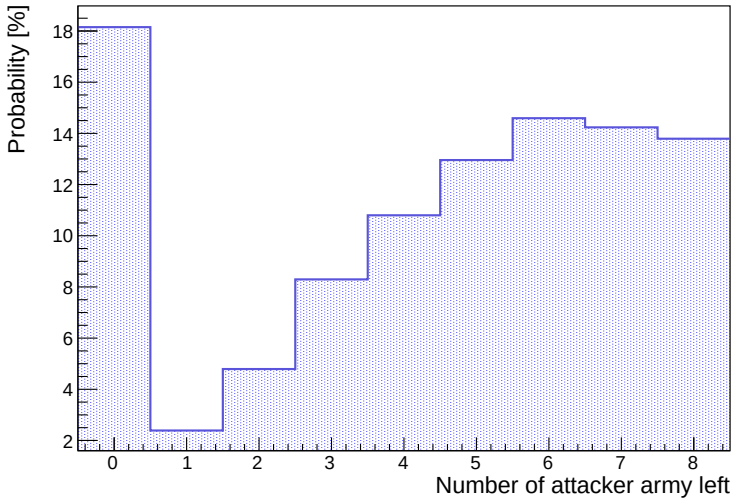

Figure 11: Probability for the attacker to remain with N_A^f armies.

$N_A^f = 0$ means that the attacker lost every single army and he will not get the territory. Since we found before that for $N_A^i = 8$ the probability to win would have been slightly larger than 80%, its complement is exaclty $P(N_A^f = 0)$ which is $\sim 18\%$.

The probability that, after the conquer, the attacker remains with at least 6 armies is the sum of probabilities of $N_A^f \geq 6$. In this case, when the attacker has $N_A^i = 8$ it is:

$$P(N_A^f \geq 6) = \sum_{j=6}^{8} P(N_A^f = j) = 41.5\%. \tag{10}$$

Let's now, with another simulation, retrieve the probabilities that the attacker conquer a territory with at least 6 armies left. The result is shown in Figure 12. Obviously in this case

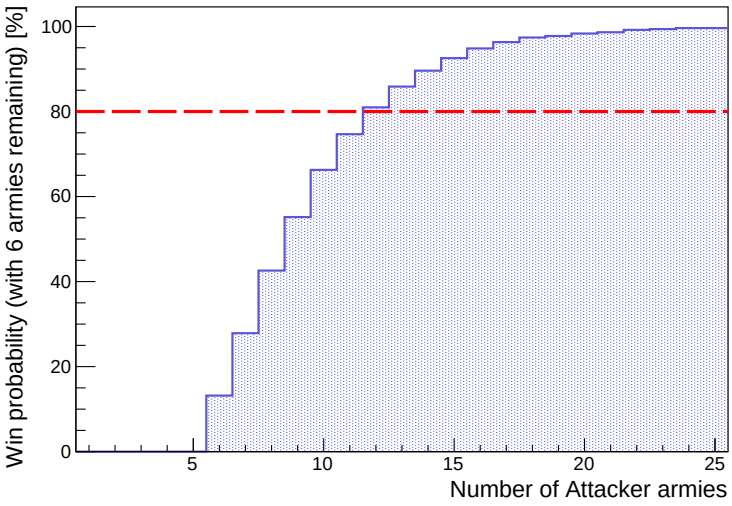

Figure 12: Probability for the attacker to conquer the territory with at least 6 armies left.

the probability that this happens is 0 if the attacker has $N_A^i < 6$. If the required probability is 80%, the attacker has to attack with at least 12 armies.

5 Monte Carlo integration

5.1 Unidimensional integration

I implemented a code to solve the following 1D integral through Monte Carlo integration:

$$I_n = \int_0^1 x^n \, \mathrm{d}x. \tag{11}$$

The expected result is $I_n = \frac{1}{1+n}$ and I compared it with the Monte Carlo output by computing the absolute value of their difference. This has been done for different $N_{extraction}$ using both the Hit or Miss Monte Carlo and computing the arithmetic mean and exploiting the mean integral theorem. The results are plotted in Figure 13.

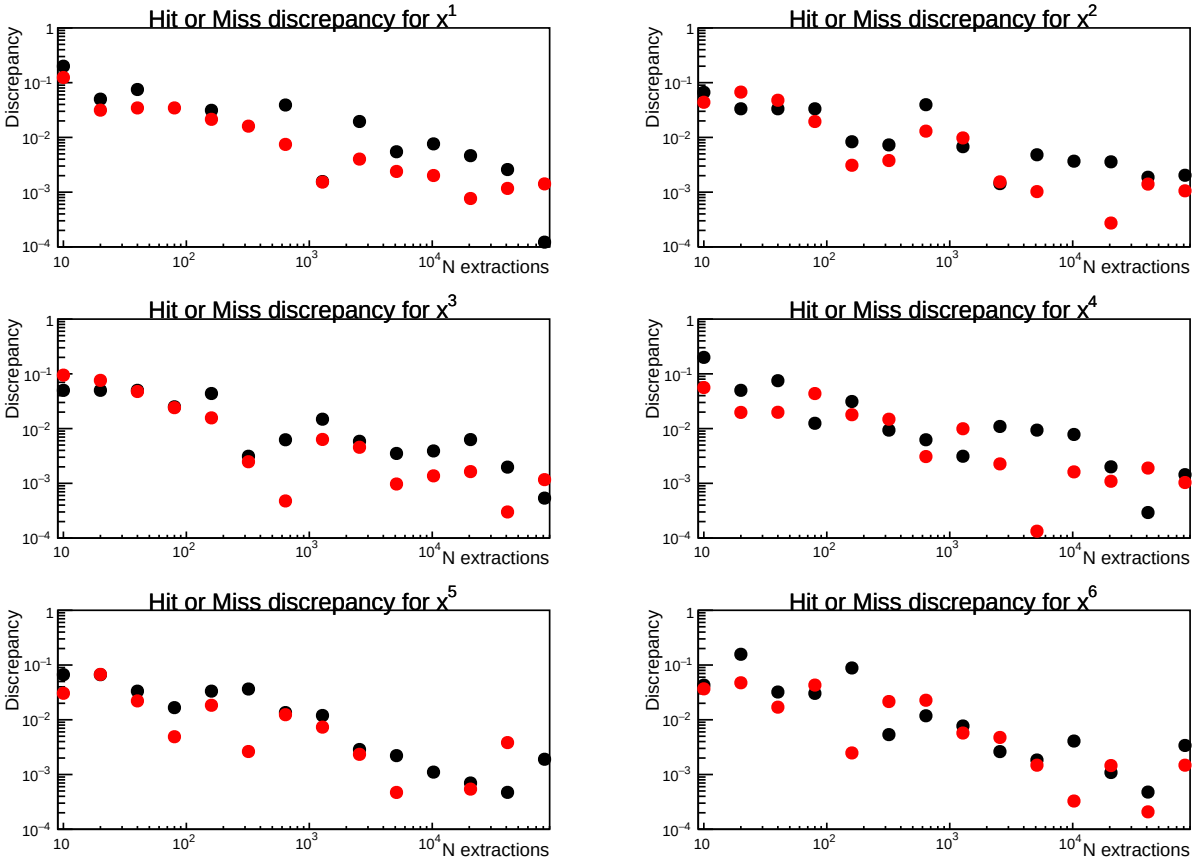

Figure 13: Absolute difference between the Monte Carlo result and I_n for different $N_{extraction}$ and different n. In black the Hit or Miss method, in red the one exploiting the mean integral theorem.

It's evident that the two approaches have approximately the same precision, however the arithmetic mean methods gains a factor 50 on the computation time.

Now I fix the extraction number $N = 10^4$ and I repeat the estimation of I_n $N_{repetition} = 100$ times. In Figure 14 the output is shown. From the central limit theorem I expected that the obtained pdf tends to a Gaussian with variance $\sigma_{MC} = \frac{k}{\sqrt{N}}$, where k depends on the Monte Carlo approach and on the integration interval. Supposing the k term in the order of unity, I need $N \sim 10^5 - 10^6$ extraction in order to get a variance $\sigma_{I_n} \sim 0.001$.

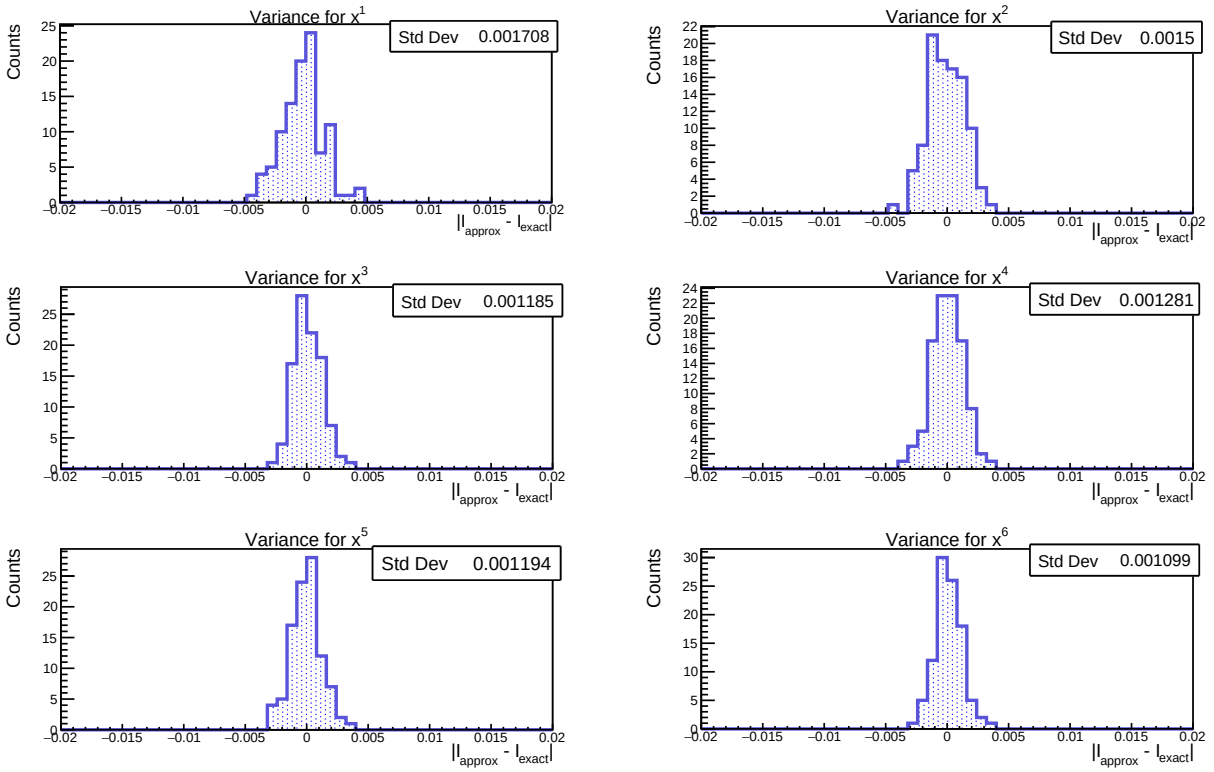

Figure 14: Distribution of I_n estimated repeating 100 times the evaluation of the integral using $N = 10^4$ extraction.

5.2 Integration in N dimensions

Now, I solve using a Monte Carlo and the mid-point summation method the integral:

$$\mathrm{d}x^D I = \prod_{i=1}^{D} \int_0^1 \mathrm{d}x_i e^{-x_i}, \tag{12}$$

where $D \in [1, 8]$ is the dimensionality of the problem. Let's define now N_h as the number of histories of the Monte Carlo and N_c the total number of cells used for the mid-point summation. For the integral evaluation, in order to maintain the same "statistics", I will employ the following table.

D	1	2	3	4	5	6	7	8
N_h	65536	256^2	40^3	16^4	9^5	6^6	5^7	4^8
N_h	65536	65536	64000	65536	59049	46656	78125	65536

I compared the results with the exact value $I_{exact} = (1 - 1/e)^D$ and plotted the absolute value of their differences in Figure 15. The Monte Carlo technique maintains its precision together with the increasing dimensionality. On the other hand, when D is low, the mid-point summation method is way more precise. However, its accuracy gets worse as the dimensionality grows. The time needed by the two methods, considering the numbers in the previous table, are comparable but absolutely negligible being lower than 1 ms.

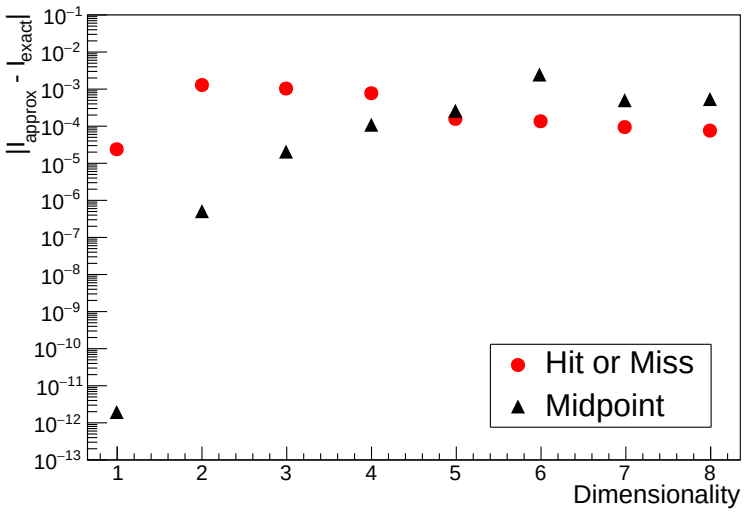

Figure 15: Absolute difference between the Monte Carlo result and the exact integral in Equation 12 for different dimensionalities.

5.3 Extra exercise

The integral in Equation 12, when computed with midpoint summation, could have been done just performing in an independent way all the integrals for fixed $i \in (1, D)$. In the end, the result was just the product of the obtained integrals. Now I want to compute the integral:

$$I = \int_0^1 \int_0^1 ... \int_0^1 (x_1^2 + x_2^2 + ... + x_D^2)\, \mathrm{d}x^D \tag{13}$$

where $D \in [1, 8]$ and the exact result is $I = D/3$. For the Monte Carlo method nothing changes. Regarding the mid-point summation the value of the function $\sum_{i=1}^{D} x_i^2$ is needed in every single cell of the D dimensional cube. In order to implement such a nested `for` loop, I introduced a recursive function. The results of the integral for different dimensionalities and both methods is reported in Figure 16. The same consideration made before about the accuracy and efficiency of the two methods at different D are valid also here.

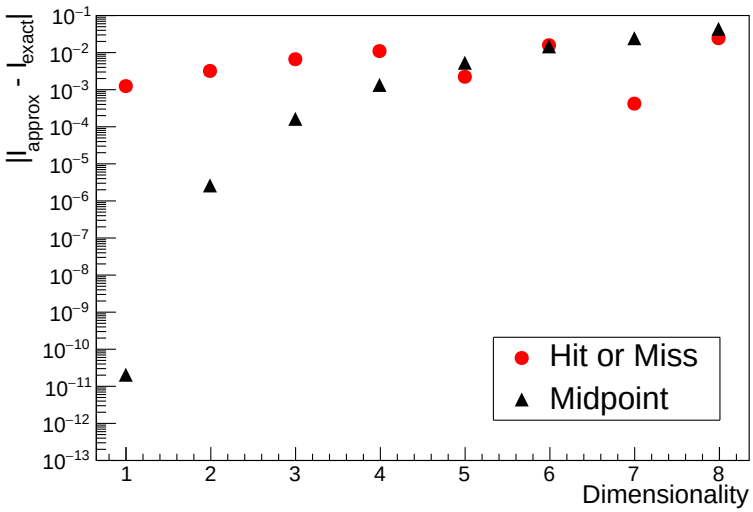

Figure 16: Absolute difference between the Monte Carlo result and the exact integral in Equation 13 for different dimensionalities.

6 Truncation errors

I want to compute the sum:

$$X = \sum_{i=1}^{N} \frac{1}{N} \tag{14}$$

by storing the variable $\frac{1}{N}$ as `float` and `double`. The `float` type occupies 32 bits in memory (1 bit taken by the sign) while the `double` occupies 64 bit (1 bit taken by the sign). I expect that, after a certain N, the number $\frac{1}{N}$ is no more well described by the single precision type and the result X no more agrees with the exact result $X = 1$. The same effect happens also for the double precision values but at an higher N. In order to visualize the different accuracy, I plot the difference between X_{float} and X_{double} in Figure 17.

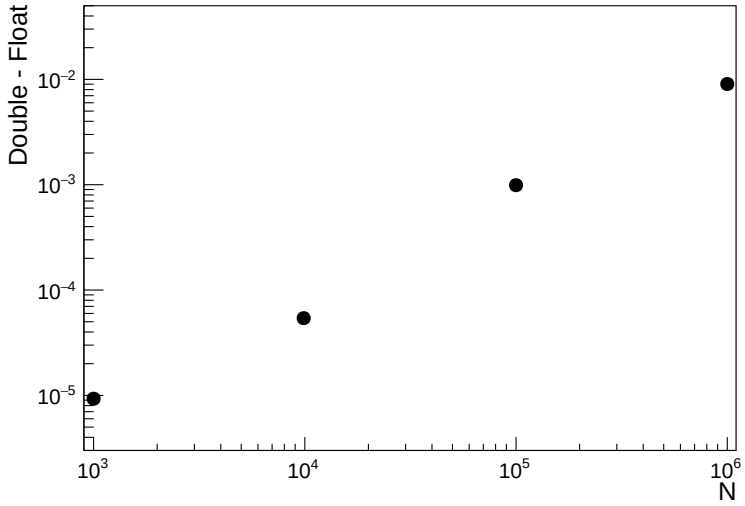

Figure 17: Differences between the results obtained storing $1/N$ as a `float` and as a `double`. For the N's considered the double precision output always agrees with exact one.

7 Tracking algorithms

The sampling of the step length s in a Monte Carlo simulation can be done in two different but equivalent ways. The first method, employed by Geant4, samples the individual steps by the distributions $p(s_i) = \mu_i e^{-\mu_i s_i}$. Then, it selects $s = \min\{s_1, ..., s_N\}$. Equivalently, as Penelope does, it's possible to directly extract s from $\mu e^{-\mu s}$ where $\mu = \sum_i \mu_i$.

In order to prove this equivalence I first apply the Geant4 sampling extracting s_i and then taking the minimum. For simplicity I consider $i \in \{1, 2\}$ and I fix $\mu_1 = 1$ and $\mu_2 = 2$. In the end, I fit the obtained distribution with $p(s) = \mu e^{\mu s}$. The results are reported in Figure 18

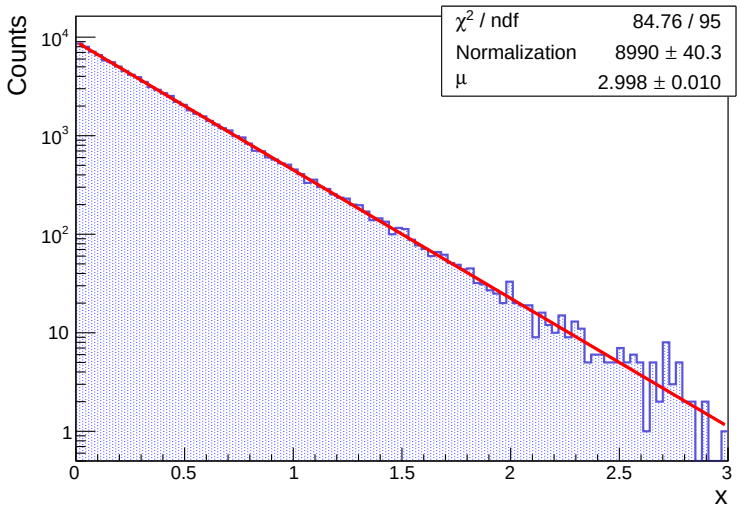

Figure 18: Distribution of s sampled in the Geant4 way.

The value of μ obtained with the fit is compatible with the expected one, that is $\mu = \sum_i \mu_i = 3$. Moreover it is also possible to define the sampling fractions associated to μ_i which are expected to be $\Gamma_i = \frac{\mu_i}{\sum_i \mu_i}$ and so, respectively 1/3 and 2/3. In fact, counting the times s_1 or s_2 have been selected, during the simulations, I obtain 0.33 and 0.66.

The prove that the two sampling approaches are equivalent can also be done theoretically. Let's start taking $i \in \{1, 2\}$ as before and consider their cumulative distribution function.

$$\begin{aligned} P(\min\{s_1, s_2\} < x) &= 1 - P(s_1 > x \wedge s_2 > x) = 1 - P(s_1 > x)P(s_2 > x) = \\ &= 1 - (1 - cdf(s_1)) \cdot (1 - cdf(s_2)) = 1 - e^{-\mu_1 x} \cdot e^{-\mu_2 x} = \\ &= 1 - e^{\mu_1 x} \cdot e^{-\mu_2 x} = 1 - e^{-(\mu_1 + \mu_2)x} \end{aligned} \tag{15}$$

That is exactly $cdf(s)$ and, if I want to retrieve the real $p(s)$, Ijust need to normalize the pdf obtaining:

$$p(\min\{s_1, s_2\}) = (\mu_1 + \mu_2)e^{-(\mu_1+\mu_2)s} \equiv p(s). \tag{16}$$

This can be easily generalized to $i \in \{1, ..., N\}$ with the very same steps. More explicitly, starting again from the cumulative function:

$$\begin{aligned} P(\min\{s_1, ..., s_N\} < x) &= 1 - P(s_1 > x \wedge ... \wedge s_N > x) = 1 - \prod_i P(s_i > x) = \\ &= 1 - \prod_i (1 - cdf(s_i)) = 1 - \prod_i e^{-\mu_i x} = 1 - e^{-\sum_i \mu_i x}. \end{aligned} \tag{17}$$

Also the results of sampling fractions can be proved in a mathematical way. The target is to compute the sampling fraction of s_i. From the previous paragraph I can write:

$$p(\min\{s_1, ..., s_{i-1}, s_{i+1}, ..., s_N\}) \equiv \sum_{j \neq i} \mu_j e^{-\sum_{j \neq i} \mu_j \tilde{s}} = \tilde{\mu} e^{-\tilde{\mu}\tilde{s}}. \tag{18}$$

An extraction $\tilde{s} = \tilde{s}^*$ has probability:

$$P(\tilde{s} \in (\tilde{s}^*, \tilde{s}^* + \mathrm{d}\tilde{s})) = \tilde{\mu} e^{-\tilde{\mu}\tilde{s}^*} \, \mathrm{d}\tilde{s}. \tag{19}$$

I require now that $s_i < \{s_1, ..., s_{i-1}, s_{i+1}, ..., s_N\}$ and this happens with probability:

$$P(s_i < \{s_1, ..., s_{i-1}, s_{i+1}, ..., s_N\}) = P(s_i < \tilde{s}^*) = (1 - e^{\mu_i \tilde{s}^*}). \tag{20}$$

Putting all together and integrating over all possible $\tilde{s}^*$, I obtain the probability that s_i is extracted as minimum of the set $\{s_1, ..., s_N\}$ which is exactly the sampling fraction:

$$\Gamma_i = \int_0^\infty \mathrm{d}\tilde{s}^* \tilde{\mu} e^{-\tilde{\mu}s^*} - \tilde{\mu} e^{-\mu s^*} = -e^{-\tilde{\mu}s^*} + \frac{\tilde{\mu}}{\mu} e^{-\mu s^*} \Big|_0^\infty = 1 - \frac{\tilde{\mu}}{\mu} = \frac{\mu_i}{\mu}. \tag{21}$$

8 Sampling of an interaction

The aim of the exercise is the final state sampling following a photo-electric effect of a γ-ray with $E = 1\,\text{MeV}$ on the K-shell of Pb, whose energy is $U = 88.01\,\text{keV}$.

8.1 Photo-electron

After the absorption of the initial γ a photo-electron is produced and emitted with energy $E_e = E - U$. Its direction is isotropic in ϕ and follows in θ the Sauter's formula:

$$\frac{\mathrm{d}\sigma}{\mathrm{d}\Omega} \propto \frac{\sin^2\theta}{(1-\beta\cos\theta)^4}\left[1 + \frac{1}{2}\gamma(\gamma-1)(\gamma-2)(1-\beta\cos\theta)\right] \tag{22}$$

where the quantities β and γ are:

$$\beta = \frac{\sqrt{E_e(E_e + 2m_2c^2)}}{E_e + m_e c^2} \sim 0.93 \quad \text{and} \quad \gamma = 1 + \frac{E_e}{m_e c^2} \sim 2.78. \tag{23}$$

In this case, θ and ϕ are the polar and azimuthal angles with respect to the reference frame defined by the arrival direction of the incoming photon.

Suppose now that the γ is traveling along the z-axis, $(0, 0, 1)$. I simulate 10^4 photo-electrons and the resulting distribution in θ and ϕ are plotted in Figure 19 and Figure 20. I employed the hit or miss method extracting $\theta \in [0, \pi]$ and the relative y coordinate $\in [0, 800]$, since the maximum in θ of Equation (22) is around 720. The associated efficiency can be found dividing the area of the extraction rectangle and the integral of the function obtaining an efficiency $\epsilon \sim 11\%$.

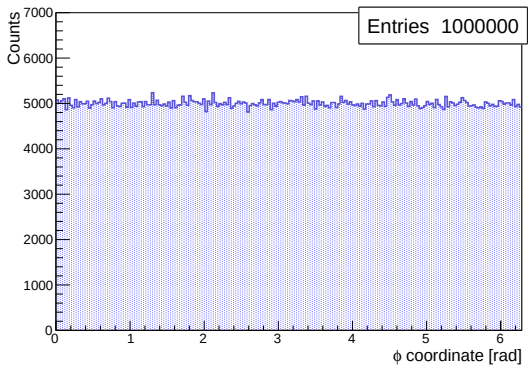

Figure 19: ϕ uniform distribution in the interval $[0, 2\pi]$.

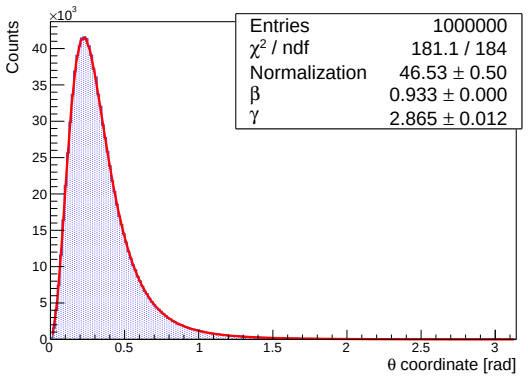

Figure 20: θ distribution sampled with the Sauter formula in the interval $[0, \pi]$.

In order to better visualize what would happen in space, I put together the two coordinates and I project the photo-electron directions on a unitary sphere as in Figure 21.

When the photon travels along the y-axis I extract the θ and ϕ coordinate in the same way and, then, I apply a rotation to restore the reference system of the γ. The result is shown in Figure 22.

8.2 Fluorescence

The remaining U energy is released by emitting two γ's. The first one is a K_α x-ray due to the vacancy movement from the K-shell to the L-shell. The latter has three levels (L_I,

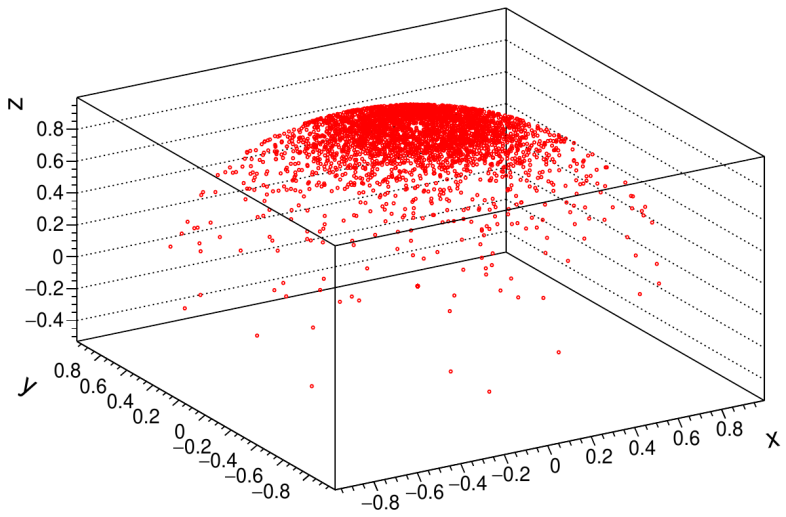

Figure 21: Distribution in space of the photo-electrons emitted by a γ in $(0, 0, 1)$ direction.

L_{II} and L_{III}) which I extract weighting for their electron occupancy, respectively 2, 2 and 4. The second γ is released with all the remaining energy. The photons directions are sampled isotropically thus extracting uniformly ϕ and $\cos\theta$. The energy spectrum is plotted in Figure 23 and the direction distribution in Figure 24.

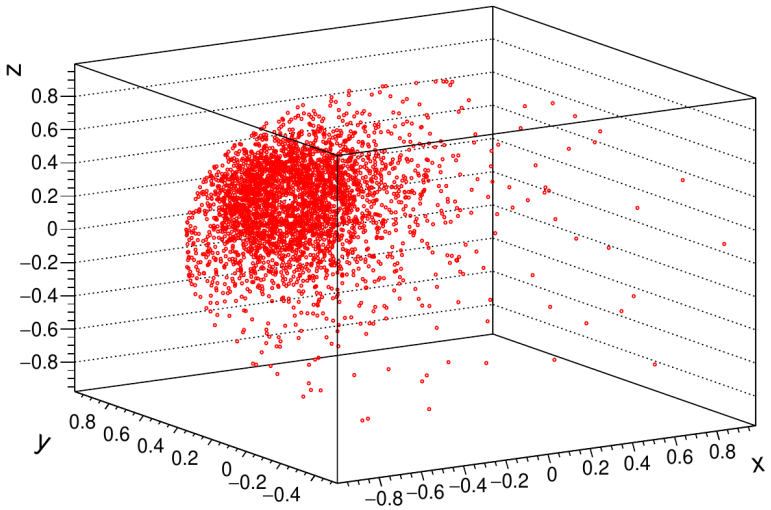

Figure 22: Distribution in space of the photo-electrons emitted by a γ in $(0, 1, 0)$ direction.

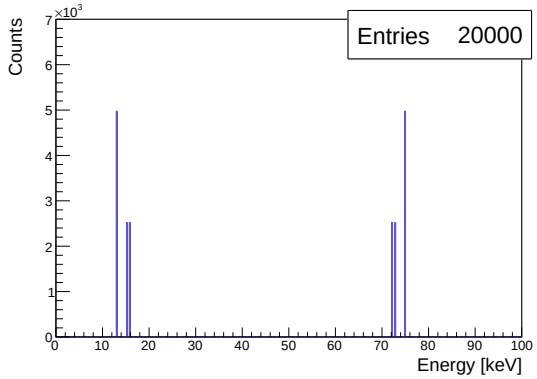

Figure 23: Energy spectrum of fluorescence photons emitted after a photo-electric effect on Pb K-shell.

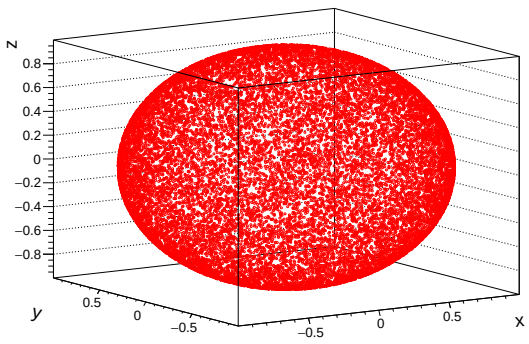

Figure 24: Fluorescence photons isotropic emission.