Università degli studi di Trieste

Corso di laurea magistrale in Fisica della materia

Laboratorio di Fisica Computazionale

Unit III
(Pseudo)random numbers with non uniform distributions; random processes

Davide Bernocco
Matricola s285296

Programming language used:
Python

Contents

1 Random numbers with non uniform distribution: Inverse Transformation Method

In this unit we are going to catch a glimpse at a couple of methods used in simulating (pseudo-)random numbers with a certain distribution.
We will start with the so called "Inverse Transformation Method": it allows us to generate pseudo random distributions with defined cumulative distribution whose inverse can be analytically evaluated.

1.1 I.T.M.: exponential distribution from uniformly generated random numbers

Let the random variable x be generated within a uniform distribution in the interval $[0, 1[$ through the intrinsic Python function `np.random.rand(n°points)`. Therefore we know $z = -\frac{1}{\lambda}\ln(x)$ will be distributed according to $\lambda e^{-\lambda z}$.

1.2 Qualitative check: histograms

From a qualitative point of view we can check whether the variable z is actually exponentially distributed using histograms. Here it is what we get from a run with

$$npoints = 10^6 \qquad\qquad \lambda = 3$$

The column heights have been suitably normalized so that the total area sums to 1.
Furthermore, for the sake of clarification, we will consider the natural logarithm of the columns: in order to avoid issues with `ln(0)` that could occur with empty bins, a small quantity of 10^{-10} have been added to each of them before taking the logarithm.

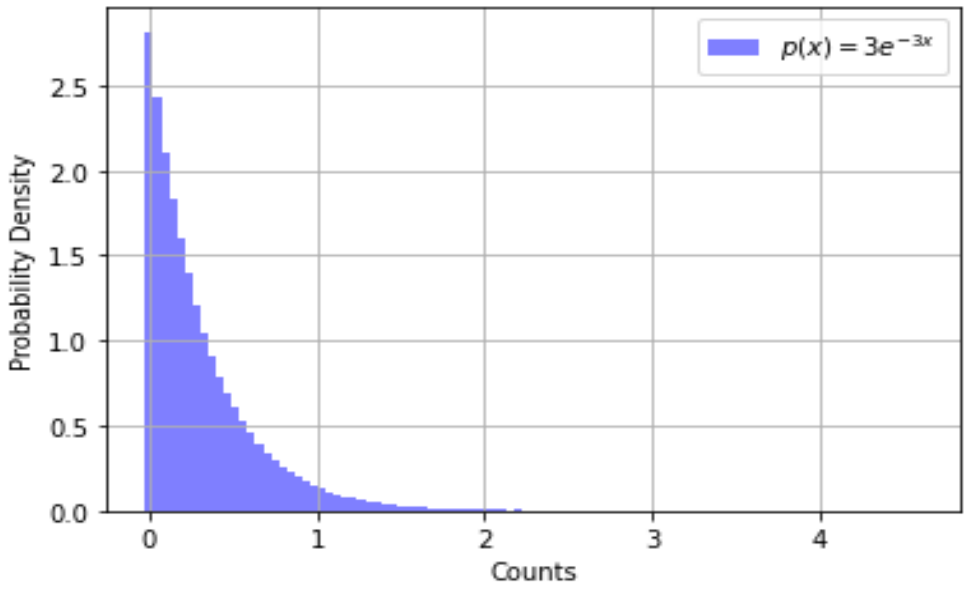

Figure 1.1: (Normalized) exponential distribution generated through the ITM.

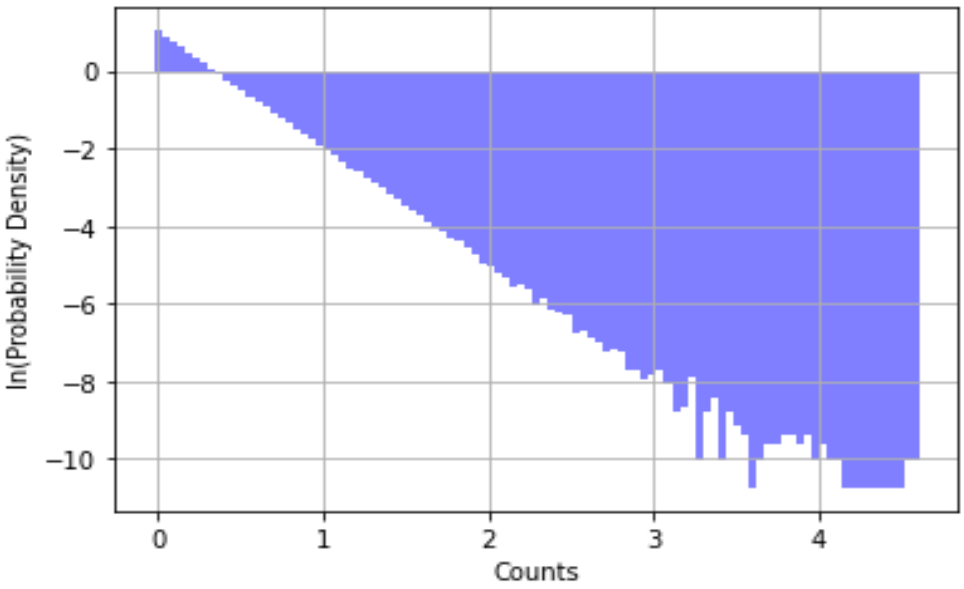

Figure 1.2: The same distribution in log scale.

1.3 Quantitative check: fitting the data with the "least-square method"

We can make a more solid estimation using the *least square method* to fit the logarithm of our data. In so doing we reduce ourselves into evaluating a linear regression which is way easier to handle especially from a computational point of view:

$$y = a + bx \equiv \ln(\lambda) - \lambda x$$

In particular, we used the `curve_fit` module from the `scipy` library.
The vertical coordinate is set to correspond to the normalized logarithm of the frequencies, associated to each bin center. The entire data set is truncated in order to cut away most of the fluctuations which affects the tail of the histogram

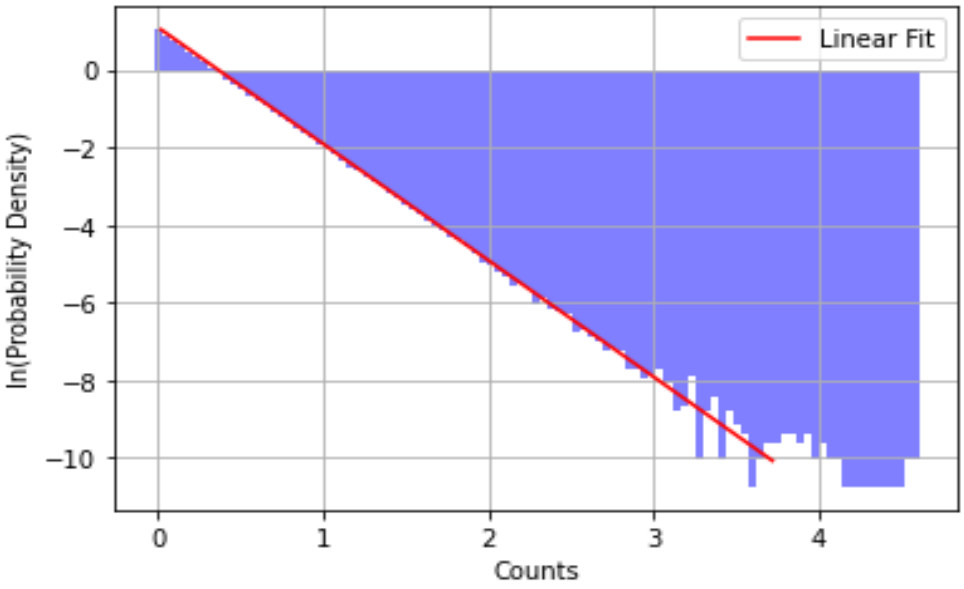

Figure 1.3: Linear regression over the logarithm of the data

Numerically, here is what we get:

$$a = 1.1 \pm 0.1$$
$$b = -3.01 \pm 0.03$$

We can thus conclude that the obtained values are compatible with the expected ones. The distribution generated with the ITM is indeed exponential.

2 Random numbers with non uniform distribution

Here we generate a random variate x in $]1, 1[$ with the "U-shaped" distribution

$$p(x) = \frac{1}{\pi} \frac{1}{\sqrt{1-x^2}}$$

using two different methods.

2.1 With ITM

The strategy is the one showed previously: we generate a set of pseudo-random numbers u in $[0, 1[$ with uniform distribution and consider $x = \sin \pi(2u - 1)$. The x shall be distributed according to the "U distribution".

2.2 With elementary operations

Here we generate two separate variables u and v with uniform distribution in $[0, 1[$, and take $x = \frac{u^2-v^2}{u^2+v^2}$ if $u^2 + v^2 \leq 1$. In so doing, x should follow the "U shaped" distribution as well.

We report the two overlapping histograms to show, in a qualitative way, the equivalence of the tho methods

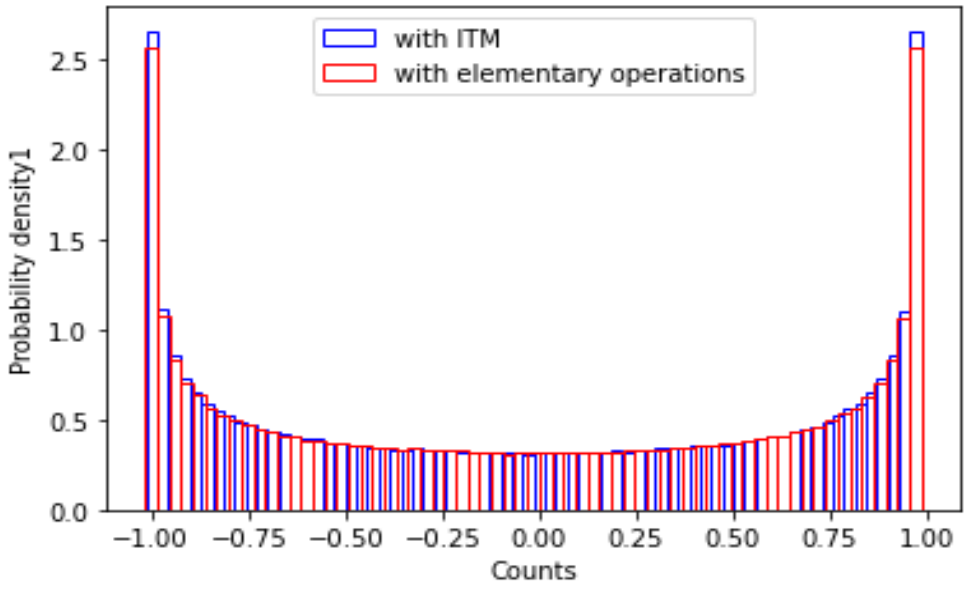

Figure 2.1: "U shaped" distribution: comparison between the two methods

3 Random numbers with gaussian distributions: Box-Muller algorithm

This method allow us to generate distributions which cannot be computed through the ITM and for which the rejection method is inefficient. A typical example is the widespread gaussian distribution.
During the class we were shown different procedures whose code structures are similar but that differs by some algorithmic details; here you can find the result taken from the case which resulted in a lower computational cost.

3.1 Optimized BM algorithm

The key idea is passing through the 2D gaussian distribution taking advantage of the coordinate change into polar ones. This allow us to rewrite the two-dimensional SND into a product of an exponential and a uniform distribution. Hence we consider two different random variables X and Y uniformly generated over $[-1, 1[$ and selected so that $R^2 = X^2 + Y^2$ is uniformly distributed over $]0, 1]$. In so doing we get rid of trigonometric functions and end up with to two statistically independent one-dimensional gaussian distributions x and y.

$$\begin{cases} x = r\cos\theta = \sqrt{-2\ln X}\cos 2\pi Y \\ y = r\sin\theta = \sqrt{-2\ln X}\sin 2\pi Y \end{cases}$$

Since the goal is producing a single set of random numbers, we take advantage of this redundancy to further optimize the code: every two calls the algorithm will use one of the two random numbers already generated in the previous call.

From a run with $npoints = 10^6$ we get:

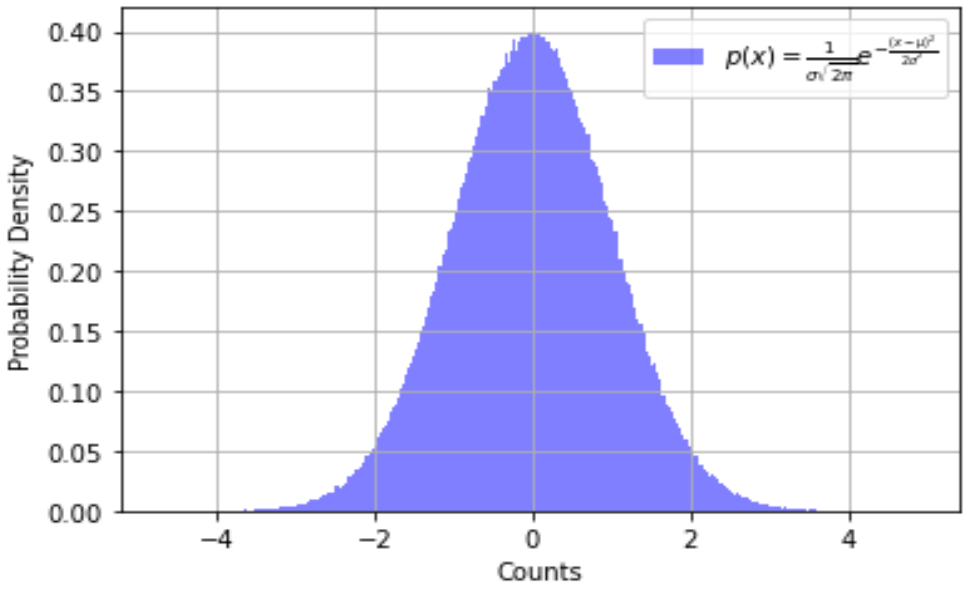

Figure 3.1: Simulation of the SDN with the Box-Muller algorithm

With the intrinsic functions `np.mean` and `np.std` we can numerically evaluate the first and the second momenta of the generated distributions and compare them with the expected value $\mu = 0$ and $\sigma = 1$:

$$\overline{x} = 0.00 \pm 0.04 \qquad\qquad s = 1.000 \pm 0.001$$

where the uncertainties on the parameters corresponds to the associated standard errors.
In conclusion, we can say that the distribution generated with the BoxMuller algorithm is effectively gaussian.

4 Simulation of radioactive decay

We are now ready to apply the acquired knowledge to simulate a physical random process such as the radioactive decay. Our parameters will be the initial number of nuclei $N(0)$ and the rate of decay $0 < \lambda < 1$; the expected distribution is the typical negative exponential.

4.1 Numerical simulation with a chosen decay parameter λ

From a computational point of view the algorithm generates a uniformly random variable in $[0, 1[$ for each one of the not yet-decayed nuclei and compares it to λ: if it is less than this fixed constant the number of atoms is decreased by one, otherwise the number is left unaltered. Once the check has been carried out for all the atoms, the game restart cycling over the remaining nuclei until they are all decayed. We can get the sense of what is happening as always by visualizing it; here the graph from a run with

$$N(0) = 1000 \qquad \lambda = 0.1$$

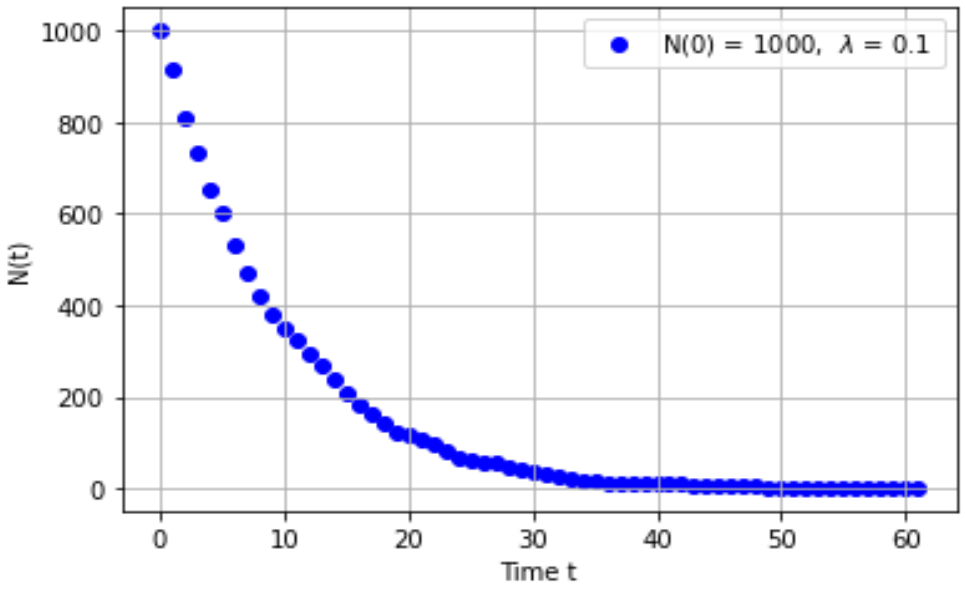

Figure 4.1: Simulation of the radioactive decay

4.2 Quantitative check: fitting the data with the "least-square method"

In order to verify that the trend of our data is effectively exponential we can perform the interpolation we used in the first session, applying it to the logarithm of the generated values applying the linear fit as shown in *Fig. 4.2*:

$$y = a + bx \equiv \ln(N(0)) - \lambda x$$

We now have to compare the numerical parameters with $\ln 1000 \simeq 6.91$ and $-\lambda = -0.1$:

$$a = 7.09 \pm 0.05$$
$$b = -0.12 \pm 0.07$$

We can conclude that, because of the compatibility, the simulated random process actually follows an exponential law.

4.3 Varying the initial number of particles with fixed λ

What happens to the generated data when the initial number of nuclei is changed (keeping λ constant)?

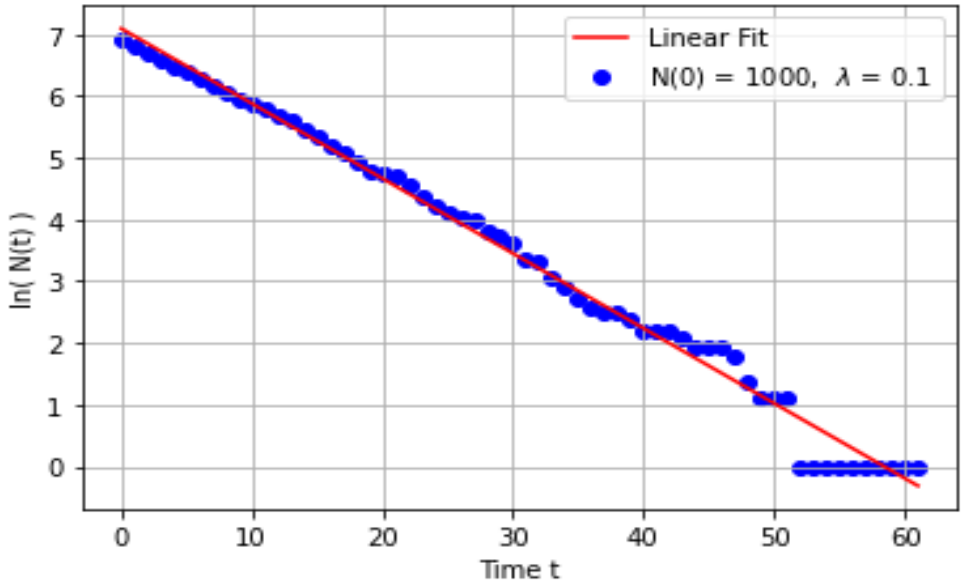

Figure 4.2: Linear fit for the radioactive decay - log scale

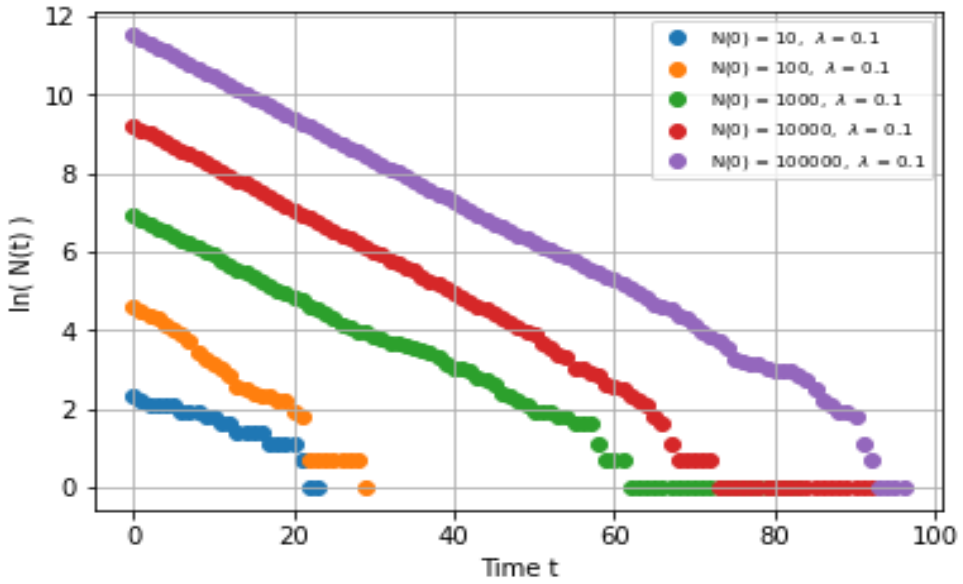

Figure 4.3: $lnN(t)$ for increasing $N(0)$ and $\lambda = 0.1$

We can observe in *Fig. 4.3* smaller stochastic deviations[1] on the tail of the distribution as $N(0)$ increases: this behaviour is due to the intrinsic nature of stochastic simulations which start to give reliable results with large numbers.

4.4 Varying λ keeping $N(0)$ constant

An analogue question can be posed for λ (with fixed $N(0)$).
Since the constant λ represents the probability for each nucleus to decay in a certain Δt, in the algorithm its value will influence the rate at which the number of left particles decreases at each step. In particular we expect to find steeper trends spread over shorter iterations as λ approaches to one. And this is exactly what emerges from the simulation:

[1] If compared to the full length of each data set

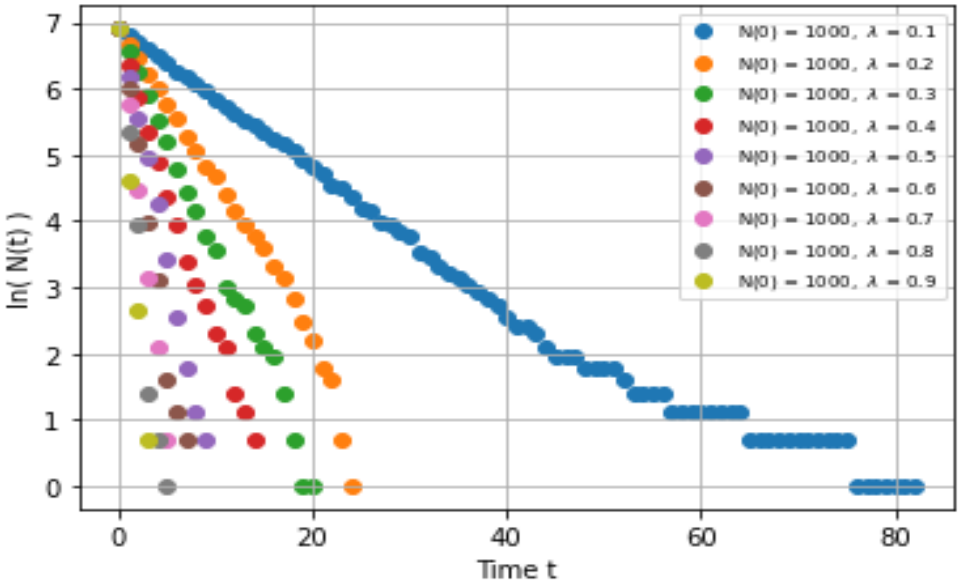

Figure 3.2: $lnN(t)$ for increasing λ and $N(0) = 1000$

.