Monte Carlo Simulations

Jerónimo Noé Acito Pino

Facultad de Matemática, Astronomía, Física y Computación
Universidad Nacional de Córdoba

Abstract

This work presents a series of Monte Carlo simulations implemented in Fortran to explore foundational concepts in statistical mechanics and numerical integration. The simulations include random walks in two and three dimensions, numerical integration using the rejection method and uniform sampling, inverse cumulative distribution function (CDF) sampling, and statistical validation of the Law of Large Numbers. For all cases, statistical indicators such as mean displacement, efficiency, distribution histograms, and convergence behavior were analyzed. The simulations confirm expected theoretical predictions, including the Gaussian behavior of random walks and convergence of integration error with $\frac{1}{\sqrt{N}}$. All code is modular, open-source, and designed for clarity and reproducibility.

1 Introduction

Monte Carlo methods are a class of computational algorithms that rely on repeated random sampling to obtain numerical results. Their flexibility makes them particularly suitable for tackling problems involving stochastic processes, high-dimensional integrals, and statistical distributions. Since their development in the mid-20th century, these methods have become indispensable across physics, finance, engineering, and other disciplines.

In this work, we implement several fundamental Monte Carlo simulations using Fortran. These include random walks in two and three dimensions to study diffusive behavior, numerical integration via the rejection sampling method, non-uniformly distributed random numbers via the inverse transform method, statistical convergence demonstrations via the Law of Large Numbers, and uniform-sampling-based Monte Carlo integration. Each program has been written in a modular fashion with clearly defined inputs, making it straightforward to modify or extend. Our aim is to both visualize and validate theoretical predictions through simulation, offering insight into the power and simplicity of Monte Carlo approaches.

Firstly, section 2.1 deals with random walks as a succession of independent steps of length 1. Many walkers are simulated and relevant physical quantities (mean displacement and mean quadratic displacement) are calculated for the specified number of steps, averaging over all the walkers' trajectories. The program is general enough to calculate random walks in N dimensions, and individual trajectories can be saved for later analysis.

Section 2.2 showcases the rejection method. To this effect the bounds of a rectangle are specified and the program calculate the integral for the function specified. The program considers the positivity of the function to get the positive and negative accepted trials (above and below the x axis respectively), then subtract and scale them to get the integral. The sum, on the other hand, is used to calculate the efficiency of this method.

In the section 2.3, we utilize the inverse cumulative distribution function (CDF) method to generate random numbers with a given point density function (PDF). To this effect, we integrate numerically the chosen PDF to get the CDF, invert it numerically (by array indexing). Then, by selecting random numbers from a uniform distribution, and using the calculated inverse of the CDF, we generate randomly distributed numbers obeying the chosen PDF.

Section 2.4 presents a simple method to illustrate a particular case of the Law of Large Numbers. The sum of N independent random numbers sampled from a uniform distribution is calculated and multiplied by $\frac{1}{N}$ in order to get a resulting number between 0 and 1. The distributions' moments and histograms are analysed, this last one with the aid of a short video.

Lastly, Monte Carlo integration is analysed in section 2.5. The sampling is performed over a uniform distribution and the error of the method investigated.

2 Computational Details and Data Analysis

All programs were made in Fortran, compiled with gfortran, and computed on a computer with Intel Core i5-9300H CPU @ 2.40GHz. The programs and modules used are available on github: https://github.com/MrMxyzptlk-jpg/Monte-Carlo-Simulations.git

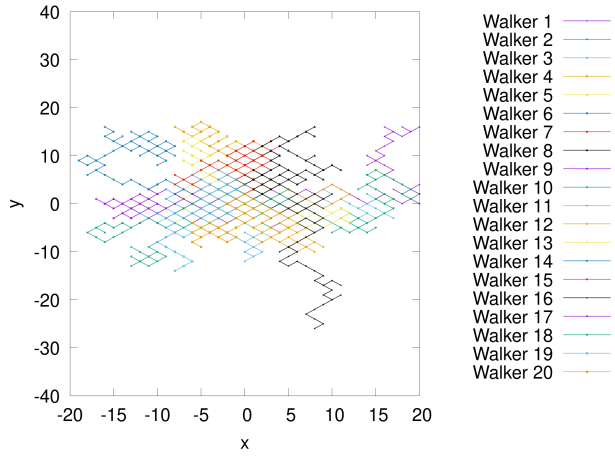
All programs are made with the same structure: a main program, a "modulos" directory containing all the relevant modules, an "input.nml" file and a "compile.sh". The "compile.sh" file contains the order of modules compilations and the flags used. The calculation's variables for each program are contained in the "input.nml" file. All programs use double precision defined in the "modulos/presicion.f90" module. All output files are created in a "datos" directory which needs to be created if not already present. For the random number generator we have used the RNG implemented by Marsaglia and Zaman[1].

2.1 Random Walks

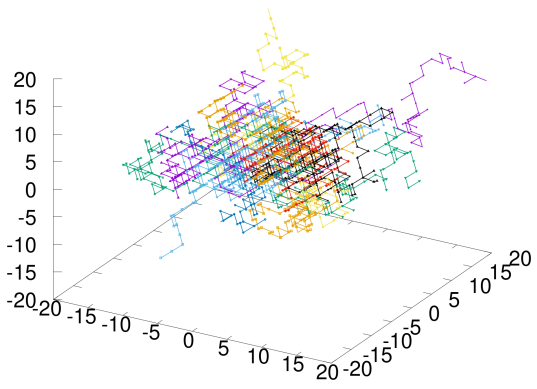
For the random walks we initialized 10^5 different random walks, which take steps in each direction randomly, of length 1, all independent of the others. The mean displacement and mean quadratic displacement are calculated. The program allows for the specification of the minimum and maximum number of steps taken, thus the aforementioned quantities are calculated for each number of steps allowed. Finally, the trajectories of 20 random walkers are saved for later plotting. All calculations were done in 2D and 3D. A sample "input.nml" file is shown below.

```
&calculation
    dim          = 2,
    walkers      = 100000,
    min_steps    = 1,
    max_steps    = 100,
    walkers_saved = 20,
    save_walks   = .true.
/
```

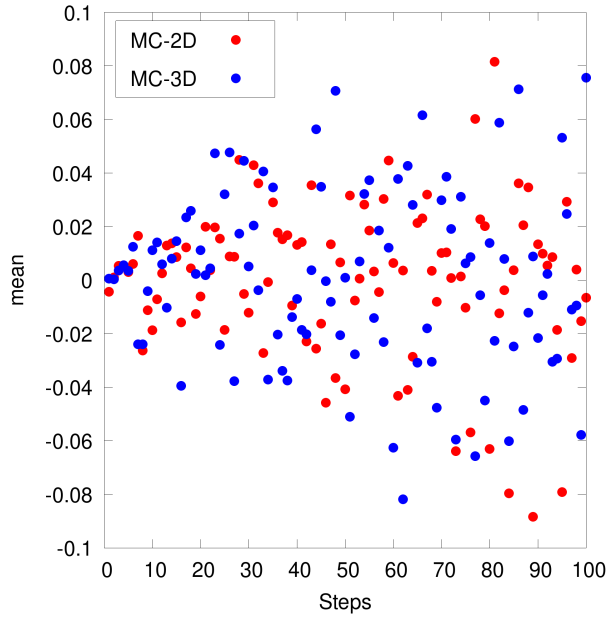
The results are plotted in Figure 1. The paths taken by the walkers appear intuitively random and the mean displacement in 2D and 3D are close to 0, as expected. The standard deviation in both cases also follows the theoretical prediction $\sim \sqrt{d \cdot N}$ for d dimensions. For a more interesting analysis, we have made videos of these calculations, with QR codes in the appendix A1.1. The QR code in Figure 6 links to a video of the distribution of the final positions of the walkers as a function of the steps, for the 2D random walk. The QR codes in Figure 7a and Figure 7b link to videos of the evolution of 20 walkers in 3D and the paths traced respectively. From the first video we see the distribution of the random walks tends to a gaussian distribution, which widens when the number of allowed steps increases, as predicted by Einstein in 1905 [2]. In the other two videos we see that the walkers behave like particles in a gas, diffusing with time and with random paths.



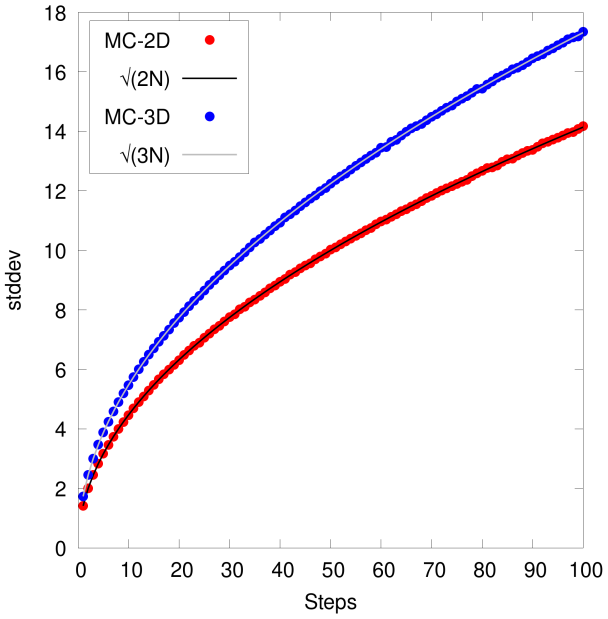
(a) Path taken by 20 random walkers in 2D.



(b) Path taken by 20 random walkers in 3D (same color indexing as for the 2D case).



(c) Mean displacements for the 2D and 3D random walks.



(d) Standard deviations (stddev) for the 2D and 3D random walks compared to the theoretical values.

Figure 1: Random walks in 2D and 3D.

2.2 Rejection Method

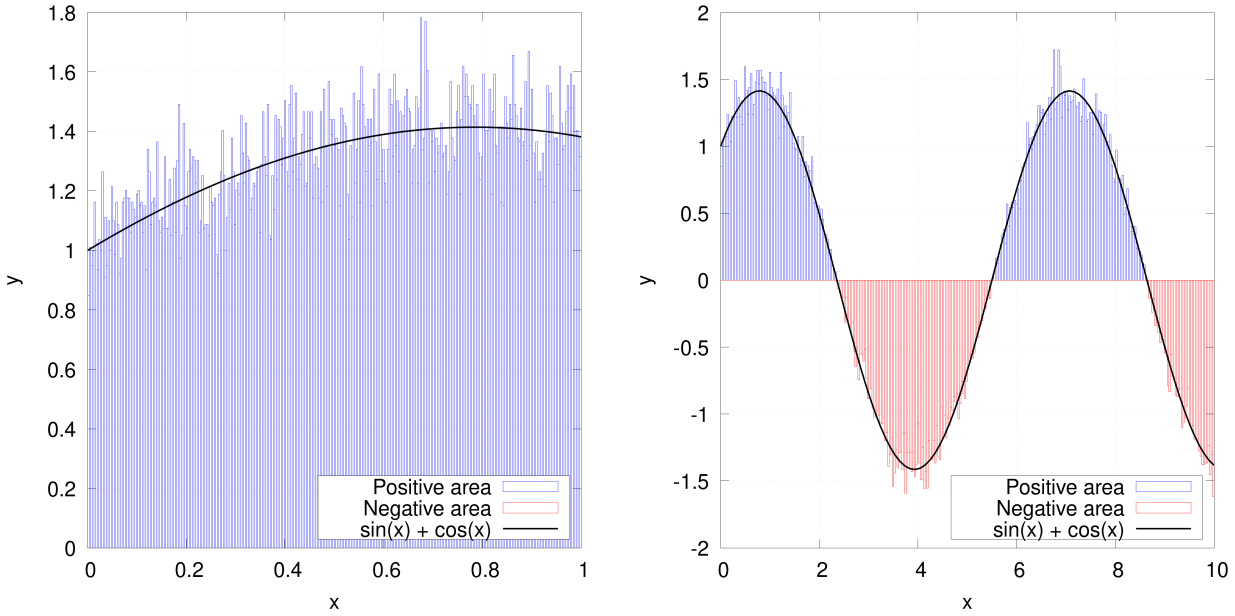
For the integration we have selected 10^5 Monte Carlo samples and a simple function $f(x) = sin(x) + cos(x)$ defined in a "funciones" module. The box in which the sampling is carried out is defined in the input file, with care that the y bounds enclose the function. For the y bounds we have chosen [0,2] and [-2,2], while for the integration limits (x bounds) we chose [0,1] and [0,10] respectively. A sample "input.nml" file is shown below:

```
&calculation
    x_lowerLim = 0,
    x_upperLim = 1,
    y_lowerLim = -2,
    y_upperLim = 2,
    samples    = 100000
/
```

The integrations' results are shown in Table 1 while the histograms created by the rejection method are displayed in Figure 2. The rejection method has an efficiency ($efficiency = \frac{accepted}{rejected}$) around 0.3 and 0.2 for each integration, and the absolute error of the method is less for the integration with a higher success rate.

Bounds	MC integral	analytic integral	Abs error	Efficiency
[0,1]	1.303920	1.301169	0.002751	0.3259800
[0,10]	1.359600	1.295050	0.064549	0.23075

Table 1: Table of the results for the rejection method integration.



(a) (b)

Figure 2: Histograms generated by the rejection method.

2.3 Inverse Cumulative Distribution Function (CDF) method

The program is able to generate random numbers for a distribution with PDF of the form $P(x) = (k+1) \cdot x^k$. The exponent is specified in the input file and was chosen to be $k = 3$. Furthermore, the integration of this function is done using the trapezium method (the number of points also specified in the input file) so as to get the CDF. Finally, a 10^5 random numbers are generated from the inversion of the CDF. A sample "input.nml" file is shown below:

```
&calculation
    exponent     = 3,
    Integration_points  = 1000,
    MC_samples = 100000
/
```

The resulting random numbers generated with this program are plotted in Figure 3, as well as the original PDF for comparison. Excellent agreement with the expected distribution is obtained using this method.

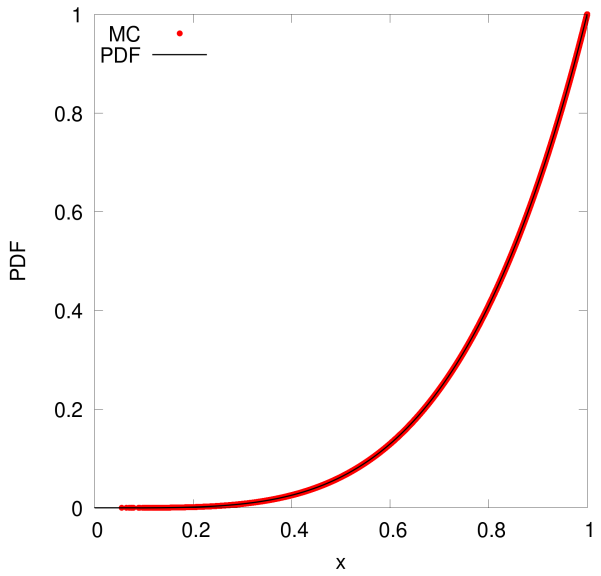


Figure 3: Monte Carlo generated random numbers (red dots) compared to the analytical PDF.

2.4 The Law of Large Numbers

For the summation we have selected N variables ranging from 1 to 10 in steps of 1, and a sample size of 10^5 chosen for each N. A sample "input.nml" file is shown below:

```
&calculation
    N_variables = 100,
    samples = 100000
/
```

The 4 first moments of the distributions generated are shown in Figure 4. As expected, the mean and skewness oscillate around $\frac{1}{2}$ and 0 respectively, while the variance and kurtosis tend to 0. This results show that the distribution tends to concentrate around $\frac{1}{2}$ due to the way we summed the variables ($X = \frac{\sum_{i=1}^{N} x_i}{N}$). This process seems to create a Dirac-delta-like distribution, at least from the behaviour of the calculated moments.

A clearer picture of the problem is obtained by generating a histogram of the resulting distribution. To this end we have created a video of the resulting distributions for each N, the link of which is in Figure 8 of the appendix A1.1. From this video we see that the initial distribution is indeed uniform. Then, for N=2 we get a triangular distribution, as theory would predict. After this point, as N grows, the resulting distributions approximate a gaussian distribution. This is a visual representation of the Law of Large Numbers, for the particular case with the random variables following a uniform distribution.

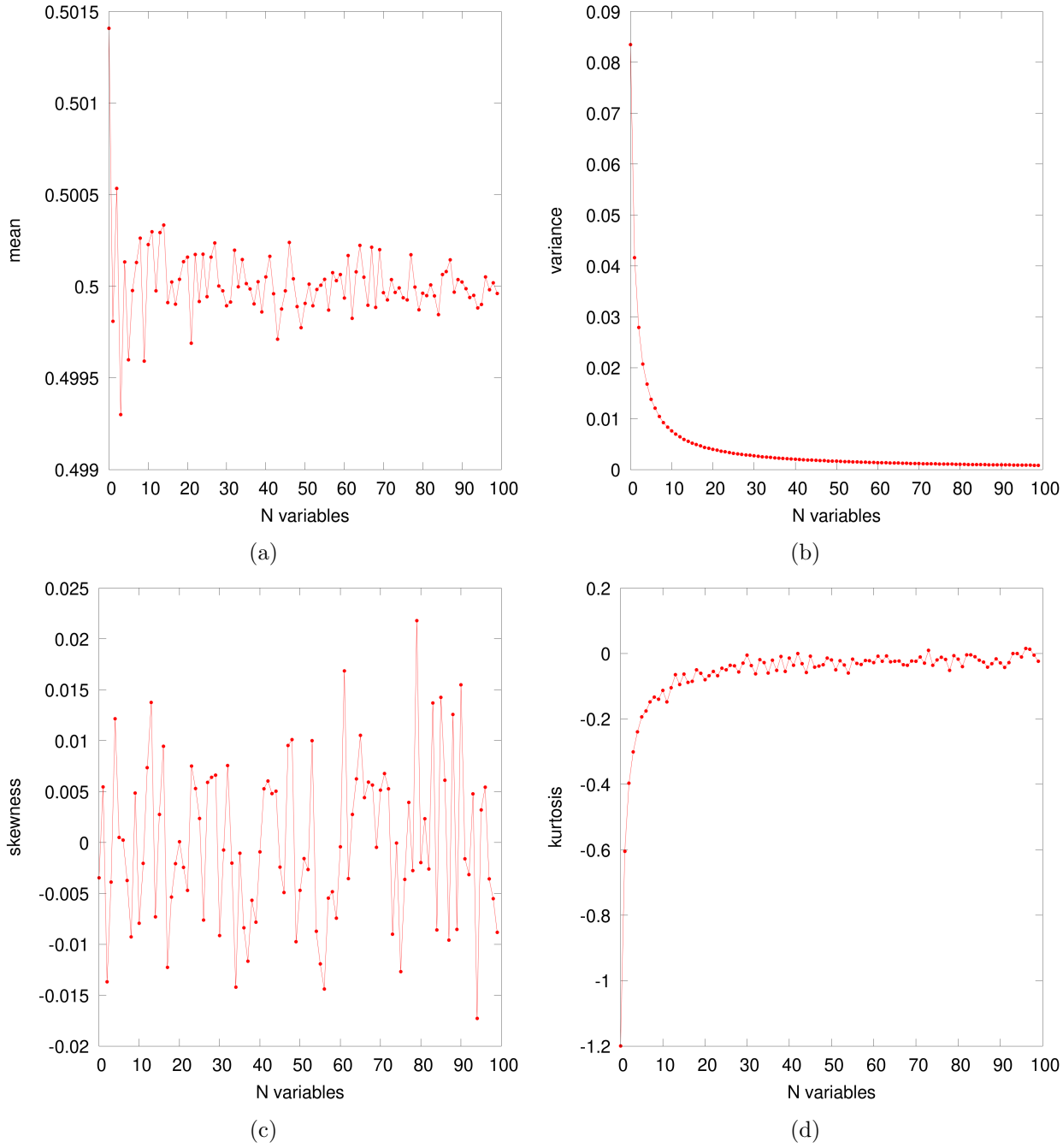


Figure 4: Plots of the 4 first moments of the generated distributions of the sum of N uniformly distributed random variables.

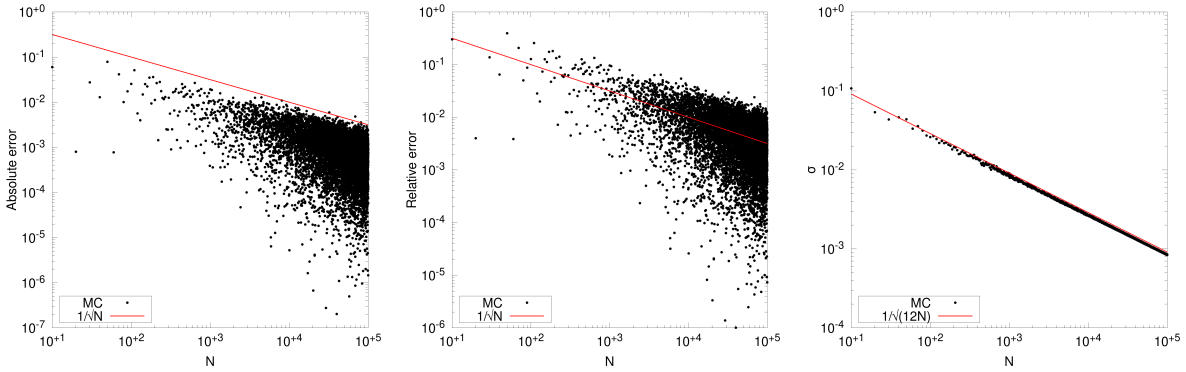
2.5 Monte Carlo Integration (uniform sampling)

For the integration we have selected N Monte Carlo samples ranging from 10 to 10^5 in steps of 10. A simple function $f(x) = x^k$ defined in a "funciones" module is chosen as the integrand. The integration limits chosen were [0,1], and the exponent set to $k = 4$. A sample "input.nml" file is shown below:

```
&calculation
    exponent     = 4,
    lower_lim    = 0,
    upper_lim    = 1,
    max_sample_exponent = 10000
/
```

The results were compared to the analytic solutions and the measures of error were plotted in Figure

5. The theory predicts that the Monte Carlo integration error is of order $\mathcal{O}(\frac{1}{\sqrt{N}})$ where N is the sample size. We see that this value is an upper bound for the absolute error and the standard deviation (σ_I), but represents the most likely outcome for the relative error. We also see that there is a noticeable variation of the error for the absolute and relative cases (which only differ by a factor of the magnitude of the analytic integral) whilst there is a clear linear trend for the standard deviation. This is due to the fact that the measure of error for MC integration is $\sigma_I = \frac{V}{\sqrt{N}}\sigma_f$ where f is the PDF of the random numbers used. Given that the sampling was done over a uniform distribution, with variance $\frac{1}{12}$ we end up with an estimation of error for the MC integration of $\sigma_I = \frac{V}{\sqrt{12 \cdot N}}$. This is in good agreement with the simulations run.



(a) Absolute error vs. sampling nuber (N).

(b) Relative error vs. sampling nuber (N).

(c) Standard deviation (σ) vs. sampling nuber (N).

Figure 5

References

[1] George Marsaglia and Arif Zaman. Some portable very-long-period random number generators. *Computer in Physics*, 8(1):117–121, 01 1994.

[2] A. Einstein. Über die von der molekularkinetischen theorie der wärme geforderte bewegung von in ruhenden flüssigkeiten suspendierten teilchen. *Annalen der Physik*, 322(8):549–560, 1905.

Apéndice

A1.1 QR codes for videos

Figure 6: QR code for 2D random walk histogram as a function of time.

(a) QR code for particles' diffusion by 3D random walk.

(b) QR code for particles' path during the diffusion by 3D random walk.

Figure 7

Figure 8: QR code for histogram of the distributions generated by sum of random variables (Law of Large Numbers).