Directed Reading Program:

Exploring Monte-Carlo methods through numerical integration, random variable generation, and random walks

Hugo Belzil, mentored by Arturo Arellano Arias

Winter 2024 - McGill University

Contents

Links to the Google Colab Notebooks (Python):

- Numerical Integration
- Random variable generation
- 2D random walks

1 Introduction

Throughout this semester of the directed reading program, we had the chance to explore different topics related to stochastic processes and see how Monte-Carlo (MC) methods could help address some of these mathematical problems, where deterministic methods fail due to their intractability or high computational cost. MC methods have numerous applications in various fields of science, such as nuclear engineering, quantitative finance, cybersecurity, and many others.

We started by looking at integration problems and how Monte-Carlo simulations can help approximate intractable integrals, including integrals of functions over domains in $\mathbb{R}$ and $\mathbb{R}^2$. In fact, Monte-Carlo numerical integration is computationally expensive and is typically adapted to very high-dimensional problems.

We continued the DRP by examining how we can generate continuous random variables. This is crucial for MC simulations, where sampling from distributions like Gaussians is needed. This applies, for example, to quantitative finance, where the price of an asset can be modelled as a Brownian Motion. It was assumed in this section that our program (or computer) can generate continuous uniform random variables ($U \sim \mathcal{U}[a,b], \quad a,b \in \mathbb{R}$). This is required for generating variables using the Inversion method (covered in section 3.1) and for 2-dimensional random walks (covered in section 4).

Finally, we explored the behavior of 2-dimensional random walks on finite grids (see figure 5) and observed how Monte-Carlo simulations were able to approximate the true solution derived in section 4.1. We also looked at how the number of functions could impact the precision of the solution.

Before diving into the content, some refresher on MC methods may be needed to understand why they are such useful tools. MC methods are closely related to the Law of Large Numbers (LLN), which states that for a sequence of independent and identically distributed sample points $X_1, X_2, ...$ with mean $\mu < \infty$, we have

$$\lim_{n\to\infty} \frac{X_1 + X_2 + \cdots + X_n}{n} = \mu \qquad \text{with probability 1}$$

In simpler words, the LLN states that if we know the distribution of some random process, we can compute the mean of this process by taking a sequence of the same distribution and calculating its mean.

Take the tossing of a fair coin as an example. You toss a coin 100 times, and you want to know the fraction of tails that you can expect to get (0.5 in this case). One way to approach this problem would be to toss a coin 100 times and count the tails. You would get a fraction somewhat close to 0.5, but chances are you could also get a fraction much larger or smaller than 0.5 (like 0.3, or 0.75 for example). The LLN tells us that if we want to know precisely the fraction of tails in n throws that one can expect, we should make n very large, and count the number of tails obtained. As n goes to infinity, the fraction of tails we obtain will converge to the true fraction we can expect (0.5 in this case) with probability 1.

This example is a bit "overkill" as we can figure out easily that the expected fraction is 0.5. But some random processes have means that are much more complicated to calculate. In this case, we can simulate these processes a large number of times and record their output. Then, we can look at the mean of these outputs, which will converge to the true mean of the process thanks to the LLN.

2 Numerical integration

In this section, we will focus on computing definite finite integrals using Monte-Carlo simulations. We will illustrate how these simulations can be used to approximate an integral with an example.We consider an arbitrary function f. We show how to use Monte Carlo simulations to approximate the integral

$$I := \int_{[0,1]^d} f(x)\,dx, \quad \text{where } d \in \mathbb{N} \tag{1}$$

Observe that in the 1-dimensional case ($d = 1$), any integral of the form $\int_a^b f(x)dx$ can be transformed into an equivalent integral on the interval $[0, 1]$ by making the substitution $u(x) = \frac{1}{b-a}x + \frac{a}{b-a}$. Applying this substitution we obtain

$$\bar{f} = \frac{1}{b-a}\int_a^b f(x)dx = \frac{1}{1-0}\int_0^1 f((b-a)u + a)du = I, \tag{2}$$

This equivalence simplifies our approach, allowing us to focus solely on integrals over $[0, 1]$.
The idea is now to find an unbiased Monte-Carlo estimator I_N to approximate our integral I. Quite intuitively, we will generate a sequence of N points uniformly distributed on $[0, 1] : \{x_1, x_2, ..., x_N\}$ and compute the average ($\bar{f}$) of the function evaluated at each x_i:

$$I_N = \frac{1}{N}\sum_{i=1}^{N} f(x_i). \tag{3}$$

By the law of large numbers, we have that

$$\lim_{N\to\infty} I_N = \int_0^1 f(x)dx \tag{4}$$

2.1 2-dimensional case

We can extend this analysis to integrals in two dimensions. Suppose $U = [0, 1]^2$. Then

$$\bar{f} = \underbrace{\frac{1}{\text{Area}(U)}}_{=1} \iint_{[0,1]^2} f(x, y)\,dx\,dy = \iint_{[0,1]^2} f(x, y)\,dx\,dy \tag{5}$$

As for the one-dimensional case, we generate a sequence $\{(x_m, y_m)\}_{N\geq 1}$ for which all x_i's and all y_i's are uniformly distributed on $[0, 1]$ and compute the average of the function evaluated at all points (x_i, y_i) of this sequence, that is

$$I_N = \frac{1}{N}\sum_{i=1}^{N} f(x_i, y_i) \tag{6}$$

Once again, by the Law of Large Numbers, we have that this estimator converges to the value of the double integral we wish to compute, i.e:

$$\lim_{N\to\infty} I_N = \iint_U f(x, y)\,dx\,dy \tag{7}$$

Observe that any double integral over the interval $[a, b] \times [c, d]$ where $a, b, c, d < \infty$ can be converted into an integral over the interval $[0, 1]^2$ via an appropriate change of variables given by $g(x, y)$. If $g : [0, 1]^2 \to [a, b] \times [c, d]$ is a differentiable bijection, and Jg denotes its Jacobian, we have:

$$\iint_{[a,b]\times[c,d]} f(x, y)\,dx\,dy = \iint_{[0,1]^2} f(g(x, y))|\det(Jg)|\,dx\,dy,$$

Exactly as for the one-dimensional case, this allows us to focus exclusively on double integrals over the interval $[0,1]$. Note that we showed the analysis for simple and double integrals, but numerical integration using Monte-Carlo methods can be extended to any higher dimensions. In fact, it turns out that this method is typically used when dealing with high-dimensional domains, where other numerical integration techniques fail to apply.
Now that we know more about the theory, we will investigate the error analysis of these computations.

2.2 Error analysis

One drawback of relying on random techniques is the error of the approximation to the true value of the integral. We hope to find an estimator that is as accurate as possible, so it may be important for us to know "how well" the estimator does. Luckily, we can quantify this error using the Koksma-Hlawka inequality (section 1.2.3).

2.2.1 Discrepancy

To investigate our bounds of error given by our Monte-Carlo estimator I_N, let us introduce a new notion. The discrepancy $D(\{x_i\}_{i=1}^N)$ of a sequence $\{\mathbf{x_i}\}_{i=1}^N$ in $\mathbb{R}^d$ is given by:

$$D(\{\mathbf{x_i}\}_{i=1}^N) = \sup_{y\in[0,1]^d} \left| \frac{1}{N}\sum_{i=1}^{N} \left(\mathbf{1}_{[0,y_1]\times\ldots\times[0,y_d]}(x_i)\right) - \prod_{k=1}^{d} y_k \right|$$

The discrepancy of the sequence previously defined on $[0,1]^2$ is a measure of "how well" the sequence fills in the square $[0,1]^2$ uniformly. The lower the value of the discrepancy, the better the sequence fills the cube uniformly, and the more accurate will be our estimator as we will see later.

2.2.2 Hardy Variation of a function

Another important notion needed to find an upper bound is the following : the Hardy Variation of a function f on $[a,b]$. It is denoted by V_f and defined as:

$$V_f = \sup\left\{ \sum_{i=1}^{n} |f(x_i) - f(x_{i-1})| : a \leq x_0 < x_1 < \ldots < x_n \leq b \right\}$$

Note however that this quantity is in fact quite complicated to compute, especially in higher dimensions. Therefore, we will not delve into the details about this definition, and we will see that since V_f is a constant which we have no control over unlike the discrepancy of our sequence, it does not matter that much.

2.2.3 A bound of error: the Koksma-Hlawka inequality

Tying together the 2 previously introduced notations (discrepancy of a sequence and Hardy variation of a function), the Koksma-Hlawka inequality can give us an upper bound of the error of the Monte-Carlo estimator :

$$\left| \frac{1}{N}\sum_{i=1}^{N} f(x_i) - \int_{[0,1]} f(x)\,dx \right| \leq V_f \cdot D(\{\mathbf{x_i}\}_{i=1}^N) \tag{8}$$

As long as the function f is sufficiently regular, the variation of f denoted by V_f is a finite constant. However, it turns out to be difficult to compute or bound V_f. By (8), to make the desired estimator more accurate, one tries to come up with a sequence $\{x_n\}$ with really low discrepancy.

2.3 Example

Consider the following integral :

$$I = \int_0^1 \sqrt{x} e^{x^2} dx$$

It is very likely that $f(x) = \sqrt{x} e^{x^2}$ does not have an elementary anti-derivative making the use of the fundamental theorem of calculus impossible here. Therefore, we will try to calculate I using a Monte-Carlo estimator and compare it with the result obtained with Gauss-Legendre quadrature. Using `NumPy`, we approximated I with 20 weights and nodes and obtained

$$I = \int_0^1 \sqrt{x} e^{x^2} dx = \frac{1}{2} \int_{-1}^1 \sqrt{\frac{x+1}{2}} e^{(\frac{x+1}{2})^2} dx \approx \sum_{i=1}^{20} \omega_i \tilde{f}(x_i) = 1.070752088 \tag{9}$$

In Table 1 we reported the values computed with the Monte-Carlo estimator, for 10 points up to 10 000 000 points in $[0, 1]$. The error is relative to the Gauss-Legendre quadrature value calculated in (9).

Number of points in [0,1]	**Monte-Carlo estimator**	**Relative Error**
10	0.676271	36.841413 %
10^2	1.075085	0.404684 %
10^3	1.091594	1.946525 %
10^4	1.052200	1.732590 %
10^5	1.073763	0.281199 %
10^6	1.070650	0.009580 %
10^7	1.070727	0.002314 %

Table 1: Values of our estimator and relative error with respect to Gauss-Legendre quadrature result

It seems clear that as the number of points increases, the Monte-Carlo estimator converges to the value obtained using the Gauss-Legendre quadrature method (which must be extremely accurate using 20 nodes and weights). The graph in Fig. 1 below indeed confirms this result. Notice however that we obtained a better accuracy of the estimator with 100 points than using 10 000 points. This turns out to be a coincidence, and it is just that the 100 randomly chosen points happen to estimate the integral more accurately than the random sample of 10 000 points.

The graph below shows the accuracy of our estimator for this specific integral in function of the number of points in $[0, 1]$ chosen at random (going from 1 to 3000). Observe that we indeed get more variance (larger errors) when the estimator is calculated using a small number of points in $[0, 1]$

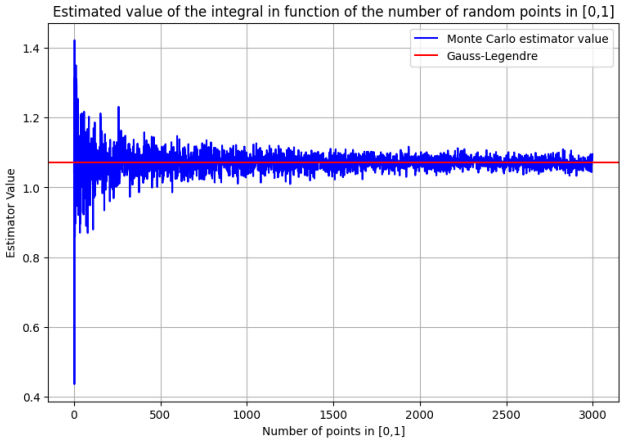


Figure 1: Accuracy of the estimator as a function of the number of points, compared with the Gauss-Legendre quadrature result.

3 Generation of continuous random variables

We investigated how to generate continuous random variables such as Gaussians, exponentials, and Cauchy variables. Generating random variables is crucial for processing many different types of Monte-Carlo simulations since they rely on randomness.

3.1 Inversion Method

Theorem 3.1. *Let $U \sim \mathcal{U}[0,1]$. Let X be a continuous random variable, with support $I \subseteq \mathbb{R}$. If the cumulative distribution function of X, given by $F_X(x)$ has a continuous inverse function $F_X^{-1}(x)$, then:*

$$F_X^{-1}(U) \sim X$$

Proof. F and F^{-1} are both increasing functions on $\mathbb{R}$. Moreover, $\forall\ u \in \mathbb{R}, F^{-1}(F(u)) = u$ if F is continuous at u. Thus, $\forall\ a \in I$, we have:

$$\begin{aligned}
\mathbb{P}(F_X^{-1}(U) \leq a) &= \mathbb{P}(F_X(F_X^{-1}(U)) \leq F_X(a)) \\
&= \mathbb{P}(U \leq F_X(a)) \\
&= \int_0^{F_X(a)} du \\
&= F_X(a)
\end{aligned}$$

Therefore, the CDF of $F_X^{-1}(U)$ is equal to $F_X(x)$, which implies that $F_X^{-1}(U) \sim X$. □

Thus, to generate a continuous random variable whose c.d.f. is continuous and can be expressed as a closed-form bijective function, it is sufficient to be able to generate a uniform variable on $[0,1]$.

3.1.1 Examples

Several common continuous random variables can be generated using the inversion method. Below are a few of them

Exponential

For $X \sim \exp(\lambda)$, the density function and the cumulative distribution function are given respectively by

$$f_X(x) = \mathbf{1}_{(0,\infty)}(x)\lambda e^{-\lambda x} \quad \text{and} \quad F_X(x) = \int_0^x \lambda e^{-\lambda t} dt = 1 - e^{-\lambda x}.$$

From this, we deduce that

$$F_X^{-1}(x) = \frac{-1}{\lambda} \ln(1-x).$$

By Theorem 3.1, in order to generate an exponential variable, we generated a sequence of 15 000 points uniformly distributed on $[0,1]$ using `NumPy`, and applied the function F_X^{-1} to the sequence. We then plotted the frequency barplot of the observations in Figure 2.

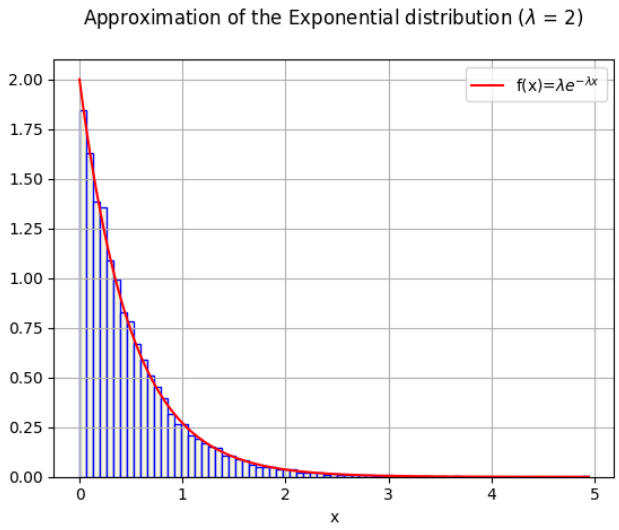


Figure 2: Distribution of the exponential variable ($\lambda = 2$) using the inversion method

One can see that the generated points indeed seem to follow an exponential distribution.

Cauchy Variable

We can use the same inversion method to generate points that follow a Cauchy distribution. Its p.d.f is given by $f(x) = \frac{1}{\pi}\left[\frac{\sigma}{(x-\mu)^2+\sigma^2}\right] \forall\, x \in \mathbb{R}$. This distribution has 2 parameters: $\sigma \in \mathbb{R}_{>0}$ and $\mu \in \mathbb{R}$. The parameter μ is the median of the distribution, while σ is said to be the scale parameter which determines how quickly the tails of the distribution decay. Larger values of σ result in wider distributions with heavier tails, while smaller values of σ result in narrower distributions with lighter tails.

The Cauchy distribution has the particularity that it does not have a finite expectation and variance. We will omit this proof and jump straight to the computation of the inverse of the cumulative distribution.

For $X \sim \text{Cauchy}(\mu, \sigma)$, the density function and the cumulative distribution function are given respectively by

$$f_X(x) = \mathbf{1}_{(\mathbb{R})}(x)\frac{1}{\pi}\left[\frac{\sigma}{(x-\mu)^2+\sigma^2}\right] \text{ and } F_X(x) = \frac{1}{\pi}\int_{-\infty}^{x}\frac{\sigma}{(t-\mu)^2+\sigma^2}dt = \frac{1}{\pi}\left[\arctan\left(\frac{x-\mu}{\sigma}\right)\right] + \frac{1}{2}$$

We deduce that

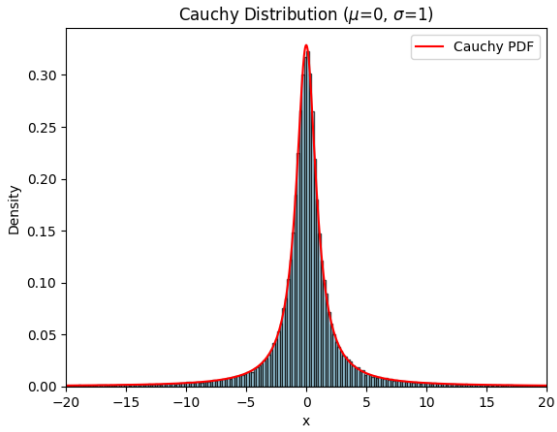
$$F_X^{-1}(x) = \sigma\tan\left[\left(x-\frac{1}{2}\right)\pi\right] + \mu.$$

As for the previous example, we generated 100 000 points uniformly distributed on $[0, 1]$ and applied the inverse cumulative distribution function to this sequence, for different values of μ and σ. On the histograms in Figure 3, we restricted the support of our generated distribution to $[-20, 20]$ in order to avoid displaying histograms over large intervals which would make them hard to read. For each of the distributions shown below, the sequences of 100 000 points are independent. The maximum value and minimum values for each sequence were added underneath the graphs, to "highlight" the fact that Cauchy distributions are heavy-tailed.

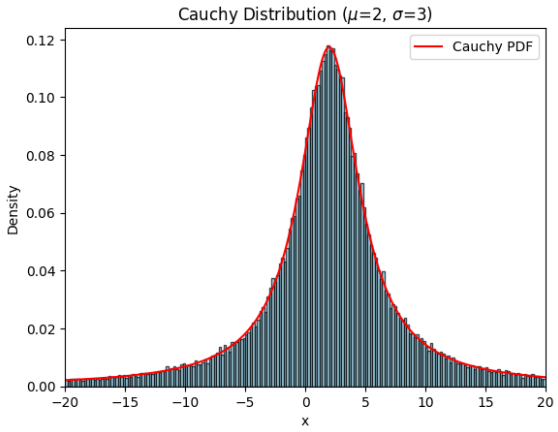
The histogram of the distribution on the truncated interval $[-20, 20]$ corresponds to the conditional probability $\mathbb{P}(X \in \cdot\ | X \in [-20, 20])$. In particular, by the formula for conditional probability,

$$\mathbb{P}(X \in \cdot\ | X \in [-20, 20]) = \frac{\mathbb{P}(X \in \cdot\ , X \in [-20, 20])}{\mathbb{P}(X \in [-20, 20])}. \tag{10}$$

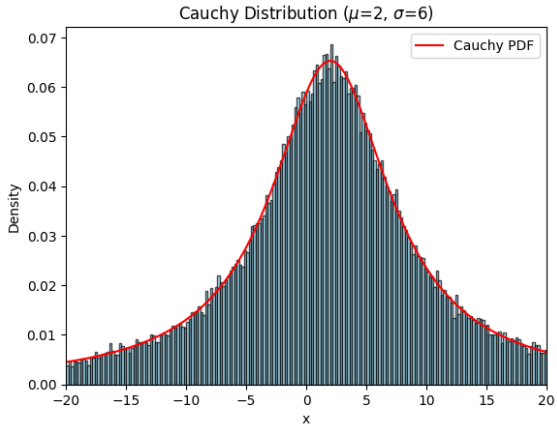
To obtain the histograms in Figure 3, this implies that we have to scale the p.d.f. of the Cauchy distribution over the interval $[-20, 20]$ by the scalar $\mathbb{P}(X \in [-20, 20])^{-1}$ where $X \sim \text{Cauchy}(\mu, \sigma)$.
This value corresponds to the inverse of the integral of the Cauchy p.d.f. on $[-20, 20]$, which we automatically calculate using the `integrate.quad` method from the library `SciPy`.



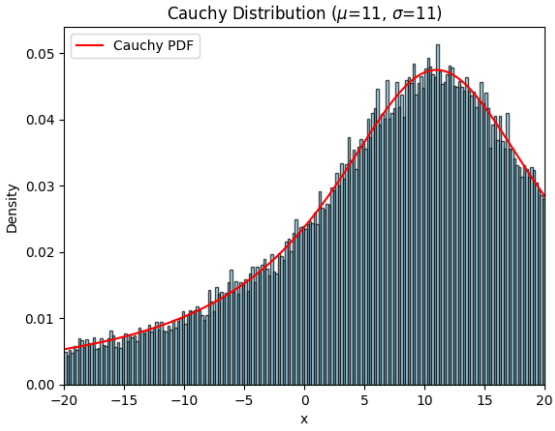
(a) Maximum value generated : 62 993.74
Minimum value generated : -43 252.64



(b) Maximum value generated : 188 983.25
Minimum value generated : -129 755.93



(c) Maximum value generated : 377 964.49
Minimum value generated : -259 513.88



(d) Maximum value generated : 692 942.23
Minimum value generated : -475 768.11

Figure 3: A few Cauchy distributions generated using the inverse method function, on the interval $[-20, 20]$. In red, $f_X(x) \times \mathbb{P}(X \in [-20, 20])^{-1}$.

3.2 Box-Muller Method

One drawback of the inversion method is that it does not allow us to generate normal variables. This is because the Gaussian integral does not have a closed-form solution, which makes it impossible to compute the inverse of the c.d.f of a normally distributed random variable. The Box-Muller method is another technique used to generate normally distributed random variables.

Theorem 3.2. *Let Θ and R^2 be two independent random variables, such that $\Theta \sim \mathcal{U}[0, 2\pi]$ and $R^2 \sim \exp(\frac{1}{2})$. Then, we have that $R\cos(\Theta) \sim \mathcal{N}(0,1)$ and $R\sin(\Theta) \sim \mathcal{N}(0,1)$.*

Proof. Suppose $X, Y \sim \mathcal{N}(0,1), X$ and Y are independent . Then, we can write the 2-dimensional vector (X, Y) in polar coordinates, i.e. $(X, Y) = (R\cos(\Theta), R\sin(\Theta))$. Since X and Y are random, independent quantities, R and θ are also random (and independent, although this is not obvious at first glare, but we'll come to it).
The joint p.d.f of X and Y is given by

$$f_{X,Y}(x,y) = f_X(x)f_Y(y) = \frac{1}{\sqrt{2\pi}}e^{\frac{-x^2}{2}}\frac{1}{\sqrt{2\pi}}e^{\frac{-y^2}{2}} = \frac{1}{2\pi}e^{\frac{-(x^2+y^2)}{2}}$$

Observe that the change to polar coordinates gives us the following relations:

$$R^2 = X^2 + Y^2, \quad \Theta = \arctan\left(\frac{Y}{X}\right)$$

Our goal is now to find the p.d.f. of the joint distribution of R^2 and Θ. Let us denote this p.d.f. by $f_{R^2,\Theta}(r,\theta)$. Let us make the following substitutions:

$$r = x^2 + y^2, \quad \theta = \arctan\left(\frac{y}{x}\right)$$

In order to make these substitutions, we need to multiply $f_{X,Y}$ by the determinant of the Jacobian $J = \frac{\partial(x,y)}{\partial(r,\theta)}$. We have,

$$|J| = \frac{\partial(x,y)}{\partial(r,\theta)} = \left[\frac{\partial(r,\theta)}{\partial(x,y)}\right]^{-1} = \begin{vmatrix} \frac{\partial r}{\partial x} & \frac{\partial r}{\partial y} \\ \frac{\partial \theta}{\partial x} & \frac{\partial \theta}{\partial y} \end{vmatrix}^{-1} = 2^{-1} \implies |J| = \frac{1}{2},$$

where $\theta \in [0, 2\pi)$ and $r \geq 0$.

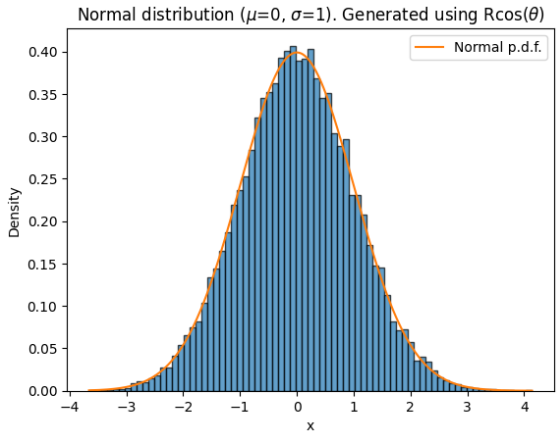
Thus, the joint p.d.f. of R^2 and Θ is now given by $f_{R^2,\Theta}(r,\theta) = f_{X,Y}(x,y)|J|\mathbb{1}_{\{\theta\in[0,2\pi)\}}\mathbb{1}_{\{r\geq 0\}}$. Hence,

$$f_{R^2,\Theta}(r,\theta) = f_{X,Y}(x,y)|J| = \frac{1}{2\pi}e^{\frac{-(x^2+y^2)}{2}}\frac{1}{2} = \underbrace{\frac{1}{2\pi}\mathbb{1}_{\{\theta\in[0,2\pi)\}}}_{A} \cdot \underbrace{\frac{1}{2}e^{\frac{-r}{2}}\mathbb{1}_{\{r\geq 0\}}}_{B}$$

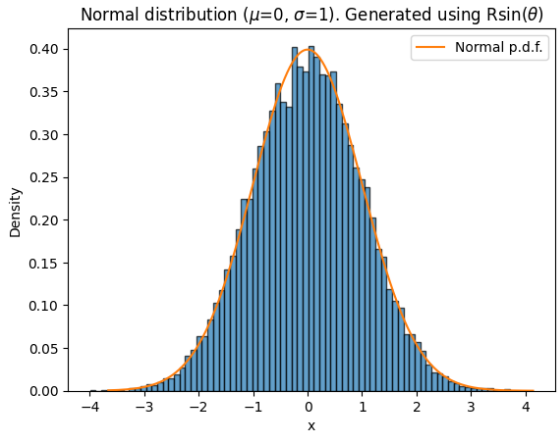
Observe that B is the p.d.f of R^2, which happens to be the p.d.f of an exponential distributed variable with parameter $\lambda = 2$. Similarly, A is exactly the p.d.f of a uniformly distributed variable on $[0, 2\pi]$ which we could assign to the p.d.f of Θ. Since the joint p.d.f of Θ and R^2 is the product of both probability distribution functions, this shows that R^2 and Θ are independent. Since $X = R\cos(\Theta)$ then $X \sim \mathcal{N}(0,1)$. Similarly, $Y = R\sin(\Theta)$, and hence $Y \sim \mathcal{N}(0,1)$.

□

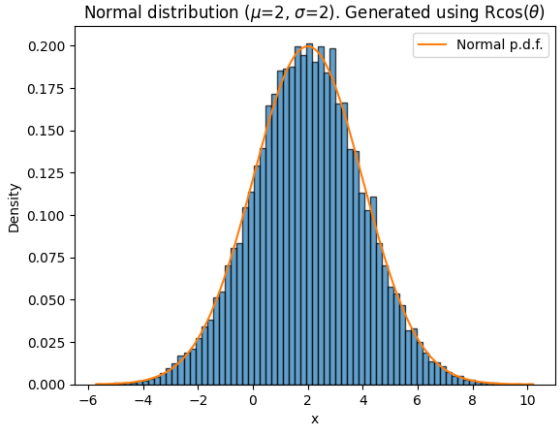
In Figure 4 are some plots of histograms of distributions generated using the Box-Muller method. To do this, we used the module `numpy.random` to generate one sample of 20 000 points uniformly distributed on $[0, 2\pi]$, as well as a sample of 20 000 points distributed exponentially with parameter $\lambda = \frac{1}{2}$. We proceeded to compute two sequences of 20 000 points, the first one using $R\cos(\theta)$ and the second one using $R\sin(\theta)$.



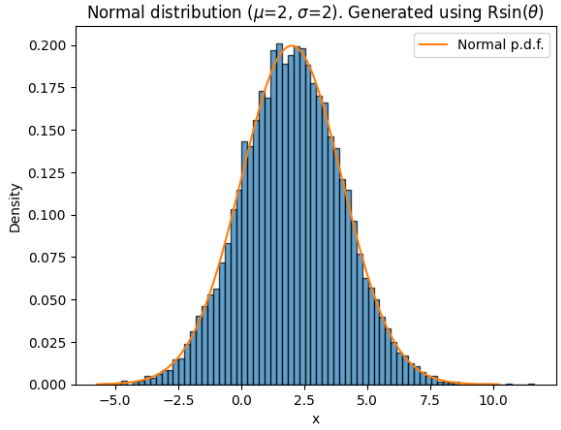
(a) $\mathcal{N}(0,1)$, using $R\cos(\theta)$
Maximum value generated : 4.14
Minimum value generated : -3.66



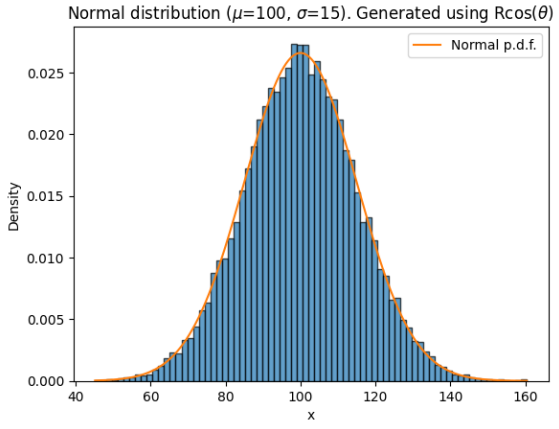
(b) $\mathcal{N}(0,1)$, using $R\sin(\theta)$
Maximum value generated : 3.71
Minimum value generated : -3.99



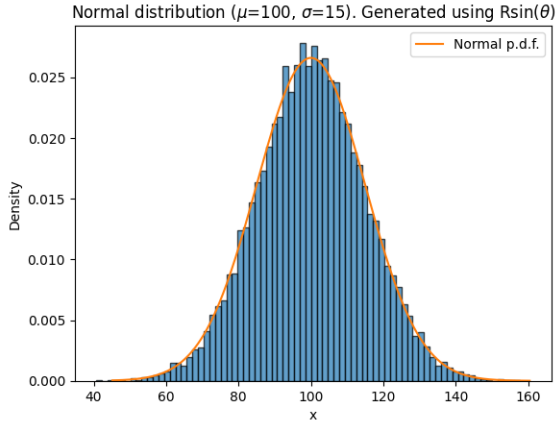
(c) $\mathcal{N}(2,4)$, using $R\cos(\theta)$
Maximum value generated : 10.21
Minimum value generated : -5.71



(d) $\mathcal{N}(2,4)$, using $R\sin(\theta)$
Maximum value generated : 11.62
Minimum value generated : -5.43



(e) $\mathcal{N}(100,225)$, using $R\cos(\theta)$
Maximum value generated : 160.44
Minimum value generated : 45.16



(f) $\mathcal{N}(100,225)$, using $R\cos(\theta)$
Maximum value generated : 157.76
Minimum value generated : 40.74

Figure 4: Different normal distributions obtained using the Box-Muller method. For each distribution, the two histograms plotted are computed using the same sequence of values for θ

4 2D random walk

Finally, we concluded this semester of DRP by investigating the 2-dimensional random walk on a 50×50 grid. This grid that we will refer to as "C" has a boundary and we investigate how the initial point of a random walk impacts the location of the exit point on the boundary.

Precisely, let us look at a subset of the boundary that we denote by A, that consists of half of the bottom-right corner, ranging from entry 25 to 50. The entire boundary of C is denoted by ∂C. The graph below shows a grid exactly formatted as the one we are studying, except it is 10×10 in terms of dimensions, rather than 50×50. The boundary here ranges from points $(5, 0)$ to $(10, 0)$ and from $(10, 0)$ to $(10, 5)$, where it would range from $(25, 0)$ to $(50, 0)$ and from $(50, 0)$ to $(50, 25)$ on our studied grid.

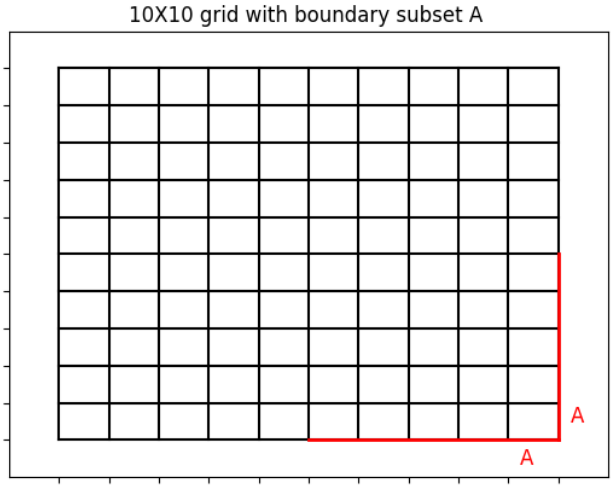


Figure 5: A grid of the same format as the one that we will be studying. Notice the red boundary A in the bottom right corner.

4.1 True solution

We will come up with a system of equations that will help us get the probability for each point that a random walk starting at this specific point leaves the grid through A.

Let S denote the random walk on the grid. Let us define the function $r : \{(x, y) : (x, y) \in C\} \mapsto [0, 1]$ which maps the point (x, y) to the probability that a random walk on S starting at (x, y) would leave C through A. So far, we do not know the closed form of this function but we'll manage to compute it in a recursive way:

Derivation By the Law of Total Probability:

$$\begin{aligned} r(x, y) &= \mathbb{P}(S \text{ exits } C \text{ through } A \mid S_0 = (x, y)) \\ &= \sum_{i \in \{-1, +1\}} \mathbb{P}(S \text{ exits } C \text{ through } A, S_1 = (x + i, y) \mid S_0 = (x, y)) \\ &\quad + \sum_{j \in \{-1, +1\}} \mathbb{P}(S \text{ exits } C \text{ through } A, S_1 = (x, y + j) \mid S_0 = (x, y)) \end{aligned}$$

Let B denote the event "S exits C through A". Then,

$$
\begin{aligned}
&\sum_{i\in\{-1,+1\}} \mathbb{P}(B, S_1 = (x+i, y) \mid S_0 = (x, y)) \\
&+ \sum_{j\in\{-1,+1\}} \mathbb{P}(B, S_1 = (x, y+j) \mid S_0 = (x, y)) \\
&= \sum_{i\in\{-1,+1\}} \frac{\mathbb{P}(B, S_1 = (x+i, y), S_0 = (x, y))}{\mathbb{P}(S_0 = (x, y))} \\
&+ \sum_{j\in\{-1,+1\}} \frac{\mathbb{P}(B, S_1 = (x, y+j), S_0 = (x, y))}{\mathbb{P}(S_0 = (x, y))}
\end{aligned}
$$

We can multiply the first fraction by

$$\frac{\mathbb{P}(S_1 = (x+i, y), S_0 = (x, y))}{\mathbb{P}(S_1 = (x+i, y), S_0 = (x, y))}$$

and the second fraction by

$$\frac{\mathbb{P}(S_1 = (x, y+j), S_0 = (x, y))}{\mathbb{P}(S_1 = (x, y+j), S_0 = (x, y))}$$

in order to get

$$
\begin{aligned}
&\sum_{i\in\{-1,+1\}} \frac{\mathbb{P}(B, S_1 = (x+i, y), S_0 = (x, y))}{\mathbb{P}(S_0 = (x, y))} \\
&+ \sum_{j\in\{-1,+1\}} \frac{\mathbb{P}(B, S_1 = (x, y+j), S_0 = (x, y))}{\mathbb{P}(S_0 = (x, y))} \\
&= \sum_{i\in\{-1,+1\}} \frac{\mathbb{P}(B, S_1 = (x+i, y), S_0 = (x, y))\mathbb{P}(S_1 = (x+i, y), S_0 = (x, y))}{\mathbb{P}(S_0 = (x, y))\mathbb{P}(S_1 = (x+i, y), S_0 = (x, y))} \\
&+ \sum_{j\in\{-1,+1\}} \frac{\mathbb{P}(B, S_1 = (x, y+j), S_0 = (x, y))\mathbb{P}(S_1 = (x, y+j), S_0 = (x, y))}{\mathbb{P}(S_0 = (x, y))\mathbb{P}(S_1 = (x, y+j), S_0 = (x, y))}
\end{aligned}
$$

Observe now that first fraction fraction can be rearranged to give

$$
\begin{aligned}
&\frac{\mathbb{P}(B, S_1 = (x+i, y), S_0 = (x, y))\mathbb{P}(S_1 = (x+i, y) \mid S_0 = (x, y))}{\mathbb{P}(S_0 = (x, y))\mathbb{P}(S_1 = (x+i, y) \mid S_0 = (x, y))} \\
&= \underbrace{\frac{\mathbb{P}(B, S_1 = (x+i, y), S_0 = (x, y))}{\mathbb{P}(S_1 = (x+i, y), S_0 = (x, y))}}_{\mathbb{P}(B \mid S_1 = (x+i, y), S_0 = (x, y))} \underbrace{\frac{\mathbb{P}(S_1 = (x+i, y), S_0 = (x, y))}{\mathbb{P}(S_0 = (x, y))}}_{\mathbb{P}(S_1 = (x+i, y) \mid S_0 = (x, y))}
\end{aligned}
$$

Since S is a Markov Chain, we have:

- 1. $\forall\, 0 \leq m < n \quad \mathbb{P}(S_n = \mathbf{x_n} \mid S_m = \mathbf{x_m}, \ldots, S_1 = \mathbf{x_1}, S_0 = \mathbf{x_0}) = \mathbb{P}(S_n = \mathbf{x_n} \mid S_m = \mathbf{x_m})$
  In other words, conditioned on the past, the future position of the random walk S depends on the most recent past. Hence, $\mathbb{P}(B \mid S_1 = (x+i, y), S_0 = (x, y)) = \mathbb{P}(B \mid S_1 = (x+i, y))$
- $\mathbb{P}(S_1 = (x+i, y) \mid S_0 = (x, y)) = \frac{1}{4}$

The second fraction can be rearranged in the same way. Using these 2 facts, we can simplify a lot the expression above

$$
\begin{aligned}
r(x,y) &= \sum_{i\in\{-1,+1\}} \frac{\mathbb{P}(B, S_1=(x+i,y), S_0=(x,y))\mathbb{P}(S_1=(x+i,y), S_0=(x,y))}{\mathbb{P}(S_0=(x,y))\mathbb{P}(S_1=(x+i,y), S_0=(x,y))} \\
&+ \sum_{j\in\{-1,+1\}} \frac{\mathbb{P}(B, S_1=(x,y+j), S_0=(x,y))\mathbb{P}(S_1=(x,y+j), S_0=(x,y))}{\mathbb{P}(S_0=(x,y))\mathbb{P}(S_1=(x,y+j), S_0=(x,y))} \\
&= \sum_{i\in\{-1,+1\}} \mathbb{P}(B \mid S_1=(x+i,y), S_0=(x,y))\mathbb{P}(S_1=(x+i,y) \mid S_0=(x,y)) \\
&+ \sum_{j\in\{-1,+1\}} \mathbb{P}(B \mid S_1=(x,y+j), S_0=(x,y))\mathbb{P}(S_1=(x,y+j) \mid S_0=(x,y)) \\
&= \sum_{i\in\{-1,+1\}} \frac{\mathbb{P}(B \mid S_1=(x+i,y))}{4} + \sum_{j\in\{-1,+1\}} \frac{\mathbb{P}(B \mid S_1=(x,y+j))}{4} \\
&= \frac{1}{4}\Big[r(x+1,y)+r(x-1,y)+r(x,y+1)+r(x,y-1)\Big]
\end{aligned}
$$

We have now found a recursive formula for the function $r(x,y)$. Recall that $r(x,y)$ is the function that maps to the probability that a random walk starting at $(x,y) \in C \setminus \partial C$ exits C through A. The region $C \setminus \partial C$ corresponds to the set of $(50-2)\cdot(50-2) = 48^2 = 2304$ points that lie inside C and that are not on the boundary. Hence, we can construct a system of 2304 linear equations to solve for $r(x,y)$ at any point in $C \setminus \partial C$, given by

$$
\begin{cases}
4r(x,y) = r(x+1,y)+r(x-1,y)+r(x,y+1)+r(x,y-1), & \text{If } (x,y) \in C \setminus \partial C \\
r(x,y) = 0, & \text{If } (x,y) \in \partial C \setminus A \\
r(x,y) = 1, & \text{If } (x,y) \in A
\end{cases}
$$

We built the system of 2304 equations using `NumPy` and solved for the value of $r(x,y)$ at all points. We then plotted the heat map of the solution:

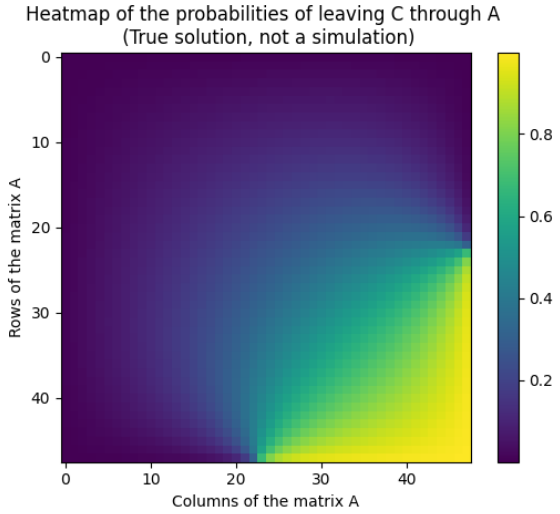


Figure 6: Heat map of the values of $r(x,y)$ at each point on C

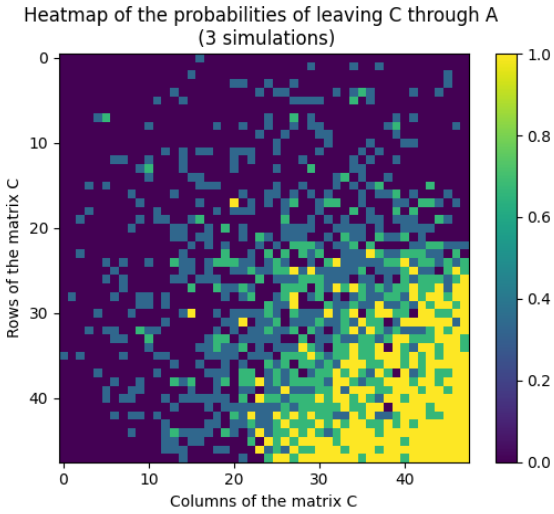
4.2 Approximating the true solution with Monte-Carlo simulations

We have seen in the previous section that for each initial point (x_0, y_0) at which a random walk would start on a 50×50 grid, we can obtain the exact probability that the walk would leave the grid C through A. However, one could also decide that the math involved is not worth the head ache and try to "bruteforce" this problem by simulating multiple random walks on the same grid and infer these probabilities.

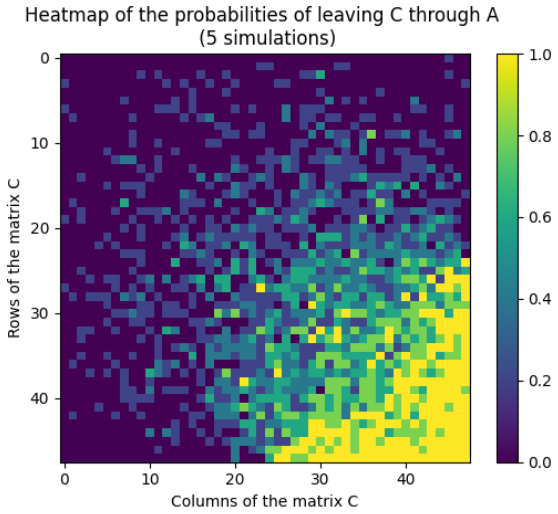
We used this approach to show that the average of the Monte-Carlo simulations converge to the true solutions. To achieve this, we created a Python function that takes as input the number of random walks to emulate at each point of the grid. Our algorithm works as follows:

1. Select the number N of simulations to run.
2. Generate a 48×48 matrix as a `NumPy` array. The matrix' entries are 0 for the moment. Let
3. Start N random walks at each point of the grid (each entry of the matrix). When a random walk leaves C through A, add 1 to the entry of the point at which the RW started.
4. After the RW have all exited C, divide each entry of the matrix by N. Now, each entry of the matrix represents the frequency at which RWs starting at the corresponding point leave C through A.

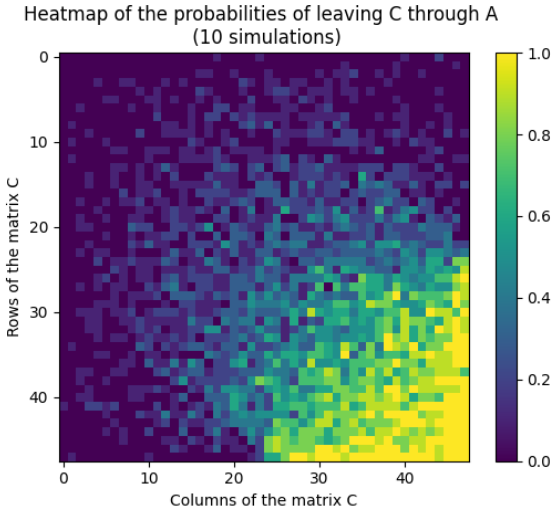
We ran those simulations by making N vary, and plotted heat maps using the library `Matplotlib`. We let $N \in \{3, 5, 10, 20, 50, 100\}$



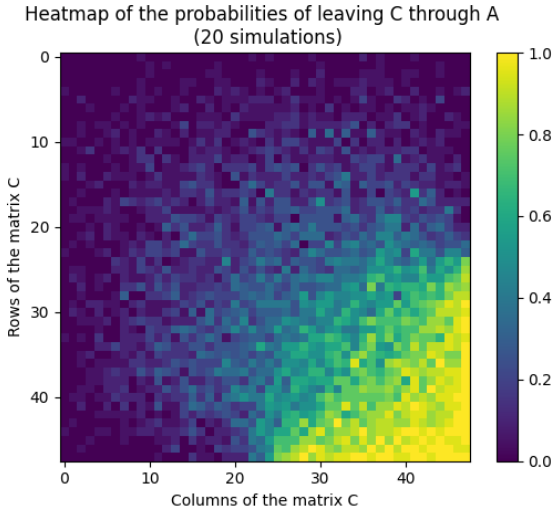
(a) 3 random walks initialized at each point



(b) 5 random walks initialized at each point



(c) 10 random walks initialized at each point



(d) 20 random walks initialized at each point

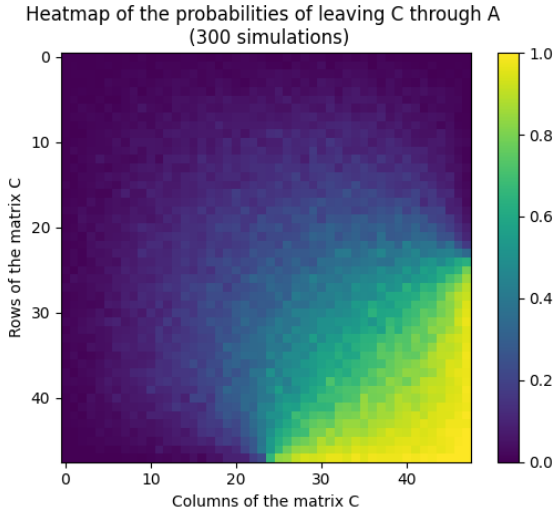
Figure 7: Heat Maps of the probabilities of leaving C through A in function of the number of random walks started at each point on the grid.

Before looking at higher number of simulations, let's analyze these 4 heat maps. One can easily see that for every different value of N, the RWs started close to the region A tend leave C through A with very high probability (this corresponds to the large yellowish or green cluster on the bottom-right corner of each simulation).
However, we notice that for low numbers of simulations (mostly for 3 and 5), there are a few outliers, lying in the upper half of the grids, even sometimes in the upper left half of the grids. Unsurprisingly, it turns out that this is due to the fact that using 3 or 5 simulations is just not enough for obtaining frequencies close to the true probabilities obtained above. As mentioned in the introduction, Monte-Carlo simulations rely on the Law of Large Numbers, which itself requires N (here) to be large in order to approximate the true probabilities.

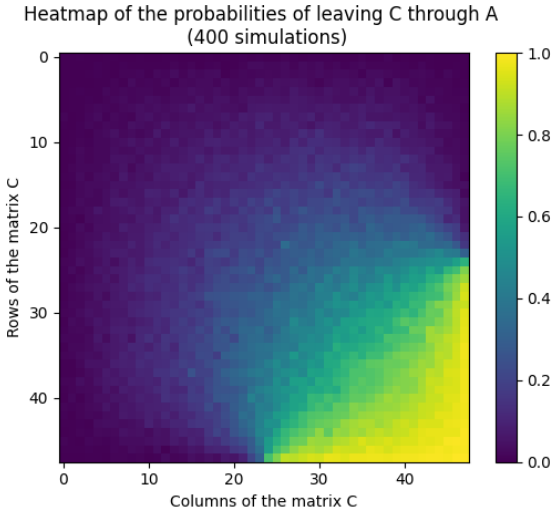
Below are the same heat maps except we used much larger numbers of simulations.

Note : these heat maps were obtained by running our program locally rather than using the Google Colab

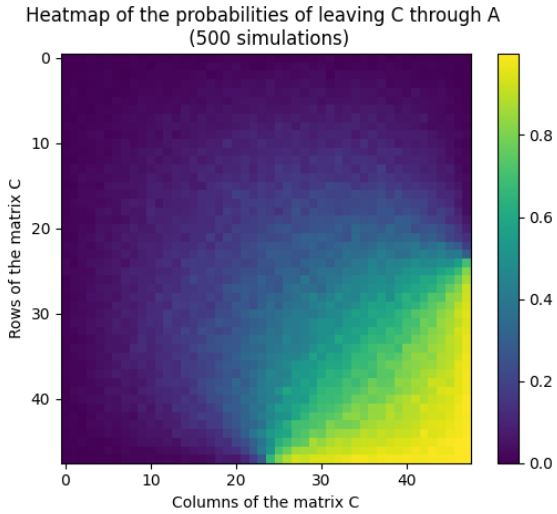
Notebook. This was done in order to accelerate the computations (the running times are given in the sub-captions of each heat maps below.)



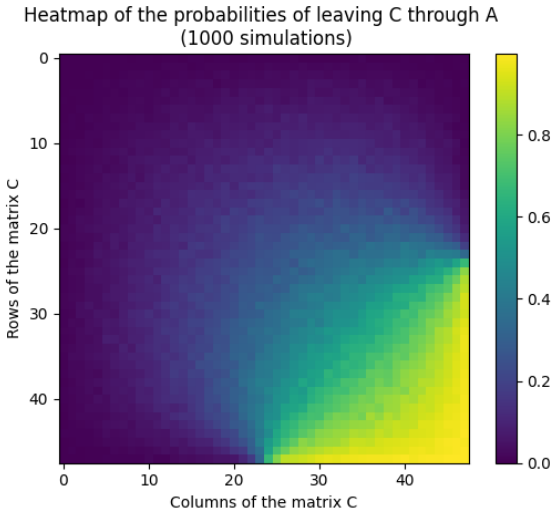
(a) 300 random walks initialized at each point
Running time : 785 seconds (13m 05s)



(b) 400 random walks initialized at each point
Running time : 802 seconds (13m 22s)



(c) 500 random walks initialized at each point
Running time : 999 seconds (16m 39s)



(d) 1000 random walks initialized at each point
Running time : 2014 seconds (33m 34s)

Figure 8: Heat Maps of the probabilities of leaving C through A in function of the number of random walks started at each point on the grid.

4.3 Error Analysis of the Monte-Carlo simulations

The heat maps are a nice tool to visualize if our simulations from section 3.2 do well compared to the true solution obtained in section 3.1. However, they are only visual tools, and we would prefer a metric to quantify the error of each simulation. This will allow us to observe how the number of random walks simulated influence the error of the simulations. The metric we introduce is the Root Mean Squared Error (RMSE). Let y_{ij} denote the true probability of leaving C through A when the random walk S starts at $(i.j)$ Also, let $\hat{y}_{ij}$ denote the probability of the same event, but this time simulated with n random walks. We have

$$RMSE = \sqrt{\frac{1}{2304}\sum_{i=1}^{48}\sum_{j=1}^{48}(y_{ij} - \hat{y}_{ij})^2}$$

Observe that if S_T denotes the 48×48 matrix of true solutions, and $\tilde{S}_n$ denotes the 48×48 matrix of probabilities estimated with n random walks, the Frobenius Norm of $\frac{S_T - \tilde{S}_n}{48}$ becomes the RMSE:
Let $\tilde{E}_n = \frac{1}{48}(S_T - \tilde{S}_n)$

$$\begin{aligned}\|\tilde{E}_n\|_F &= \left\| \frac{S_T - \tilde{S}_n}{48} \right\|_F \\ &= \sqrt{\underbrace{\frac{1}{48^2}}_{=2304}\sum_{i=1}^{48}\sum_{j=1}^{48}(y_{ij} - \hat{y}_{ij})^2} \\ &= RMSE\end{aligned}$$

By the Law of Large Numbers, we have

$$\lim_{n\to\infty} \text{RMSE} = \lim_{n\to\infty} \|\tilde{E}_n\|_F = 0$$

Below is a plot of the RMSE as a function of the number of random walks per simulation.

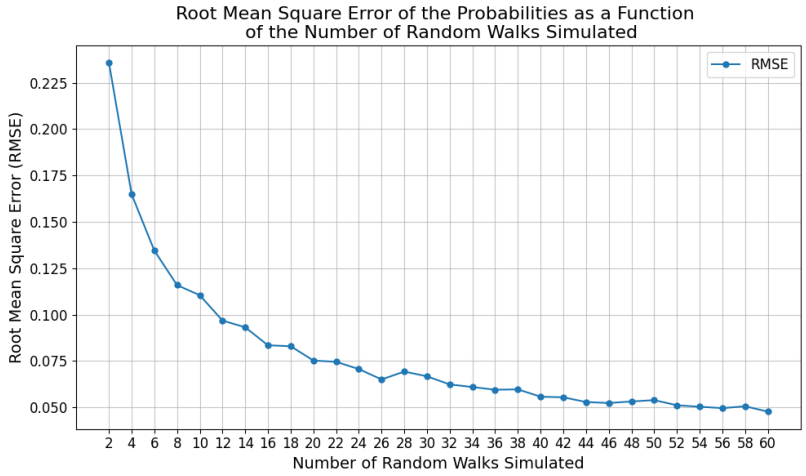


Figure 9: RMSE of our model as a function of the number of random walks

One can easily see that indeed, the error tends to decrease as the number of random walks increase. In the table below, we report the RMSE for other numbers of simulations.

n	**RMSE** $= \|\tilde{E}_n\|_F$
3	0.187013
5	0.146881
10	0.105036
20	0.076647
50	0.051558
100	0.041524
300	0.030604
400	0.028538
500	0.028277
1000	0.025170

Table 2: Recorded RMSE's in function of the number of random walks simulated

5 Acknowledgements

I would like to thank my mentor Arturo Arellano Arias, for his invaluable help and advice throughout the semester. Arturo's approach as a mentor has not only enhanced my ability to independently explore and understand new topics but also significantly strengthened my mathematical foundation. His guidance has been instrumental in the development of this write-up.

I would also like to thank the DRP commitee for allowing me to embark on this really enriching semester.

References

[1] Emmanuel Gobet, *Monte-Carlo Methods and Stochastic Processes : from linear to non-linear*, Chapman and Hall/CRC Press, 2016.

[2] Robert P. Dobrow, *Introduction to Stochastic Processes with R*, Wiley, 2016.

[3] Sheldon Ross, Chapter 5 - Generating Continuous Random Variables, Editor(s): Sheldon Ross, Simulation (Fifth Edition), Academic Press, 2013, Pages 69-96, ISBN 9780124158252, https://doi.org/10.1016/B978-0-12-415825-2.00005-X. (https://www.sciencedirect.com/science/article/pii/B978012415825200005X)

[4] Amri Muhaimin, *Muller-Box Transformation*, Medium, November 25, 2020. Available: https://medium.com/@amrimuhaimin/muller-box-transformation-1c4143bf7a92.