# Monte-Carlo method to solve PDEs numerically

Adrien Lemercier

December 2019

**Abstract**

In this paper, a probabilistic method to calculate numerically an approximation of the solution of partial derivative equations (PDE) of the form $\Delta f - \gamma f + \delta f^2 = 0$ with $\gamma \geq \delta \geq 0$ and with Dirichlet conditions on the boundary : $f(x, y) = \varphi(x, y)$ for some function $\varphi$ is shown. This method uses Markov Chains, in the form of random walks, and is very easy to implement.

Then, a stochastic algorithm directly based on the method presented here has been implemented, and approximate solutions to three different equations belonging to the class of equations described above are shown. Finally, a short analysis of the algorithm's performance is made.

## Contents

# 1 Equations we will solve

We will present a method that enables us to compute an approximation of the function $f$ defined on the domain $\mathbb{D}$ and satisfying :

$$\Delta f - \gamma f + \delta f^2 = 0$$

with $\gamma \geq \delta \geq 0$ and under the Dirichlet condition : $f(x, y) = \varphi(x, y)$ on the boundary $\partial\mathbb{D}$.

Let's introduce the notations that will be used in what follows :

- $\Delta f = \partial_x^2 f + \partial_y^2 f$ is the Laplacian.
- $\mathbb{D} = [0, 1]^2$ is the domain on which we will study the equations here. Notice that the method presented here is very easy to generalize to more complex **bounded** domains (bounded is crucial), even in **higher dimension**.
- $\partial\mathbb{D} = \overline{\mathbb{D}} \setminus \mathring{\mathbb{D}} = \{0, 1\} \times [0, 1] \cup [0, 1] \times \{0, 1\}$ is the boundary of this domain.
- $\varphi$ is a given function, setting the boundary condition of $f$.
- $L$ is an integer used to discretize the problem. From a theoretical perspective it can be any integer. We will discuss what value to take in practice later.
- $\mathbb{D}_L = [0, 1]^2 \cap \frac{1}{L}\mathbb{Z}^2$ is the discretized domain.
- $\partial\mathbb{D}_L = \partial\mathbb{D} \cap \frac{1}{L}\mathbb{Z}^2$ is the discretized boundary.
- $\varphi_L(i, j) = \varphi(i/L, j/L)$ is the discrete function obtained from the continuous boundary function $\varphi$.

# 2 Presentation of the Monte-Carlo method

## 2.1 Definition of a random walk

Let's $\{x_n\}_{n\geq 0}$ be the random walk defined on $\mathbb{D}_L$ as follows : if $x_n$ is on the boundary $\partial\mathbb{D}_L$, then $x_{n+1} = x_n$, else $x_n$ go up, down, left or right with probability $1/4$ for each possibility. We call this random walk the **single random walk**.

Then we define the random walk $\{X_n\}_{n\geq 0}$ : each term is a vector of variable dimension that is called $N_n$. Each coordinate $X_n^{(i)}, i \in \{1, ..., N_n\}$ of this vector can **die** with probability $\alpha$, **duplicate** with probability $\beta$, or with probability $1 - \alpha - \beta$ **move as the single random walk** described above. If $X_n^{(i)}$ duplicates we just add its value in a new coordinate of $X_{n+1}$, no matter the exact location. If $X_n^{(i)}$ dies, we remove the corresponding coordinate for $X_{n+1}$. It is clear from this description that the number of coordinates can vary for different values of $n$.

## 2.2 Definition of a discrete function

Let's define first $T = \inf\{n \in \mathbb{N}, X_n^{(i)} \in \partial\mathbb{D}_L \ \forall i \in \{1, ..., N_n\}\}$ and $\tau = \inf\{n \in \mathbb{N}, X_n = [\,]\}$ where $[\,]$ is the empty vector. Note that $T$ and $\tau$ are **stopping time** with values in $\mathbb{N} \cup \{+\infty\}$, and that $T \leq \tau$.

Then we define a function $F$ on $\mathbb{D}_L$ as follows :

$$F(i, j) = \mathbb{E}\left(\mathbb{1}_{T<\tau} \prod_{k=1}^{N_T} \varphi_L(X_T^{(k)}) \mid X_0 = [(i, j)]\right) \ \forall (i, j) \in \mathbb{D}_L$$

## 2.3 Equation satisfied by this discrete function

The function $F$ defined in the previous subsection satisfies the following equation :

$$\frac{1-\alpha-\beta}{4}\overline{\Delta}F(i,j) - (\alpha+\beta)F(i,j) + \beta F^2(i,j) = 0 \quad \forall (i,j) \in \mathbb{D}_L \backslash \partial\mathbb{D}_L$$

where $\overline{\Delta}F(i,j) = F(i+1,j) + F(i-1,j) + F(i,j+1) + F(i,j-1) - 4F(i,j)$ is the discrete Laplacian (we call it this way for a reason that will appear in subsection 2.5).

Let's prove this equality here. First, observe we have :

$$\mathbb{E}\left(\mathbb{1}_{T<\tau}\prod_{k=1}^{N_T}\varphi_L(X_T^{(k)}) \mid X_1 = [\,], X_0 = [(i,j)]\right) = 0 \tag{1}$$

$$\mathbb{E}\left(\mathbb{1}_{T<\tau}\prod_{k=1}^{N_T}\varphi_L(X_T^{(k)}) \mid X_1 = [(i',j')], X_0 = [(i,j)]\right) = F(i',j') \tag{2}$$

$$\mathbb{E}\left(\mathbb{1}_{T<\tau}\prod_{k=1}^{N_T}\varphi_L(X_T^{(k)}) \mid X_1 = [(i,j),(i,j)], X_0 = [(i,j)]\right) = F^2(i,j) \tag{3}$$

(1) is true because $T = \tau = 1$ since $(i,j) \in \mathbb{D}_L\backslash\partial\mathbb{D}_L$, (2) is true because $\{X_n\}_{n\geq 1}$ is a time-shifted process starting at $(i',j')$. (3) is correct because the two coordinates of $X_1$ generate each two **independent** random walks starting at $(i,j)$.

Then using the property $\mathbb{E}(\mathbb{E}(X \mid Y)) = \mathbb{E}(X)$ on conditional expectation we finally get the desired equation.

## 2.4 Boundary conditions satisfied by this discrete function

If $(i,j) \in \partial\mathbb{D}_L$ then $F(i,j) = \varphi_L(i,j) = \varphi(i/L, j/L)$. This is simply because $T = 0$ in this case.

## 2.5 This discrete function approximates the continuous function which is solution of the initial equation

Let $F'(i,j) = f(i/L, j/L)$. Applying the Taylor's theorem to $f$, we get asymptotically (when $L \to +\infty$) $\Delta f(i/L, j/L) = L^2\overline{\Delta}F'(i,j)$, and therefore $\overline{\Delta}F' - \frac{\gamma}{L^2}F' + \frac{\delta}{L^2}F'^2 = 0$ on $\mathbb{D}_L$.

We **choose** $\alpha = \frac{\gamma-\delta}{4L^2}$ and $\beta = \frac{\delta}{4L^2}$. Since $\gamma \geq \delta \geq 0$, a condition we set at the beginning, everything is fine for a large enough $L$ : $\alpha$ and $\beta$ are positive and their sum is less than 1, so the random walk $\{X_n\}_{n\geq 0}$ described in 2.1 is well-defined.

With those values of $\alpha$ and $\beta$, $F'$ and $F$ satisfie asymptotically the same equation **and** the same border conditions, and thus $F \simeq F'$.

Hence, $F(i,j)$ tends toward $f(i/L, j/L)$ when $L$ tends toward $+\infty$.

## 2.6 Description of an algorithm to calculate an approximate solution

The implementation of an algorithm that simulates $F(i,j)$ is then really easy : for each couple $(i,j) \in \mathbb{D}_L$ we compute $K$ times the value $\mathbb{1}_{T<\tau}\prod_{k=1}^{N_T}\varphi_L(X_T^{(k)})$ starting with $X_0 = [(i,j)]$, and we compute the arithmetic mean of the obtained values. This is simply a Monte-Carlo approximation of the expected value (this works thanks to the law of large numbers).

After choosing the value of the parameter $L$, running an algorithm computing all the $F(i,j)$ gives us an approximation of the solution $f$ to the equation, because $f(i/L, j/L) \simeq F(i,j)$.

This raises the following question : **what are the optimal values of simulation parameters $L$ and $K$ ?** A priori the bigger the better : a larger $K$ gives a more accurate estimation (thanks to central limit theorem) and a larger $L$ gives more information on $f$ since the discretization is finer. But of course it will also require more time to get computed : here we have to make a **trade-off**. I have personally chosen $L = 40$ and $K = 100$, which leads to a computing time of approximately one minute on an usual laptop, and I find the simulations' quality satisfying.

# 3 Illustration of the method

## 3.1 Laplace's equation $\Delta f = 0$

Laplace's equation is very classic and appears in various fields. It is for example a special case of the heat's equation in Physics : if $\varphi$ is the temperature on the boundary $\partial\mathbb{D}$ of a squared box represented by $\mathbb{D}$, and if $\varphi$ doesn't change with time, then $f$ represents the temperature when the equilibrium is reached.

Laplace's equation belongs to the class of the equations we study here, and we can therefore compute an approximate solution with the algorithm described in 2.6.

Here is what we get with a boundary function $\varphi$ equal to 1 on the vertical edges and 0 on the horizontal ones.

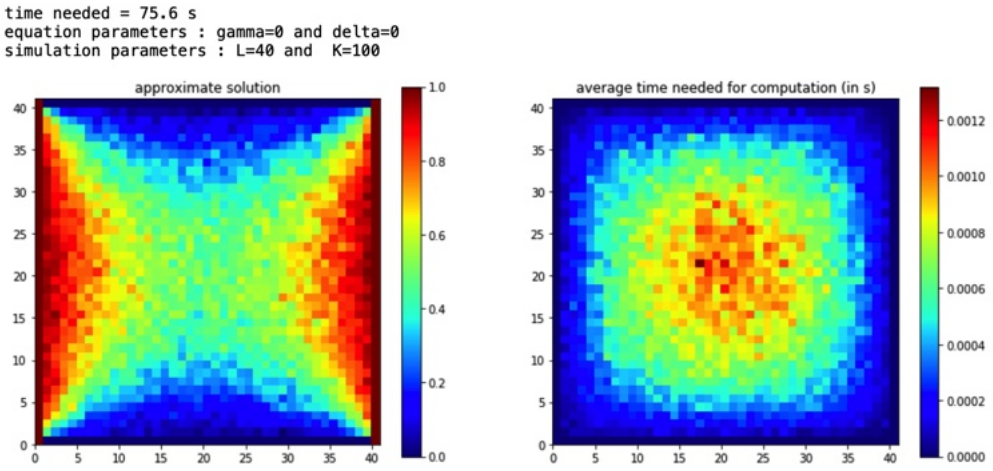
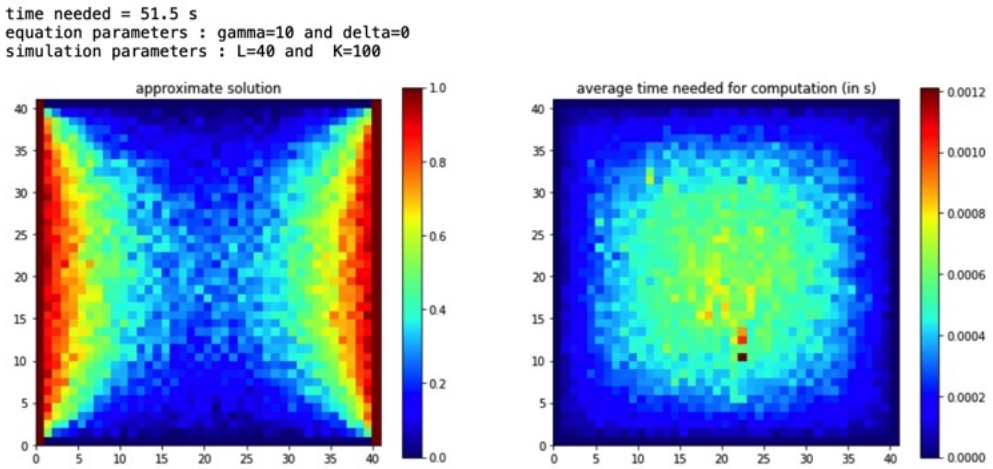


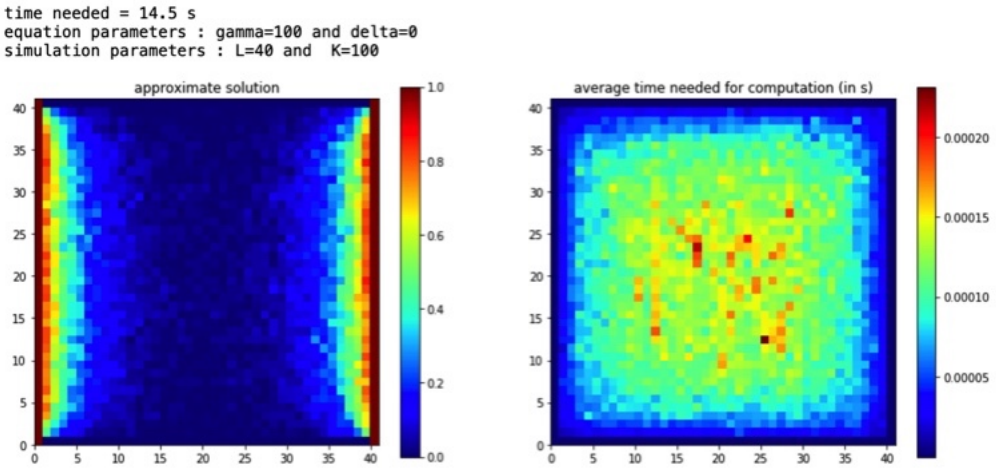

Figure 1: numerical simulation on Laplace's equation.

## 3.2 Equation $\Delta f - \gamma f = 0$

With the same boundary function $\varphi$ and $\gamma = 10$, here is what we obtain.



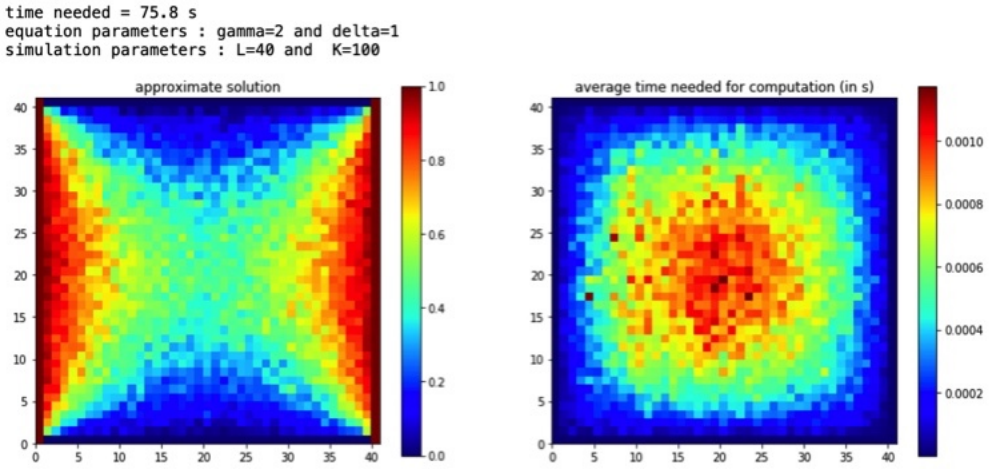

And with $\gamma = 100$.




With those two simulations, one can conjecture the effect on the solution $f$ of increasing the value of $\gamma$ in the equation : the boundary function $\varphi$ being fixed, the value of $f$ on $\mathring{\mathbb{D}}$ decreases when $\gamma$ increases.

### 3.3 Equation $\Delta f - 2f + f^2 = 0$

Let's simulate an approximate solution with the same boundary function $\varphi$.




### 3.4 Comments and discussion

We can observe a few things from the numerical simulations :

- There are inconsistencies in the approximate solutions produced. For example, some "pixels" should be redder or bluer according to the color of their neighbours, since the function $f$ is very unlikely to have a local extremum here. The approximate solutions should also share the same symmetries as $f$, which is overall the case but not pixel-by-pixel. There is a simple reason for those two observations : **the algorithm is stochastic.** Choosing a bigger $K$ diminish those inconsistencies, but then the running time is longer.
- The bigger the parameter $\gamma$, the faster the execution. This is intuitive : since the random walk is more likely to die, on average it stops earlier, and so the algorithm.
- On the contrary, the bigger the parameter $\delta$, the longer the execution. This is because when a coordinate of the random walk duplicates, the full process is likely to end later because we have two identical and independent processes running. So on average the random walk die or reach the boundary later, and thus the running time is longer.

The running time depends on the parameters as follows :

- It is linear in $K$.
- It is asymptotically more than quadratic in $L$ ($\Omega(L^2)$) : not only the number of cells is $L^2$, but the time to compute a cell is longer since the distance to the boundary (measured in number of cells) is on average higher. If we assume that the expected value of the running time for one simulation starting on one cell is linear in the distance of that cell to the boundary, which seems roughly true, then the running time is linear in $L^3$ ($\Theta(L^3)$).
- It is decreasing in $\gamma$.
- It is increasing in $\delta$.

Finally, note that the running time of this algorithm can be reduced very easily by using **parallel computing**.