Solving simple PDEs using Markov chains

Gaëtan Ecrepont

May 2023

Abstract

In this paper, we discuss a probabilistic method relying on Markov chains to solve partial differential equations (PDEs) of the form $\Delta f - \gamma f = 0$ with Dirichlet conditions. We will implement our solution in two dimensions on the unit square domain with basic 0/1 boundary conditions. Nonetheless the method presented can be generalized in arbitrary dimension, with much more complex domains and boundary conditions.

Contents

1 Introduction

The Laplace equation $\Delta f = 0$ naturally arises in many branches of physics such as electrostatics, gravitation and fluid dynamics [1]. Although simple, this equation cannot be solved analytically except for very simple boundary conditions. However, numerical methods were devised to provide approximate solutions. In this paper we present one such method, which uses random walks to generate pointwise estimates of the solution f given boundary conditions.

More precisely, our goal is to solve the PDE

$$\Delta f - \gamma f = 0 \tag{1}$$

where $\gamma > 0$ and $f : D \subset \mathbb{R}^d \to \mathbb{R}$, with Dirichlet conditions φ *i.e.* $f(x) = \varphi(x)\ \forall x \in \partial D$.

We will limit ourselves to $d = 2$, $D = [0,1]^2$ and $\varphi \equiv 0$ on the vertical bounds and $\varphi \equiv 1$ on the horizontal bounds, *i.e.* $\varphi(0, x_2) = \varphi(1, x_2) = 1\ \forall x_2 \in [0,1]$ and $\varphi(x_1, 0) = \varphi(x_1, 1) = 0\ \forall x_1 \in]0,1[$.

With such domain and boundary conditions, Equation (1) becomes the usual steady-state heat equation when $\gamma = 0$, and otherwise $\gamma > 0$ introduces "friction" (or "loss") in the heat transfer. Thus we are essentially looking at the steady-state distribution of temperature in a square box with temperature $T_{\min}$ on two opposites sides and $T_{\max}$ and the two other sides.

We will first describe our method in Section 2 and then implement it to compute actual approximations of the solution in Section 3. Section 4 will be devoted to further discussion.

2 Method

The Laplace equation ($\gamma = 0$)

In this subsection we will assume that $\gamma = 0$, yielding the Laplace equation

$$\Delta f = 0 \tag{2}$$

The method presented here computes estimations of f in a finite and predefined subset $G \subset D$. More precisely, G is a *meshgrid* of D with resolution N, that is $G = \{(\frac{i}{N}, \frac{j}{N}) \mid 0 \leq i, j \leq N\}$.

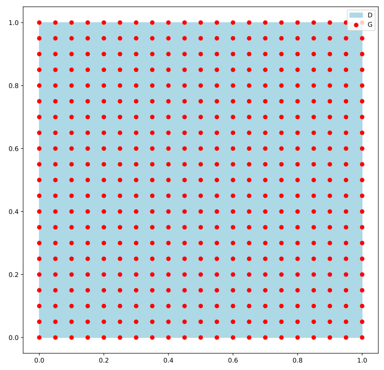

Figure 1: *Meshgrid G* for $N = 20$

Once N has been fixed (a high value of N means better resolution but more computation), we introduce the graph $\Gamma = \{0, \dots, N\}^2$. We then consider a discrete random walk $(X_n)_{n\geq 0}$ on Γ starting at $X_0 = x \in \Gamma$. At each step, the walk can go four directions: up, down, left or right. All directions have the same probability $\frac{1}{4}$. Thus, (X_n) is a Markov chain on Γ, and as such we can introduce the stopping time $T_{\partial\Gamma} = \inf_{n\geq 0} \{X_n \in \partial\Gamma\}$.

With these definitions, $T_{\partial\Gamma} < \infty$ a.s. and we can thus define

$$F(x) = \mathbb{E}_x[\varphi(X_{T_{\partial\Gamma}})] \ \forall x \in \Gamma \tag{3}$$

Intuitively, since $X_{T_{\partial\Gamma}}$ is where the random walk ends (by hitting one of the four edges of the square), $F(x)$ represents the expected value of the boundary function φ computed at this end position, knowing that the random walk started at position x.

Let's now look at the first step of the trajectory, which brings the walk to a neighbor y of x. There are four possibilities for y, and if we condition the expected value in $F(x)$ based on that first step $X_1 = y$, we get

$$F(x) = \sum_{y\sim x} \mathbb{E}_x[\varphi(X_{T_{\partial\Gamma}})|X_1 = y]\mathbb{P}_x(X_1 = y) = \sum_{y\sim x} \mathbb{E}_y[\varphi(X_{T_{\partial\Gamma}})]\frac{1}{4} = \frac{1}{4}\sum_{y\sim x} F(y) \tag{4}$$

where $y \sim x$ means that y is a neighbor of x. Note that we used $\mathbb{E}_x[\varphi(X_{T_{\partial\Gamma}})|X_1 =$

$y] = \mathbb{E}_y[\varphi(X_{T_{\partial\Gamma}})]$, which is a direct consequence of the strong Markov property.

Equation (4) can be rewritten $\bar{\Delta}F = 0$ where $\bar{\Delta}$ is the discrete Laplace operator: $\forall f \in \mathcal{F}(\mathbb{R}^2, \mathbb{R}), \overline{\Delta} f = \frac{1}{4(\Delta x)^2} \sum_{y\sim x}(f(y) - f(x))$. It is also clear that $F = \varphi$ on the boundaries $\partial\Gamma$. Thus F is the (unique) solution to Equation (2) with Dirichlet conditions φ.

Lastly, since we have expressed $F(x)$ in the form of an expected value, we can approximate it using Monte Carlo simulation. For each $x \in \Gamma$, we simulate $k >> 1$ random walks on Γ that start at position x and when the walk hits the boundaries we compute $\varphi(X_{T_{\partial\Gamma}})$. We then average these values to compute our estimate of $F(x)$. Again, higher values of k will provide better estimates but will increase runtime (linearly in k).

Adding a friction term ($\gamma > 0$)

When now look at Equation (1) again but this time with $\gamma > 0$, thus adding a new term $-\gamma f$ which can be interpreted as friction or loss in the heat transfer. We want to reuse the same method as previous, but we somehow need to make the term f appear. The trick is to add a new state $\dagger$ to Γ.

We thus introduce $\tilde{\Gamma} = \Gamma \cup \{\dagger\}$ and we again consider a random walk (X_n) on $\tilde{\Gamma}$, but this time there is a probability $0 < \alpha < 1$ to go from any position $x \in \Gamma$ to $\dagger$; all four directions are still equally likely, such that the probability to go up, down, left or right is $\frac{1-\alpha}{4}$. Finally, the state $\dagger$ is absorbent. We define the stopping time $T_\dagger = \inf_{n\geq 0}\{X_n = \dagger\}$ and

$$F(x) = \sum_{y\sim x} \mathbb{E}_x[\varphi(X_{T_{\partial\Gamma}})\mathbf{1}_{T_{\partial\Gamma} < T_\dagger}] \tag{5}$$

Just like before, we look at the first step of the trajectory, but this time it can bring us to a neighbor y of x or to $\dagger$. However the case $X_1 = \dagger$ does not appear in the sum since $\mathbb{E}_x[\varphi(X_{T_{\partial\Gamma}})\mathbf{1}_{T_{\partial\Gamma}<T_\dagger}|X_1 = \dagger] = 0$ given that $T_{\partial\Gamma} > T_\dagger$ in that case.

We thus obtain $F(x) = \frac{1-\alpha}{4}\sum_{y\sim x} F(y)$, *i.e.* $\alpha F(x) = \frac{1-\alpha}{4}\overline{\Delta}F(x)(\Delta x)^2$ *i.e.* $\overline{\Delta}F - \gamma F = 0$ where $\gamma = \frac{4\alpha(N)^2}{1-\alpha}$ since $\Delta x = \frac{1}{N}$.

Note that $\phi : \alpha \mapsto \frac{4\alpha N^2}{1-\alpha}$ is a bijection between $]0, 1[$ and $]0, +\infty[$. Therefore we can pick $\gamma > 0$ and then select the corresponding $\alpha = \phi^{-1}(\gamma)$.

Lastly, we still have $F = \varphi$ on the boundaries. Thus F is the solution to Equation 1 and since we have expressed $F(x)$ as an expected value, we can again approximate it using Monte Carlo simulation. We generate random walks on $\tilde{\Gamma}$ and follow the same protocol as before to compute an estimate of the expected value.

3 Results

First results

The method was implemented in Python in a Jupyter notebook. Because Python is rather slow we had to limit ourselves to small values of N and k ($N = 100$ and $k = 10$) to obtain the following results. Note the red vertical boundaries indicating hot surfaces ($\varphi \equiv 1$) and the horizontal boundaries indicating cold surfaces ($\varphi \equiv 0$).

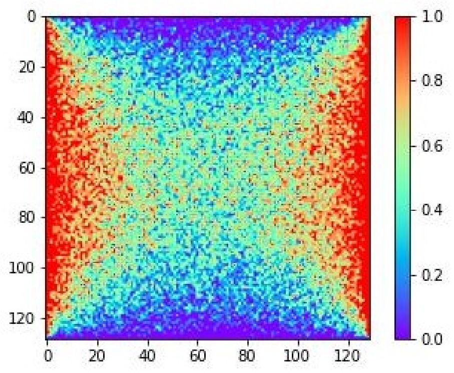

Figure 2: $\gamma = 0$ (no friction)

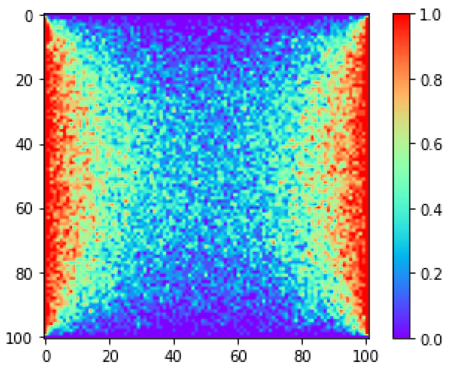

Figure 3: $\gamma = 20$ (friction)

As expected, the heat dissipates more quickly when we add a friction term.

Reducing runtime using Java

In order to get a better resolution, we decided to switch from Python to faster, compiled programming language. We picked Java for convenience and runtime was approximately divided by 30, which allowed for a significant increase in N, from 100 to 400; k was also increased from 10 to 30.

We thus obtained the following grids, which are much more satisfying to the human eye.

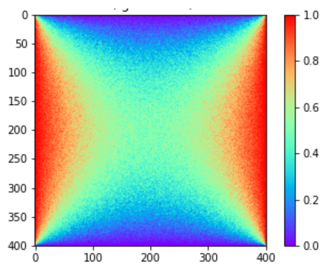

Figure 4: $\gamma = 0$

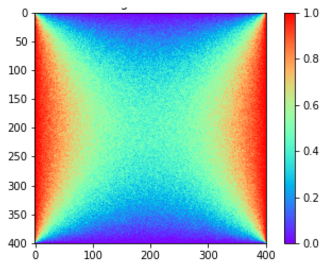

Figure 5: $\gamma = 1$

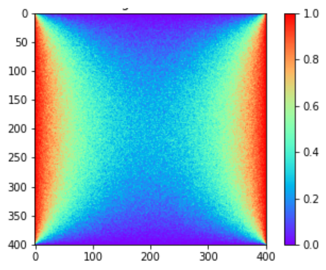

Figure 6: $\gamma = 5$

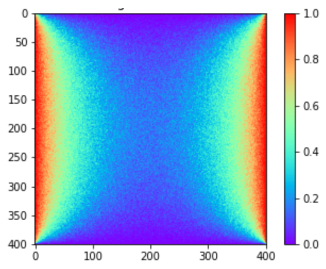

Figure 7: $\gamma = 10$

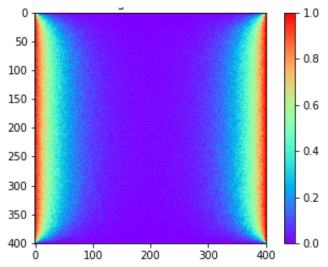

Figure 8: $\gamma = 50$

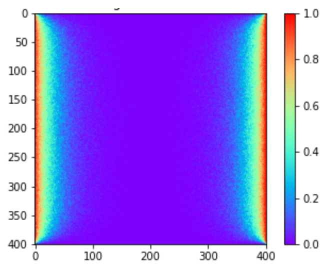

Figure 9: $\gamma = 100$

As expected, we find that the heat dissipates faster as *gamma* increases. For $\gamma > \gamma^\star = 100$, the heat is almost instantly absorbed and therefore $f \simeq 0$ except near the hot surfaces.

4 Discussion

Runtime

When performing simulations, it appears that the closer x is to the center of the square, the longer it is to compute $F(x)$. We also notice that runtime decreases dramatically once γ reaches the threshold $\gamma^\star$.

These observations can be explained easily by proposing rough estimates of the expected values of stopping times $T_{\partial\Gamma}$ and $T_\dagger$. If we assume that $\alpha = 0$, (X_n) becomes the usual two-dimensional symmetric random walk and we know from the central limit theorem that a random walk on $\mathbb{Z}^2$ starting a $x = 0$ hits the circle of radius R in about R^2 steps. Hence we have $T_{\partial\Gamma} \sim \mathrm{d}(x, \partial\Gamma)^2$ where

$\mathrm{d}(x, \partial\Gamma)^2$ is the distance between x and the boundaries of the square. Likewise, since at every step we have a probability α to be absorbed by $\dagger$, $T_\dagger \sim \frac{1}{\alpha}$. Plugging in $\alpha = \frac{1}{1+\frac{4N^2}{\gamma}}$ yields $T_\dagger \sim \frac{4N^2}{\gamma}$.

If we focus on the points which are most expensive to compute, *i.e.* the points near the center of the square, we have $\mathrm{d}(x, \partial\Gamma) \simeq \frac{N}{2}$ such that $T_{\partial\Gamma} \sim \frac{N^2}{4}$. Solving $T_\dagger \sim T_{\partial\Gamma}$ yields $\bar{\gamma} = 16$. We can thus identify two regimes:

- for $\gamma >> \bar{\gamma}$, $T_\dagger < T_{\partial\Gamma}$ *i.e.* most walks end up absorbed, hence the overwhelming cold
- for $\gamma << \bar{\gamma}$, $T_\dagger > T_{\partial\Gamma}$ *i.e.* most walks end up hitting the boundaries, hence the high computation time

This modelling is rather simple and yet these theoretical results are strikingly close to the practical observations.

Generalization

Regarding the potential for generalizations of this method, note that

- boundary conditions given by φ can be made arbitrarily complex
- we can increase the dimension d of the problem simply by generating random walks in dimension d instead of 2
- the domain $D \subset \mathbb{R}^d$ can be much more complex then the unit hypercube

Parallelization

Since most of the computer workload consists in generating the same random walks several times, we could easily speed up runtime by another order of magnitude by implementing basic CPU parallelization. However this was not our goal and wasn't implemented because the Java speedup was already satisfying.

References

[1] "Laplace's equation". In: *Wikipedia* (May 9, 2023). URL: `https://en.wikipedia.org/wiki/Laplace%27s_equation`.