Monte Carlo Method for Solving Laplace and Poisson Equations

Ankur Shrikant Sonawane
Electrical Department
Indian Institute of Technology BHU
s.ankursonawane@gmail.com

***Abstract*— The Poisson Equation, and its special case, the Laplace Equation, are important partial differential equations within electrostatics, as they describe the electric potential field caused by a particular charge distribution. Solving the Poisson equation for a given set of boundary conditions is a fundamental problem within the field. This report describes an algorithm for a Monte Carlo Method solution to the Poisson and Laplace equations. Monte Carlo Method is a parallelizable non-deterministic numerical approach towards solving this problem. A Python implementation is also provided for three versions of the Monte Carlo Method: Fixed Step, Semi-Floating and Full Floating. Several examples including the one-dimensional and two-dimensional parallel plate capacitors are performed to illustrate the function of the implementation.**

Keywords—Poisson Equation, Laplace Equation, Monte Carlo Methods, Electrostatics, Parallel Plate Capacitor, Python Implementation

I. Poisson and Laplace Equation

Poisson equation is one of the most important equations in applied mathematics and has applications in such fields as astronomy, heat flow, fluid dynamics, and electromagnetism and describes stationary (time-independent) problems such as torsional deflection of a prismatic bar, stationary heat flow, distribution of electrical potential or filtration through porous media[1]. It is an elliptic partial differential equation which describes equilibrium and is a part of Boundary Value Problems (BVP). It takes the general form of:

$$\nabla^2\varphi = f\ ;\ \varphi(x) = g(x)$$

Here, φ is a scalar field, f is any given function of space and g is the boundary condition. In electrostatics, we can derive the Poisson Equation from the fundamental Maxwell's Equations as follows

$$\nabla \cdot E = \frac{\rho}{\epsilon}$$

Here ρ is the free charge volume density, ϵ is the permittivity and E is the electric field at a given point. Considering the electric potential field φ we have come to the Poisson Equation.

$$\nabla\varphi = -E$$

$$\nabla^2\varphi = -\frac{\rho}{\epsilon}$$

The Laplace Equation is a special case of the Poisson Equation where f is known to be zero [2]. This is the case of calculating the electric potential when there are no charges in the region. The Laplace Equation has the advantage of having a computationally easier solution.

According to the uniqueness theorem[3] for the Poisson Equation, there exist an infinite number of solutions for the scalar field φ but a unique solution for the gradient of the scalar field $\nabla\varphi = E$. This means that there is a unique electric field described by any given charge distribution.

However, finding an analytical solution to this equation is difficult, as with many other partial differential equations. So, several numerical methods have been devised to solve them approximately with varying levels of computational cost and accuracy.

II. Finite Difference Equations

In numerical analysis, finite-difference equations (FDEs) are discretizations used for solving differential equations by approximating them with difference equations [4]. FDEs allow numerical methods like Monte Carlo and Gauss Siedel which require iteration over discrete differences to work on continuous differential equations [5]. In 1D the Laplace operator is approximated to an FDE as follows:

$$\varphi\,(x-h) = \varphi(x) - h\frac{\partial\varphi}{\partial x} + h^2\frac{\partial^2\varphi}{2\partial x^2}\ldots$$

$$\varphi\,(x+h) = \varphi(x) + h\frac{\partial\varphi}{\partial x} + h^2\frac{\partial^2\varphi}{2\partial x^2}\ldots$$

Adding while ignoring higher order terms,

$$\frac{\partial^2\varphi}{\partial x^2} = \nabla^2\varphi\,(x) \approx \frac{\varphi\,(x-h) - 2\,\varphi\,(x) + \varphi\,(x+h)}{2h^2}$$

Similarly, any other differential operator can be converted to a difference operator. For now, let us substitute this into the Poisson Equation to get a difference equation

$$\frac{\varphi\,(x-h) - 2\,\varphi\,(x) + \varphi\,(x+h)}{2h^2} \approx f$$

$$\varphi\,(x) \approx -fh^2 + \frac{\varphi(x-h)}{2} + \frac{\varphi(x+h)}{2}$$

And for the Laplace equation, $f = 0$

$$\varphi\,(x) \approx \frac{\varphi(x-h)}{2} + \frac{\varphi(x+h)}{2}$$

Generalizing for any number of dimensions we get:

$$\varphi\,(x) \approx -fh^2 + \langle\varphi(x \pm h)\rangle$$

III. Monte Carlo Methods

Monte Carlo Methods are a class of computational solutions that can be applied to vast ranges of problems which rely on repeated random sampling to provide generally approximate solutions [6], [7]. Monte Carlo is a stochastic approach, in which series of simulations (trials), representing the analysed problem, with randomly selected input values, are performed. Among these trials, a specified number of properly defined successes is achieved. The ratio between the number of success trials to the number of all trials, scaled by dimensional quantity (e.g., area, function value) allows for the estimation of the unknown solution, providing the number of trials is large enough.

They can be understood as procedures for solving non-probabilistic problems by probabilistic type methods [8]. They are used in cases where analytical or deterministic numerical solutions don't exist or are too difficult to implement. In deterministic models, a particular state (either continuous or discrete) is uniquely assigned to input data and no element of randomness occurs. On the other hand, the probabilistic class of mathematical models specifies governing differential equations that combine one or more random variables (with assigned probability distribution function) with other non-random variables [9].

A. *Procedure for solving the Poisson Equation with Monte Carlo Method*

1) To apply MCM to the poisson equation, we first have to understand the concept of a random walk (RW). Consider a rectangular lattice that covers the domain *Ω* of *φ* with each lattice point at a distance h from its neighbours. Starting from a point *A* in this lattice, we move randomly to one of the closest neighbours of the point in each step, with equal probability.

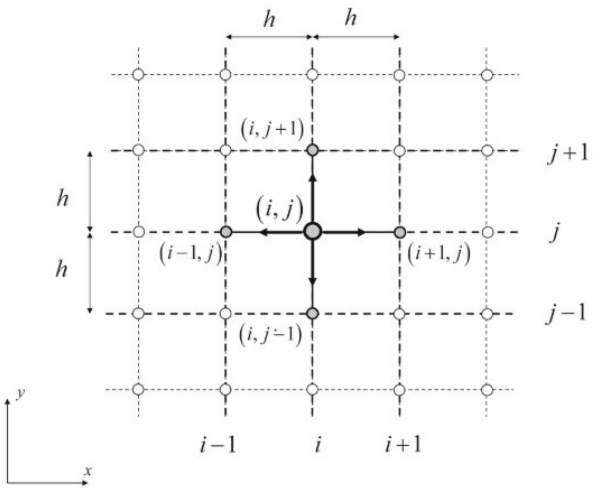

Figure 1: Lattice in a 2D space. Beginning from point (i, j) we randomly move to any of its closest four neighbours

2) We repeat this type of movement until we reach the boundary point *a* of the domain at which point the random walk *w* concludes. Compute a term *F* as

$$F(w) = \sum_{\substack{i \in interor\ points\ in \\ random\ walk}} f(i)h^2$$

3) Repeat this RW for a finite number of times (*N*) and calculate the probability of the walk starting at *A* and concluding at each boundary point *a* as *P_A(a)*

$$\mathrm{P_A(a)} = \frac{\begin{array}{c} Number\ of\ times \\ \underline{random\ walk\ ends\ at\ a} \end{array}}{N}$$

4) Using the known values of boundary condition g at each boundary point *a* compute the quantity *u(A)* as

$$u(A) = \sum_{a \in boundary\ points} g(a)\, P_A(a) - \sum_{for\ all\ w} F\,(w)$$

5) Calculate u for all points in the domain. This is the approximate solution to the differential equation.

$$\varphi(A) \approx u(A) \qquad A \in Lattice\ points\ in\ \Omega$$

B. *Proof that the above solution converges to φ*

The proof of this method utilizes FDE for approximating the Poisson Equation [10].

1) Suppose *A* lies on the boundary of the of the domain. Then since all random walks will terminate at A itself since it is the boundary point. This means that *$P_A(A)$* = 1 and *u(A)* = *g(A)*. Since we know that *g(A)*= $\varphi(A)$, we have *u(A)*=$\varphi(A)$
2) Now, for the other possibility, let *A* be an interior point. From the expression of *u* we can clearly deduce that *u(A)* will be the averge of u for all the neighbours of A minus *fh^2* term. So this is equal to the finite difference approximation to the Poisson equation.
3) So, for either case, *u* is equal to the finite difference equation which itself is an approximation of the Poisson equation. Hence as h → 0 and N → ∞, we can say that MCM solution converges to the Poisson Equation

C. *Advantages and other types of Monte Carlo Method*

Compared to other numerical methods, MCM is very suitable for parallelization. Each RW, being independent of each other, can be run in parallel for a point. Moreover, since the field at each point is calculated independently, it is also trivial to parallelize over all points.

The method described above uses **fixed step** sizes and predefined directions for random walks, and is also called standard random walk. Apart from the standard RW, two main types exist, namely **semi floating** RW (step size is fixed, though the move direction is not limited) and **full floating** random walk (step size is not preassigned and changes at each step, whereas the move direction is not limited)[11]. Some methods don't use mesh or lattice system at all and are thus termed as meshless methods [12].

IV. Experiments

Simulations are performed using an implementation of MCM written in the Python language with varying boundary conditions and charge distributions. Both one- and two-dimensional domains are considered. Unless specified, the standard random walk algorithm is used.

A. One dimentional capacitor (Laplace Equation)

A capacitor is constructed out of two infinite metal plates spaced 10 cm apart from one another. One plate is placed at 5V potential with respect to the other and the space between the plates is free of charges. The number of random walks is taken to be 400 and the number of lattice points is taken to be 15. The potential is measured in one dimension only.

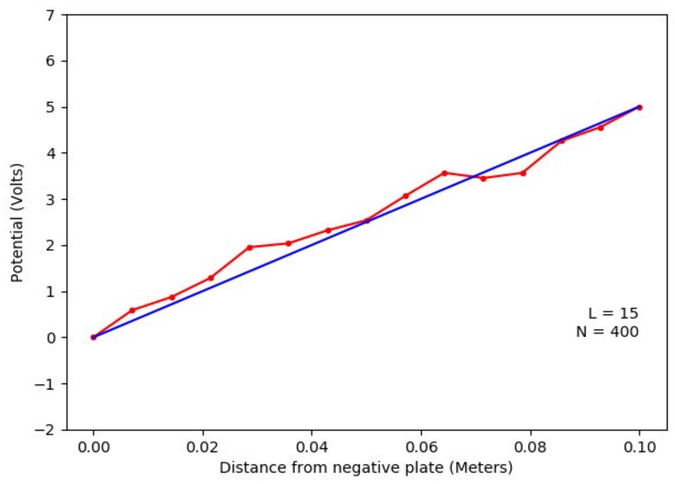

Figure 2: Calculated voltage (red) vs actual voltage in a one-dimensional capacitor

B. Variation of solution with number of random walks (Laplace Equation)

Keeping the same conditions as A, only the number of random walks is varied from 10 to 1000 with increments of 20 and the error between the analytical solution and the MCM solution is recorded and plotted. Additionally, a time lapse gif of the field vs distance graph is plotted in addendum 1.

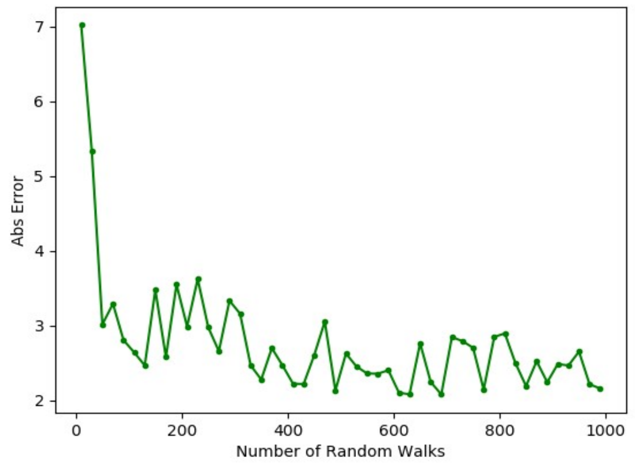

Figure 3: Absolute error vs number of random walks

C. Variation of solution with number of lattice points (Laplace Equation)

Similarly, only the number of lattice points is varied from 2 (trivial solution) to 30, keeping the number of random walks constant at 400. Since here the error does not show any meaningful quantity, a time lapse gif of the field vs distance graph is plotted and the results are shown in addendum 2.

D. One dimensional capacitor with a linear charge distribution (Poisson Equation)

A capacitor with the same physical parameters as A is considered but with a linearly varying charge distribution between the plates. The charge varies from *-1fC/m* at the negative plate to *+1fC/m* at the positive plate.

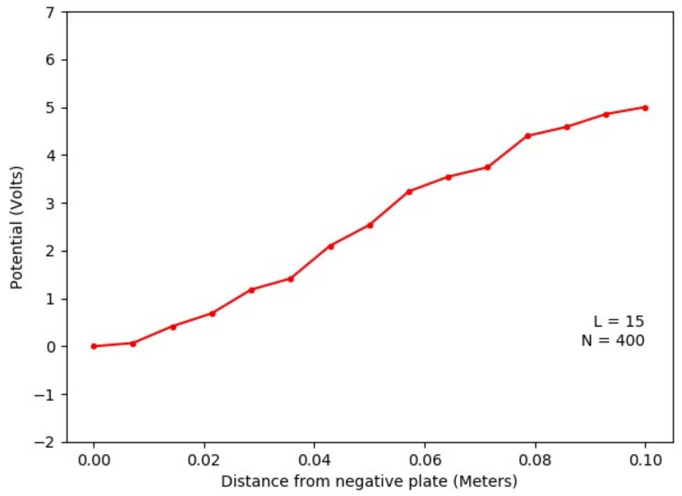

Figure 4: Calculated voltage in a one-dimensional capacitor with linear charge distribution

E. Two dimentional capacitor (Laplace Equation)

Here, the boundary conditions are changed slightly to allow for easier simulation. While the capacitor plates are charged as above, the other two boundaries of the two-dimensional 10cm x 10cm simulation region are taken to be at 0V to avoid having to consider the open boundary conditions within an actual finite capacitor.

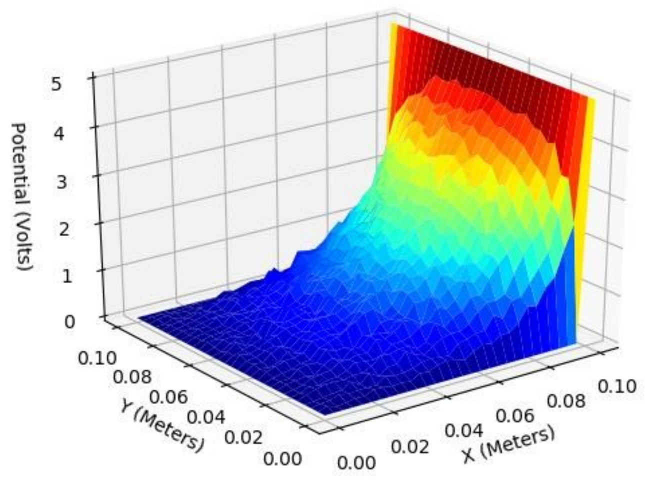

Figure 5: Surface plot of voltage in a two-dimensional capacitor

F. *Two dimentional metal box with a spherical charge in centre (Poisson Equation)*

Here, the capacitor is replaced with an uncharged metal box with all the walls at 0V. A spherical charge is placed at the centre of the box with a charge density of *-60 fC/m²*.

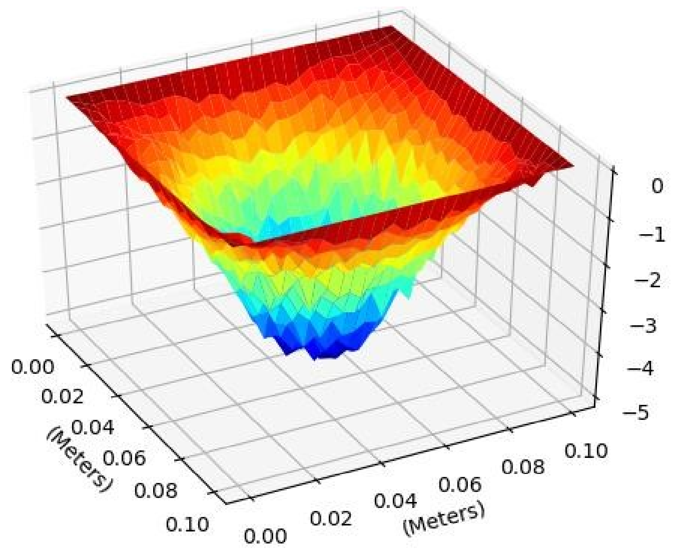

Figure 6: Surface plot of potential Metal box with negative charge at centre

G. *Two dimentional metal box with a two oppositely charged spheres (Poisson Equation)*

Two oppositely charged spheres are place at 10/3cm and 20/3cm from one wall of the box. Both spheres have a charge density of magnitude *60 fC/m²*.

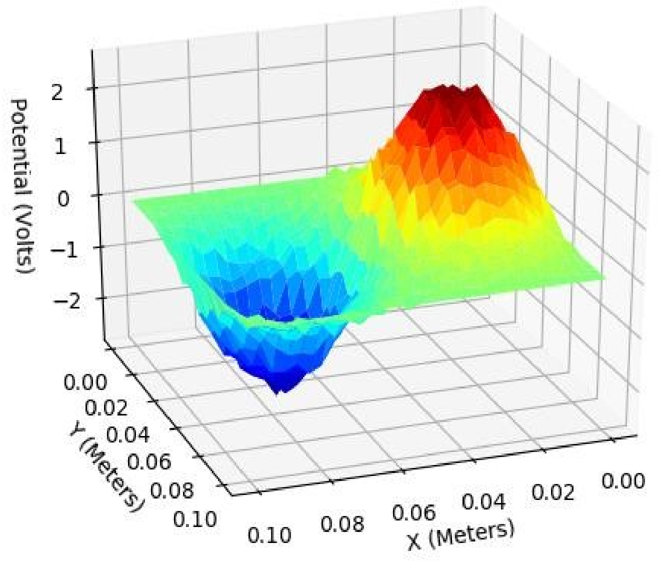

Figure 7: Surface plot of voltage in a metal box with two opposite charges inside

H. *Two dimentional metal box with a spherical charge at center using semi-floating random walk algorithm (Poisson Equation)*

The experiment F is repeated with a positive charge and the algorithm is changed to the semi floating MCM, where the distance in each step is random but only in 4 directions.

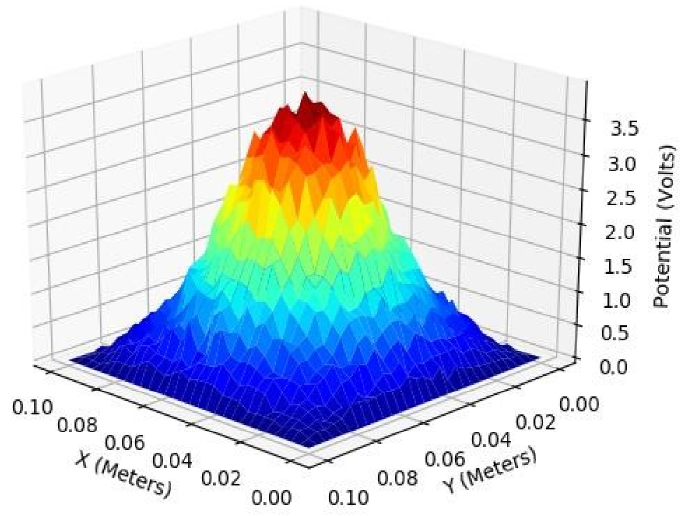

Figure 8: Surface plot of voltage in a metal box with positive charge at centre

I. *Two dimentional metal box with a spherical charge at center using full floating random walk algorithm (Poisson Equation)*

Here, experiment H is repeated but with the full floating MCM in which both the direction and distance of each random step is randomized.

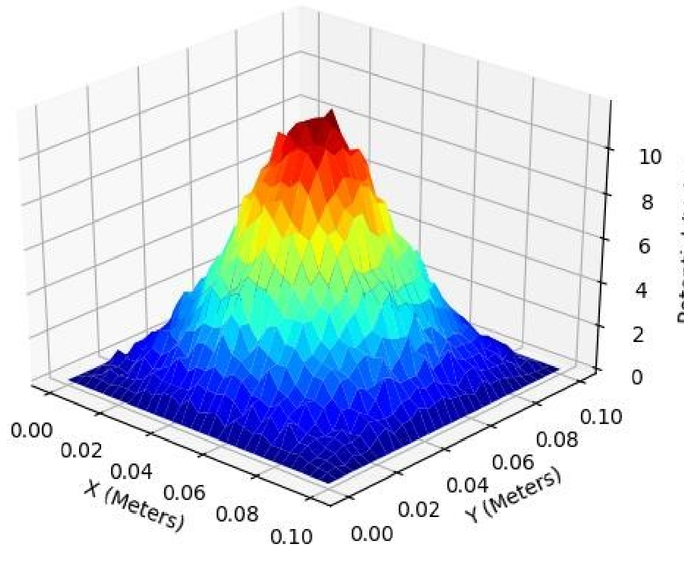

Figure 9: Surface plot of voltage in a metal box with positive charge at centre

V. Implementation Details

The code[1] for the implementation was written in the Python3 version of the language. The structure of the code is as follows

a) *Figures* contains the figures obtained from the simulations
b) *montecarlo.py* contains the code for A, B, C and D
c) *montecarlo2d.py* contains the code for the E, F and G
d) *montecarlo2d_floating.py* contains the code for H and I

VI. Conclusion and Discussion

For the simplest case, in experiment A, the expected linear curve was followed by the simulation. Similarly, the expected curve was followed in D. By varying the number of random walks in B, it was determined that the MCM solution converges to the analytical solution eventually as the absolute error gradually falls to 0. However, at around 400 random walks, increasing the number offers diminishing returns compared to increased computational time. Reduction in error is minimal after about 1000 walks. In the case of varying the number of lattice points in C, the solution approximates the equation better, however the time required for the computation also increased linearly.

In the case of the two-dimensional capacitor in E, the graph diverges significantly from the expected linear curve, for the field within a capacitor. However, it can clearly be seen that the cause of this is the fact that the field is clipped to zero at the edges of the simulation. Due to these boundary conditions, the curve takes on a 'saddle' shape. In F and G, the properties of free charges and the fields surrounding them are explored. In H and I, excessive additional processing time was used and the solutions were differing near the centre of the charges.

Thus, MCM was implemented in Python for solving Poisson and Laplace equations and several experiments were performed on various combinations of boundary conditions and charge distributions.

References

[1] "Poisson's Equation," *www*. [Online]. Available: https://en.wikipedia.org/wiki/Poisson%27s_equation.

[2] "Laplace's Equation," *www*. [Online]. Available: https://en.wikipedia.org/wiki/Laplace%27s_equation.

[3] "Uniqueness theorem for Poisson's equation." [Online]. Available: https://en.wikipedia.org/wiki/Uniqueness_theorem_for_Poisson%27s_equation.

[4] "Finite Difference Method," *www*. [Online]. Available: https://en.wikipedia.org/wiki/Finite_difference_method.

[5] J. Nagel, "Solving the Generalized Poisson Equation Using the Finite-Difference Method (FDM)," 2011.

[6] "Monte Carlo Method," *www*. [Online]. Available: https://en.wikipedia.org/wiki/Monte_Carlo_method.

[7] M. Mascagni, "Monte Carlo method for partial differential equations."

[8] R. Ahmed, "The Monte Carlo method and some applications," *LisMath Semin. IST Lisbon, 27th May 2016*, no. May, 2016.

[9] S. J. Farlow, *PDE for Scientists and Engineers*. 1993.

[10] J. F. Reynolds, "A Proof of the Random-Walk Method for Solving Laplace's Equation in 2-D," *Math. Gaz.*, vol. 49, no. 370, pp. 416–420, 1965.

[11] A. Haji-Sheikh and E. M. Sparrow, "The Floating Random Walk and Its Application to Monte Carlo Solutions of Heat Equations," *SIAM J. Appl. Math.*, vol. 14, no. 2, pp. 370–389, 1966.

[12] S. Milewski, "A Matlab software for approximate solution of 2D elliptic problems by means of the meshless Monte Carlo random walk method," *Numer. Algorithms*, vol. 83, no. 2, pp. 565–591, 2020.

[1] https://github.com/s-ankur/montecarlo-pde