Monte Carlo solutions to PDE

Physics Background

In Chapter 9 of Mark Newman and Lab 8, I learned to use different methods to find solutions to Laplace's equation in 2D which is in the form of:

$$u_{xx} + u_{yy} = 0$$

In fact, there are a lot of different equations in physics are can be represented as Laplace's equation. For example, we will be exploring the temperature distributions $T(x, y)$ in heat conducting materials in 2D. We can represent this problem as a Laplace's equation:

$$\nabla^2 T = 0$$

Meaning that in a steady state and close system, the overall temperature must be constant. We know from mathematics that there exists a unique solution to our Laplace's equation given the boundary conditions. However, in most cases, these exact solutions are not computer friendly and often hard to formulate. Therefore, this project aims to explore how to use Python to help us find solutions to Laplace equation by comparing the methods used in labs and the Monte Carlo method.

Computational Background

First allow me to remind the method used in the textbook which is Jacobi method, overrelaxation and Gauss-Seidel replacement. In essence, they start with initial boundary conditions, then traversing the array and updating the value by calculating the central difference, i.e.

$$u_{i,j} = \frac{1}{4}\left(u_{i+1,j} + u_{i-1,j} + u_{i,j-1} + u_{i,j+1}\right)^{1}$$

This works because we used relaxation method to transform the Laplace's equation in the form of the solution of a set of linear simultaneous equations that was derived from the solutions of a partial differential equation.

However, note that if we want to compute a single point value, we must first finish finding solutions for the whole grid. On the other hand, we can use Monte Carlo methods to find solution of Laplace's equations on a single point.

First remember that Monte Carlo is a type of algorithm or process that relies on repeating random sampling to obtain or approximate some discrete results. We can start by randomizing the process of calculating the average of a point's neighbors.

Instead of taking the average of all 4 neighbors, consider the following procedure: we start with the chose point, then continue to "walk" to continue either up, down, left, or right. Each direction has a probability of $\frac{1}{4}$. The walk will decide the direction randomly at each step. Once the walk reaches a boundary point, then we record the value of that boundary

[1] Equation (9.8) from Mark Newman

point. Refer to Figure 1 for an example walk. Then, we repeat the random walk N times and calculate the average frequency of each boundary point reached by the walks. Finally, we multiply the frequency and the corresponding boundary point value and take the sum to get our solution at the chose starting point.

The following section will explain why the method described above works as a method to find solutions of Laplace's equation at a chose point.

Why does it work

Intuitively, we can picture a square box with boundaries of all 0 K except the top which is 100 K. If we picked a point closer to the top boundary, we would expect a value closer to 100 K. On the other hand, if we picked a point closer to the bottom boundary, we would expect a value closer to 0 K. Therefore, if we start a random path from a point closer to the top boundary, the path is more likely to end at the top boundary. Thus, the random walk method's result is expected to be close to 100K but smaller than it.

Now we can formally to show how Monte Carlo method can be used to find solutions to a Laplace's equation.

If our selected starting point is a boundary point, then the walk will immediately stop and record that point's value. Then, one can simply follow that the averaged value will be its original value.

On the other hand, if our selected starting point is not a boundary point. Then we can first assume there is a uniform probability of $\frac{1}{4}$ for picking any of the four directions. Let x be a point with a, b, c, d as its neighbors. The expectation value of x can be formulated as:

$$E[x] = \frac{1}{4}E[a] + \frac{1}{4}E[b] + \frac{1}{4}E[c] + \frac{1}{4}E[d]$$

This is because every direction is equally likely, thus x's expectation value must be the average of a, b, c, d's expectation value.

If we further expand the expectation value of a, b, c, d, once we reach a boundary point, its expectation value is the discrete boundary point value. Then this value now became part of the expectation value of the previous point, and so on, tracing back to the starting point. That is the reason why we take the first boundary point's value for each walk.

Since our result is calculated by summing the product of averaged frequency and respective boundary values, we can represent the expectation value as:

$$E[x] = Avg(x) = \frac{1}{4}\left(Avg(a) + Avg(b) + Avg(c) + Avg(d)\right)$$

Where $Avg(a)$ can also be formulated as:

$$Avg(a) = b_0 f_0 + b_1 f_1 + \cdots + b_n f_n$$

Where b_i is the value at i-th boundary point, f_i is the averaged frequency of random walks reaching point i.

We can now see a close resemblance to the equation we arrived with the Gauss-Seidel method's equation. Therefore, we can use Monte Carlo method to find the solution to Laplace's equation.

General Approach and Pseudocode

First, we would want to find out the probabilities of going to each direction. If our Laplace's equation is in the form of $u_{xx} + u_{yy} = 0$, then the probabilities is $\frac{1}{4}$ as discussed.

However, I will demonstrate with a different Laplace's equation:

$$u_{xx} + \sin(x)\, u_{yy} = 0$$

First, use relaxation to reformulate the Laplace's equation, in other words, find the equation to express $u_{i,j}$. We can break down the Laplace's equation into:

$$u_{xx} = \left[u_{i,j+1} - 2u_{i,j} + u_{i,j-1}\right]/h^2$$

$$u_{yy} = \left[u_{i+1,j} - 2u_{i,j} + u_{i-1,j}\right]/k^2$$

$$\sin x = \sin x_j$$

Which can give us the following equation:

$$u_{i,j} = \frac{u_{i,j+1} + u_{i,j-1} + \sin x_j \left(u_{i+1,j} + u_{i-1,j}\right)}{2\left(1 + \sin x_j\right)}$$

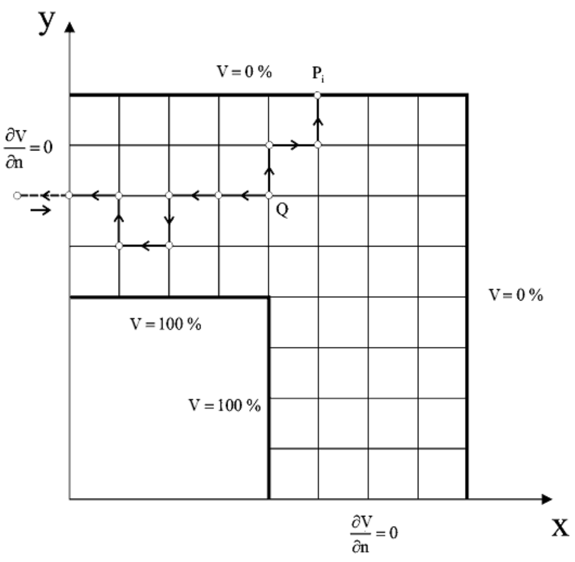
We can also double check that this equation can be randomized for Monte Carlo method by taking the sum of the coefficients:

$$\frac{1}{2\left(1 + \sin x_j\right)} + \frac{1}{2\left(1 + \sin x_j\right)} + \frac{\sin x_j}{2\left(1 + \sin x_j\right)} + \frac{\sin x_j}{2\left(1 + \sin x_j\right)} = 1$$

It indeed sums to 1 which matches the Law of Probability, therefore we can transform this such that choosing both $u_{i,j+1}$ and $u_{i,j-1}$ has a probability of $\frac{1}{2(1+\sin x_j)}$, while choosing both $u_{i+1,j}$ and $u_{i-1,j}$ has a probability of $\frac{\sin x_j}{2(1+\sin x_j)}$.

After obtaining the probabilities of choose each of the 4 neighbors, we can now pick a point as our starting point and start the random walk.

We can see from Figure 1 that demonstrates a random walk from a starting point Q. Note that it stops when the path hits a boundary point.


y
V = 0 %
P_i
$\frac{\partial V}{\partial n} = 0$
Q
V = 0 %
V = 100 %
V = 100 %
$\frac{\partial V}{\partial n} = 0$
X
Figure 1 Random Path demonstration

The pseudocode for the Monte Carlo method of finding solutions to Laplace's equation for the entire grid:

```
For every point (i,j) in Grid:
    For each walk in range of N (Complete N random walks):
        Create an array to store counts of boundary points reached
        While the path has not reached a boundary point:
            Generate a probability at each step with the suitable
            probabilities assigned to each direction
            If Up then go to the point above the current point (i-1,j)
            If Down then go to the point below the current point (i+1, j)
            If Left then go to the left of the current point (i, j-1)
            If Right then go to the right of the current point (i, j+1)
        Record the boundary point reached
    Take the sum of the averaged counts of boundary points reached multiplied by
    the boundary value
    Save the sum as the value at point (i,j)
```

It is very important to note that the overall runtime of this method is dominated by the while loop, since we cannot decide when will the path reach a boundary point. Theoretically, the path can end up in a cycle path depending on the probabilities of each direction. Nonetheless, the process of finding everything can be very long.

Therefore, the best use of this method is to find solution for a particular point. So, we do not need to loop through every point of the grid. This proves to be much more efficient to find a single point as we can see in the coming sections.

Example Problem

In the section, we will examine the difference between using Gauss-Seidel relaxation with replacement and overrelaxation and Monte Carlo method.

The problem that we want to solve is that we want to find the distribution of temperature under steady state conditions and in the absence of heat sources. To simplify the comparison, the boundary condition is that only the top boundary has initial temperature of 25 ^oC, and the rest are 0 ^oC.

Using Gauss-Seidel with replacement and relaxation method gives us the following result:

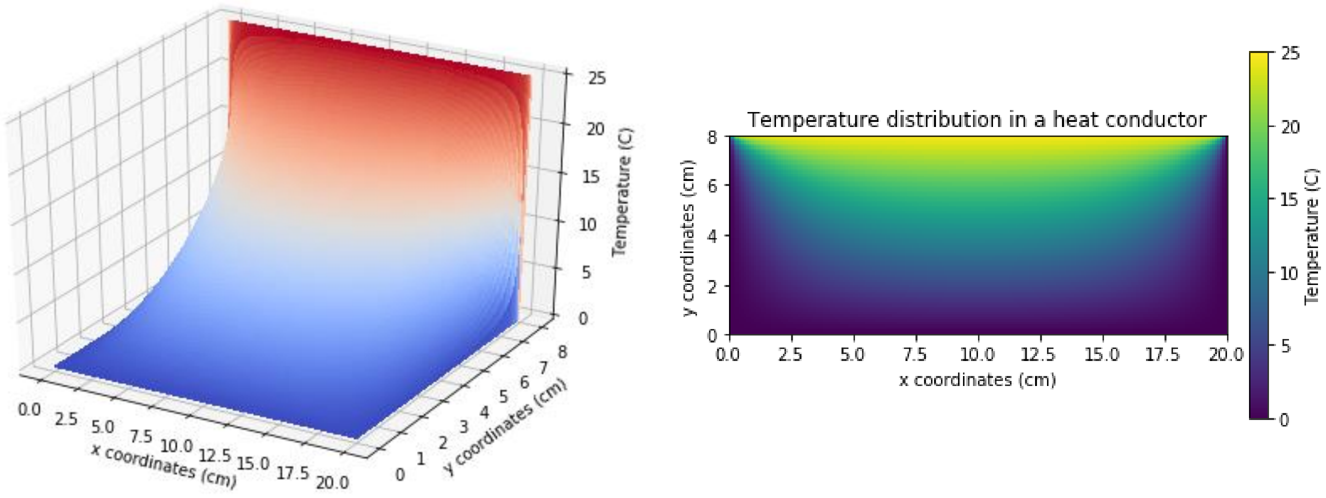


Figure 2 results using Gauss-Seidel Method

We can now find solution at specific points, i.e. temperature at the point x = 2.5 cm, y = 1 cm is 1.639 ^oC.

Using the Monte Carlo method describe above, use the same initial boundary condition to calculate the temperature at x = 2.5 cm, y = 1 cm and the temperature for the whole grid.

Solution

In the code attached, note that I have written 3 functions which is different than the pseudocode. However, I just took each loop out for clarity; **`monteWalkto`** performs a single walk, **`walkToBoundary`** repeats the random walk until it hits the boundary, **`walkNtimes`** repeats N walks and calculate the approximate value by taking the average of boundary points hit. You can also see my functions as a breakdown of the pseudocode provided.

We can simply use the last function to get approximate solution to the Laplace's equation at x = 2.5 cm, y = 1 cm. After running several times, I obtain the temperature as 1.75 ^oC which is pretty close to the value we obtained from the finite-difference method. However, note that the result's accuracy is dependent on the number of walks you take. With higher number of walks, the result would be more accurate. This is due to the nature of any experiment taken by random inputs. Also, note that the variance between each N-runs can be high, this can be reduced if we perform multiple N-runs and take the average again.

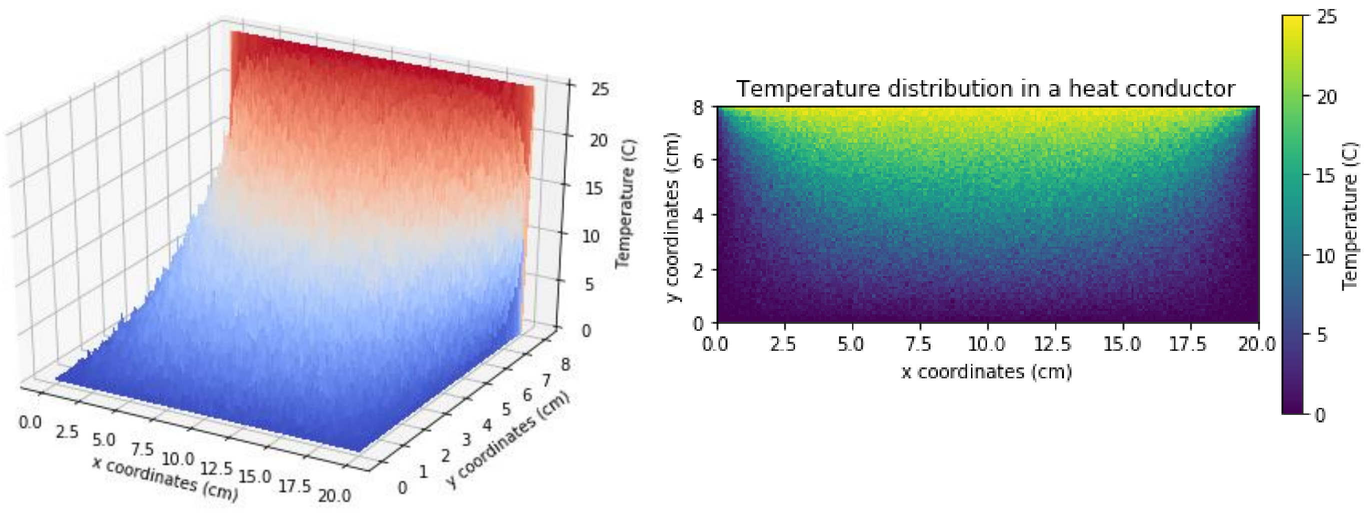


Figure 3 results using Monte Carlo method with N=100

To generate the graph similar to Figure 2, we must perform N walks for each point. Note this will take a long time since the runtime is mostly dependent on the termination of while loop for the walk to a boundary point, which can be theoretically never finish running. It took me about 1 hour to finish running the entire grid which consist of $81 \times 201 = 16281$ points. However, we only need to perform the random walk on non-boundary points, meaning only $79 \times 199 = 15721$ points which is still relativity large. The result is shown above:

We can see that the general distribution of heat is relatively the same as the distribution we obtained from Figure 2. However, note that there are many artifacts or random noise in the graph. This can be easier to visualize in the 3D graphs. We can see the surface of Figure 2 is much smoother than the surface of Figure 3. This is again due to the nature of Monte Carlo process, since our random paths is based on random choices at each point, meaning there are chances that our approximation might not be correct. Therefore, in order to increase the precision, we can increase the size of N which increases the number of times of repeating random paths taken at each point. However, this will significantly increase the total run time as mentioned above. To conclude our results, we can see that the overall distribution is still within our expectation (Figure 2).

Discussion

As we can see from our example problem, the Monte Carlo method is not suited for finding solutions of Laplace's equation for the entire grid due to the long computation time. Moreover, we can calculate the percentage error of the results from Monte Carlo method using the results from Gauss-Seidel method as reference ground truth. The percentage error is about 7.76% which can be acceptable in some situation. However, I do not see a point in spending more computation time for a less accurate method.

On the other hand, the Monte Carlo method excels at approximating the solution of Laplace's equation at a given point since we can directly start the paths from the chosen point. We don't even need to know the value of the starting point, instead the boundary value is all the information we need. In fact, the time needed for the Monte Carlo method to approximate the point value in the sample problem is only 0.343 s with N = 300. On the other hand, the Gauss-Seidel method needs 42.68 s to find the value at the same point after finding the values of the entire grid. Thus, we can repeat the Monte Carlo method for about $\frac{42.68}{0.343} \approx 120$ times and get a higher accuracy. Note that the approximations by the Monte Carlo has a certain amount of variance between each run. However, the overall time reduced is far more beneficial if accuracy is not a major concern.

Therefore, the Monte Carlo method is very good at giving approximates of solution to the Laplace's equation at a point or small region of the grid. This can be applicable for situations where we only care about the solutions at certain areas of the grid. For example, if we only want to find the temperature surrounding myself inside a room.

Note

On the last bit of the code, I have added a method to calculate the probabilities mentioned in the General Approach section. I have chosen numpy.random.choice which takes in a list of choices and the respective probabilities. Therefore, once you derived the probabilities for your own problem, i.e. another form of Laplace's equation. Then you can use similar method and perform the Monte Carlo method of approximating the solution at a point.

Conclusion

We can conclude that the Monte Carlo method is a great method for approximating solutions of Laplace's equation for a given point or small region of the grid compared to the finite difference methods such as Gauss-Seidel method. However, one should not use this method to find the solution of the entire grid as this is shown to be ineffective.

References

Kokorus, Mario & Sokolija, Kemo. (2013). Solving Laplace Differential Equation Using Markov Chains in Monte Carlo Method. 2013 24th International Conference on Information, Communication and Automation Technologies, ICAT 2013. 10.1109/ICAT.2013.6684079.

Daria Apushkinskaya. (2015). Lecture 21, PDE and Boundary-Value Problems, Saarland University. Accessed from https://www.math.uni-sb.de/ag/fuchs/PDE14-15/pde14-15-lecture-21.pdf