

PDE and Boundary-Value Problems

Winter Term 2016/2017

Lecture 21

Saarland University

17 February 2017

Purpose of Lesson

- To explain the basic philosophy of Monte Carlo methods and suggest how they can be used to solve various problems.
- To show how random games (Monte Carlo methods) can be designed whose outcomes approximate solutions to differential equations. A specific game (tour du wino) is described whose outcome is the finite-difference approximation to a Dirichlet problem inside a square. The game is extended to include solutions to other problems as well.

Monte Carlo Methods (an Introduction)

- The basic idea here is that **games of chance** can be played (generally on a computer) whose outcomes approximate solutions to real-world problems.
- First of all Monte Carlo methods are procedures for solving **nonprobabilistic-type problems** (problems whose outcome does not depend on chance) by **probabilistic-type methods** (methods whose outcome depends on chance).
- The general philosophy of Monte-Carlo methods is illustrated on the next page.

Probabilistic game


The outcome of the game \hat{p}

(Like the fraction of heads in tossing a coin, throwing darts, and so forth)

Deterministic problem

The answer to the problem is P

(Like evaluation an integral, solving a PDE, and so forth)

Outcome = \hat{P}  Answer = P
Approximation

Evaluating an Integral

- To illustrate the method, suppose we wanted to evaluate the integral

$$I = \int_a^b f(x) dx$$

(a nonprobabilistic problem).

- To use the Monte Carlo method, we would devise a game of chance whose outcome was the value of the integral (or approximates the integral).
- There are, of course, many games that we could devise; the actual game we used would depend on the accuracy of the approximation, simplicity of the game, and so on.

Evaluating an Integral (cont.)

- An obvious game to evaluate the integral would be throwing darts at the rectangle

$$R = \{(x, y) : a \leq x \leq b, 0 \leq y \leq \max f(x)\}.$$

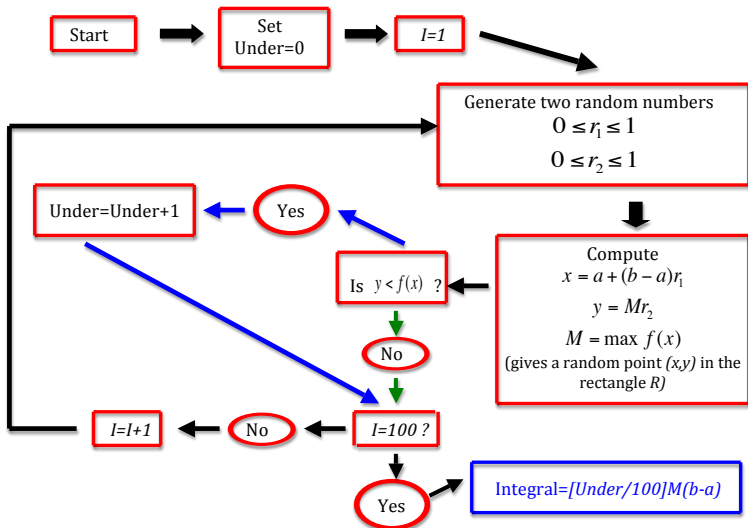
- It's fairly obvious that if we randomly toss 100 or so darts at the rectangle R enclosing the graph, then the fraction of darts hitting below the curve times the area of R will estimate the value of the integral.
- Hence, our outcome of the game

$$\hat{I} = [\text{fraction of tosses under } f(x)] \times (\text{area of } R)$$

is used to estimate the true value of the integral I .

Evaluating an Integral (cont.)

- To carry out the actual computation on a computer, we would have to generate the sequence of random points in some way (we'll discuss this shortly) and have the computer play the dart tossing game.
- Let us assume for the time being that we have a sequence of random points. The flow diagram on the next page illustrates how the computer would attack this problem. (See flow diagram to evaluate $\int_a^b f(x)dx$ by the Monte Carlo method (100 tosses)).



Random Numbers

- Before going on to apply this technique to the solution of PDEs, we discuss the important topic of random numbers.
- In the integral just considered, it was necessary to generate a sequence of random points $P_i = (x_i, y_i)$ that fell inside rectangle R . In other words, the x -coordinate would have to be a random number in the interval $[a, b]$, while y_i must be in $[0, M]$.

Random Numbers (cont.)

- To find random numbers inside specific intervals, we start with a basic sequence of random numbers r_i (uniformly distributed) inside $[0, 1]$.

- It's obvious then that if we want a random number x_i inside $[a, b]$, we just compute

$$x_i = a + (b - a)r_i$$

- So everything comes down to the question, how do we generate a sequence of random numbers $\{r_i, i = 1, 2, \dots\}$ uniformly distributed in $[0, 1]$.

Residue Algorithm for Generating Random Numbers

To generate a sequence of **random integers** (between 0 and P), we use the **residue algorithm**.

- 1 Pick the first random integer any way you like between 0 and P (P was picked in advance).
- 2 Multiply this random integer by some fixed integer M (picked in advance).
- 3 Add to that product another fixed integer K (picked in advance).
- 4 Divide the resulting sum by P and pick the **remainder** as the new random integer. Now go back to step 2 and repeat steps 2-4 until you have enough random integers.

- This residue algorithm can be written as

$$r_{i+1} \equiv (Mr_i + K) \bmod P \quad i = 0, 1, 2, \dots$$

which says, if we are given a random integer r_i , then to compute a new one r_{i+1} , we multiply by M , add K , divide by P , and pick the remainder.

Remarks

- If we choose, for example, $P = 100$ in our random-number generator, the remainders will be one of the integers $0, 1, 2, \dots, 99$, and, hence, our entire process will start repeating before long.
- In fact, our random numbers might be

15, 71, 43, 7, 43, 7, 43, 7, (Cycle of two numbers)

and, hence, our method is no good.

- The ideal situation is to generate the entire residue class $\{0, 1, 2, \dots, 99\}$ in a **random fashion** before starting to repeat.

Remarks (cont.)

- It can be proven mathematically that if the numbers M , K , and P are chosen according to certain rules, then no matter how we pick the first random number r_0 , the algorithm will generate the entire residue class.
- So, if we pick P very large (like 2^{40}), we are assured that (for practical purposes) the process will never repeat.
- It is possible to generate random samples from various statistical distributions other than the uniform distribution $f(x) = 1$, $0 < x < 1$ (the usual random-generator).
- Computer programs are available to generate random samples from the binomial, gamma, normal, and many other distributions.

Monte Carlo Solution of PDEs

We show how we can design a game to approximate solution of the Dirichlet problem:

Problem 21-1

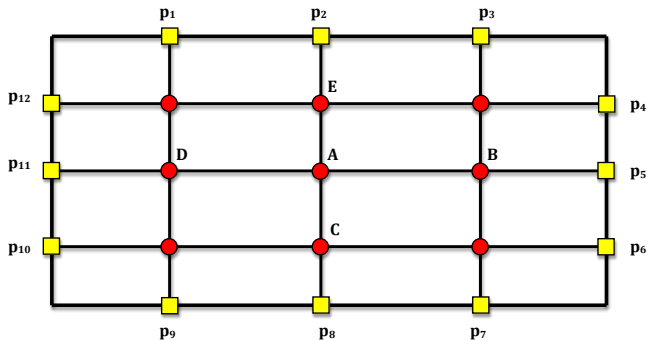
To find a function $u(x, y)$ that satisfies

$$\text{PDE: } u_{xx} + u_{yy} = 0, \quad 0 < x < 1, \quad 0 < y < 1$$

$$\text{BC: } u(x, y) = g(x, y) = \begin{cases} 1, & \text{On the top of the square} \\ 0, & \text{On the sides and} \\ & \text{bottom of the square} \end{cases}$$

To illustrate the Monte Carlo method in this problem, we introduce a game called [tour du wino](#). To play it, we need a board on which grid lines are drawn (see next page).

Board for tour du wino



A = starting points for game

■ = end points p_i

● = interior grid points

g_i = reward for ending at p_i

How Tour du Wino is Played

1. The wino starts from an arbitrary point (point A in our case).
2. At each stage of the game, the wino staggers off randomly to one of the four neighboring points. (In our case, the neighbors of A are B , C , D and E , and the probability of going to each of these neighbors is $1/4$.)
3. After arriving at a neighboring point, the wino continues this process wandering from point to point until eventually hitting a **boundary point** p_j . He then stops, and we record that point p_j . This completes one **random walk**.

How Tour du Wino is Played (cont.)

4. We repeat steps 1-3 until many random walks are completed. We now compute the fraction of times the wino had ended up at each of the boundary points p_i .
5. Suppose the wino receives a reward g_i (g_i is the value of the BC at p_i) if he ends his walk at the boundary point p_i , and suppose that the goal of the game is to compute his average reward $R(A)$ for all this walks. The average reward is

$$R(A) = g_1 P_A(p_1) + g_2 P_A(p_2) + \cdots + g_{12} P_A(p_{12})$$

The game is completed with the determination of $R(A)$.

Probability of Random Walk Ending at p_i

Boundary point p_i	$P_A(p_i) =$ fraction of times the wino ends at p_i	$g_i =$ reward for ending at p_i
1	0.04	1
2	0.15	1
3	0.03	1
4	0.06	0
5	0.17	0
6	0.05	0
7	0.06	0
8	0.15	0
9	0.03	0
10	0.06	0
11	0.16	0
12	0.04	0

Reason for Playing Tour du Wino

It turns out that the average reward is the approximate solution to our Dirichlet problem at Point A . This interesting observation is based on two facts.

- 1 Suppose the wino started at a point A that was on the **boundary** of the square. Each resulting random walk ends immediately at that point, and the wino collects the amount g_j . Thus, his average reward for starting from a boundary point is also g_j .
- 2 Now suppose the wino starts from an interior point. Then, the average reward $R(A)$ is clearly the average of the four average rewards of the four neighbors

$$R(A) = \frac{1}{4} [R(B) + R(C) + R(D) + R(E)]$$

- We ask why the wino's average reward $R(A)$ approximates the solution of the Dirichlet problem at A . We have seen that $R(A)$ satisfies two equations

$$R(A) = \frac{1}{4} [R(B) + R(C) + R(D) + R(E)] \quad (A \text{ an interior point})$$

$$R(A) = g_i \quad (A \text{ a boundary point})$$

- If we let g_i be the value of the boundary function $g(x, y)$ at the boundary point p_i , then our two equations are exactly the two equations we arrived at when we solved the Dirichlet problem by the finite-difference method.

- That is, $R(A)$ corresponds to $u_{i,j}$ in the finite-difference equations

$$u_{i,j} = \frac{1}{4} (u_{i-1,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1}) \quad (i,j) \text{ an interior point}$$

$$u_{i,j} = g_{i,j} \quad g_{i,j} \text{ the solution at a boundary point } (i,j)$$

- Hence, $R(A)$ will approximate the true solution of the PDE at A .

Solution of Laplace's Equation by the Monte Carlo Method

These rules give the solution at one point inside the square.

1. Generate several random walks starting at some specific point A and ending once you hit a boundary point. Keep track of how many times you hit each boundary point.
2. After completing the walks, compute the fraction of times you have ended at each point p_i . Call these fractions $P_A(p_i)$.
3. Compute the **approximate solution** $u(A)$ from the formula

$$u(A) = g_1 P_A(p_1) + g_2 P_A(p_2) + \cdots + g_N P_A(p_N)$$

where g_i is the value of the function at p_i and N is the number of boundary points.

Solution to a Dirichlet Problem with Variable Coefficients

The game tour du wino can be modified to solve more complicated problems, as in the following example.

Problem 21-2

To find a function $u(x, y)$ that satisfies

$$\text{PDE: } u_{xx} + (\sin x) u_{yy} = 0, \quad 0 < x < \pi, \quad 0 < y < \pi$$

$$\text{BC: } u(x, y) = g(x, y) \quad \text{on the boundary of the square}$$

- To solve Problem 21-2, we replace u_{xx} , u_{yy} and $\sin x$ by

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

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

$$\sin x = \sin x_j$$

and plug them into the PDE.

- Making these substitutions and solving for $u_{i,j}$ gives

$$u_{i,j} = \frac{u_{i,j+1} + u_{i,j-1} + \sin x_j (u_{i+1,j} + u_{i-1,j})}{2(1 + \sin x_j)} \quad (21.1)$$

- Look carefully at (21.1). The coefficients of $u_{i+1,j}$, $u_{i-1,j}$, $u_{i,j+1}$, and $u_{i,j-1}$ are positive and sum to one. In other words, $u_{i,j}$ is a **weighted average** of the solutions at the four neighboring points.
- Hence, we modify our game so that the wino doesn't stagger off to each neighbor with probability $1/4$, but, rather, with a probability equal to the coefficients of the respective term.

- In other words, if the wino is at the point (i, j) , he then goes to the point:

$$(i, j + 1) \text{ with probability } \frac{1}{2(1 + \sin x_j)}$$

$$(i, j - 1) \text{ with probability } \frac{1}{2(1 + \sin x_j)}$$

$$(i + 1, j) \text{ with probability } \frac{\sin x_j}{2(1 + \sin x_j)}$$

$$(i - 1, j) \text{ with probability } \frac{\sin x_j}{2(1 + \sin x_j)}$$

- Other than this slight modification, the game is exactly the same as before.

Remarks

- Observe that once the fractions $P_A(p_i)$ (the fraction of times the wino ends at p_i) are computed, we can then find the solution $u(A)$ for any other boundary conditions g_i just by plugging the $P_A(p_i)$ into the formula

$$u(A) = g_1 P_A(p_1) + g_2 P_A(p_2) + \cdots + g_N P_A(p_N).$$

That is, we don't have to recompute new random walks.

- In many cases, a researcher wants to find the solution of a PDE at only one point. If the boundary is fairly complicated and if the PDE involves 3 or 4 dimensions, then Monte Carlo methods may come to rescue.

Remarks (cont.)

- In fact, Monte Carlo methods were originally developed to study difficult neutron-diffusion problems that were impossible to solve analytically.