Estimation of π using Monte-Carlo method

06 November 2023 23:21

Theory

Monte Carlo methods, or Monte Carlo experiments, are a broad class of computational algorithms that rely on repeated random sampling to obtain numerical results. The underlying concept is to use randomness to solve problems that might be deterministic in principle.

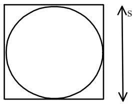
They are often used in physical and mathematical problems and are most useful when it is difficult or impossible to use other approaches.

Procedure

To estimate the value of π using Monte-Carlo experiments we generate random points and check if they fall inside the square and a circle inscribed in the square.

Then we compute the ratio of points inside the circle and points inside the square to estimate the value of π.

Let the side of square be 's' then the radius of circle inscribed is 's/2'.



probability that the point resides in the square ∝ Area of square = s^2

probability that the point resides in the circle ∝ Area of circle = $\frac{\pi s^2}{4}$

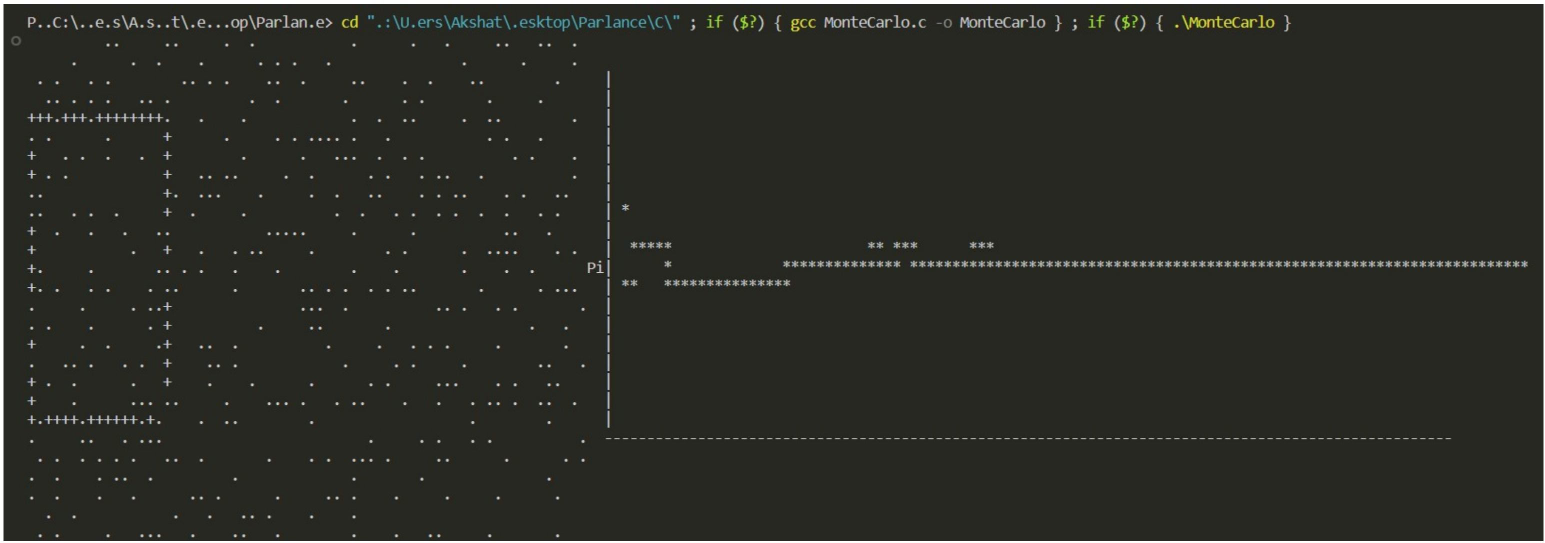
So, 4/(number of points in square/number of points in circle) = π

The code is completely in C and library used is "windows.h" . In the code, I've displayed the square and the random points generated in the form of an animation.

CODE:

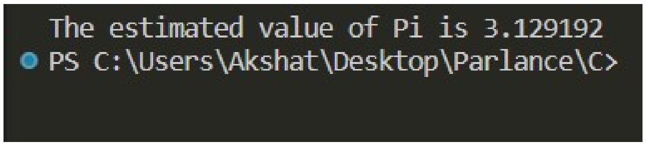
```c
#include <stdio.h>
#include <time.h>
#include <math.h>
#include <stdlib.h>
#include <windows.h>
int main()
{
  int abscissa;      // abscissa of random point generated
  int ordinate;      // ordinate of random point generated
  int in_circle = 0; // number of random points inside the circle of radius side/2
  int in_square = 0; // number of random points inside the square of side = side;
  DWORD dwWritten = 0;
  int side = 8000; // side of square
  int i = 0;
  // area assumed to start from (0,0) to (side,side)
  srand(time(NULL)); // seeding the random number generator
  HANDLE hout = GetStdHandle(STD_OUTPUT_HANDLE); // using windows api functions to display points at specific
coordinates
  if (hout == INVALID_HANDLE_VALUE)
  {
    return EXIT_FAILURE;
  }
  // drawing square
  for (int i = 0; i < side / 500; i++) // dividing by 500 such that it displays on the screen as screen width and height
are much less compared to RAND_MAX
  {
    const char *line = "+";
    COORD top = {i, 5};
    COORD left = {0, i + 5};
    COORD bottom = {i, (side / 500) + 5};
    COORD right = {side / 500, i + 5};
    WriteConsoleOutputCharacter(hout, line, 1, top, &dwWritten);
    WriteConsoleOutputCharacter(hout, line, 1, left, &dwWritten);
    WriteConsoleOutputCharacter(hout, line, 1, bottom, &dwWritten);
    WriteConsoleOutputCharacter(hout, line, 1, right, &dwWritten);
  }
  // drawing x and y axes
  while (i != 20)
  {
    COORD y = {68, i + 3};
    COORD pi = {66, i + 3};
    WriteConsoleOutputCharacter(hout, "|", 1, y, &dwWritten);
    if (i + 3 == 13)
    {
      WriteConsoleOutputCharacter(hout, "Pi", 2, pi, &dwWritten);
    }
    i++;
  }
  i = 0;


  while (i != 100)
  {
    COORD x = {i + 68, 22};
    WriteConsoleOutputCharacter(hout, "-", 1, x, &dwWritten);
    i++;
  }
  i = 0;

  while (i != 30000) // 30000 is the number of samples
  {
    abscissa = rand();
    ordinate = rand();
    const char *point = ".";
    COORD c = {abscissa / 500, ordinate / 500}; // dividing by 500 such that it displays on the screen as screen width
and height are much less compared to RAND_MAX
    WriteConsoleOutputCharacter(hout, point, 1, c, &dwWritten);
    //Sleep(10); // we can vary this according to speed of animation we want
    double radius = sqrt(pow((double)abs(abscissa - (side / 2)), 2) - pow((double)abs(ordinate - (side / 2)), 2)); //
calculating the radius of circle
    if (abscissa < side && ordinate < side)
    {
      in_square++;
    }
    if (radius < (double)side / 2)
    {
      in_circle++;
    }
    // to draw graph
    COORD coordinate = {70 + (i/10), (int)((double)4 / ((double)in_square / (double)in_circle)) + 10};
    WriteConsoleOutputCharacter(hout, "*", 1, coordinate, &dwWritten);
    i++;
  }
  system("cls");
  Sleep(100);
  printf("The estimated value of Pi is %lf", (double)4 / ((double)in_square / (double)in_circle));
  return 0;
}
```

Output:



```
P..C:\..e.s\A.s..t\.e...op\Parlan.e> cd ".:\U.ers\Akshat\.esktop\Parlance\C\" ; if ($?) { gcc MonteCarlo.c -o MonteCarlo } ; if ($?) { .\MonteCarlo }
```

Note: the square displayed doesn't look like a square because the x and y offsets vary in the terminal screen but number of symbols used to depict the side of the square is equal i.e. side/500.



```
The estimated value of Pi is 3.129192
PS C:\Users\Akshat\Desktop\Parlance\C>
```

Note: More number of samples can be taken to get more accurate value.