Calculating the value of Pi (π): A Monte Carlo Scheme in Scilab

R. Moulick

Department of Physics
School of Chemical Engineering and Physical Sciences
Lovely Professional University
Phagwara, Punjab – 144411.

Abstract

Scilab is an open source software widely used in computational studies these days. In this article the aim is to demonstrate the working of the software by calculating the value of pi (π). The calculation is based on the Monte Carlo algorithm. With higher number of runs the accuracy of the calculation increases. The Calculations are done with uniformly generated random numbers.

I. Introduction

The value of pi is a well-known quantity having a value of 3.14159265359. The digits after the decimal value however do not stop here and the chain is indefinitely long. By definition, the quantity pi is the ratio of the circumference to the diameter of a circle. However, calculating the value of pi using many different techniques is a mathematical fun in itself.

In this article, we intend to describe a procedure of calculating the value of pi using a standard Monte Carlo scheme. The problem is also well known and often it is used as a beginner problem in learning Monte Carlo simulation. The calculation is based on the theories of probability. The detailed mathematical analysis will follow in the next section of this article. Scilab, on the other hand is a free software which is gaining popularity because of its wide field of application and prompt calculating power [1]-[4]. Scilab provides the power of open source exploration along with high scientific accuracy and accountability.

In the following, section II deals with the detailed mathematical structure of the problem, section III deals with writing the code in Scilab platform and the results have been discussed in section IV. Finally, the paper is concluded in section V.

II. Mathematical Analysis of the problem

In order to frame the mathematical part of the problem let us refer to the figure 1. The figure shows a circle embedded in a square. Let the radius of the circle be 'r', then, the sides of the square will be '2r'.

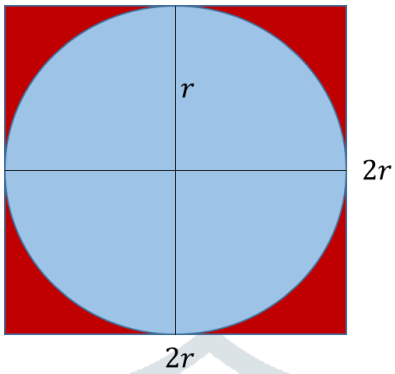

Figure 1: Circle embedded in a square

Monte Carlo methods are concerned with the theories of probability and computationally this is associated with the generation of the random numbers [5]. Let us assume, the rain drops are allowed to fall on the square at random. In that case, few raindrops will fall on the circle and others will fall outside it. However, it is ensured that each rain drop is fallen within the square area and not outside it. Thus, we get the probability of the raindrops falling on the circle is,

$$\mathrm{P} = \frac{\text{Area of the circle}}{\text{Area of the square}}$$

Therefore,

$$P = \frac{\pi r^2}{4r^2}$$

In this way we get,

$$\pi = 4 \times P$$

Thus, it is seen that the value of pi is four times the probability of raindrops falling on the circle. In the following section, detailed coding exercise is discussed.

III. Coding Exercise

While coding in Scilab, we generate uniformly distributed random numbers. The flowchart is shown in figure 2. Let C denote the number of random points falling within the circle and S counts for all the points falling

on the square. In the beginning, both of these variables are initialized to zero. Then, generate two random points 'r_x' and 'r_y' and verify whether the condition $r_x^2 + r_y^2 \leq r^2$, where r is the radius of the circle. If the condition is satisfied, the points are within the circle, else they fall on the square only. Consequently, the count of C is increased, subject to the satisfaction of the condition; the count of S is increased in each iteration step however. The Scilab code is presented below in Figure 3:

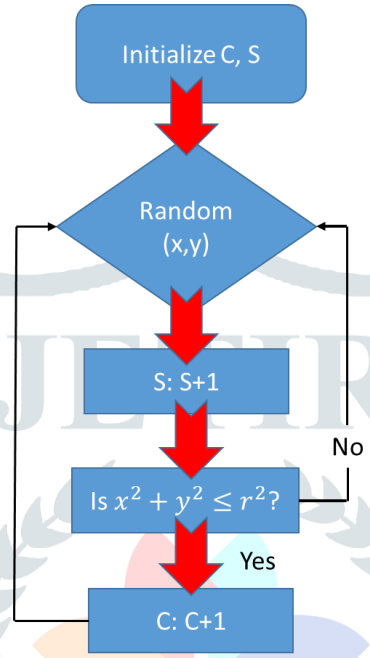

Figure 2: Flowchart for solving the problem

```
clc; clear;
r = 1; // radius of the circle chosen
c = 0; // Initialize C
s = 0; // Initialize S
Maxstep = 5000;

for it = 1:Maxstep

    rx = r*rand();
    ry = r*rand();
    r2 = rx^2 + ry^2;

    if (r2<=r^2)then
        c=c+1;
        plot(rx,ry,'b.','MarkersiZe',5)
    else
        plot(rx,ry,'r.','MarkersiZe',5)
    end
    s = s+1;
    if(modulo(it,100)==0)then
        disp(it,'it=')
    end
end

pi = 4*(c/s);
disp(pi,'Pi=')
```

Figure 3: The Scilab code

IV. Results and Discussions

Figure 4 shows the results of the run for 5000 iteration steps. The dots represent the points, as generated by r_x and r_y. The blue dots represent those points for which the condition is satisfied whereas, the red dots represent the points for which the condition is not satisfied. We can see a clear demarcation whereby the blue boundary is separated from the red one. The value of pi is determined from the ratio $4 \times \left(\frac{C}{S}\right)$. The ultimate value from this run has been found to be 3.1232.

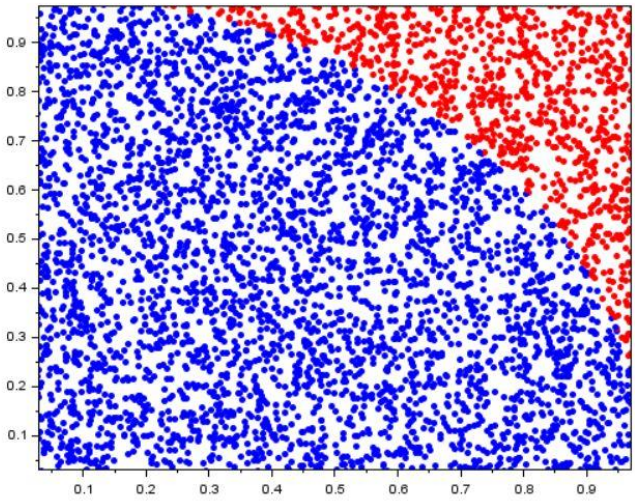

Figure 4: The random number distribution over the square

V. Conclusions

Scilab is a very effective software for doing important scientific calculation. One such example has been explained in this article. The problem has been tackled using uniformly distributed random numbers, however, one may also try it out with normally distributed random numbers. It has been observed that increasing the 'Maxstep' variable in the code increases the precision of the value of pi.

References:

[1] Stephen L. Campbell, Jean-Philippe Chancelier and Ramine Nikoukhah, Modeling and Simulation in Scilab/Scicos, Springer, (2006).

[2] Scilab for beginners, Scilab Enterprise, (2013).

[3] Scilab Reference Manual, Scilab Group.

[4] Michael Baudin, Programming in Scilab, (2011).

[5] Gilberto E. Urroz, Probability Distributions with SCILAB, (2001).