Monte Carlo simulations to estimate Pi

Rik King
Peter Anderson

Abstract

A simple Monte Carlo simulation, using functions from various R packages is explored for calculating π using a variety of polygons to circumscribe a unit circle. Starting with a square, higher-order polygons, starting with the hexagon where the ratio of the circle area to the enclosing area to 90.68%, are explored for improved results. From there the number of sides, n, of the circumscribing polygon is progressively increased. The standard error of the estimate is reduced as the number of polygon sides is increased. One of several possible variance reduction methods is discussed.

Keywords: Barycentric Coordinates, Hit or Miss, R, Random variables, Simple Monte Carlo.

Introduction

The value of the transcendental constant π is known to at least 31 trillion decimal places, but even so, interest in finding just the first few digits of the constant using simple processes, has grown over the years; much fostered by internet usage. Figure 1(a) is central to a common Monte Carlo (MC) simulation, where computer-generated random numbers, represented by dots in a square, are classified as to whether they also fall within an enclosed circle. The logic is as follows (Anderson, 2020). If it is assumed that the points are totally random, then:

$$\frac{No\ of\ points\ in\ circle(n)}{No\ of\ points\ in\ square(N)} = \frac{Area\ circle}{Area\ square}. \quad (1)$$

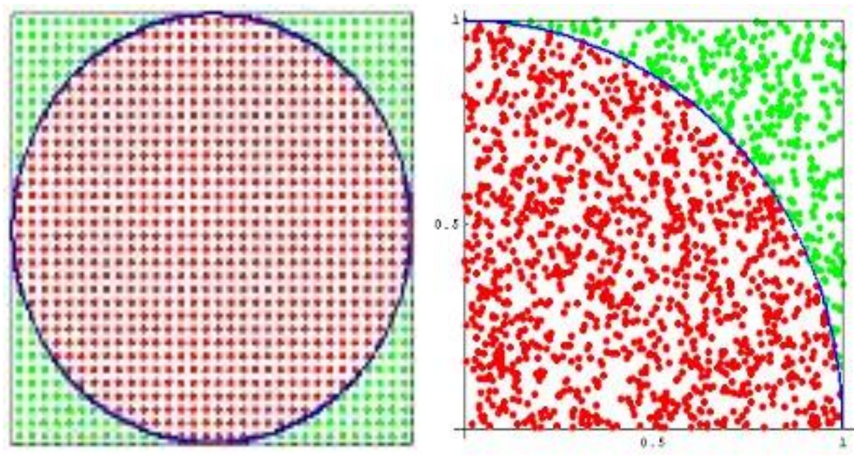
That is

$$\frac{n}{N} = \frac{\pi}{4}. \qquad (2)$$

From the simulation standpoint, the left and right sides of equation (2) provide two different estimators for the value of ▯, and so, π itself.

The estimators

Figure 1(b) shows a set of X-Y axes set up on the circle/square surface. Any point of the square with coordinates (x, y) will be some distance *d* from the centre where $d = \sqrt{x^2 + y^2}$.



(a) Circle. (b) Quadrant.

Figure 1: Random points.

A point (x, y) lies inside the circle if $d = \sqrt{x^2 + y^2} \leq 1$. Points are either inside or not inside - a "Hit or Miss" (HM) scenario. Simulating *d* is equivalent to simulating a Bernoulli random variable with probability $\frac{\pi}{4}$ ascertained from the right side of equation (2). Thus, naming the estimator as θ_{HM} gives:

$$\widehat{\theta_{HM}} = \sqrt{x^2 + y^2} \leq 1. \qquad (3)$$

$\widehat{\theta_{HM}}$ is an unbiased estimator of $\frac{\pi}{4}$ with Bernoulli variance $N\,\frac{\pi}{4}(1 - \frac{\pi}{4})$ and where *N* the number of trials.

Returning to Figure 1(b) and the RHS of equation (2), the ratio of areas "inside" where random points fall, to the unit area of the quadrant, is just $\frac{\pi}{4}$. Standard calculus can find $\frac{\pi}{4}$ from

$$A = \int_0^1 y\, dx \tag{4}$$

and since for points on the quadrant arc, $x^2 + y^2 = 1$, this becomes[i]

$$\hat{\theta}_I = \int_0^1 \sqrt{(1 - x^2)}\, dx. \tag{5}$$

This simulation estimator $\hat{\theta}_I$ is actually a special case of a general form:

$$\hat{\theta}_I = \int_a^b f(x)\, dx, \quad \text{with } a = 0,\ b = 1.$$

The general integral is approximated by averaging N samples of some function f at uniform random points within an interval. With a set of N uniform random variables $X_i\ \varepsilon\ (a,b)$, and with pdf $\frac{1}{(b-a)}$, the Monte Carlo estimator is:

$$\widehat{\theta_I} = (b-a)\sum_{i=1}^{N} f(X_i).$$

Therefore, there are two different estimators:

(i) $\widehat{\theta_{HM}} = x^2 + y^2 \leq 1$ and (ii) $\hat{\theta}_I = \sqrt{1 - x^2}$.

The variance of $\hat{\theta}_I$ will always be less than the variance of $\widehat{\theta_{HM}}$. Intuitively, this must be so, because (ii) involves the simulation of only one random variable, as opposed to two; and the intuition is easily confirmed algebraically. Interestingly, the latter is derived from the former by the conditioning of x on y. Most of what follows in this article will prefer the estimator $\hat{\theta}_I$.

Alternative circumscribing polygons

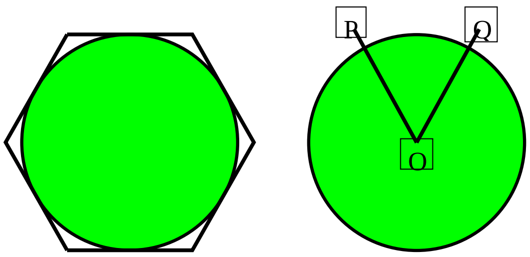
In connection with the circle-in-the-square-problem, not a lot appears to have been written for the cases when the circle becomes circumscribed by some other figure, e.g. a polygon (although the original arithmetic way of Archimedes used just this technique). Is it the case that the spread of the variance of the simulations might be reduced by adopting a different basic arrangement e.g. instead of the circle being enclosed in a square, where it occupies 75.8 percent of the bounding area, would better results be obtained if the circle were enclosed by a higher-order polygon?

Table 1: Circle area as % of Polygon

Polygon	Square	Pentagon	Hexagon	Octagon
% occupied	78.5	86.4	90.68	94.8

Table 1 shows the ratios of some possible figure areas. So, for example, a circumscribed hexagon would bring the ratio of the circle area to the enclosing area to 90.68 percent. This is then a good point to begin a general discussion through the example of a hexagon, moving to the case of more general figures later (it was also a waypoint for Archimedes' arithmetrical/geometrical estimation of the upper limit on the value of π).

Figure 2 shows: (a) a regular hexagon enclosing a circle of unit radius, and (b) one of six triangles comprising the hexagon, and a sector of the circle enclosed. As previously, the centre of the circle is (0,0), with radius 1.



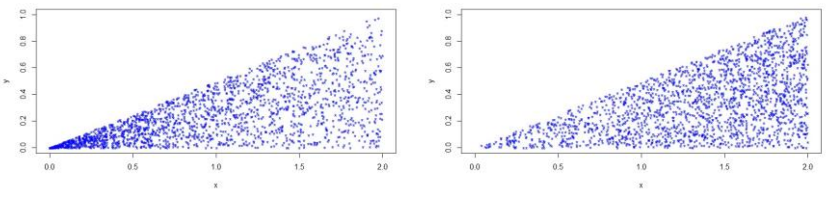
(a) Hexagon, Circle (b) Triangle in Hexagon

Figure 2: Circle and Hexagon

The HM estimator

In the case of the circumscribing hexagon (Figure 2), the triangle shown consists of a tangent and two circle radii extended. Its area includes the circle sector. What is now required is to simulate a set of N random numbers which cover the triangle and ascertain the number n of those numbers also falling within the sector of the circle. Unfortunately, generating random points within a triangular area is more involved than producing random points within a quadrant, which is the appropriate simulation technique for Figure 1(b).

In the case of the quadrant, N random points are generated by R commands such as x = runif (N, 0, 1) and y = runif (N, 0, 1), along the X and Y axes respectively, and uniformly cover the 2D square area. It turns out that an arbitrary triangular area is not covered uniformly by similar commands. Observe $N = 2000$ random points of a triangle in Figure 3 below, one of which was affected by the above method, the other by the correct method (see the programs tri.non uniform.R and tri.uniform.R in the Appendix for the generation of the two triangles)



(a) Non-Uniform. (b) Uniform.

Figure 3: Non-Uniform/Uniform points.

It can be seen that (b) is the way to achieve a pattern of points where there are no obvious gaps in coverage. In fact, (b) relies on a system of barycentric coordinates (Burkhardt, 2014). To use this more complicated procedure is essential since the whole validity of Monte Carlo simulation is posited on random points being uniformly distributed.

Barycentric coordinates

Let the vertices of a general triangle be A, B, C, each with Cartesian vector coordinates (x, y). Let r and s be random numbers where $0 < r < 1$ and $0 < s < 1$. The following intermediate quantities are needed for the coordinates of points P(x, y) lying within the triangle: $ea = 1.0 - \sqrt{s}$, $eb = (1.0 - r) * \sqrt{s}$, $ec = r * \sqrt{s}$.

Then, the barycentric coordinates P are:

$$P = ea * A + eb * B + ec * C. \qquad (6)$$

The inverted triangle (figure 4) has a vertex angle at point O, which is $\frac{\pi}{6}$ radians, so the semi-vertex angle is $\frac{\pi}{12}$. Thus, the length of the tangent side of the triangle is $2\tan(\frac{\pi}{12})$, and the set of triangle(x, y) coordinates to be transformed into barycentric coordinates is therefore:

$$(-\tan(\tfrac{\pi}{12}), 1),\ (0, 0),\ (\tan(\tfrac{\pi}{12}), 1).$$

Fortunately, R incorporates a package 'uniformly' which generates uniform distributions for different geometric shapes. For a triangle, if given the Cartesian vertices' coordinates, it implements the above transformations and generates a vector of N random variates of points. The tabulation below shows estimates of pi and the standard error of estimates for different values of N, from 50 replications of the program polyHM.R (see code in the Appendix).

Table 2: Estimator HM

Hexagon		
N	**pi**	**s.e.**
1000	3.133	0.03
10000	3.142	0.009
100000	3.141	0.003
500000	3.1417	0.0014

But as it is the estimator with the higher variance, the topic of the $\widehat{\theta_{HM}}$ estimator will not be pursued, the other estimator being preferred.

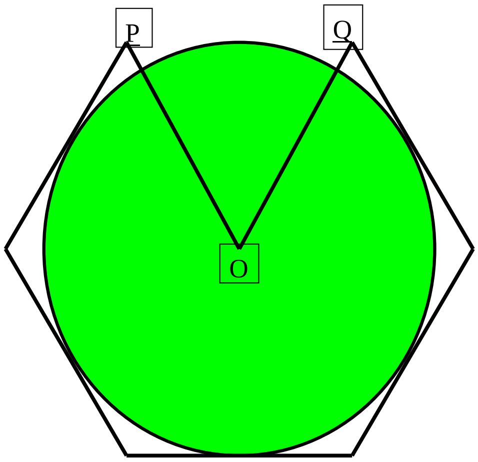


Figure 4: Hexagon and circle

The I estimator

A circumscribing Hexagon (Figures 4 & 5) provides a good illustrative starting point for the I estimator. It shows integration for the sector of the arc RS of $y = \sqrt{1 - x^2}$. The area includes the 2 triangles AOR and BOS; these need to be deducted later for the final result. The advantage is that, from the shape of the figure, barycentric coordinates are not needed.

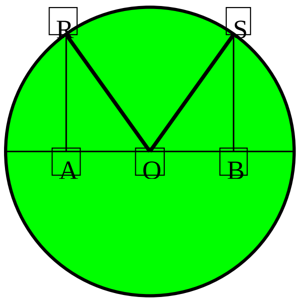


Figure 5: Sector of Hexagon

As with the previous .R program, polyI.R is general, based on the angle of the sector which will always be $\frac{2pi}{n}$, where n is the number of polygon sides. The base angles of the triangles will be (0.5)*(pi - $\frac{2pi}{n}$) and the ± cos of this angle gives the integration limits a and b. The total area of the two triangles (0.5)*sin(1 - $\frac{2pi}{n}$). The program polyI.R employs simulation to estimate the sector area ORS, returning an estimate of *pi*, and the standard error of that estimate. As previously, one argument (n) of polyI(n, N) is the number of sides of the circumscribing polygon.

The following lines of R code will average the results of 50 runs:

```
    estims <− replicate (50 , expr = polyI (n, N))
    colMeans(t( estims ) )
  # n = 4 ,5 ,6 ,8 ,     with N the number of simulations .
```

Table 3: A typical set of results for polyI

Standard Errors				
N	n = 4	n = 5	n = 6	n = 8
1000	0.015	0.010	0.0075	0.0043
10000	0.0048	0.003	0.0023	0.0013
100000	0.0015	0.0010	0.00075	0.00043
500000	0.0006	0.0004	0.00033	0.00019

A typical set of results is shown in Table 3. Clearly, the standard error of the estimate is reduced as the number of polygon sides is increased, which accords with the theoretic probability density considerations (Ebert, 2020).

More decimal places

Even for the best of the results, viz: the octagon with N = 500000, with a standard error of the order of 0.00019, the *pi* estimate is certain only to 3 decimal places. Better is possible.

One variance reduction strategy which is applicable is that of a control variate. A suitable function for the *pi* estimates above is $f(x) = x^2$; the mean value of x^2 in (a, b) is known. It is:

$$\frac{1}{(b-a)} \int_a^b x^2 dx.$$

The program polyIcv.R adds this control variate to polyI.R. Table 4 shows the pi estimates and standard error for 50 runs of polyIcv.R for the case of the Octagon, using the following lines of code:

```
estims <− replicate (50 , expr = polyIcv (8 , N))
colMeans(t( estims ))
```

Table 4: pi estimates and standard error for 50 runs of polyIcv.R

Octagon		
N	$\hat{pi}$	s.e.
1000	3.14158	0.000046
10000	3.14159	0.000013
100000	3.1415928	0.000004
500000	3.1415928	0.000002

The last result with standard error of the order of 10^{-6} is quite a satisfactory one, obtained by restricting the number of simulations and there exists the potential for other forms of variance reduction, but that extension will not be continued here.

Summary and conclusion

A simple Monte Carlo simulation, using functions from various R packages, was used to achieve reasonable estimates of the early digits of π using a variety of polygons to circumscribe a unit circle. Polygons were segmented into triangles and barycentric coordinates were used to provide a uniform distribution of points within the triangles from the Monte Carlo simulation tool. The number of polygon sides were increased thereby increasing the ratio of the circle area to the enclosing figure and leading to closer approximations to pi. The standard error of the estimate was reduced as the number of polygon sides increased. One of several possible variance reduction methods was discussed.

References

Anderson E. *Calculation of Pi using the Monte Carlo Method* Retrieved 25 August 2020, from http://www.evananderson/pi/monte-carlo-demo.tcl.

Burkhardt. J, (2018). Computational geometry lab. Retrieved 25 August 2020, from https://people.sc.fsu.edu/~jburkardt/classes/cg_2007/cg_lab_monte_carlo_triangles.pdf

Ebert T. *The Monte Carlo Method*. Retrieved 25 August 2020, from http://web.csulb.edu/~tebert/teaching/lectures/552/mc/mc.pdf

Authors

Rik King (PhD) is a retired Australian academic and now continues his research interests in financial mathematics and simulation methods.

Peter K. Anderson, (PhD) is Head, Department of Mathematics & Computing Science, DWU. Email: panderson@dwu.ac.pg

Appendix

```
# tri . non−uniform
#−−−−−−−−
# non−uniformly distributednumbers over # the triangle (0 ,2) ,
(0 ,0) , (2 ,1)
N <− 2000
x <− 2∗runif (N);
y <− (x/2)∗ runif (N);
rnums <− matrix(c(x , y) , nrow = N);
plot .new() plot(rnums , xlim = c(0 ,2) , ylim = c(0 ,1) , pch = ’∗
’ , col = ”blue” , xlab = ’x ’ , ylab = ’y ’ )
#−−−−
# tri . uniform
#−−−−
# uniformly distributed numbers over # the triangle (0,2), (0,0),
(2,1)

library ( uniformly )
N <− 2000 tri . uniform <− runif in triangle (N, c(0,0) ,c(2,0)
,c(2,1))
plot .new() plot( tri . uniform , xlim = c(0,2) , ylim = c(0,1) ,
pch = ’∗ ’ , col = ”blue” , xlab = ’x ’ , ylab =’y ’ )

#−−−−
# polyHM
#−−−−

library ( uniformly )
polyHM <− function (n, N){ g <− function(u, v){uˆ2 + vˆ2}
tp <− tan( pi/n)
rit <− runif _in_triangle (N, c(−tp ,1) , c(0 ,0) ,c(tp ,1))
x <− rit [ ,1]; y <− rit [ ,2]; p <− g(x , y)
inside <− p[p <= 1]; num <−length( inside )
pd <− (n∗tan( pi/n )); Ex _ val <− (num/N);
pi_est <− Ex_val∗pd; se <− pd∗sd( inside )/sqrt(N)
```

```
return(c( piest , se )) } # end function
# example
estims <− replicate (50, expr= polyHM(6,1000))
colMeans(t( estims ) )

#−−−−−−−−
# polyI
#−−−−−−−−
polyI <− function(n, N){
 g <− function(u){sqrt(1 − uˆ2)}
 x <− cos ((0.5)∗( pi − 2∗pi/n))
 b <− x ; a <− − x ; Y <− runif (N, a , b)
 tri . area <− 0.5∗sin ((n − 2)∗pi/n)
 estim <− n∗((g(Y))∗(b−a) − tri . area )
 me <− mean( estim ); se <− sd( estim )/sqrt(N)
return(c(me, se ))} # end           function
# example estims <− replicate (50 , expr = polyI (6 , 500000))
 colMeans(t( estims ) )
#−−−−−−−
# polyIcv
#−−−−−−−
polyIcv <− function(n,N){ options( digits = 8)
g <− function(u){sqrt(1 − uˆ2)} x <− cos ((0.5)∗( pi − 2∗pi/n))
 a <− − x ; b <− x ; Y <− runif (N, a , b );

#− control variate Z−
 Z <− Yˆ2;
 cv <− function(w){wˆ3/3} # the mean for xˆ2
 Zm <− (1/(b − a))∗(cv(b) − cv(a) )
#−−−−−−−
 # simulated mean of Y Yms <− mean(Y); tri . area <− 0.5∗sin
 ((n − 2)∗pi/n)
#−−−−−−−
 X <− g(Y); Xms <− mean(X)
 c <− −sum ((X − Xms)∗(Z − Zm))/sum((Z − Zm)ˆ2)
```

```
tmp <− (b −a)∗(X + c ∗(Z − Zm))
estim <− n∗(tmp − tri . area )
me <− mean( estim ); se <− sd( estim )/sqrt(N)
return(c(me, se )) }# end function
# example estims <− replicate (50 , expr = polyIcv (8 ,
 500000)) colMeans(t( estims ) )
#
```
