3. Monte Carlo Integration

SM Shafqat Shafiq

2023-08-12

Monte Carlo Integration:

Monte Carlo integration is a numerical method that uses random numbers to approximate definite integrals, using a non-deterministic/stochastic approach.

Key-idea behind Monte Carlo: Break down the area under the curve into individual rectangles, with the height of each rectangle corresponding to f(x_i), which is the value of each point x_i belonging to sequence of random points (x_n: n belongs to Naturals), evaluated under f(x).

Then, using the trapezoid rule for calculating area under the curve, the result of the integration is evaluated as the the sum of all f(x_i), divided by the length of the sequence (x_n), and multiplied by the length of the interval over which it is being integrated.

As n (length of the sequence) tends to infinity, by LLN the MC integration of f(x) converges to the true value of the integration of f(x).

Load libraries, set seed:

```
set.seed(123)
library(DescTools)
library(plot3D)
```

```
## Warning in fun(libname, pkgname): couldn't connect to display ":0"
```

Example 1: Start with a simple function: f(x) = x^2

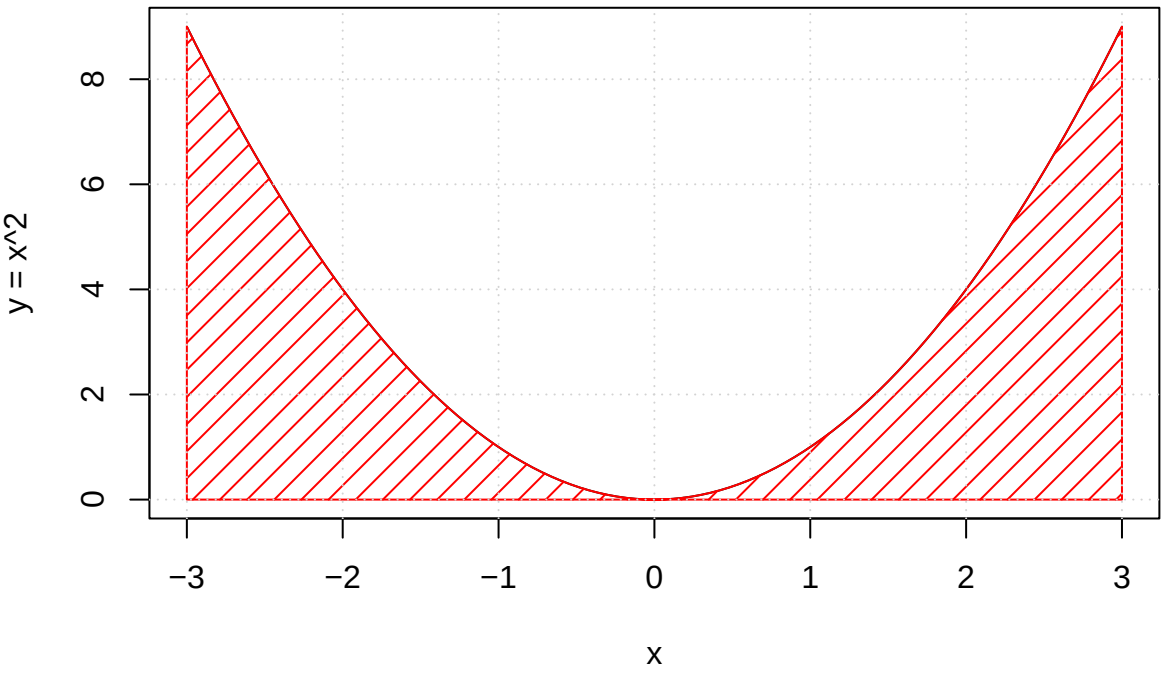
1.1 Write the function:

```
f <- function(x) {
  x^2
}
```

1.2 Plot:

We will estimate the area under the curve from (-3, 3)

```
curve(f, from = -3, to = 3, ylab = "y = x^2", xlab = "x")
Shade(f, breaks = c(-3,3), col = "red")
grid()
```



From Calculus, we know that integrating x^2 from [-3, 3] will give us 18. Let's now use Monte Carlo integration to approximate our integral, which is the shaded area under the graph.

1.3 Perform Monte Carlo Approximation:

First, generate 1000 Random Points from Uniform [-3, 3], that will be the sequence (x_n).

```
random_points <- runif(1000, -3, 3)
```

Then perform Monte Carlo approximation by evaluating the function at each of the points in the sequence of random points generated above.

```
mean(6*f(random_points))
```

```
## [1] 17.8355
```

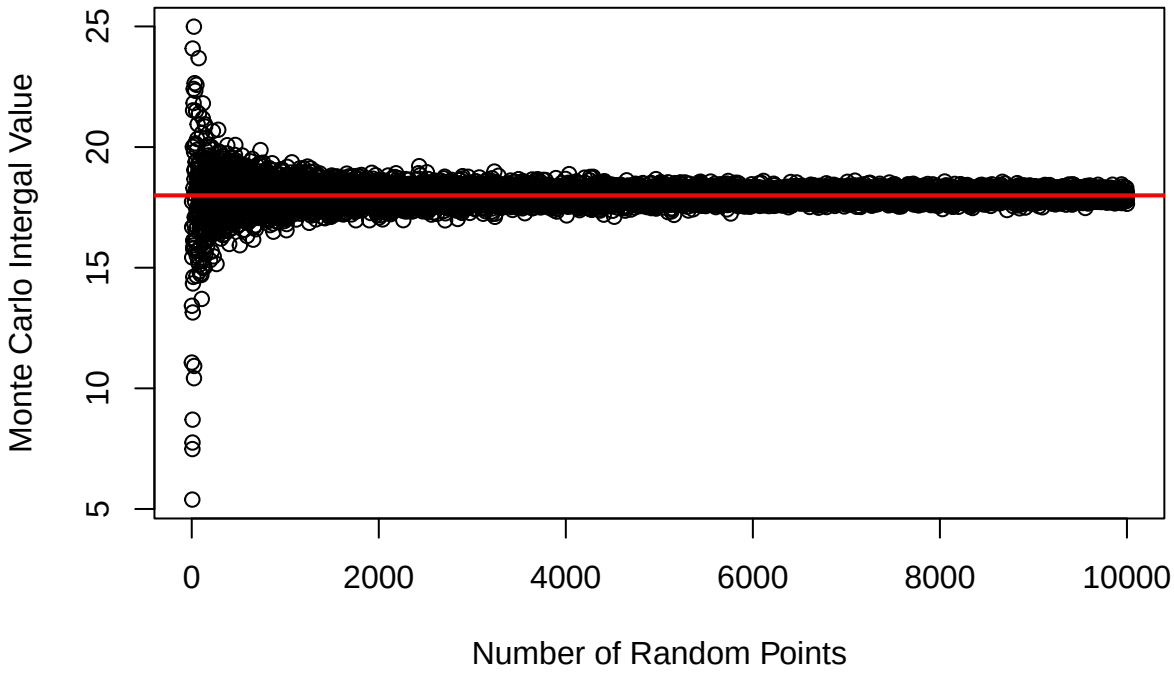
1.4 Check accuracy:

We would now like to check how the accuracy of our MC estimate is affected by the length of the sequence of random numbers. This can be done using a for-loop, and then plotting the results.

```
sequence_length <- c(1:10000)
area_estimates <- numeric(length(sequence_length))

for (i in 1:length(sequence_length)) {
  random_points <- runif(i, -3, 3)
  area_estimates[i] <- mean(6*f(random_points))
}
```

1.5 Plot the accuracy:

```
plot(area_estimates, xlab = "Number of Random Points",
     ylab = "Monte Carlo Intergal Value")
abline(h = 18, col = "red", lwd = 2)
```



As expected, due to the Law of Large Numbers, as n increases, our Monte Carlo approximation of the function f(x) = x^2, evaluated between the interval [-3, 3], approximates to the true value of 18.

Example 2: Approximate Pi

Let's first define an appropriate function:

```
g(x, y) = 1, if x^2 + y^2 <= 1

g(x, y) = 0, if x^2 + y^2 > 1

-1 <= x, y <= 1
```

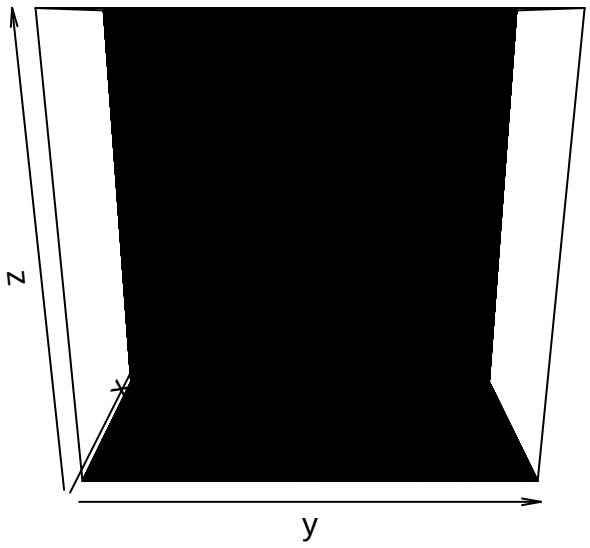
2.1 Write the function:

```
g <- Vectorize(function(x, y) {
    if ((x^2 + y^2) <= 1) {

  return(1)

    } else

  0

}, vectorize.args = c("x", "y"))
```

2.2 Plot

It's better to use R's native connectivity with Plotly to render 3D graphs.

```
x <- y <- seq(from = -1, to = 1, length = 1000)
z <- outer(x, y, g)
persp(y, x, z)
```



2.3 Perform Monte Carlo Approximation:

As per geometric argument, volume under this function is equal to pi, since we are looking at a cylinder with a radius and height of 1 (V = pi* r^2 * h).

Let's approximate this volume using Monte Carlo integral. First, generate 1000 random points for both x and y.

```
rand_points_x <- runif(1000, -1, 1)
rand_points_y <- runif(1000, -1, 1)
```

Compute Monte Carlo Integral (Length = 2*2 = 4) by evaluating the MC integral at the randomly generated points.

```
4 * mean(g(rand_points_x, rand_points_y))
```

```
## [1] 3.192
```

2.4 Check accuracy:

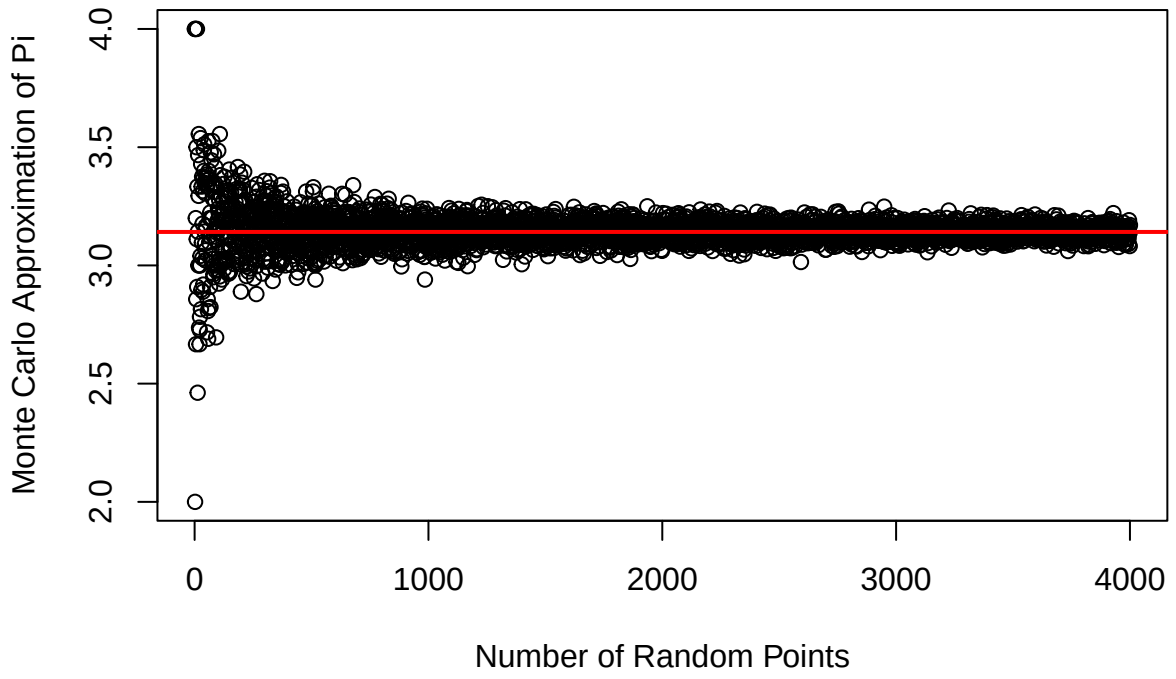
As before, evaluate how the accuracy of the MC approximation is affected by the number of random points generated.

```
area_estimates_pi <- numeric(4000)

for (i in 1:length(area_estimates_pi)) {

  rand_points_x <- runif(i, -1, 1)
  rand_points_y <- runif(i, -1, 1)

  area_estimates_pi[i] <- 4*mean(g(rand_points_x, rand_points_y))

}
```

2.5 Plot the accuracy:

Once again, as shown in the previous example, the MC approximation reaches the true value of Pi as n tends to infinity.

```r
plot(area_estimates_pi, xlab = "Number of Random Points",
     ylab = "Monte Carlo Approximation of Pi")
abline(h = pi, col = "red", lwd = 2)
```



Monte Carlo integration is a powerful tool to evaluate functions that don't have any closed-form solutions. Real-world applications such as Claims-Distributions, Survival Analysis, and others, tend to throw up complicated PDFs that often exhibit such characteristic (i.e. no availability a of closed-form solution).

In such cases, MC integration is incredibly helpful.

List of cool references:

1. MIT: "Monte Carlo in a Nutshell" https://ocw.mit.edu/courses/2-086-numerical-computation-for-mechanical-engineers-fall-2014/30d8b5da0c8e6ad44987b3563bc32dab_MIT2_086F14_Monte_Carlo.pdf
2. Princeton: "Monte Carlo Integration" https://www.cs.princeton.edu/courses/archive/fall11/cos323/notes/cos323_f11_lecture15_monte.pdf
3. Wichita State University: "Simulation and Monte Carlo" https://www.math.wichita.edu/~cma/stat774/ch3.pdf
4. Statistical Computing with R, Chapter 5: "Monte Carlo Integration and Variance Reduction"