The Method of Monte Carlo

STA 326 2.0 Programming and Data Analysis with R

Dr Thiyanga S Talagala

Contents

1. Introduction

A simulation model is a computerized mathematical technique that imitates a real life situation.

Objectives:

1. Set up simulation models.
2. Run a simulation model.
3. Interpret the simulation output and relate it to the the real-world situation that is being modelled.

What are Monte Carlo Simulations?

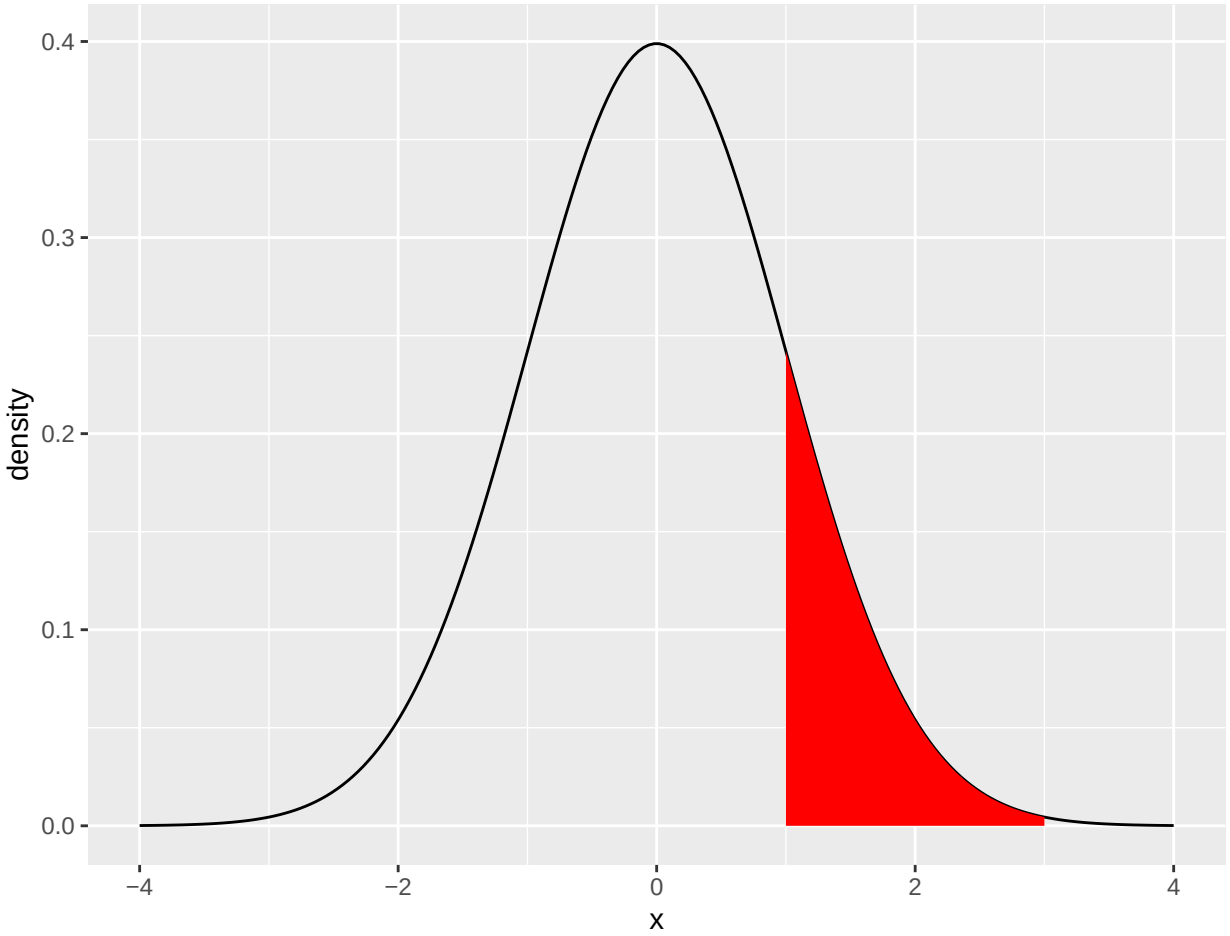
> "MCS studies are computer-driven experimental investigations in which certain parameters, such as population means and standard deviations that are known a priori, are used to generate random (but plausible) sample data (Mooney 1997). These generated data are then used to evaluate the sampling behavior of one or more statistics of interest. This process of generating and analysing data is repeated over many iterations and differing conditions that are thought to influence the sampling behavior of the statistic of interest (e.g., through increasing sample size, mean differences, variability)."
>
> Play It Again: Teaching Statistics With Monte Carlo Simulation, Matthew J. Sigal & R. Philip Chalmers, Journal of Statistics Education

Main steps:

1. Design: Factors of interest under study
2. Simulate dataset
3. Analyse: Extract relevant statistic
4. Interpret

The best way to understand is to go through a bunch of examples, so let's go!

Suppose $X \sim Normal(0, 1)$ and we want to find $P(1 < X < 3)$ as visualized below.



Using `base R` functions:

```
pnorm(3) - pnorm(1)
```

```
[1] 0.1573054
```

Let's try to estimate this using Monte Carlo simulations. We can simply generate 100,000 random numbers from the standard normal distribution and see how many values are between 1 and 3.

```
runs <- 100
sims <- rnorm(runs, mean = 0, sd = 1)
head(sims)
```

```
[1]  1.5921676 -0.9247866 -0.7182562  0.6046786 -0.9414394  0.7646799
```

```
mc.integral <- sum(sims >= 1 & sims <= 3)/runs
mc.integral
```

```
[1] 0.15
```

```
mc.integral
```

```
[1] 0.15
```

```
runs <- 100000
sims <- rnorm(runs, mean = 0, sd = 1)
head(sims)
```

```
[1] -0.1178349 -0.2342004 -0.6512459 -0.4910658 -0.6147532 -0.9765902
```

```
mc.integral <- sum(sims >= 1 & sims <= 3)/runs
mc.integral
```

```
[1] 0.15882
```

```
mc.integral
```

```
[1] 0.15882
```

Which isn't too far off from the 0.1573054 that `pnorm(3) - pnorm(1)` gives us.

In our Monte Carlo simulations so far, we have seen that the more times we repeat the underlying random process, the closer our estimate is likely to be to the actual value.

This is a consequence of a theorem in the subject of probability that is known as the **Law of Large Numbers**.

2. Applications

2.1 Estimate Area (The "hit-or-miss" method)

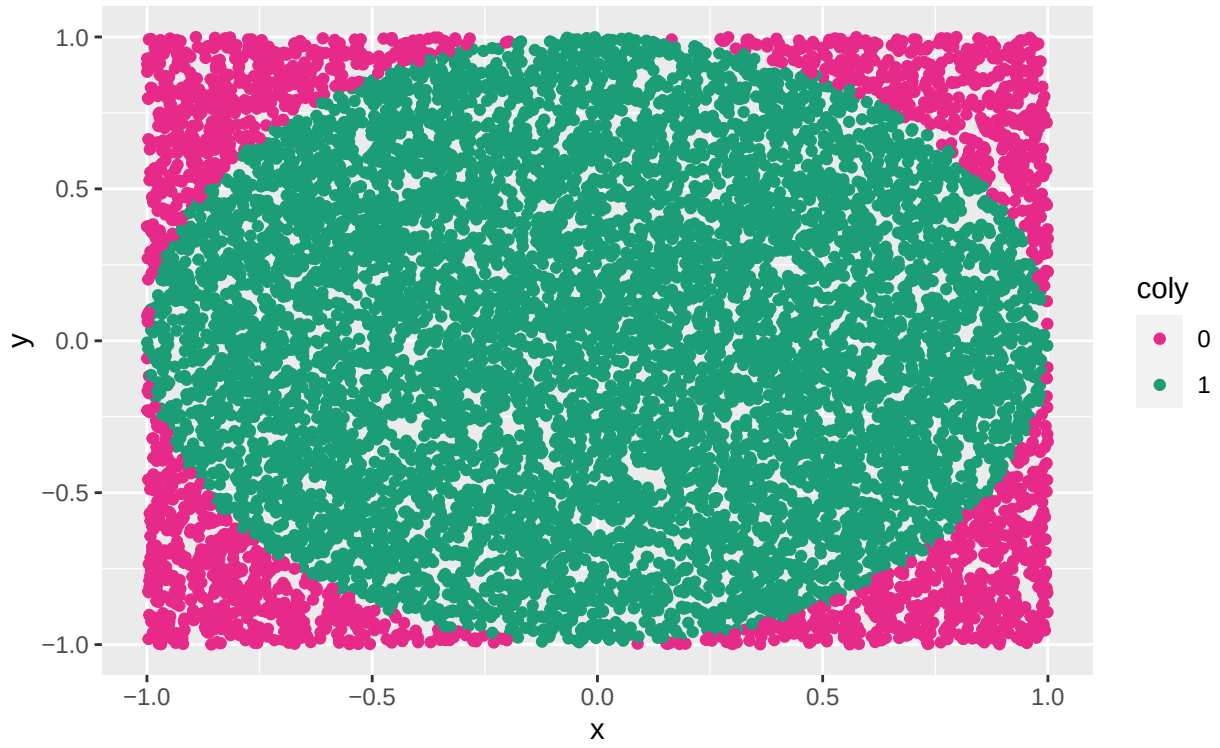
Example 2.1.1: Approximating Pi = 3.14616

$$\frac{A_{circle}}{A_{square}} = \frac{\pi r^2}{4r^2}$$

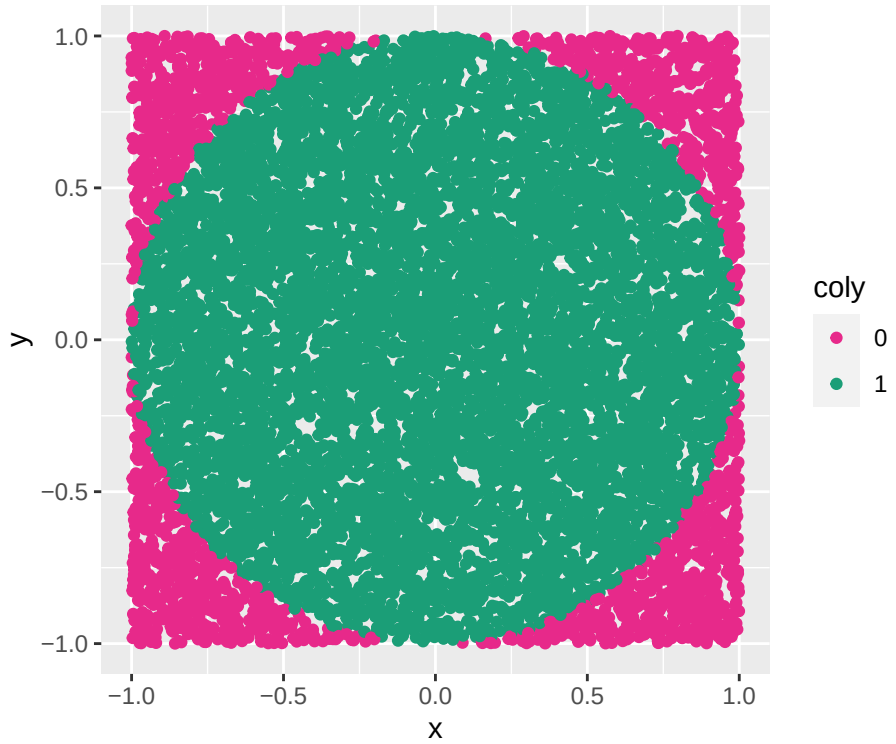
Equation of the unit circle center around 0: $x^2 + y^2 = r^2$

```
library(tidyverse)
x <- runif(10000, -1, 1)
y <- runif(10000, -1, 1)
fx <- x^2 + y^2
coly <-  ifelse(fx <= 1, 1, 0)
coly <- as.factor(coly)
pidf <- data.frame(x=x, y=y, coly=coly)
```

```
# without coord_equal()
ggplot(pidf, aes(x=x, y=y, col=coly)) + geom_point()  +
   scale_colour_manual(values = c("#e7298a", "#1b9e77"))
```




```
# with coord_equal()
ggplot(pidf, aes(x=x, y=y, col=coly)) + geom_point() +
   scale_colour_manual(values = c("#e7298a", "#1b9e77")) +
  coord_equal()
```



```
compute_pi_mc_sim <- function(n){
  x <- runif(n, -1, 1)
  y <- runif(n, -1, 1)
  count <- 0
  for(i in 1:n){
    if(x[i]^2 + y[i]^2 < 1 | x[i]^2 + y[i]^2 == 1){
      count = count + 1

    } else {
      count
    }

  }

 pi <- (count/n) * 4
 pi

}
compute_pi_mc_sim(100)
```

```
[1] 3.08
```

```
compute_pi_mc_sim(1000)
```

```
[1] 3.144
```

```
compute_pi_mc_sim(10000)
```

```
[1] 3.1336
```

YOUR TURN: What other ways can you write this R function to perform the same task ?

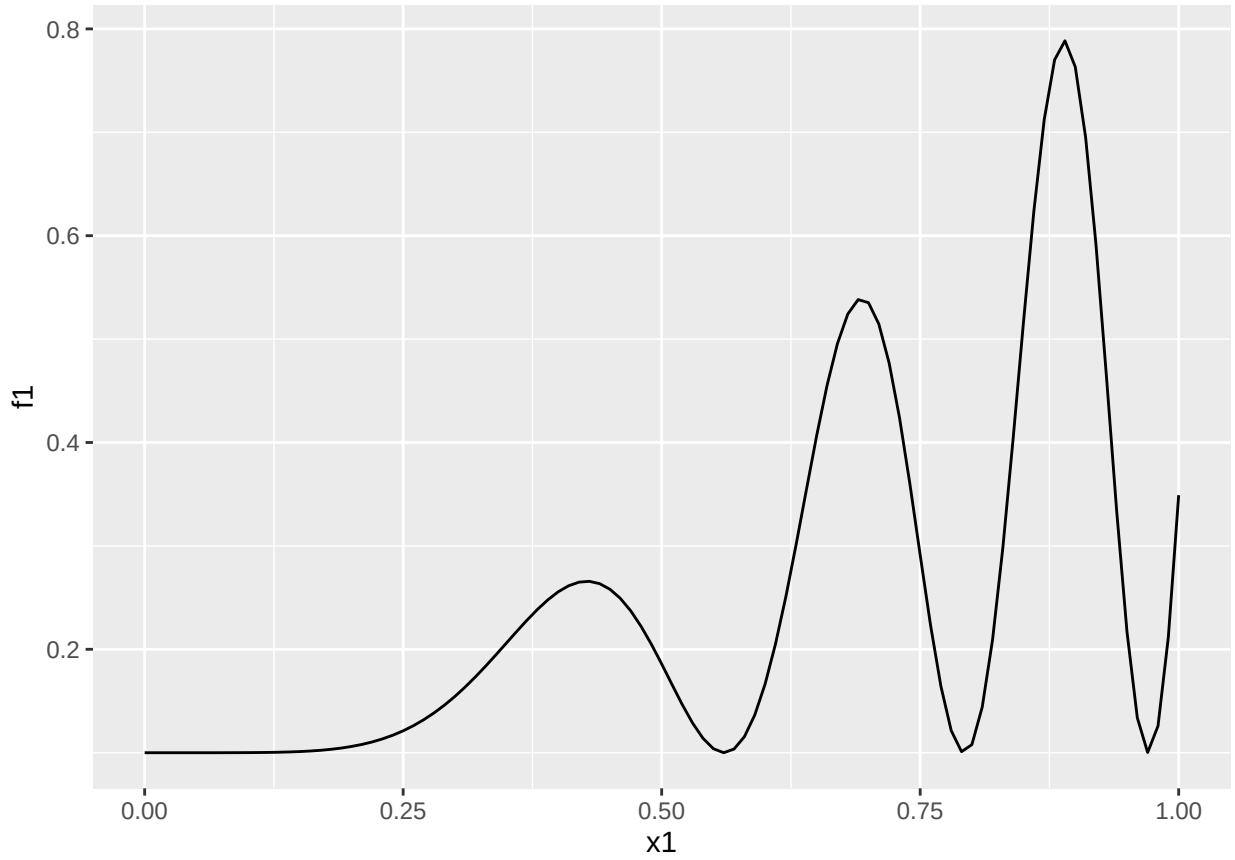
Write a function to approximate π using Monte Carlo simulations.

Example 2.1.2: YOUR TURN

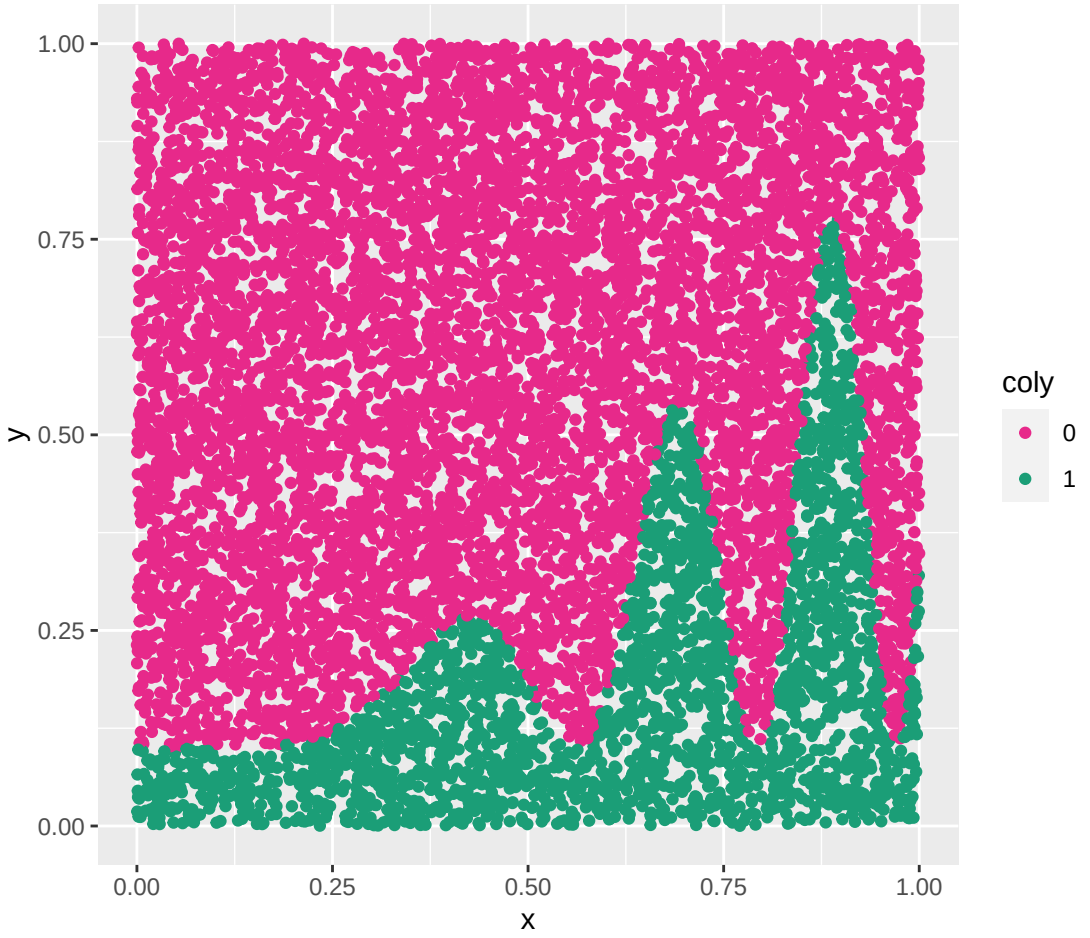
Estimate area under the curve within $x \in [0, 1]$ using Monte Carlo simulation approach.

$f(x) = sin(10x)^{2sin(x)}x + 0.1$

```
x1 <- seq(0, 1, 0.01)
f1 <- ((sin(10*x1^2))^2*sin(x1))*x1+0.1
df1 <- data.frame(x=x1, y=f1)
ggplot(df1, aes(x=x1, y=f1))+geom_line() + coord_equal()
```




```
set.seed(2020)
x = runif(10000, min =0 , max =1 )
y = runif(10000, min =0 , max =1 )
fx <- ((sin(10*x^2))^2*sin(x))*x+0.1
coly <-  ifelse(y < fx, 1, 0)
coly <- as.factor(coly)
df2 <- data.frame(x=x, y=y, coly=coly)
ggplot(df2, aes(x=x, y=y, col=coly)) + geom_point() +
  scale_colour_manual(values = c("#e7298a", "#1b9e77")) +
  coord_equal()
```



2.2 Monte Carlo Integration

Suppose we want to calculate the integral $\int_a^b g(x)dx$ for a continuous function g over the closed and bounded interval [a, b]. If the anti-derivative of g does not exist, then numerical integration is in order. A simple numerical technique is the method of Monte Carlo. We can write the integral as

$$\int_a^b g(x)dx = (b-a)\int_a^b g(x)\frac{1}{a-b}dx = (b-a)E[g(X)],$$

where X has the $uniform(a, b)$ distribution.

To compute the estimated value first a set of random numbers X_1, X_2, X_n of size n is, then compute $Y_i = (b-a)g(X_i)$. Then $\bar{Y}$ is a consistent estimate of $\int_a^b g(x)dx$.

Example 2.2.1: Approximating Pi using Monte Carlo Integration.

Let $g(x) = 4\sqrt{1-x^2}$ for $0 < x < 1$. Then,

$$\pi = \int_0^1 g(x)dx = E[g(X)],$$

where X has the $uniform(0,1)$ distribution.

Write an R function to obtain point estimate and 95% confidence interval for pi using the sample sizes, 100, 1000, 10000 , 100000 and fill the blanks in the following table.

```
pi_mi <- function(n){

  random.uniform <- runif(n)
  sample_gx_values <- 4*sqrt(1-random.uniform^2)
  # point estimate
  y_bar <- mean(sample_gx_values)
  # standard error
  se <- sqrt(var(sample_gx_values)/n)
  # 95% confidence interval
  interval_estimate_lower <- y_bar - (1.96 * se)
  interval_estimate_upper <- y_bar + (1.96 * se)
  #output
  tibble::tibble(pi=y_bar,
                 CI.lower=interval_estimate_lower,
                 CI.upper=interval_estimate_upper)

}

pi_mi(100)
```

```
# A tibble: 1 x 3
     pi CI.lower CI.upper
  <dbl>    <dbl>    <dbl>
1  2.97     2.78     3.16
```

```
pi_mi(1000)
```

```
# A tibble: 1 x 3
     pi CI.lower CI.upper
  <dbl>    <dbl>    <dbl>
1  3.15     3.10     3.21
```

```
pi_mi(10000)
```

```
# A tibble: 1 x 3
     pi CI.lower CI.upper
  <dbl>    <dbl>    <dbl>
1  3.14     3.12     3.15
```

```
pi_mi(100000)
```

```
# A tibble: 1 x 3
     pi CI.lower CI.upper
  <dbl>    <dbl>    <dbl>
1  3.15     3.14     3.15
```

Example 2.2.2: YOUR TURN

Write an R function to estimate $log2$ using Monte Carlo Integration method. Obtain a point estimate, and 95% confidence interval for estimate using 10000 simulations and compare it to the true value.

Hint

$log\ 2 = \int_0^1 \frac{1}{x+1} dx.$

Example 2.2.3: YOUR TURN

Compute

$$\int_0^1 \sqrt{1-x^2} dx$$

.

i) Using the "hit-or-miss" method.

ii) Using the Monte Carlo integration approach.

3. Other applications

- Modelling the price of stock.
- Estimating probability distribution of project completion time.
- Analyzing a portfolio of investments.
- Simulating cash budgets.
- Simulation approaches to capacity planning.
- Determination of the optimal order quantity/ project profit.