Monte Carlo Simulation of areas

Mahtab Nahayati

2024-11-13

Contents

Monte Carlo Integration for $\int_1^b e^{-x^3}\,dx$

Step 1: Approximate the Integral for $b = 6$ Using Monte Carlo Integration

1. **Define the Function**: The function we want to integrate is
$$f(x) = e^{-x^3}$$
over the interval $[1, b]$ with $b = 6$.

2. **Generate Uniformly Distributed Random Variables**: We generate a set of random variables x_i uniformly distributed between 1 and 6. We choose a large number of samples, say $N = 10,000$, to get a good approximation.

3. **Calculate the Monte Carlo Estimate**:

   Using Monte Carlo integration, we can approximate the integral as follows:
$$\int_1^6 e^{-x^3}\,dx \approx \frac{b-a}{N}\sum_{i=1}^{N} f(x_i)$$
   where $a = 1$ and $b = 6$, and x_i are our uniformly distributed samples over $[1, 6]$.

```
# Define parameters
  N <- 10000
  a <- 1
  b <- 6

  # Generate random samples and evaluate the function at these points
  x <- runif(N, min = a, max = b)
  f_x <- exp(-x^3)

  # Monte Carlo estimate of the integral
  integral_approx <- (b - a) * mean(f_x)
  integral_approx
```

```
## [1] 0.08624321
```

4. **Comparison Using Exact Integration:** To verify the accuracy of our Monte Carlo approximation, we compare it to the exact integral using R's integrate function:

```
# Compute the exact integral
exact_result <- integrate(function(x) exp(-x^3), lower = 1, upper = 6)
exact_result$value
```

```
## [1] 0.08546833
```

Step 2: Approximate the Integral for $b = \infty$ Using Monte Carlo Integration

1. **Define the Function**: The function we want to integrate is:
$$f(x) = e^{-x^3}$$
over the interval $[1, \infty)$.

2. **Choose a Suitable Density**: Since our integration range is $[1, \infty)$, we choose an exponential distribution with support $[0, \infty)$ and shift it by 1. Let $X = 1 + Y$ where $Y \sim \text{Exponential}(\lambda = 1)$. Then we approximate the integral as:
$$\int_1^\infty e^{-x^3}\,dx \approx \frac{1}{N}\sum_{i=1}^{N} \frac{e^{-(1+y_i)^3}}{\lambda e^{-\lambda y_i}}.$$

```r
# Define parameters
N <- 10000
lambda <- 1

# Generate random samples from the exponential distribution
y <- rexp(N, rate = lambda)

# Calculate the Monte Carlo estimate
integral_approx_infinity <- mean(exp(-(1 + y)^3) / (lambda * exp(-lambda * y)))
integral_approx_infinity
```

```
## [1] 0.08634902
```

Comparison Using Exact Integration: To verify the accuracy of our Monte Carlo approximation, we use R's integrate function to compute the integral with an upper limit of infinity:

```r
# Define the function
f <- function(x) exp(-x^3)

# Compute the exact integral from 1 to infinity
exact_result_infinity <- integrate(f, lower = 1, upper = Inf)
exact_result_infinity$value
```

```
## [1] 0.08546833
```

step 3: Why Monte Carlo integration works better in step 2 that step 1?

In step 1, the Monte Carlo integration used a **uniform distribution** over $[1, 6]$ for the integral

$$\int_1^6 e^{-x^3}\, dx.$$

Since e^{-x^3} decreases rapidly as x increases, most of the integral's value comes from x near 1. The uniform distribution samples points evenly across the entire interval, including regions where the function is very small, leading to a less accurate estimate.

In step 2, we used an **exponential distribution** shifted to $[1, \infty)$ for

$$\int_1^\infty e^{-x^3}\, dx.$$

This distribution places more samples near $x = 1$, where e^{-x^3} is largest, reducing variance and improving accuracy. This alignment between the sampling density and the function's decay results in a closer match with the `integrate` function's result.

Monte Carlo Area Calculation for Polar Function with Visualization

Step 1: Define and plot the Function $r(t)$ in Cartesian Coordinates

The function is given by:

$$r(t) = \left(\exp(\cos(t)) - 2 \cdot \cos(4 \cdot t) - \sin(t/12)^5\right)$$

We convert this polar function into Cartesian coordinates for plotting:

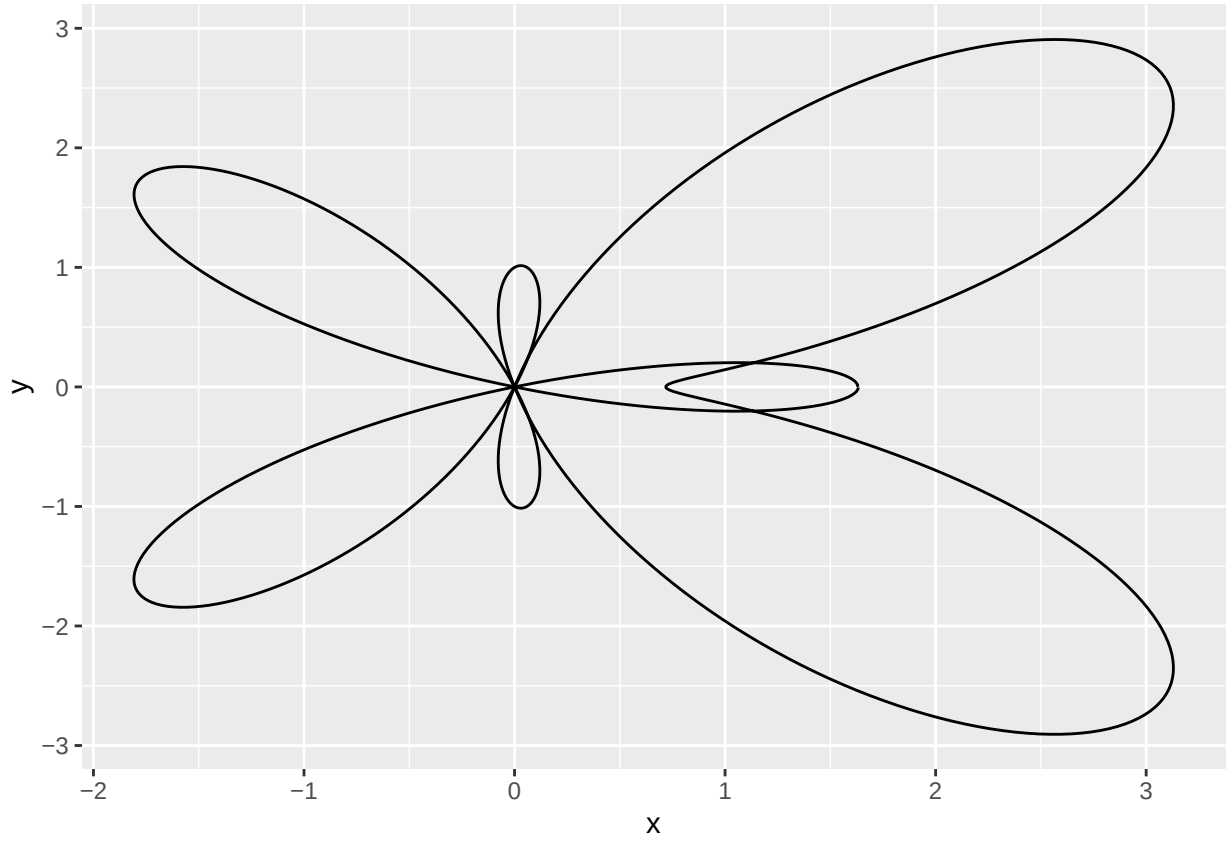
$$x(t) = r(t) \cdot \cos(t)$$

$$y(t) = r(t) \cdot \sin(t)$$

```r
# Define r(t)
r <- function(t) {
  exp(cos(t)) - 2 * cos(4 * t) - (sin(t / 12)^5)
}

# Generate t values and calculate corresponding x and y coordinates
t <- seq(by = 0.01, from = -pi, to = pi)
x <- r(t) * cos(t)
y <- r(t) * sin(t)

 #Plot the area enclosed by the function.
 library(ggplot2)
```

```
## Warning: Paket 'ggplot2' wurde unter R Version 4.3.2 erstellt
```

```r
 ggplot()+
 geom_path(aes(x=x,y=y))
```



Step 2: Generate Uniform Random Coordinates

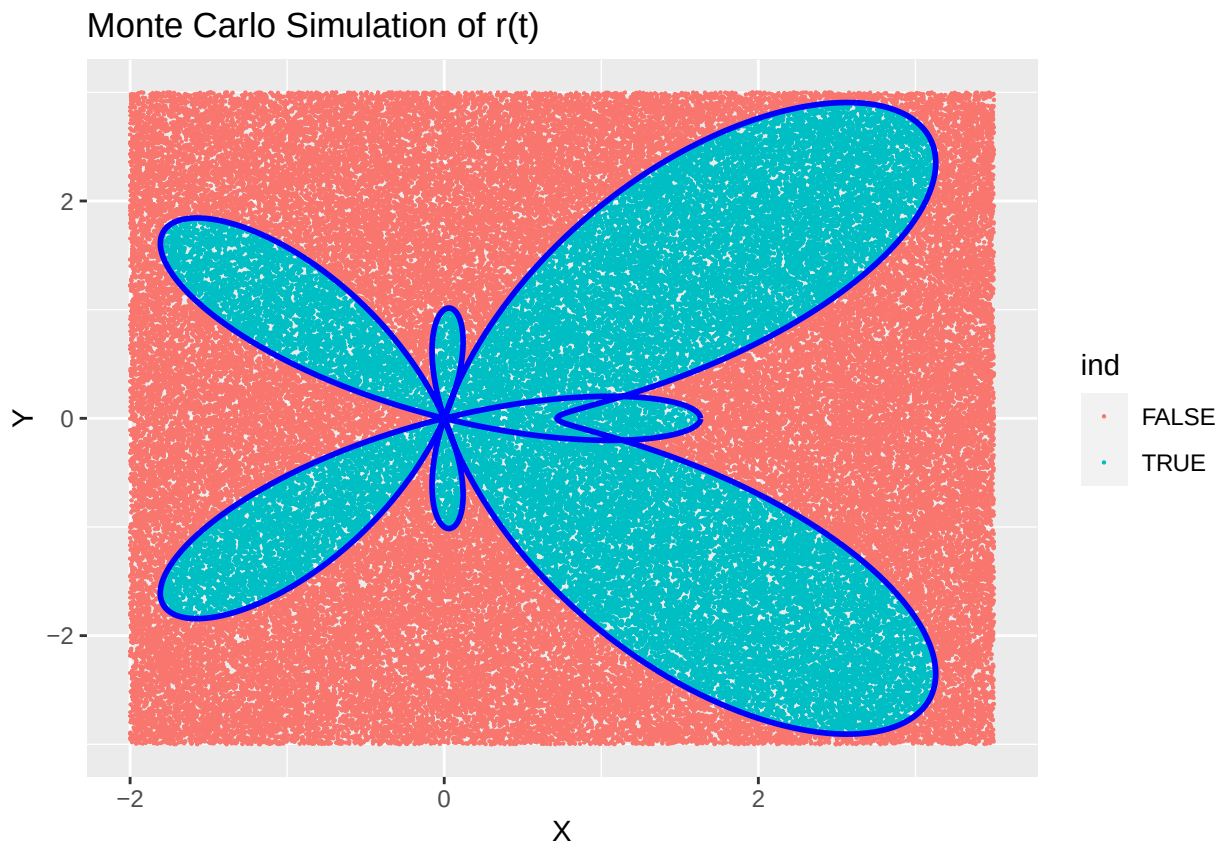
Generate uniform random coordinates within the rectangle $[-2, 3.5] \times [-3, 3]$ and determine if each point lies within the area enclosed by the function $r(t) = \left(\exp(\cos(t)) - 2 \cdot \cos(4 \cdot t) - \sin(t/12)^5\right)$ for $t \in [-\pi, \pi]$, using polar coordinates with:

$$x = r(t) \cdot \cos(t) \quad \text{and} \quad y = r(t) \cdot \sin(t)$$

An indicator function will mark whether each random point lies inside or outside the region.

```r
#generate random coordinates
 n <- 100000
 xs = runif(n, -2, 3.5)
 ys = runif(n, -3, 3)

 #check if it lies within the defined area

 inside_func <-function(x,y){
   beta <- atan2(y,x)
   rad <- sqrt(x^2 + y^2)

   r_beta <- r(beta)
   r_beta_pi <- r(beta + pi)

  return((r_beta > 0 & rad < abs(r_beta)) ||(r_beta_pi< 0& rad <abs(r_beta_pi)))

 }

 # Indicator function for each point

 ind <- logical(n)
 for (i in 1:n){
   ind[i] <- inside_func(xs[i], ys[i])
 }

 df <- data.frame(xs, ys, ind)

 # Plotting using ggplot2
library(ggplot2)
ggplot() +
  geom_point(data = df, aes(x = xs, y = ys, color = ind), size = 0.1) +
  geom_path(aes(x = x, y = y), color = "blue", size = 1) +
  ggtitle("Monte Carlo Simulation of r(t)") +
  xlab("X") +
  ylab("Y")
```

```
## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
## i Please use `linewidth` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```



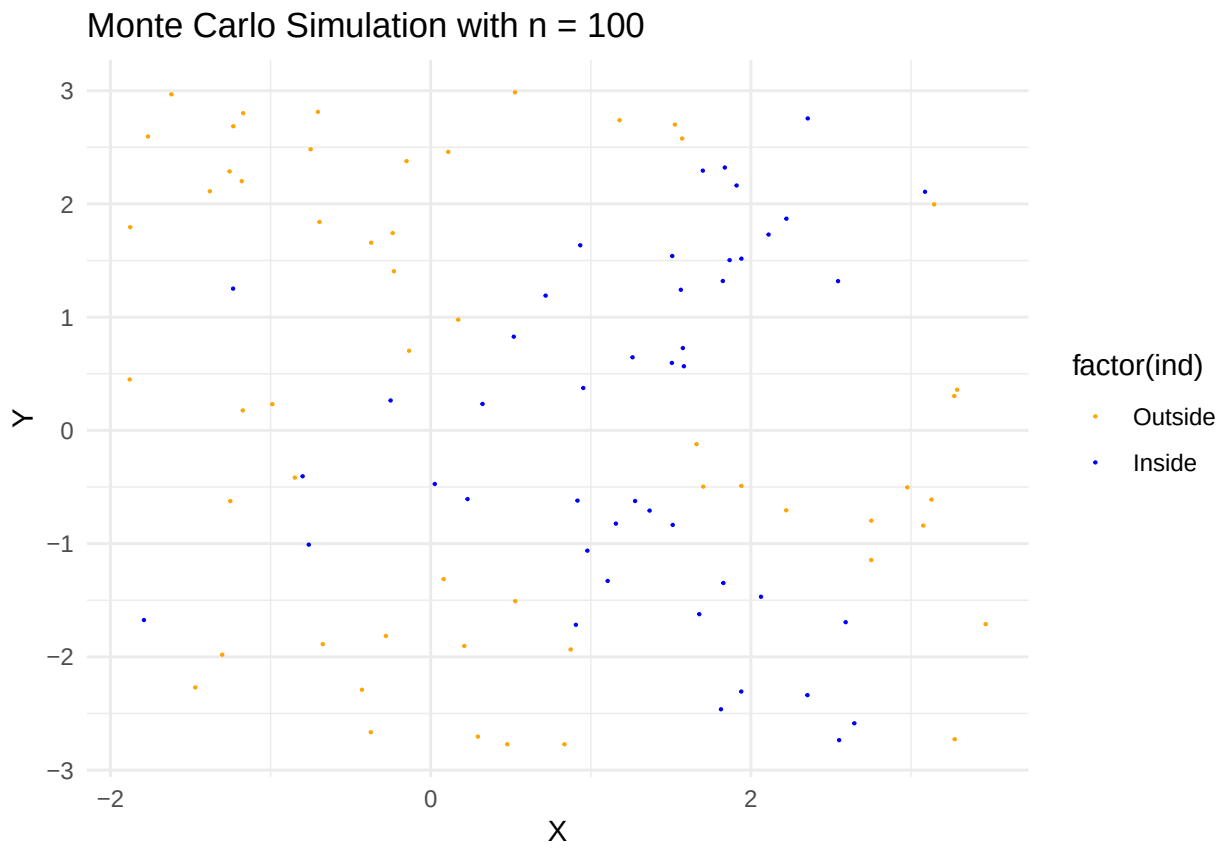
Step 3: Estimation and Visualization

Now we are going to simulate 100, 1000, 10000 and 100000 random coordiantes and calculate the percentage of the points within the enclosed area. And base on this information we want to estimate the area of the figure.

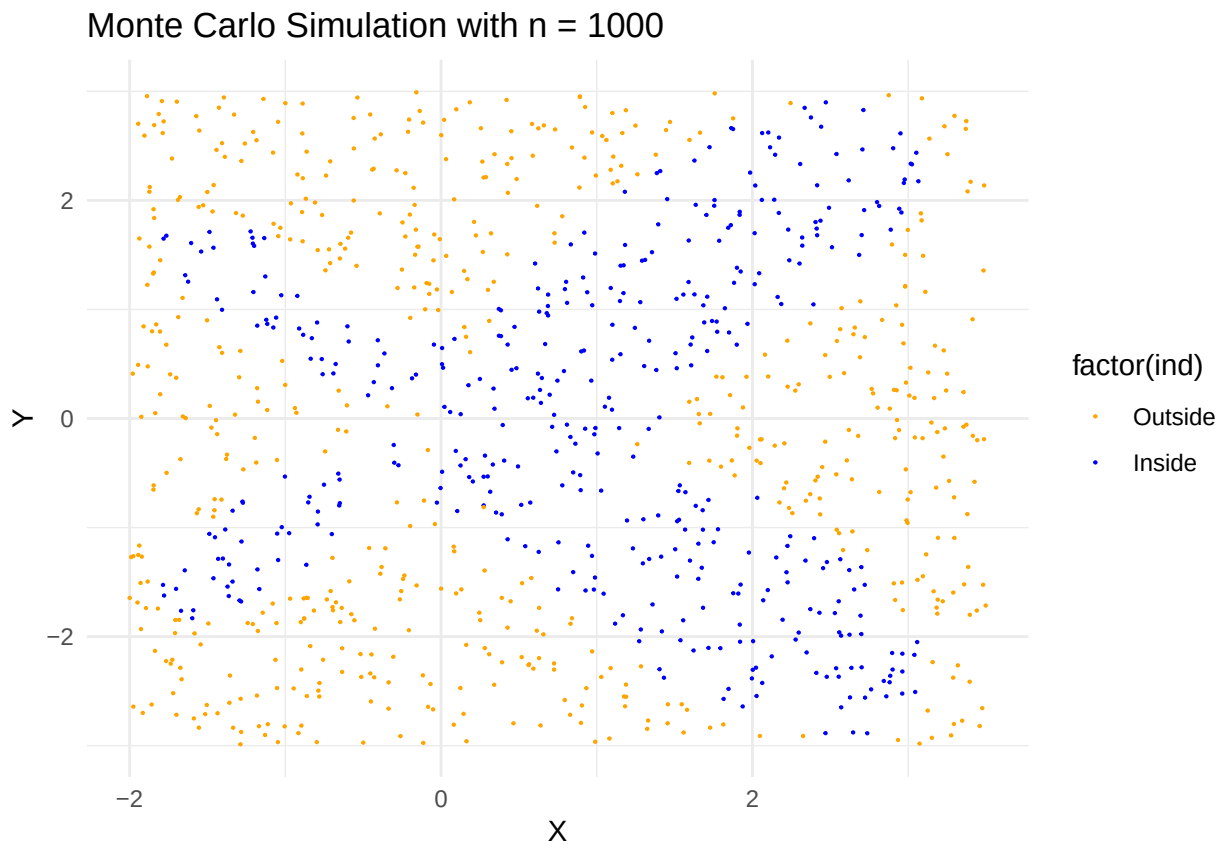
```
# Define r(t)
r <- function(t) {
  exp(cos(t)) - 2 * cos(4 * t) - sin(t / 12)^5
}

# Bounding box limits
x_min <- -2
x_max <- 3.5
y_min <- -3
y_max <- 3

# Bounding box area
bounding_box_area <- (x_max - x_min) * (y_max - y_min)

# Initialize the vector to store area estimations for each sample size
estimator <- numeric(4)

# Function to determine if a point lies within the defined area
inside_func <- function(x, y) {
  beta <- atan2(y, x)
```

```r
  rad <- sqrt(x^2 + y^2)

  r_beta <- r(beta)
  r_beta_pi <- r(beta + pi)

  return((r_beta > 0 & rad < abs(r_beta)) || (r_beta_pi < 0 & rad < abs(r_beta_pi)))
}

# Loop over different sample sizes (10^2, 10^3, 10^4, and 10^5)
for (i in 2:5) {
  n <- 10^i     # Sample size
  xs <- runif(n, min = x_min, max = x_max)  # Random x-coordinates
  ys <- runif(n, min = y_min, max = y_max)  # Random y-coordinates

  # Indicator for points within the enclosed area
  ind <- numeric(n)
  for (j in 1:n) {
    ind[j] <- inside_func(xs[j], ys[j])
  }

  # Calculate the percentage of points inside and the estimated area
  percentage_inside <- mean(ind)
  estimated_area <- percentage_inside * bounding_box_area
  estimator[i - 1] <- estimated_area

  # Print the percentage and estimated area for the current sample size
  print(paste("Sample size:", n,
              "- Percentage Inside:", round(percentage_inside * 100, 2),
              "% - Estimated Area:", round(estimated_area, 4)))

  # Create a ggplot for the current sample size, showing points inside/outside the area
  df <- data.frame(xs = xs, ys = ys, ind = ind)
  p <- ggplot() +
    geom_point(data = df, aes(x = xs, y = ys, color = factor(ind)), size = 0.1) +
    scale_color_manual(values = c("0" = "orange", "1" = "blue"), labels = c("Outside", "Inside")) +
    ggtitle(paste("Monte Carlo Simulation with n =", n)) +
    xlab("X") +
    ylab("Y") +
    theme_minimal()

  # Print the plot for the current sample size
  print(p)
}
```

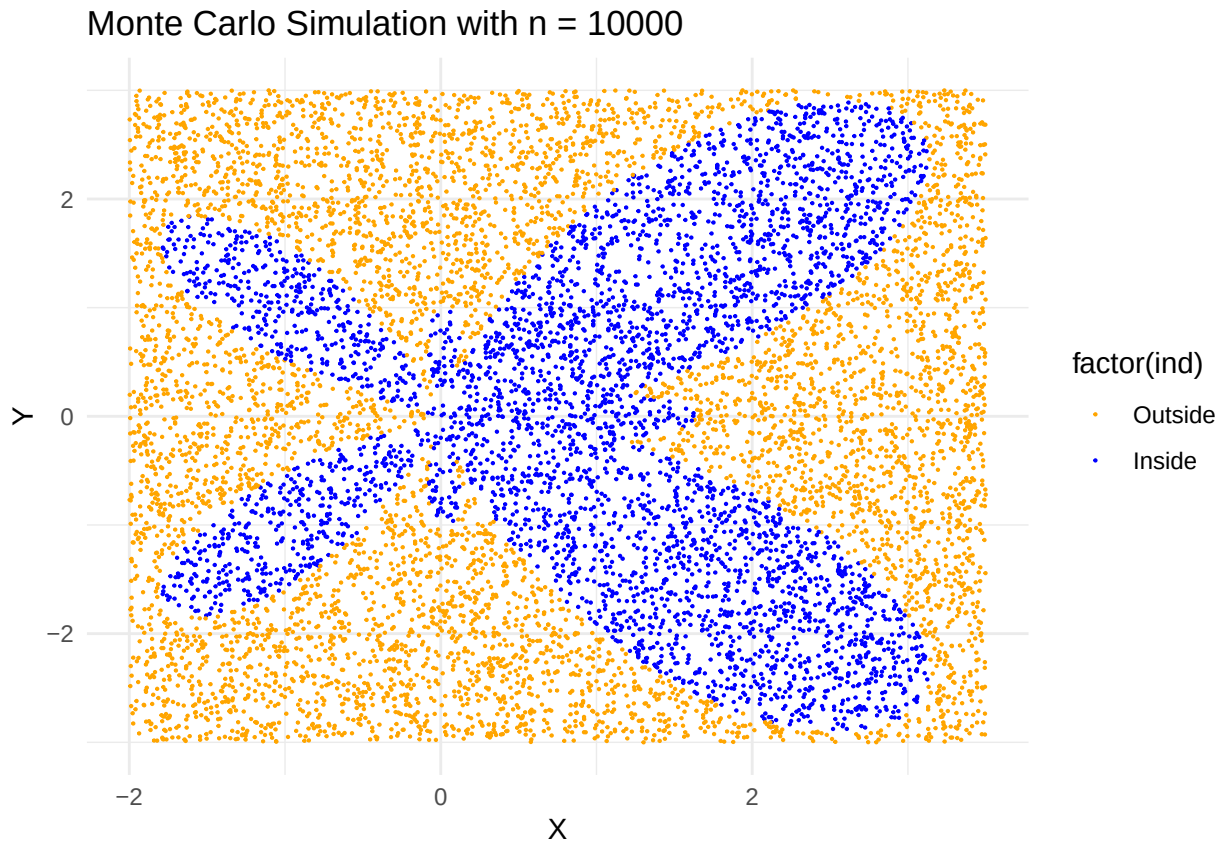
```
## [1] "Sample size: 100 - Percentage Inside: 46 % - Estimated Area: 15.18"
```



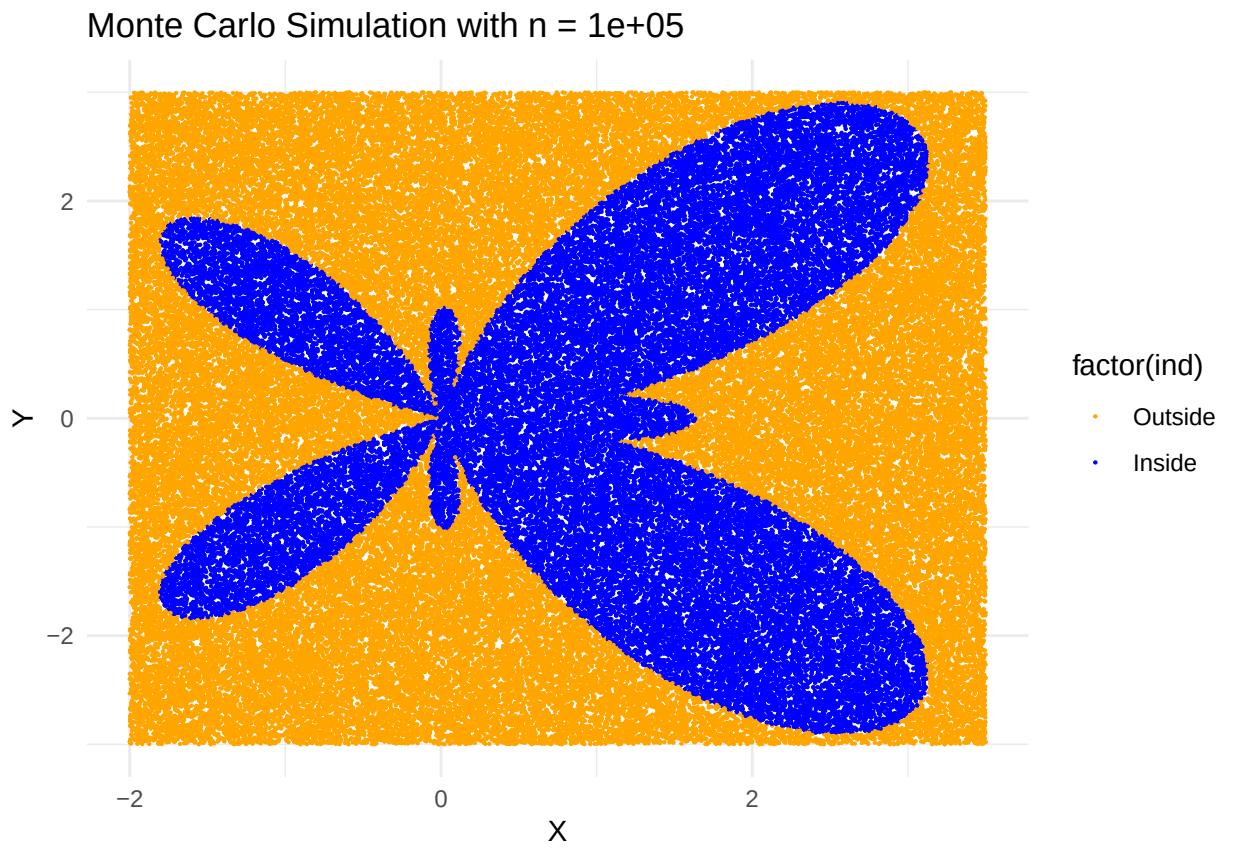

```
## [1] "Sample size: 1000 - Percentage Inside: 41.4 % - Estimated Area: 13.662"
```




```
## [1] "Sample size: 10000 - Percentage Inside: 40.83 % - Estimated Area: 13.4739"
```




```
## [1] "Sample size: 1e+05 - Percentage Inside: 39.86 % - Estimated Area: 13.1528"
```




```
# Summarize the results in a table
results <- data.frame(Sample_Size = c(10^2, 10^3, 10^4, 10^5),
                      Estimated_Area = estimator)

print(results)
```

```
##   Sample_Size Estimated_Area
## 1       1e+02       15.18000
## 2       1e+03       13.66200
## 3       1e+04       13.47390
## 4       1e+05       13.15281
```

Step 4: Functionality of Monto Carlo Simulation

Monte Carlo is a powerful statistical technique that leverage random sampling to estimate mathematical functions or model that might be difficult or impossible to compute directly.In this exercise, the Monte Carlo simulation estimates the area enclosed by the curve by generating random points within a defined bounding box. We check if each point falls inside the region defined by $r(t)$. The proportion of points inside the shape compared to the total points gives an estimate of the area's percentage of the bounding box. By multiplying this percentage by the area of the bounding box, we approximate the enclosed area. More points generally improve accuracy, as seen with larger sample sizes in the simulations.