Exercise 3 - Monte Carlo Simulation of areas

Konstantinos Vakalopoulos 12223236

2023-10-25

Contents

Task 1

Consider the integral $\int_1^b e^{-x^3} dx$

Task 1.1

Use uniformly distributed random variables to approximate the integral for b = 6 (using Monte Carlo integration). Then use the function integrate for comparison.

We are going to follow the procedure from the slides of the lecture. We assign b = 6, we set a seed (12223236) for reproducibility purposes and we generate the random values using the uniform distribution. Afterwards, we calculate the integration using the Monte Carlo method and the function integrate.

```
b <- 6

# Monte Carlo integration
set.seed(12223236)
us <- runif(100000,min=1, max=b)
cat(paste("Monte Carlo Integral:",mean(exp(-us^3)*(b-1))))
```

```
## Monte Carlo Integral: 0.0867967177023595
```

```
# Function integrate
cat("\nFunction Integral: ")
```

```
##
## Function Integral:
```

```
integrate(f = function(x){exp(-x^3)},lower = 1,upper = b)
```

```
## 0.08546833 with absolute error < 3.2e-07
```

Task 1.2

Use Monte Carlo integration to compute the integral for $b = \infty$. What would be a good density for the simulation in that case? Use also the function integrate for comparison.

In a repetitive manner and according to the lectures notes, the exponential distribution can be used. In general, due to the fact that the integral from 1 to ∞ must be calculated, it needs to be used a distribution where $x \in [0, \infty]$. One distribution where $x \in [0, \infty]$ is the exponential distribution. Also, the integral that needs to be calculated contains exponential factors (e^{-x^3}). Therefore, it is reasonable to use this specific distribution.

The below chuck of code represents the implementation of this task.

```
# exponential distribution
set.seed(12223236)
x <- rexp(100000)
cat(paste("Monte Carlo Integral:",mean(exp(-(x+1)^3)/dexp(x))))
```

```
## Monte Carlo Integral: 0.0848411938025795
```

```r
# Function integrate
cat("\nFunction Integral: ")
```

```
##
## Function Integral:
```

```r
integrate(f = function(x){exp(-x^3)},lower = 1,upper = Inf)
```

```
## 0.08546833 with absolute error < 6.2e-06
```

Task 1.3

Do you have an explanation why Monte Carlo integration agrees in 2. with integrate but not so much in 1.?

The reason why the Monte Carlo integration agrees in task 2 with integrate but not so much in task 1 is because of the sample size. In the first case, we take values from range [1,6] and the result is more accurate when we have smaller sample size. Specifically, in case where we modify the runif() function and take, for instance, a sample equals to 100, which is relatively small, the result in the first case (uniform distribution) will be more accurate compared to the second case (exponential distribution). On the other hand, if we take a large sample size, the inverse prerequisite will hold true. This can be observed for sample size equals to 100 and 1000000 below:

```r
sample <- 100

# Uniform distribution
b <- 6
set.seed(12223236)
us <- runif(sample,min=1, max=b)
cat(paste("Monte Carlo Integral:",mean(exp(-us^3)*(b-1))))
```

```
## Monte Carlo Integral: 0.0850758798194259
```

```r
# Function integrate
cat("\nGround truth for b = 6: ")
```

```
##
## Ground truth for b = 6:
```

```r
integrate(f = function(x){exp(-x^3)},lower = 1,upper = b)
```

```
## 0.08546833 with absolute error < 3.2e-07
```

```r
# Exponential distribution
set.seed(12223236)
x <- rexp(sample)
cat(paste("Monte Carlo Integral:",mean(exp(-(x+1)^3)/dexp(x))))
```

```
## Monte Carlo Integral: 0.0629374897201474
```

```r
# Function integrate
cat("\nGround truth for b = inf: ")
```

```
##
## Ground truth for b = inf:
```

```r
integrate(f = function(x){exp(-x^3)},lower = 1,upper = Inf)
```

```
## 0.08546833 with absolute error < 6.2e-06
```

```r
sample <- 1000000

# Uniform distribution
b <- 6
set.seed(12223236)
us <- runif(sample,min=1, max=b)
cat(paste("Monte Carlo Integral:",mean(exp(-us^3)*(b-1))))
```

```
## Monte Carlo Integral: 0.0856680073002578
```

```r
# Function integrate
cat("\nGround truth for b = 6: ")
```

```
##
## Ground truth for b = 6:
```

```r
integrate(f = function(x){exp(-x^3)},lower = 1,upper = b)
```

```
## 0.08546833 with absolute error < 3.2e-07
```

```r
# Exponential distribution
set.seed(12223236)
x <- rexp(sample)
cat(paste("Monte Carlo Integral:",mean(exp(-(x+1)^3)/dexp(x))))
```

```
## Monte Carlo Integral: 0.0853381089723776
```

```r
# Function integrate
cat("\nGround truth for b = inf: ")
```

```
##
## Ground truth for b = inf:
```

```r
integrate(f = function(x){exp(-x^3)},lower = 1,upper = Inf)
```

```
## 0.08546833 with absolute error < 6.2e-06
```

Therefore, the Monte Carlo simulation depends to a large extent on the sample size.

Task 2

Monte Carlo simulation shall be utilized for obtaining the area enclosed by the graph of the function $r(t) = e^{\cos(t)} - 2 \cdot \cos(4t) - \sin\left(\frac{t}{12}\right)^5$ for $t \in [-\pi, \pi]$, when using polar x-coordinates $x(t) = r(t) \cdot \sin(t)$ and y-coordinates $y(t) = r(t) \cdot \cos(t)$.

Task 2.1

Visualize the function and the area.

First, we create the function r(t).

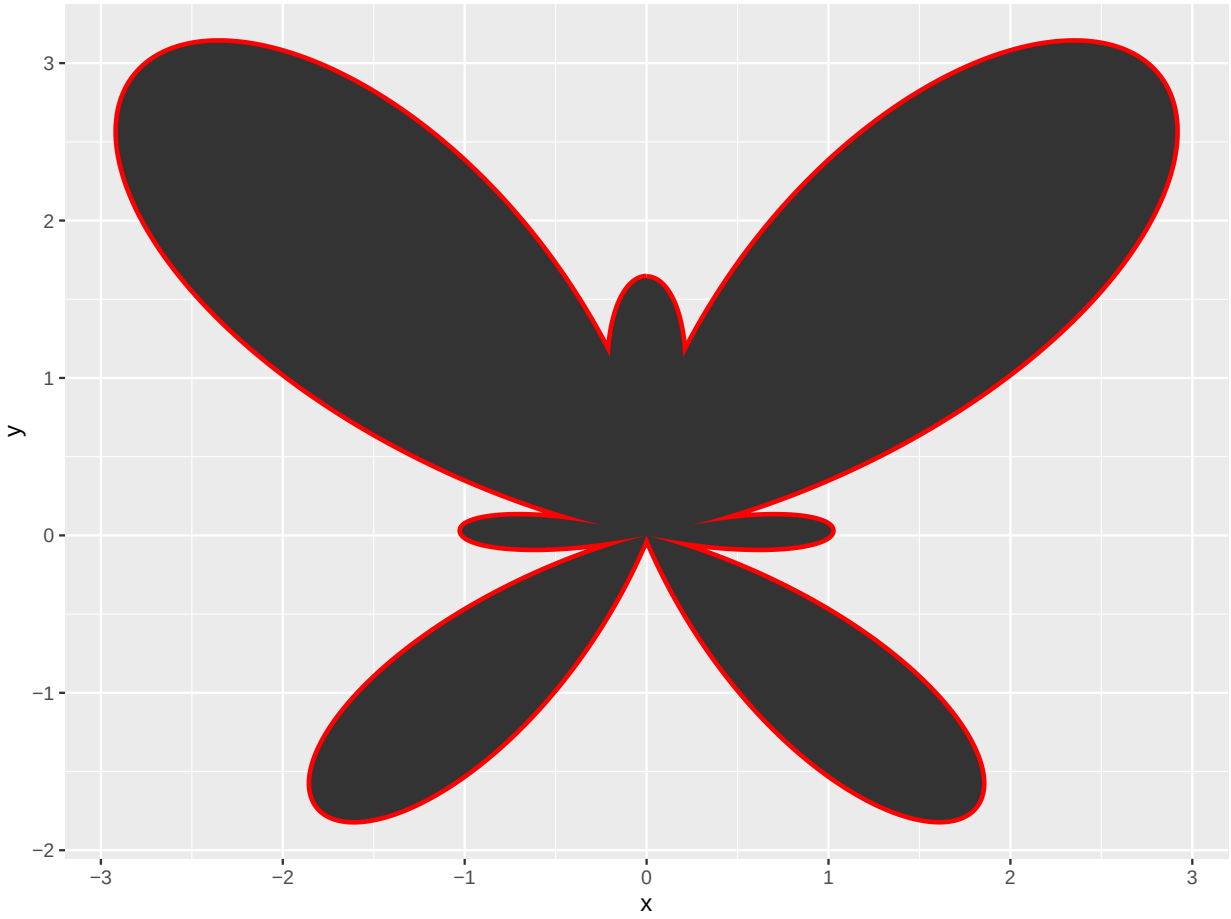
```
r <- function(t){
  exp(cos(t))-2*cos(4*t)-sin((t/12))^5}
```

Then, we take $t \in [-\pi, \pi]$, create the x(t) and y(t). Afterwards, we visualize the function and the area.

```
t <- seq(-pi,pi,length.out = 1000)
x <- r(t)*sin(t)
y <- r(t)*cos(t)
data <- data.frame(x,y)
```

```
# Import the ggplot2 library
library(ggplot2)
```

```
ggplot(data = data, aes(x = x, y = y))+
  geom_path(linewidth = 2, color = "red")+
  geom_polygon()
```



(Note: The plot is represented better in the rmarkdown. In the pdf, the area inside the function overplots the curve)

Task 2.2

Generate uniform random coordinates within the rectangle [-3,3]x[-2,3.5] and an indicator whether this point lies within the area in question.

In the following code block, an indicator function is represented. Essentially, the function takes as input the sample size and a data frame is initialized, which contains uniform random coordinates within the rectangle [-3,3]x[-2,3.5] and a logical variable inside_area. This variable is responsible for whether or not a specific point lies inside the curve from task 2.1.

The idea behind the function is that the uniform random coordinates are transformed into polar coordinates by calculating the angle θ and the distance r using the formulas: $\theta = \tan^{-1}(\frac{x}{y})$ and $r = \sqrt{(x^2 + y^2)}$, respectively. Also, it is checked if the y point is lower than zero. In case it holds true, we add to the θ the π. Therefore, it is checked if the point is inside the curve or not and it is assigned to the logical variable inside_area TRUE or FALSE, respectively.

```
indicator <- function(sample){
  set.seed(12223236)
  # Initialization of the data frame
  points <- data.frame(x = runif(sample, min = -3, max = 3),
```

```
                      y = runif(sample, min = -2, max = 3.5),
                      inside_area = logical(sample))

  for (i in 1:dim(points)[1]){

    # Check if the y value is lower or higher than zero
    if (points$y[i] > 0){
      degree <- 0
    }
    else {
      degree <- pi
    }

    # Polar Coordinates calculation
    theta_value <- atan(points$x[i]/points$y[i]) + degree
    r_value <- sqrt(points$x[i]^2 + points$y[i]^2)

    temp1 <- FALSE
    temp2 <- FALSE

    # Assign TRUE of FALSE, depending on whether the point is inside the curve
    if (r(theta_value) > 0 & (r_value < abs(r(theta_value)))) {
      temp1 <- TRUE
    }
    if (r(theta_value+pi) < 0 & (r_value < abs(r(theta_value + pi)))) {
      temp2 <- TRUE
    }
    if (temp1 && temp2) {
      points$inside_area[i] <- FALSE
    }
    else if (temp1|temp2){
      points$inside_area[i] <- TRUE
    }
    else{
      points$inside_area[i] <- FALSE
    }
  }
  return(points)
}
```

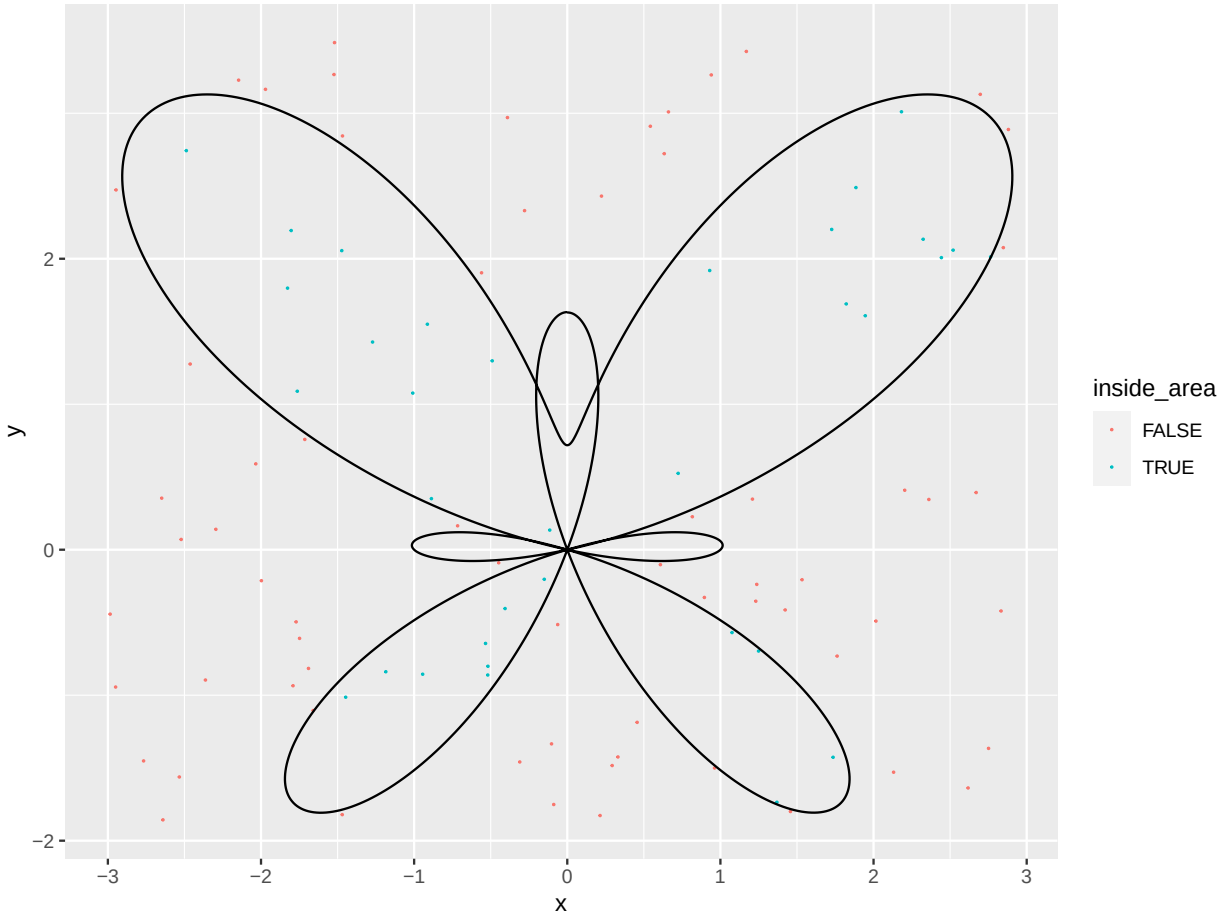
Task 2.3

Simulate 100, 1000, 10000 and 100000 random coordinates and calculate the percentage of the points within the enclosed area. Based on this information estimate the area of the figure. Summarize those values in a table and visualize them in plots of the function curve and enclosed area.

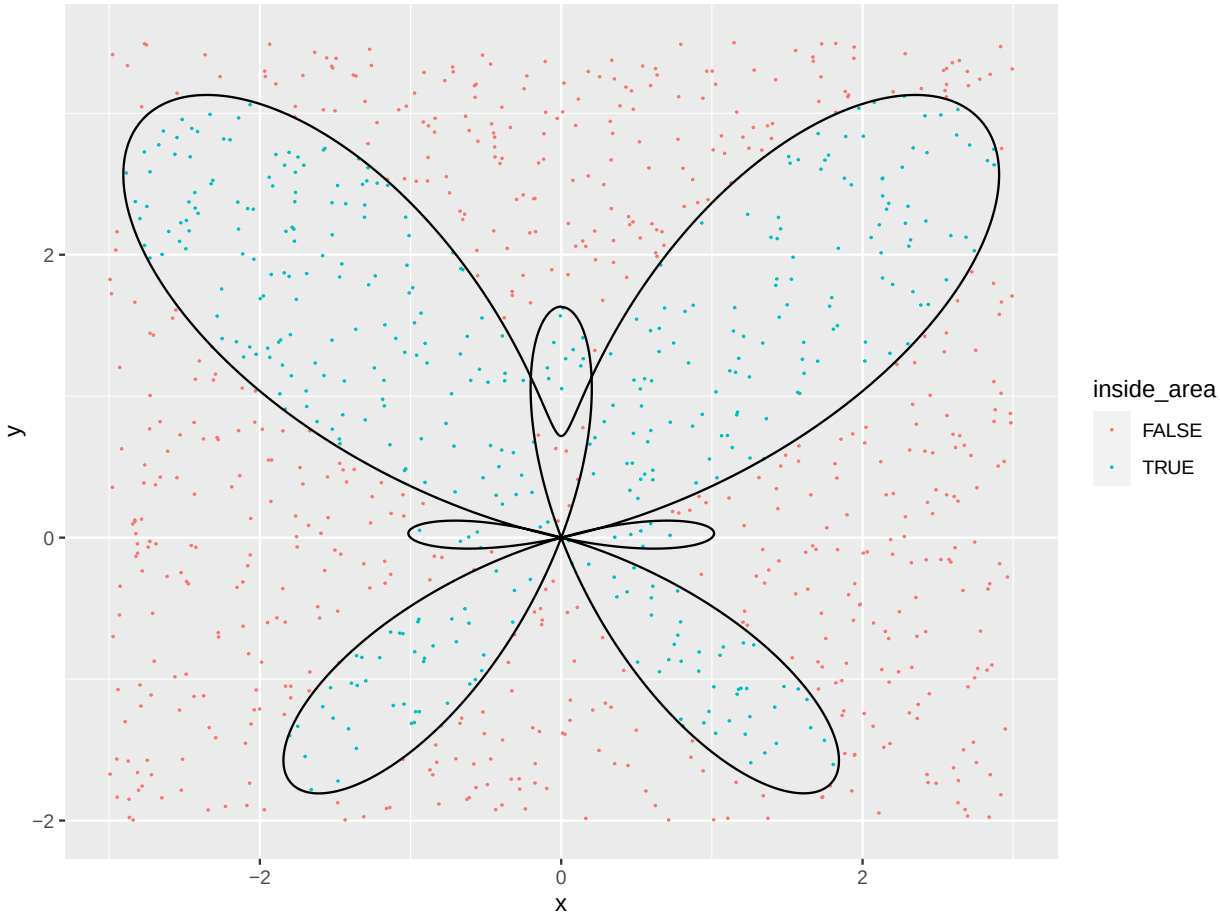
In the following plots, we visualize the function curve for the 4 different sample sizes. Essentially, the indicator function from the previous task is called four times and the points that are inside and outside of the curve are determined by the variable inside_area of the data frame.

```
simulations <- c(100, 1000, 10000, 100000)
```

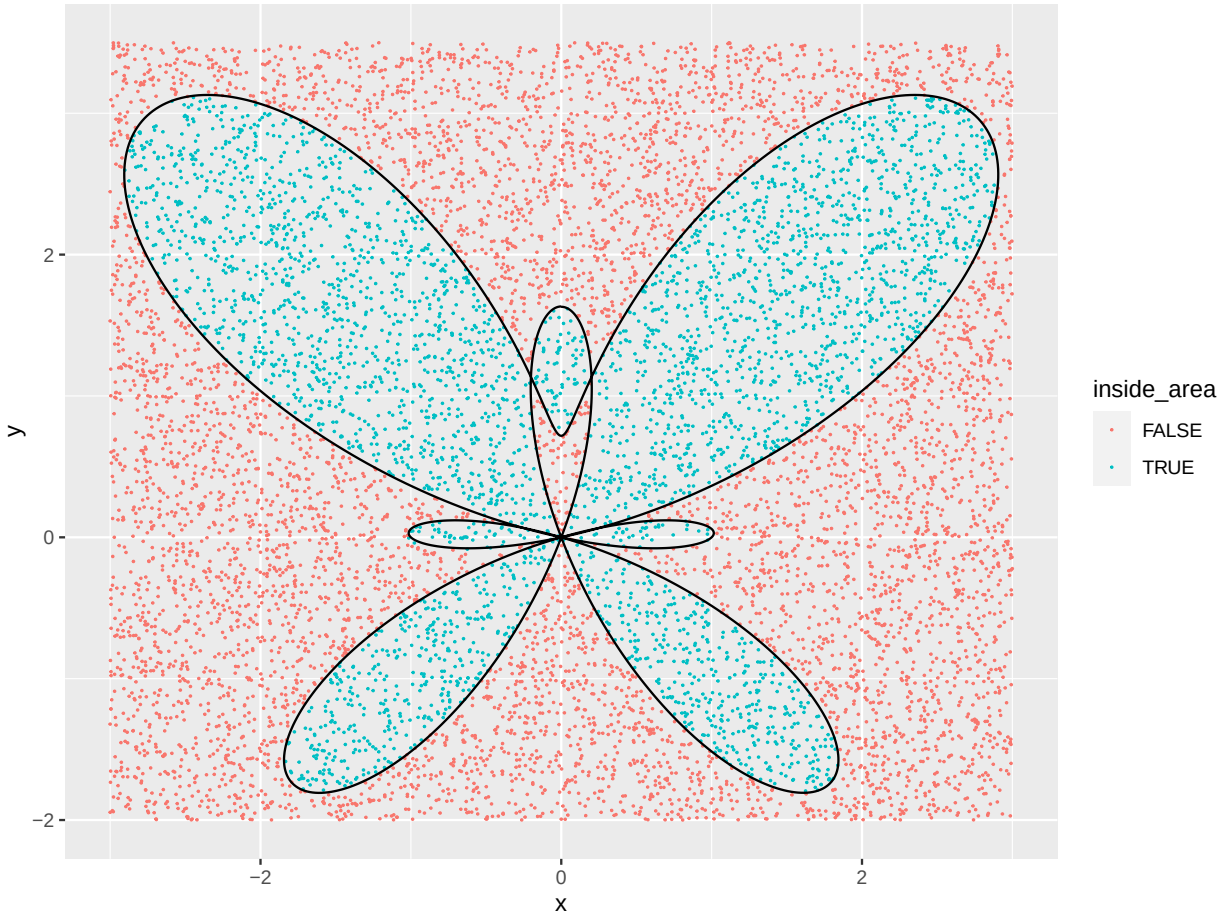
```r
# Sample 100
points <- indicator(simulations[1])
ggplot()+
  geom_point(data= points, aes(x=x, y=y, color = inside_area), size = 0.1)+
  geom_path(aes(x=x, y=y))
```



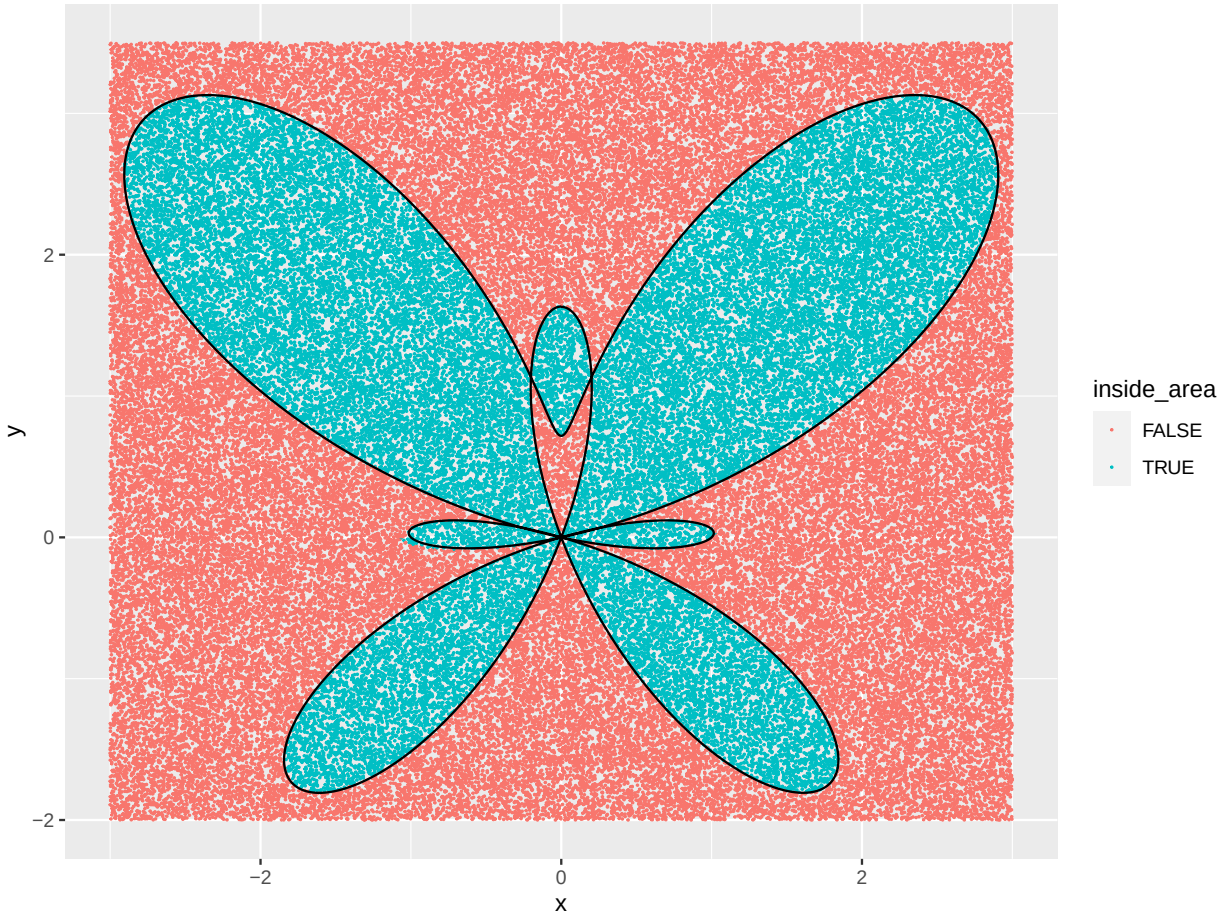

```r
# Sample 1000
points <- indicator(simulations[2])
ggplot()+
  geom_point(data= points, aes(x=x, y=y, color = inside_area), size = 0.1)+
  geom_path(aes(x=x, y=y))
```




```
# Sample 10000
points <- indicator(simulations[3])
ggplot()+
  geom_point(data= points, aes(x=x, y=y, color = inside_area), size = 0.1)+
  geom_path(aes(x=x, y=y))
```




```
# Sample 100000
points <- indicator(simulations[4])
ggplot()+
  geom_point(data= points, aes(x=x, y=y, color = inside_area), size = 0.1)+
  geom_path(aes(x=x, y=y))
```



Afterwards, the the percentage of the points within the enclosed area and the estimated area of the figure is calculated. The percentage is calculated based on the formula: $percentage = \frac{points\ inside\ the\ curve}{sample\ size}$ and the estimated area: $area = 6 \cdot 5.5 \cdot percentage$. The 6 multiplied by 5.5 is the rectangle area.

```
percentage <- c()
estimated_area <- c()
# For loop for each sample size
for (sample in simulations){

  # Call the indicator function
  points <- indicator(sample)

  # Sum the points inside the curve
  correct_points <- sum(points$inside_area==TRUE)

  # Percentage calculation
  per <- correct_points/sample

  percentage <- c(percentage,per)

  # Area calculation
  estimated_area <- c(estimated_area,per*6*5.5)  # Rectangle area
}
```

```
result.table <- data.frame(simulations, percentage, estimated_area)
```

The table below shows the percentage and area results.

```
library(knitr)
kable(result.table, col.names = c("Sample", "Percentage", "Area"),
      format = "markdown",
      caption = "Percentage and Area results")
```

Table 1: Percentage and Area results

Sample	Percentage	Area
1e+02	0.34000	11.22000
1e+03	0.40300	13.29900
1e+04	0.38040	12.55320
1e+05	0.39172	12.92676

Task 2.4

Explain the functionality of Monte Carlo simulation in your own words referring to these simulations.

Monte Carlo simulation is a powerful statistical technique used to model complex systems or processes. Specifically, there is a degree of uncertainty involved because it involves running a large number of random simulations to estimate the range of possible outcomes. By repeatedly sampling from a wide range of input parameters and running these simulations, you can gain insights into the behavior and potential outcomes of the system or the model. This is evident from the four different simulations conducted in this exercise. For each sample size, random coordinates were drawn from a uniform distribution, and it was checked whether these points belong to the desired curve. In this way, by using random values, we managed to approximate the function curve to a large extent.