Exercise 3 - Monte Carlo Simulation of areas

107.330 - Statistical Simulation and Computerintensive Methods, WS24

12433688 - Yash Lucas

25.10.2024

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 assign b = 6, we set a seed (12433688) for reproducibility purposes and we generate the random values using the uniform distribution.
- To use Monte Carlo integration to approximate the integral $\int_1^b e^{-x^3} dx$, we shall generate a uniform distribution of random values between 0 and 6.
- Afterwards, we calculate the integration using the Monte Carlo method and the function integrate.

```
set.seed(12433688)
uni_d <- runif(100000, min=1, max=6)
uni_d<-mean(exp(-1*uni_d^3))*(6-1)
uni_d
```

```
## [1] 0.0853226
```

```
mc_f_int<-integrate(f= function(x){exp(-1*x^3)}, lower=1, upper=6)
mc_f_int
```

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

Task 1.2

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

- Monte Carlo integration across an infinite domain, such [0,∞], usually results in a "fading" or "decaying" function, where the function being integrated tends to approach zero as x grows. Inefficient sampling would result from sampling too many points when the function's value is already close to zero if we used a uniform distribution to sample random points between 0 and infinity.
- It will leade to wastage of processing resources on regions where the function adds little to the integral, uniform sampling is not the best option in this case.

```
set.seed(12433688)
X <- rexp(100000)
uni_d1<-mean(exp(-1*(X+1)^3)/dexp(X))
uni_d1
```

```
## [1] 0.08563886
```

```
mc_f_int1<-integrate(f= function(x){exp(-1*x^3)}, lower=1, upper=Inf)
mc_f_int1
```

```
## 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.?

- In task 1, the Monte Carlo integration is less in agreement with integrate than in job 2. The sample size is to blame for this. In the first case, we utilize numbers from the range [1,6], and a smaller sample size produces a more accurate result. Specifically, the first example (uniform distribution) will produce a more accurate result than the second case (exponential distribution) if we change the runif() method to take, say, a sample of 100, which is a tiny quantity. However, if we utilize a high sample size, the reverse need will be true.
- Integrating for b=∞ and + Monte Carlo integration agree well because the decay pattern of the function is matched by importance sampling, which increases estimate efficiency. The structure of the integrand for b=6 is not exploited by uniform sampling in Monte Carlo integration, which results in slower convergence and poorer agreement with the more accurate integrate function.

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

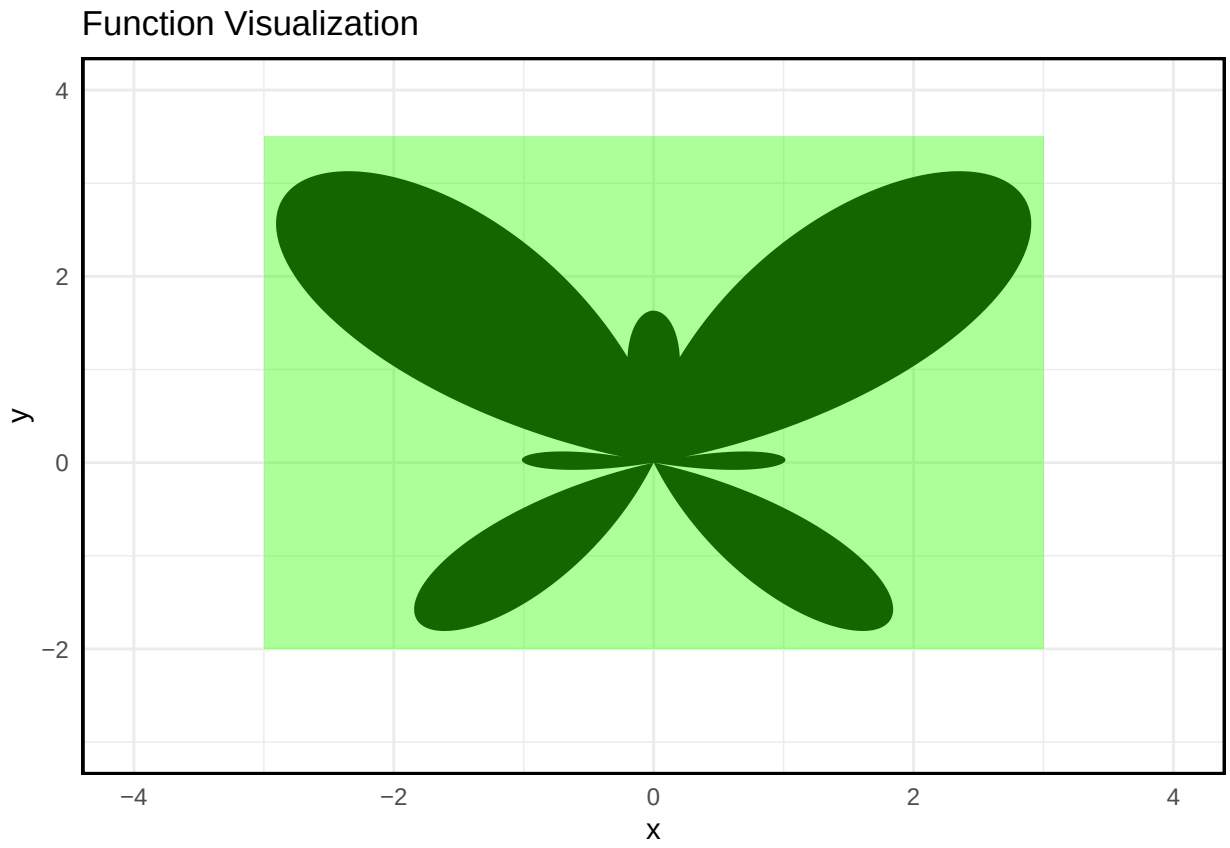
Visualize the function and the area.

- To begin, we construct the function r(t). After that, we pick $t \in [-\pi, \pi]$ and produce x(t) and y(t). We then see the region and the function.
- In summary, the code below displays a complicated curve in polar form that is specified by the function, fills in the area underneath the curve with dark green, and overlays a yellow semi-transparent rectangle on top of the plot.

```
library(ggplot2)
library(sp)

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

mppi <- seq(-pi, pi, 0.01)

x_vals <- rad(mppi) * sin(mppi)
y_vals <- rad(mppi) * cos(mppi)

df_curve <- data.frame(x = x_vals, y = y_vals)
df_polygon <- data.frame(
  x = c(-3, 3, 3, -3),
  y = c(-2, -2, 3.5, 3.5)
)
ggplot() +
  geom_polygon(data = df_curve, aes(x = x, y = y), fill = "black") +
  geom_polygon(data = df_polygon, aes(x = x, y = y), fill = rgb(0.2, 1, 0, 0.4))+
  coord_cartesian(xlim = c(-4, 4), ylim = c(-3, 4)) +
```

```
  theme_minimal() +
  theme(panel.border = element_rect(colour = "black", fill = NA, size = 1))+
  labs(x = "x", y = "y", title = "Function Visualization")
```

```
## Warning: The `size` argument of `element_rect()` is deprecated as of ggplot2 3.4.0.
## i Please use the `linewidth` argument instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```



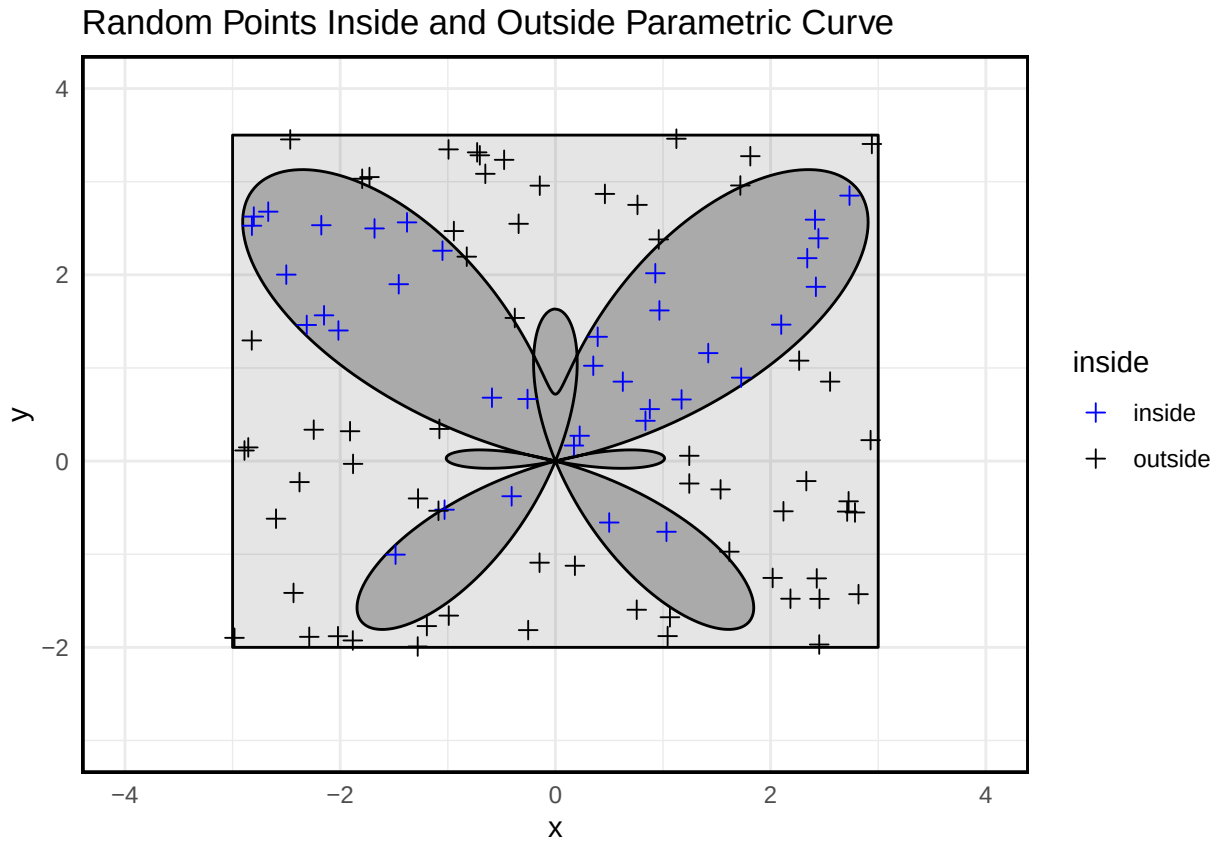
Task 2.2

Generate uniform random coordinates within the rectangle and an indicator whether this point lies with area in question.

- We generated 100 random x-coordinates that were evenly distributed between -3 and 3 and 100 random y-coordinates that were evenly spread between -2 and 3.5 using the runif function. These locations are situated inside the rectangle $[-3,3] \times [-2,3.5]$, which is the bounding box of the polar curve.

```
set.seed(12433688)
ran_x <- runif(100, -3, 3)
ran_y <- runif(100, -2, 3.5)
mycol <- point.in.polygon(ran_x, ran_y, x_vals, y_vals)
```

```r
df_box <- data.frame(
  x = c(-3, 3, 3, -3),
  y = c(-2, -2, 3.5, 3.5)
)
df_points <- data.frame(x = ran_x,y = ran_y,inside = factor(ifelse(mycol > 0, "inside", "outside")))
ggplot() +
  geom_polygon(data = df_curve, aes(x = x, y = y), fill = "gray", color = "black") +
  geom_polygon(data = df_box, aes(x = x, y = y), fill = rgb(0, 0, 0, 0.1), color = "black") +
  geom_point(data = df_points, aes(x = x, y = y, color = inside), shape = 3, size = 2) +
  scale_color_manual(values = c("inside" = "blue", "outside" = "black")) +
  coord_cartesian(xlim = c(-4, 4), ylim = c(-3, 4)) +
  labs(x = "x", y = "y", title = "Random Points Inside and Outside Parametric Curve") +
  theme_minimal() +
  theme(panel.border = element_rect(colour = "black", fill = NA, size = 1))
```



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. Summaries those values in a table and visualize them in plots of the function curve and enclosed area.

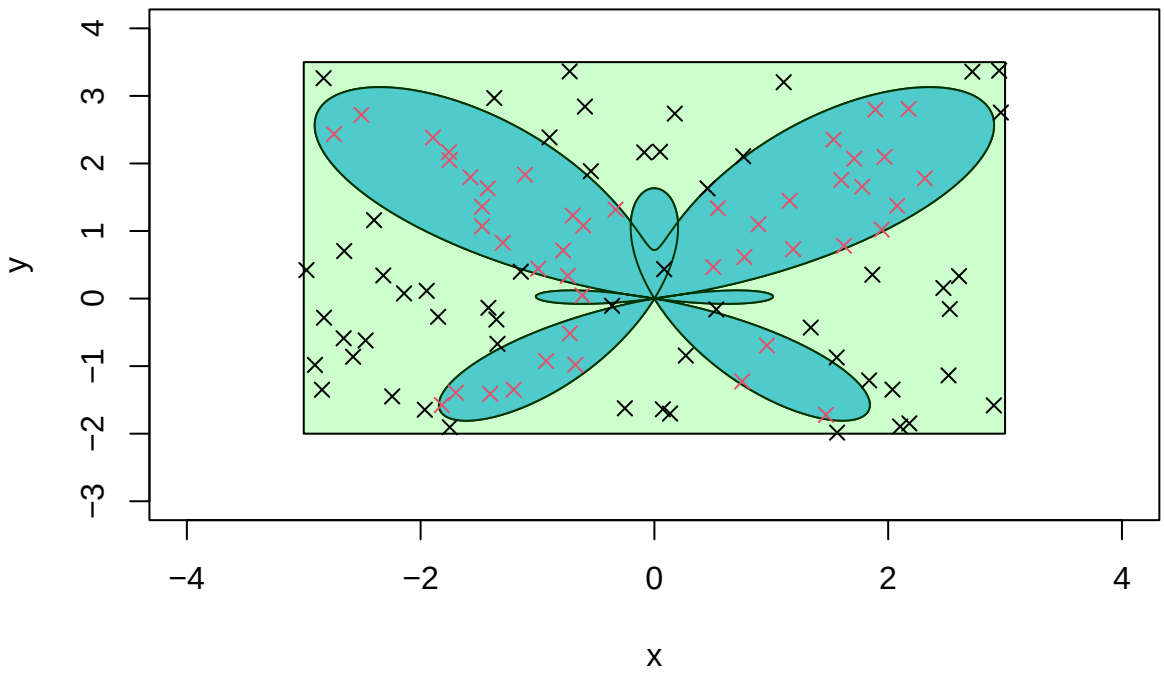
```r
find_area <- function(n) {
  ran_d_x <- runif(n, -3, 3)
  ran_d_y <- runif(n, -2, 3.5)
```

```r
  mycol <- point.in.polygon(ran_d_x, ran_d_y, rad(mppi) * sin(mppi), rad(mppi) * cos(mppi))
  estimated_area <- (sum(mycol) / n) * (6 * 5.5)
  return(estimated_area)
}
```

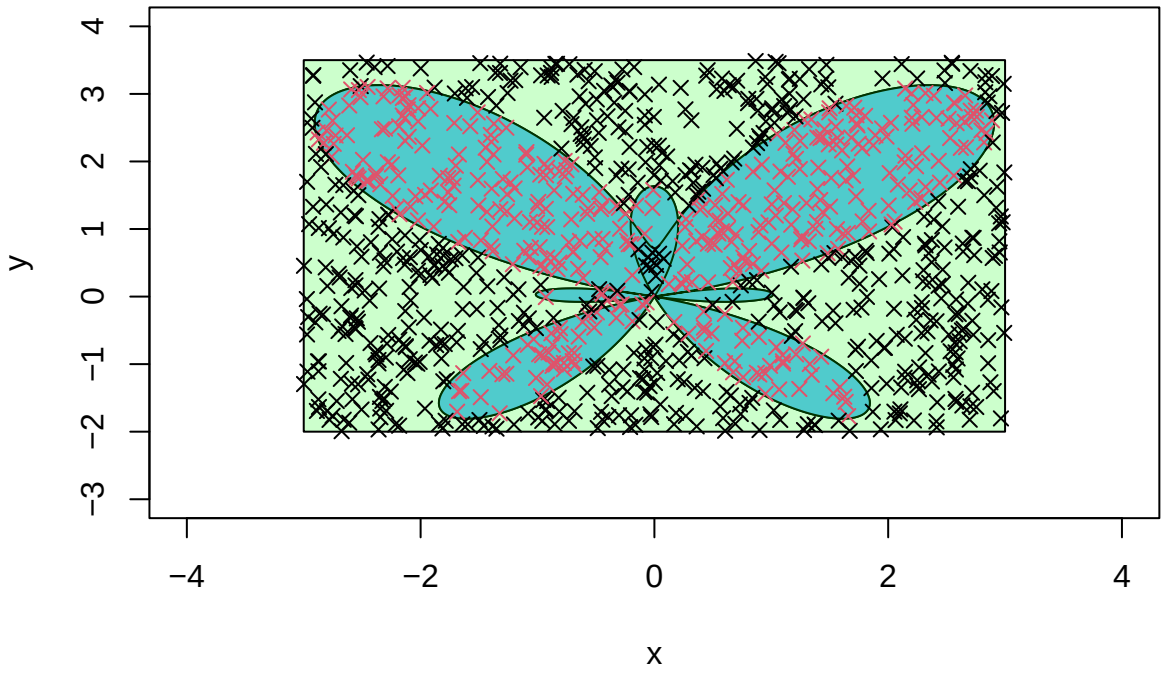
```r
plot_viz <- function(n) {
  ran_d_x <- runif(n, -3, 3)
  ran_d_y <- runif(n, -2, 3.5)
  mycol <- point.in.polygon(ran_d_x, ran_d_y, rad(mppi) * sin(mppi), rad(mppi) * cos(mppi))
  plot(rad(mppi) * sin(mppi), rad(mppi) * cos(mppi), type = 'l', xlim = c(-4, 4), ylim = c(-3, 4),xlab =
  polygon(rad(mppi) * sin(mppi), rad(mppi) * cos(mppi), col = "#65BFFF")
  polygon(c(-3, 3, 3, -3), c(-2, -2, 3.5, 3.5), col = rgb(0, 1, 0, 0.2))
  points(ran_d_x, ran_d_y, col = mycol + 1, pch = 4)
}
```

```r
table_1 <- function(samples) {
  results <- data.frame(Samples = integer(), Estimated_Area = numeric())

  for (n in samples) {
    estimated_area <- find_area(n)  # Call the function to calculate area
    results <- rbind(results, data.frame(Samples = n, Estimated_Area = estimated_area))  # Append resul
  }
  return(results)
}
```

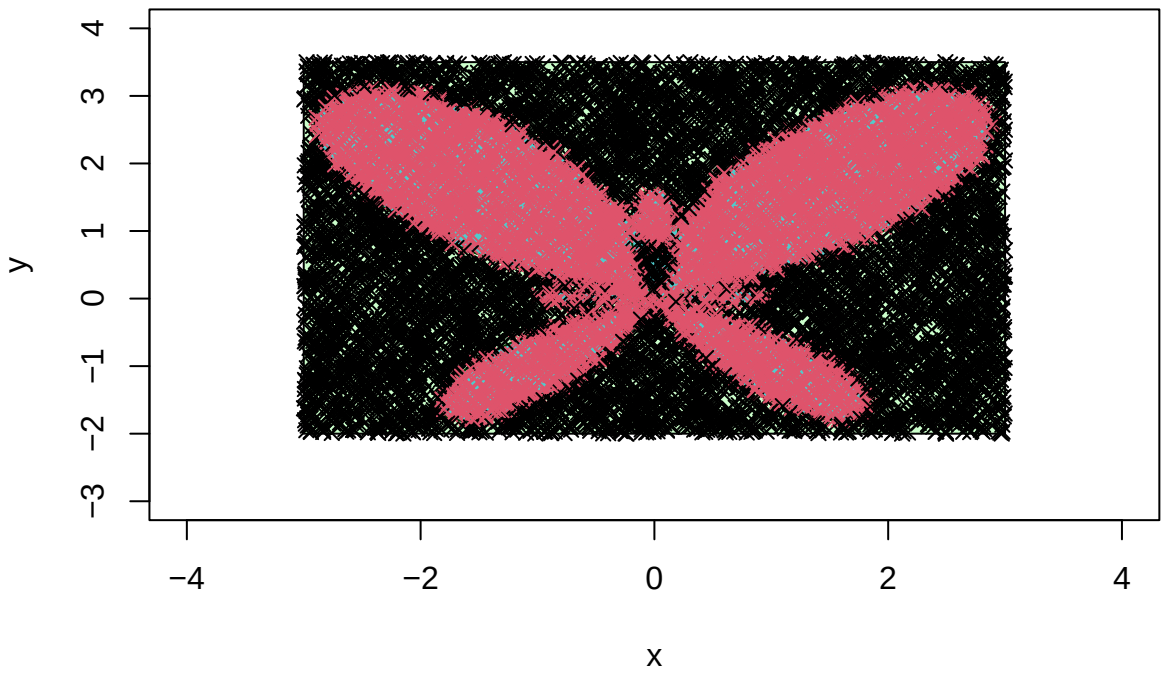
```r
plot_viz(100)
```



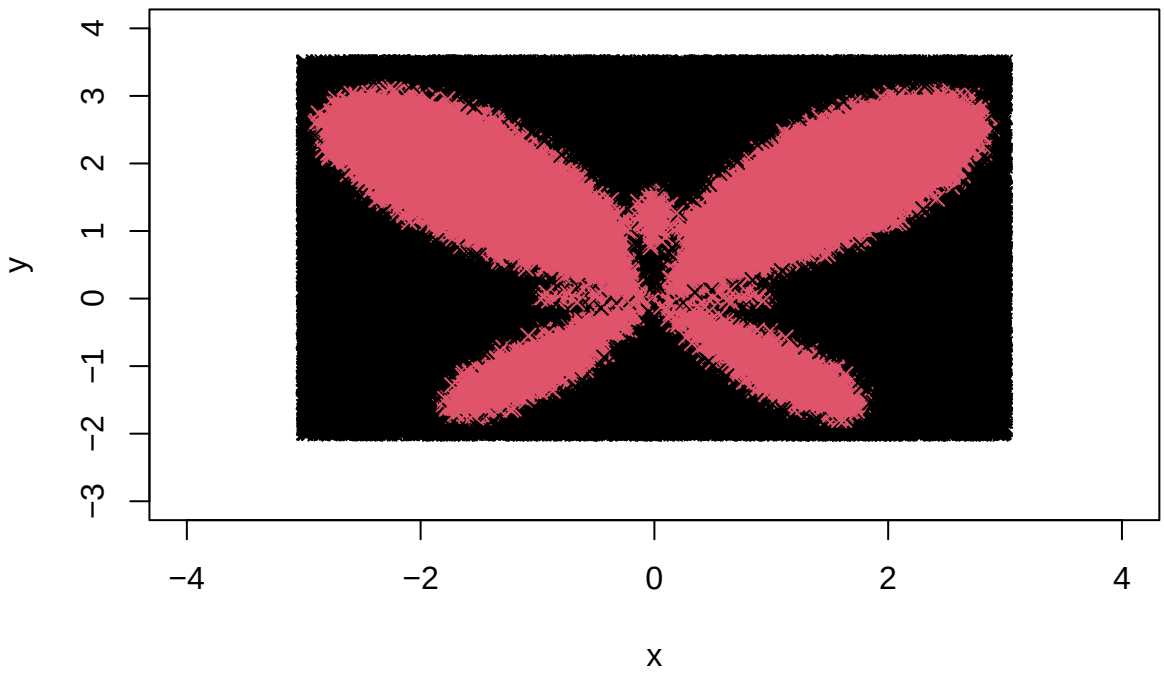

```
plot_viz(1000)
```




```
plot_viz(10000)
```




```
plot_viz(100000)
```




```
library(knitr)
samples <- c(100, 1000, 10000, 100000)
results <- table_1(samples)
options(scipen = 999)
kable(results, caption = "Sample of Areas")
```

Table 1: Sample of Areas

Samples	Estimated_Area
100	11.55000
1000	12.96900
10000	12.63240
100000	12.94557

Task 2.4

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

By using periodic random sampling, Monte Carlo simulation is a potent method for estimating complex systems that contain uncertainty or unpredictability. Each point in your exercise had an equal chance of being selected since random points were created inside a predetermined bounding box and sampled from a uniform distribution. After then, it was determined if these locations were inside or outside the curve of interest. The area or form of the curve was approximated by comparing the number of points inside the

curve to the total number of points generated. The accuracy of the approximation increases with the number of points sampled.

Because of their adaptability, Monte Carlo techniques can be used in scenarios where analytical solutions are challenging or unattainable. The behaviour of the curve, the impact of randomness and uncertainty on the results, and the effect of sample size on approximation accuracy are the main takeaways from the simulations. The results converge to a more accurate approximation as more simulations with bigger sample sizes are conducted. This procedure demonstrates how uncertainty can be modelled by Monte Carlo simulations, which makes them indispensable for challenging issues in a variety of domains.