Statistical Simulation and Computerintensive Methods

Exercise 3: Monte Carlo Simulation

Markus Kiesel | 1228952

11.10.2021

```
# set seed
set.seed(1228952)
```

Monte Carlo integration

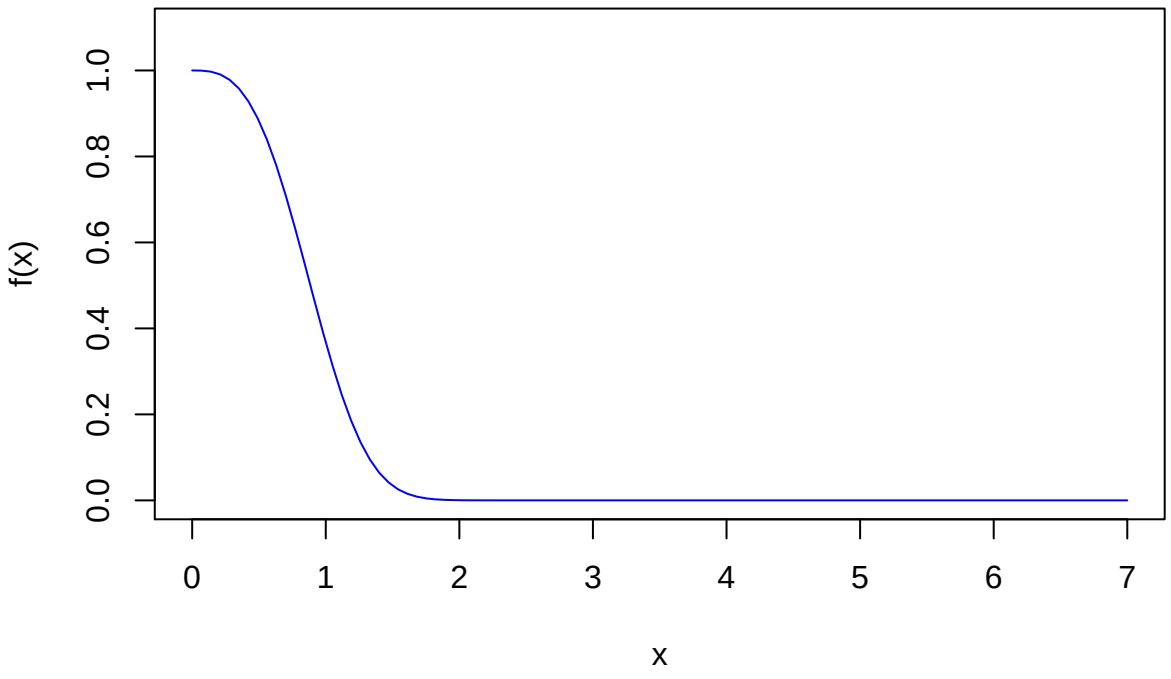
We consider the Integral $\int_1^b e^{-x^3}$.

This function can be implemented in R as follows.

```
# implementation of our function to integrate
f <- function(x) exp(-x^3)
```

Now we can have a look at the function we want to integrate.

```
# draw function f
curve(f, 0, 7, ylim=c(0, 1.1), col = 'blue')
```



Definite integral

First we look at how to calculate the integral with Monte Carlo integration for a definite integral with b = 6.

To approximate then

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

We use

$E(h(X)) = \int_1^6 \mathrm{e}^{-x^3} \cdot \frac{1}{6-1} dx$

Where $h(x) = \mathrm{e}^{-x^3}$ and $f(x) = \frac{1}{6-1}$.

to get

$\int_1^6 \mathrm{e}^{-x^3} dx = (6-1) \cdot E[h(X)]$

We can implement this in R as follows.

```
# number of simulated values
n <- 100000
# create uniform random samples in range min - max
us <- runif(n, min = 1, max = 6)
# calculate the Monte Carlo Integral
mci <- mean(f(us) * (6-1))
# calculate the Numerical Integral
ni <- integrate(f, lower = 1, upper = 6)

# Compare values:
results <- data.frame(mci, ni$value, all.equal(mci, ni$value))
colnames(results) <- c("Monte Carlo Integral", "Numerical Integral", "")
```

Table 1: Monte Carlo Integral vs Numerical Integral

Monte Carlo Integral	Numerical Integral	
0.0847331	0.0854683	Mean relative difference: 0.008676923

Infinite bounds

Now we try to use Monte Carlo integration to compute the integral for $b = \infty$.

So we want to approximate the value of the integral

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

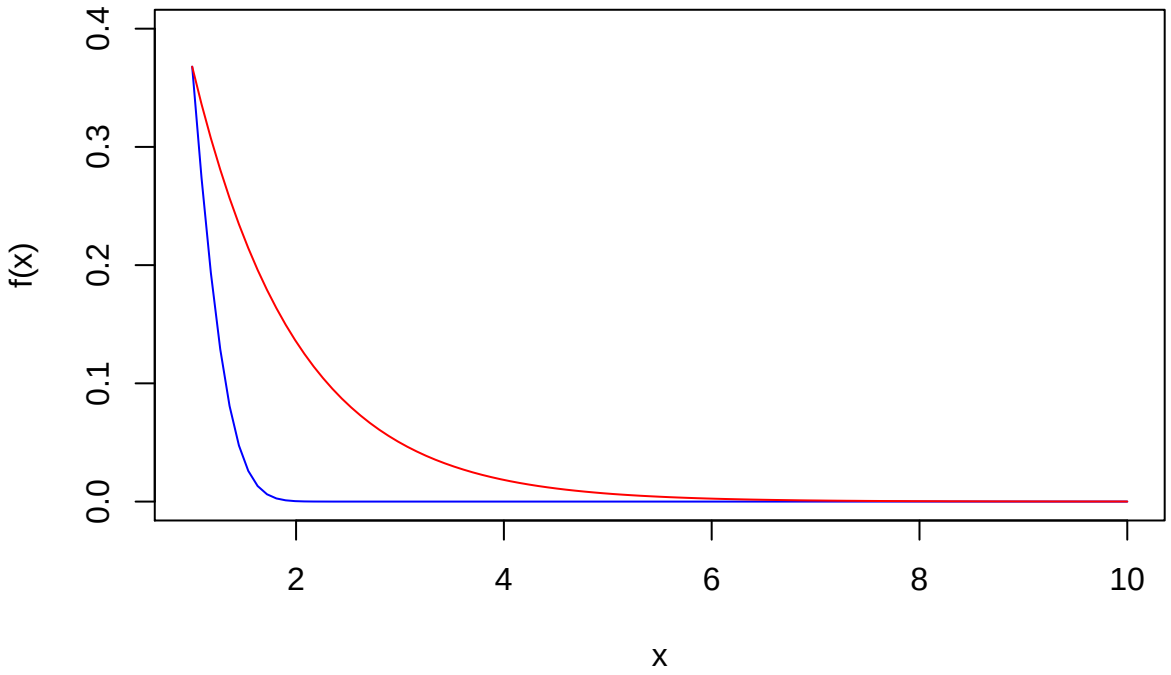
For this we need a density with the same support. Therefore, we rewrite

$\int_1^\infty \mathrm{e}^{-x^3} dx = \int_0^\infty \mathrm{e}^{-(x+1)^3} dx$

```
# rewrite function
f_r <- function(x) exp(-(x+1)^3)
```

Because the upper bound is infinite we can not use uniform distributed values. So another distribution is needed. A good candidate is the exponential distribution as we can see if we draw both functions.

```
# function to integrate in blue
curve(f, 1, 10, ylim=c(0, 0.4), col = 'blue')
# exponentioal distribution in red
curve(dexp, 1, 10, ylim=c(0, 0.4), col = 'red', add = TRUE)
```




```r
# number of simulated values
n <- 100000
# create random samples with exponential distribution
X <- rexp(n)
# calculate the Monte Carlo Integral (use rewritten function)
mci <- mean(f_r(X) / dexp(X))
# calculate the Numerical Integral (use normal function)
ni <- integrate(f, lower = 1, upper = Inf)


# Compare values:
results <- data.frame(mci, ni$value, all.equal(mci, ni$value))
colnames(results) <- c("Monte Carlo Integral", "Numerical Integral", "")
```

Table 2: Monte Carlo Integral vs Numerical Integral

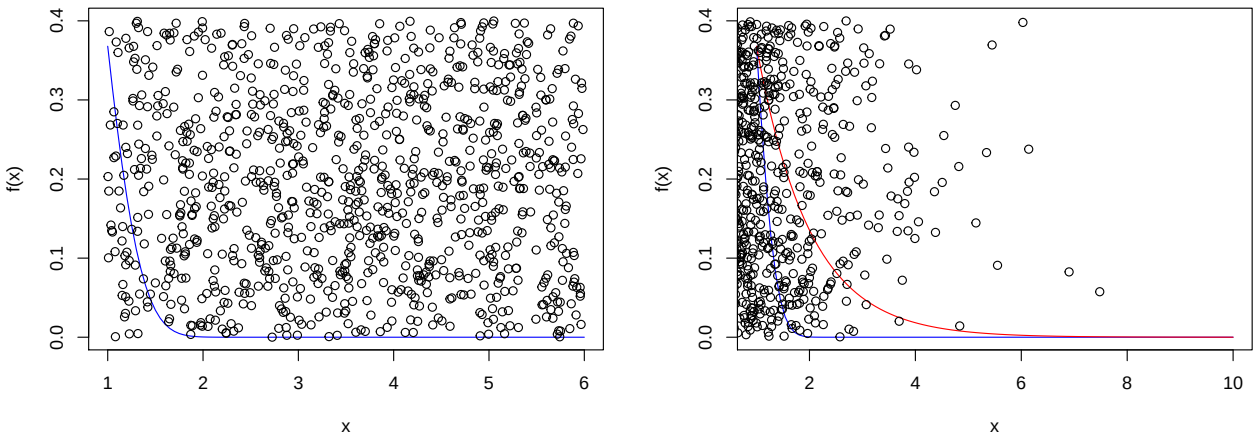
Monte Carlo Integral	Numerical Integral	
0.0853246	0.0854683	Mean relative difference: 0.001684739

The Monte Carlo Integration worked for the infinite bound better than for the definite bounds. This is the the case because for the second calculation we used an exponential distribution where most of our values where created in the area where the integral we wanted to calculate is located. Instead of sampling points randomly we use importance sampling where points are sampled more frequently where the area of the integral is large. Therefore we reduce the variance of the estimate.

To make this more clear we can visualize both integral calculations we made. We draw points on the y axis whit a uniform distribution to spread points over the whole graph just for the sake of the visualization.

```r
par(mfrow = c(1,2))
# draw left graph
curve(f, 1, 6, ylim=c(0, 0.4), col = 'blue')
points(runif(1000, 1, 6), runif(1000, 0, 0.4))
```

```r
# draw right graph
curve(f, 1, 10, ylim=c(0, 0.4), col = 'blue')
curve(dexp, 1, 10, ylim=c(0, 0.4), col = 'red', add = TRUE)
points(rexp(1000), runif(1000, 0, 0.4))
```

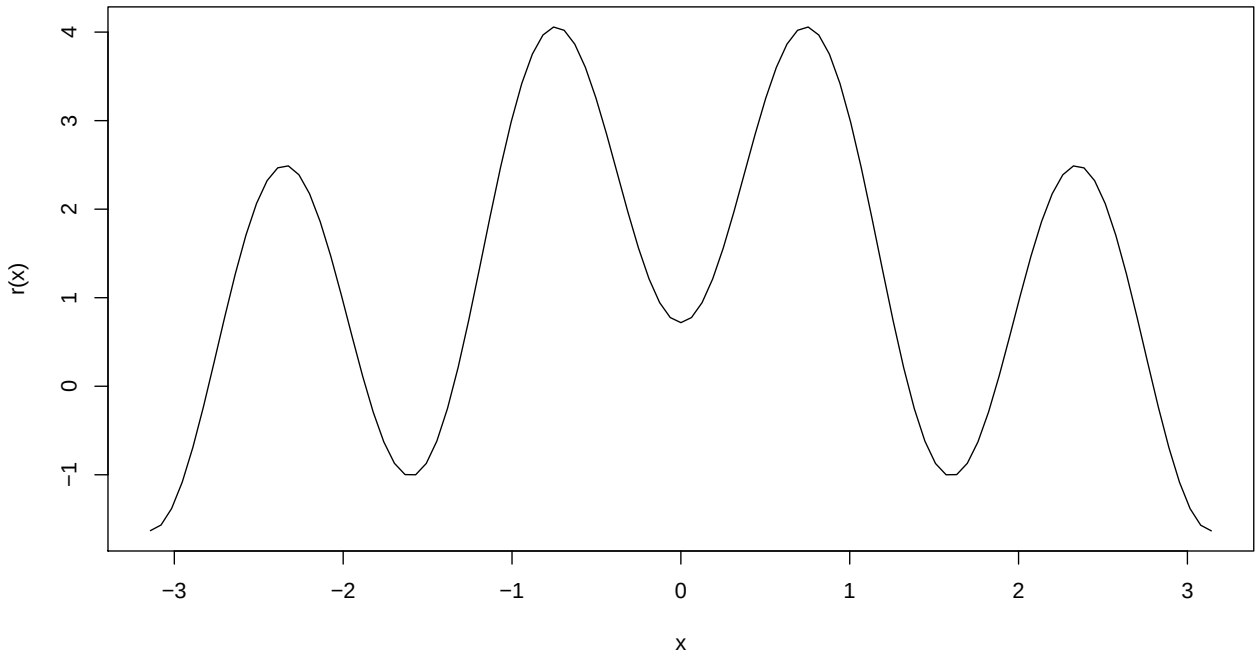


Monte Carlo Simulation of areas

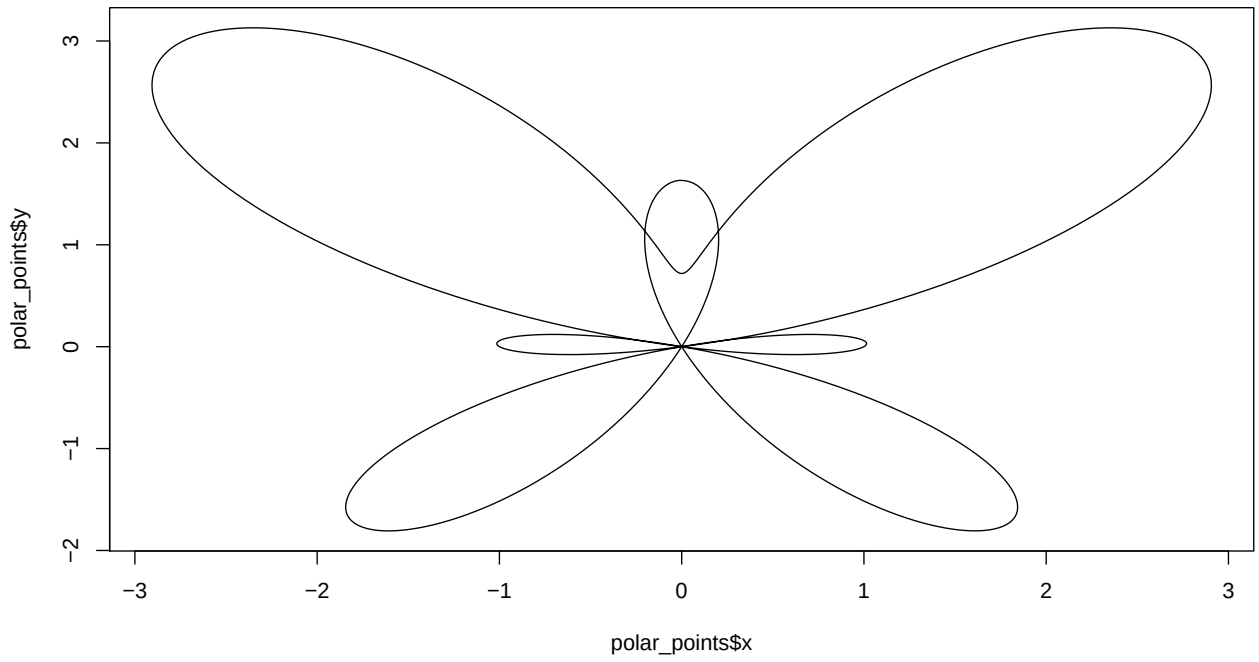
Now we want to use Monte Carlo simulation for obtaining the area enclosed by the graph of the function r(t). We first draw the function with cartesian coordinates and than transform to polar coordinates and draw the figure.

Visualizing the function and the area

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

# draw the function courve
curve(r, -pi, pi)
```




```r
# transform to polar coordinates
polar_fun <- function(t) {
  x <- r(t) * sin(t)
  y <- r(t) * cos(t)
  return(list(x = x, y = y))
}

# plot the figure
t <- seq(-pi,pi,0.01)
polar_points <- polar_fun(t)
plot(polar_points$x, polar_points$y, type="l")
```

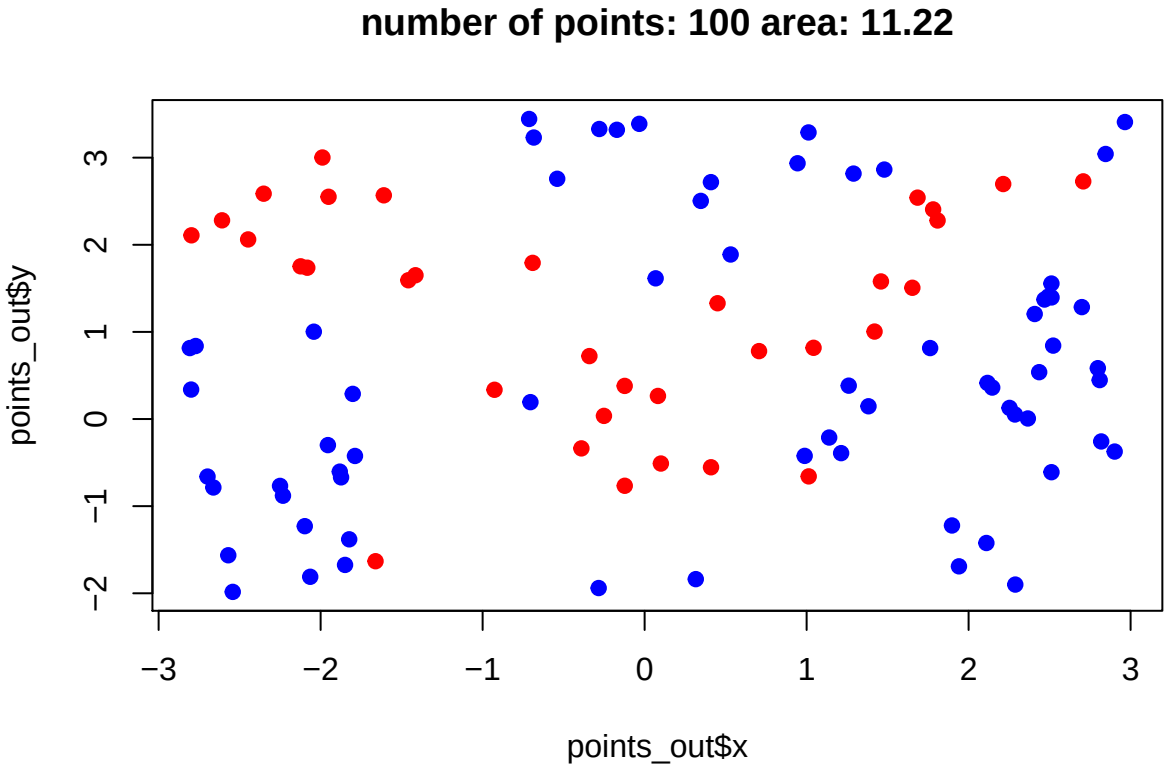


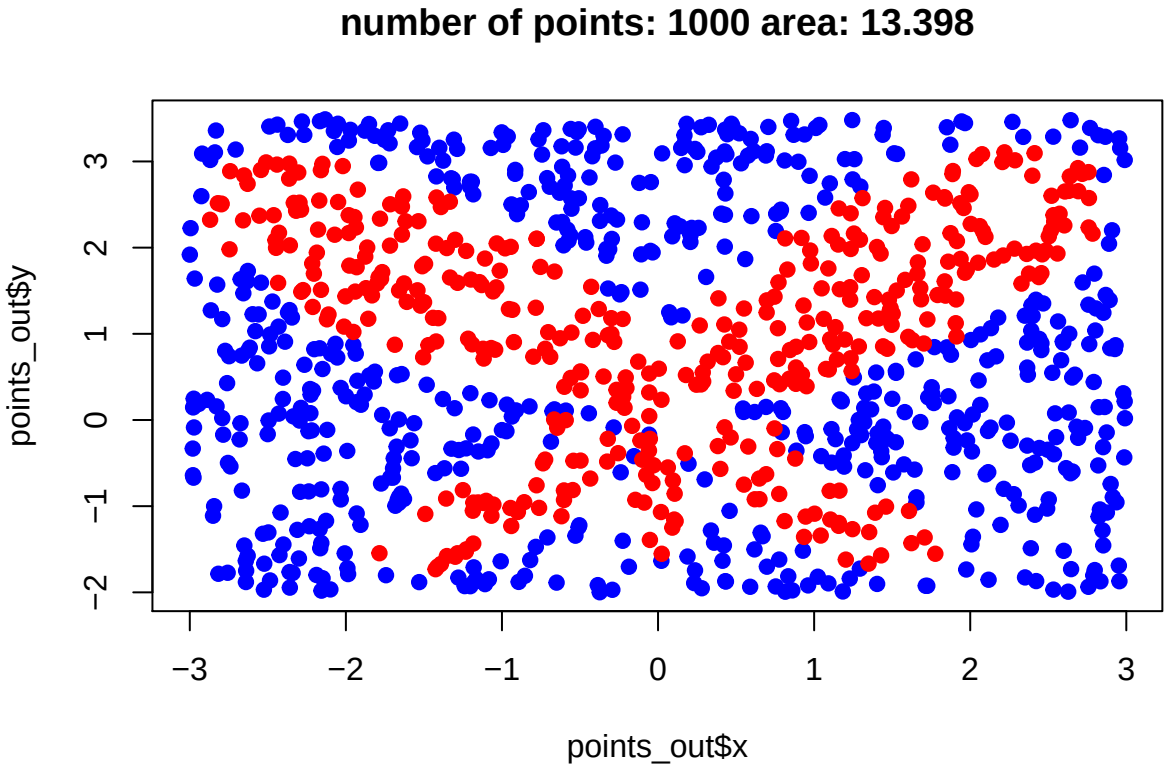
Area calculation for different simulation sizes

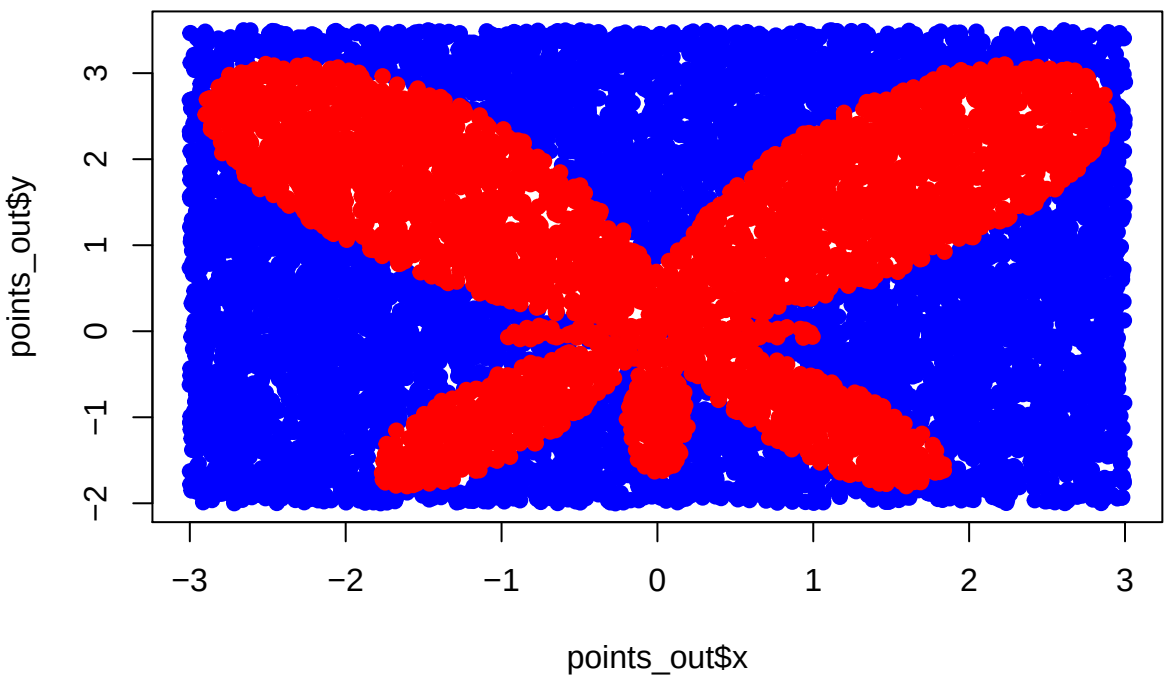
The area within this figure should be approximated with Monte Carlo simulation. For this we need a function which indicates whether a point lies within the area or not.

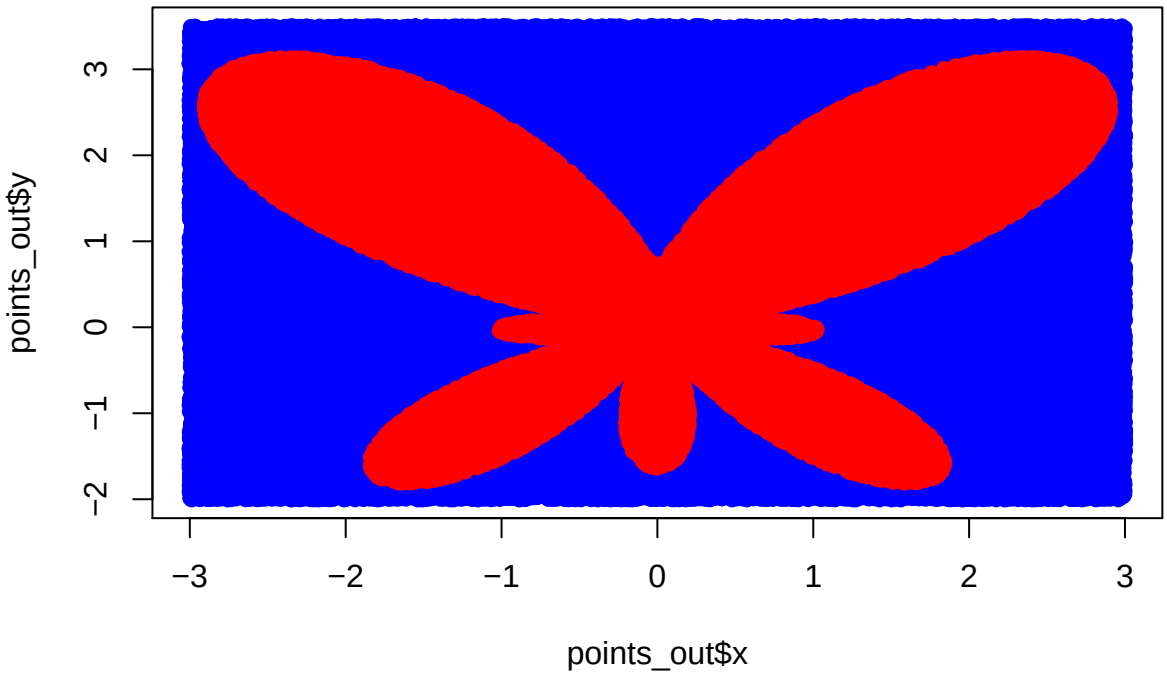
```
# indicate if point is in or outside area
points_indicate <- function(x, y) {
  # calculate polar coordinates for points
  points <- cart2pol(x, y)
  points <- points %>%
    # transform theta
    mutate(theta = theta - pi/2) %>%
    # check if radius of random point smaller radius of function
    mutate(in_area = r < abs(r(theta)))
  return(points)
}

# calculate area for a given simulation size
area_calc <- function(n) {
  # define the rectangle we use to calculate the area
  rect <- list(x = c(-3, 3), y = c(-2, 3.5))
  # generate uniform random values within rectangle
  x <- runif(n, rect$x[1], rect$x[2])
  y <- runif(n, rect$y[1], rect$y[2])
  # create points with indication if they are in the area
  points <- points_indicate(x, y)
  # get number of points within area
  n_points_in_area <- points %>%
    summarize(sum(in_area))
  # calculate percentage within area
  perc <- as.numeric(n_points_in_area) / n
  # calculate area by multiplying with rectangel
  area <- (rect$x[2] - rect$x[1]) * (rect$y[2] - rect$y[1]) * perc

  return(list(perc=perc,
              area=area,
              points=points))
}

# size of simpulation
n <- c(100, 1000, 10000, 100000)

# calculations per simulation size
perc <- numeric(length(n))
area <- numeric(length(n))
for (i in 1:length(n)) {
  result <- area_calc(n[i])

  # get points from simulation
  points <- result$points
  # filter for points inside area
  points_in <- result$points %>%
    filter(in_area == TRUE)
  # filter for points outside area
  points_out <- result$points %>%
```

```
    filter(in_area == FALSE)

  # plot points
  plot(points_out$x, points_out$y, col = "blue", pch = 19,
       main = paste("number of points:", n[i], "area:", result$area))
  points(points_in$x, points_in$y, col = "red", pch = 19)

  # get values for percentage and area
  perc[i] <- result$perc
  area[i] <- result$area
}
```




number of points: 1000 area: 13.398
points_out$y
points_out$x
−3
−2
−1
0
1
2
3


number of points: 10000 area: 13.3749
points_out$y
points_out$x
−3
−2
−1
0
1
2
3




```
# create table and print results
table <- data.frame(n, perc, area)
```

Table 3: Todo Title

n	perc	area
1e+02	0.34000	11.22000
1e+03	0.40600	13.39800
1e+04	0.40530	13.37490
1e+05	0.40761	13.45113

Although there is a problem with the shape of the area (probaply the transformation of theta values is wrong) we can show the area approximation gets better with the size of the simulation if we have a look at the table. The plots tell us that more and more points lead to a better estimate. We see clearly that the area can not be estimated with only 100 points whereas 10000 points fill the whole rectangle we use to calculate tha area of the shape.

Functionality of Monte Carlo simulation

With Monte Carlo simulation we are able to use the law of big numbers to estimate the result of various problems. We need a problem for which we can simulate inputs with a probability distribution over the domain. Then we calculate the output per simulated value and aggregate the results. With this method we can estimate the output for problems that are otherwise hard to calculate (like the area of an arbitrary shape). The method is general and can be used for a wide range of problems not only calculation the area under / within functions.