Assignment 3

Monte Carlo Methods

Roman Grebnev

2022-11-06

Contents

Task 1 . . . 1
Task 1.1 . . . 1
Task 1.2 . . . 2
Task 1.3 . . . 2
Task 2 . . . 2
Task 2.1 . . . 2
Task 2.2 . . . 3
Task 2.3 . . . 4
Task 2.4 . . . 7

Task 1

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

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.

In order to compute the integral we need to generate a random sample drawn from the univariate distribution.

Computation of the integral of the target function based on the Monte Carlo integration.

```
n <- 10^5
a <- 1
b <- 6
f<- function(x){exp(-x^3)}

set.seed(12202120)
us <- runif(n, min = a, max = b)
mean(f(us)) * (b-a)
```

```
## [1] 0.08462263
```

Monte Carlo Integral: 0.08462263

Computation of the integral of the target function.

```
integrate(f = f, lower = a, upper = b)
```

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

Numerical Integral: 0.08546833 with absolute error < 3.2e-07

Compare obtained results of integral computation to the function integrate. Obtained results are very close to each other, but they don't coincide after 5th decimal point.

Task 1.2

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

We can rewrite the integral $\int_1^\infty e^{-x^3}$ as $\int_0^\infty e^{-(x+1)^3}$. Thus the function for computation of the

```
n_inf <- 10^5
set.seed(12202120)
exp_s <- rexp(n_inf)
f_inf<- function(x){exp(-(x+1)^3)}

mean(f_inf(exp_s)/dexp(exp_s))
```

```
## [1] 0.08481072
```

Monte Carlo Integral: 0.08518384

```
integrate(function(x){exp(-x^3)}, 1, Inf)
```

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

Numerical Integral: 0.08546833

Task 1.3

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

The difference between integral from task 1.1 and task 1.2 is that we use for Monte Carlo integration the distribution that is defined from 1 to infinity in task 1.2 compared to the distribution defined from 1 to 6 in task 1.1. Thus more points are generated for integral computation based on averaging when using samples drawn from the exponential distribution. It means that the tail of distribution is much better represented in such a case compared to sampling from uniform distribution. It yields in turn better precision of integral computation.

Task 2

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

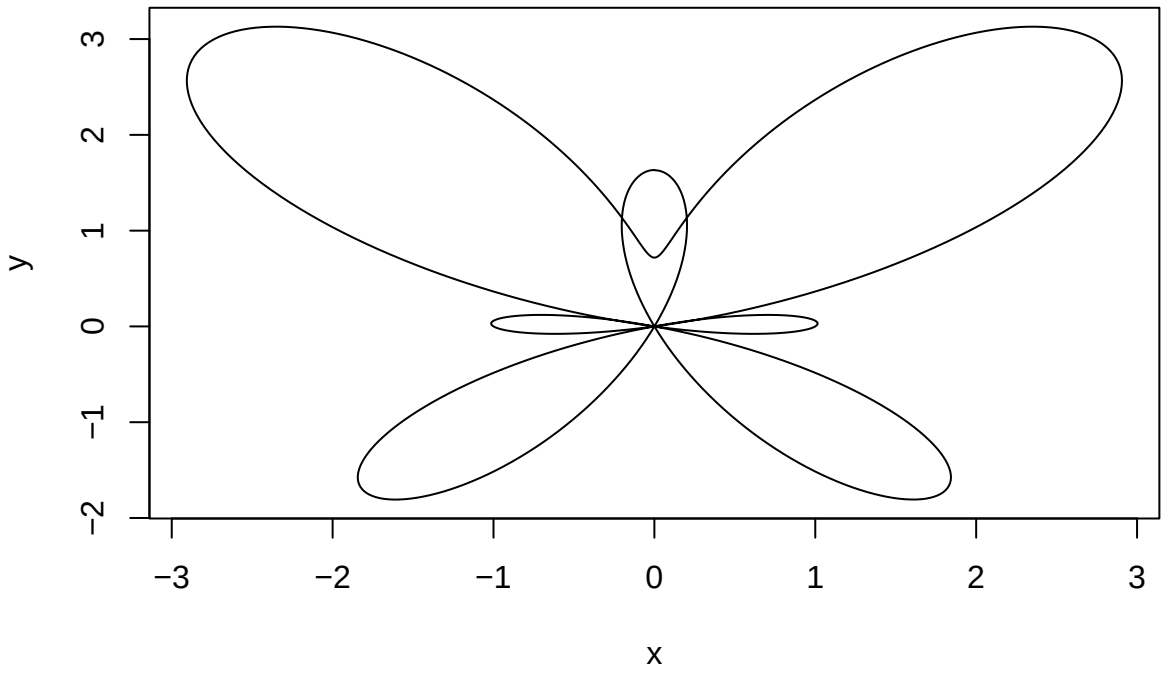
Let's define the function of interest and the polar coordinates x and y

```
r <- function(t){
  exp(cos(t))-2*cos(4*t)-sin(t/12)^5}
t <- seq(by=0.01,from = -pi, to = pi)
x <- r(t)*sin(t)
y <- r(t)*cos(t)
```

Task 2.1

Visualize the function and the area.

```
plot(x=x, y=y, type='l')
```



Task 2.2

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

In order to identify, whether particular point lies within the function defined above we can implement the algorithm that checks this condition for all of the points generated.

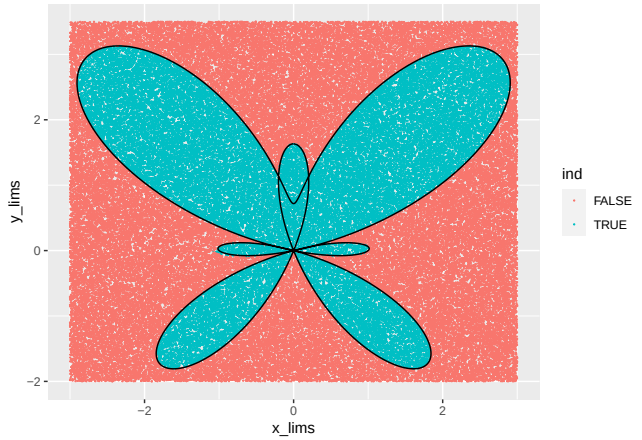
```
n_lims = 10^5
```

```
x_lims <- runif(n_lims, min=-3, max=3)
y_lims <- runif(n_lims, min=-2, max=3.5)

is_inside<- function(x, y, n) {

  if (y > 0){
    alpha <- 0
  } else {
    alpha <- pi
  }
  rad <- sqrt(x^2 + y^2)
  beta <- atan(x/y) + alpha
  if (r(beta) > 0 & (rad < abs(r(beta)))) {
    return(TRUE)
  } else if (r(beta+pi) < 0 & (rad < abs(r(beta + pi)))) {
    return(TRUE)
  } else {
    return(FALSE)
  }
}


ind <- logical(n_lims)
for (i in 1:n_lims){
```

```
  ind[i] <- is_inside(x_lims[i], y_lims[i], n_lims)
}
```

To illustrate the result of the simulation we can illustrate the outcome visually, where red color indicates that the generated point is outside of the defined function and red - that it is inside.

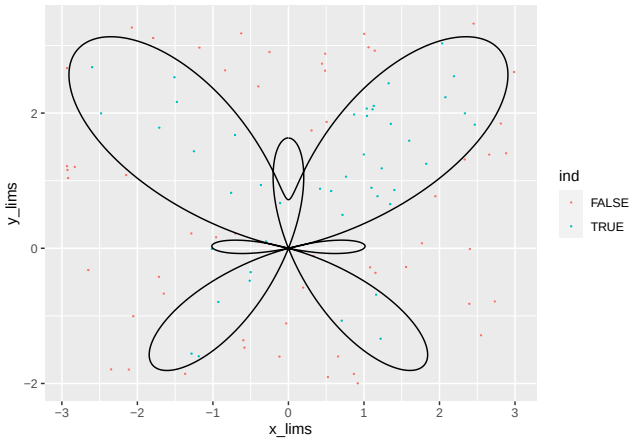
```
library(ggplot2)
df <- data.frame(x_lims, y_lims, ind)

ggplot()+
  geom_point(data= df, aes(x=x_lims, y=y_lims, color = ind ), size = 0.1)+
  geom_path(aes(x=x, y=y))
```



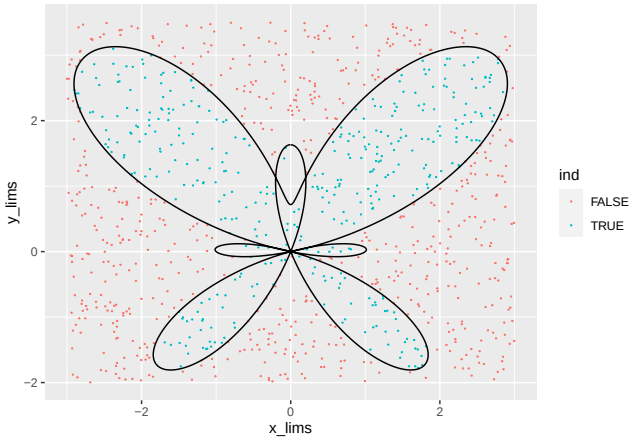
Task 2.3

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

```
n_lims = 10^2
x_lims <- runif(n_lims, min=-3, max=3)
y_lims <- runif(n_lims, min=-2, max=3.5)

ind <- logical(n_lims)

for (i in 1:n_lims){
  ind[i] <- is_inside(x_lims[i], y_lims[i])
}

library(ggplot2)
df <- data.frame(x_lims, y_lims, ind)

ggplot()+
  geom_point(data= df, aes(x=x_lims, y=y_lims, color = ind ), size = 0.1)+
  geom_path(aes(x=x, y=y))
```

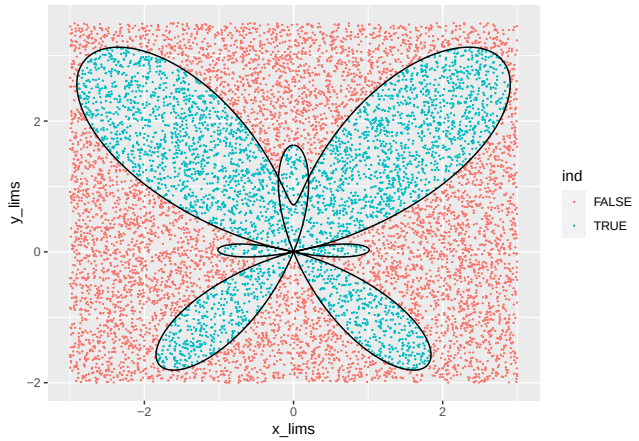



```
n_lims = 10^3
x_lims <- runif(n_lims, min=-3, max=3)
y_lims <- runif(n_lims, min=-2, max=3.5)

ind <- logical(n_lims)

for (i in 1:n_lims){
  ind[i] <- is_inside(x_lims[i], y_lims[i])
}

library(ggplot2)
df <- data.frame(x_lims, y_lims, ind)

ggplot()+
  geom_point(data= df, aes(x=x_lims, y=y_lims, color = ind ), size = 0.1)+
  geom_path(aes(x=x, y=y))
```

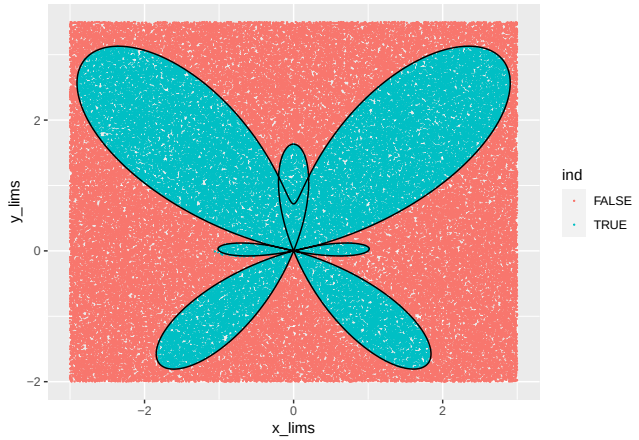



```
n_lims = 10^4
x_lims <- runif(n_lims, min=-3, max=3)
y_lims <- runif(n_lims, min=-2, max=3.5)

ind <- logical(n_lims)

for (i in 1:n_lims){
  ind[i] <- is_inside(x_lims[i], y_lims[i])
}
```

```
library(ggplot2)
df <- data.frame(x_lims, y_lims, ind)

ggplot()+
  geom_point(data= df, aes(x=x_lims, y=y_lims, color = ind ), size = 0.1)+
  geom_path(aes(x=x, y=y))
```




```
n_lims = 10^5
x_lims <- runif(n_lims, min=-3, max=3)
y_lims <- runif(n_lims, min=-2, max=3.5)

ind <- logical(n_lims)

for (i in 1:n_lims){
  ind[i] <- is_inside(x_lims[i], y_lims[i])
}

library(ggplot2)
df <- data.frame(x_lims, y_lims, ind)

ggplot()+
  geom_point(data= df, aes(x=x_lims, y=y_lims, color = ind ), size = 0.1)+
  geom_path(aes(x=x, y=y))
```



Task 2.4

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

Monte Carlo Method is used to obtain the possible range of output variables by randomly sampling the input variables from probability distributions and using particular deterministic function, which describes the relations between inputs and outputs. Method can help to evaluate the statistical properties of the output variable, because it provides the possibility to make more precise calculations by running a large number of simulations. Monte Carlo Method is applicable in various domains, which deal with random variables and uncertainty. The principle is the following: Monte Carlo Method uses probability distributions of the input variables and the model that produces the output variable, by using input variables. Thus we can observe all possible variations of output variable, which comes from using input values drawn from their probability distributions. Beside that we can identify, how sensitive the output variable is to the changes in input variables. The most challenging part of Monte Carlo Method is identifying the relationship between the output variables and the input variables.