Exercise 3 - Monte Carlo Simulation of areas

Florian Stadler

2024-10-08

Contents

1 Task 1 Monte Carlo Integration

In this task, we consider the integral

$$\int_1^b e^{-x^3}.$$

We will now compute following subtasks.

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.

```
f <- function(x){
  return (exp(-x**3))
}
monte.carlo.unif.based <- function(b, n){
  uniform.x <- runif(n,1,b)
  f.values <- f(uniform.x)
  average.f <- mean(f.values)
  int.est <- (b-1) * average.f
  return(int.est)
}
```

With the function monte.carlo.unif.based we can compute the monte carlo estimation for the given integral. Lets look at the values for
n=100, b=6: Integral ≈ 0.0975184
n=10000, b=6: Integral ≈ 0.085017
n=1000000, b=6: Integral ≈ 0.0849453
What the method does is to create a uniform sample in the integration area -> applies the function in the integral on all those data points -> takes the mean of the function values -> Multiply by the integral area. We can use the base r "integrate" function, to get the value of the integral. for b=6 we get the Integral = 0.0854683 with an error of less than $3.2 \cdot 10^{-7}$. We observe that montecarlo comes pretty close when choosing high enough n. However, this also increases the computation time by a lot and for low n the difference is higher.

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.

We cannot use the uniform distribution does not have the same support due to b = ∞. Therefore, we will use the exponential distribution and shift the integration levels. We get the equivalent integral

$$\int_0^\infty e^{-(x+1)^3}.$$

we now have the same support as the exponential distribution and can compute the montecarlo integration with an exponential distrubuted sample.

```
monte.carlo.exp.based <- function(n){
  x <- rexp(n)
  value <- mean(f(x+1)/dexp(x))
  return (value)
}
montecarlo.value<- monte.carlo.exp.based(1000000)
cat(paste("Monte Carlo Integral:",montecarlo.value))
```

```
## Monte Carlo Integral: 0.0855437214614677
```

```
integral.value <-integrate(f,1,Inf)[[1]]
cat(paste("True Integral:",integral.value))
```

```
## True Integral: 0.0854683294276731
```

```
cat(paste("The absolute difference of these methods is ",integral.value-montecarlo.value))
```

```
## The absolute difference of these methods is  -7.53920337946551e-05
```

1.3 Do you have an explanation why Monte Carlo integration agrees in 1.2 with integrate but not so much in 1.1?

By using the exponential distribution in 1.2, we naturally get more points that are closer to the range of arguments, where the integrand is bigger. The function in the integrand is larger, the closer the points are to 0. Therefore, by choosing the exponential distribution, we did importance sampling. Thus we have reduced

the variance of the estimate significantly since we have more data points at the "important" values. In 1.1, a lot more data points are at a value range, where the function is close to zero. Therefore, we need a much bigger sample size, to get more accurate results, when using uniform distribution for monte carlo estimation for this integral.

2 Task 2 Multivariable Monte Carlo Approximation (2 Dimensional)

Monte Carlo simulation shall be utilized for obtaining the area enclosed by the graph of the function

$$r(t) = e^{\cos(t)} - 2 \cdot \cos(4t) - \sin\left(\frac{t}{12}\right)^5, \text{for } t \in [-\pi, \pi],$$

when using polar x- and y-coordinates

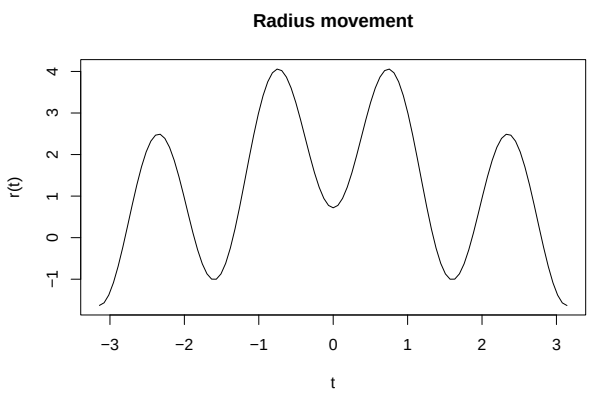
$$x = r(t)\cos(t), y = r(t)\sin(t).$$

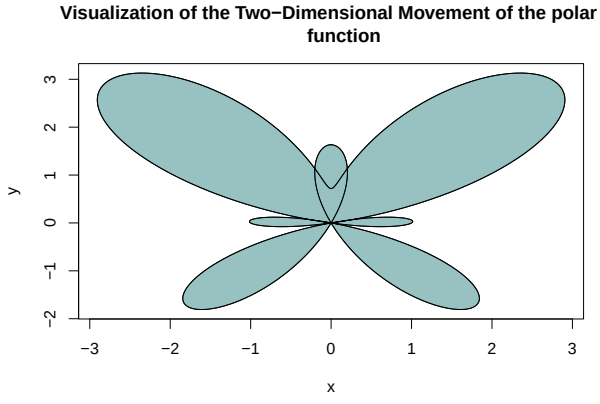
Visualization of the function We will visualize the radius movement of r and the area of the function.

```
r <- function(t) {
  return((exp(cos(t)) - 2 * cos(4*t) - sin(t/12)^5))
}
plot(r, -pi, pi, main="Radius movement", xlab="t", ylab="r(t)")

t <- seq(-pi, pi, length.out=2500)
x_coord <- r(t) * sin(t)
y_coord <- r(t) * cos(t)
plot(x_coord, y_coord, type="l", main="Visualization of the Two-Dimensional Movement of the polar
     function", xlab="x", ylab="y")

polygon(x_coord, y_coord, col = rgb(0.20, 0.53, .52, alpha = 0.5))
```





The area of the function looks like a bit like a butterfly :)