Ex3

Yelyzaveta Klysa

2023-10-13

Contents


```
set.seed(12208877)
```

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.

```
b <- 6
set.seed(12208877)
exp_f <- function(x) exp(-x^3)
x1 <- runif(100000, min=1, max=b)
mc <- mean(exp_f(x1)) * (b - 1)
cat("Monte Carlo Integral: ", mc)
```

```
## Monte Carlo Integral:  0.08650901
```

```
ri <- integrate(exp_f,lower = 1,upper = b)[[1]]
cat("Real Integral: ", ri)
```

```
## Real Integral:  0.08546833
```

```
all.equal(mc, ri)
```

```
## [1] "Mean relative difference: 0.01202977"
```

The difference between the 2 calculated integrals is quite fine, but still is bigger than 0.01.

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.

The formula can be changed the way that we will be able to use exponential distribution, specifically as $\int_0^\infty e^{-(x+1)^3}$.

```
set.seed(12208877)
x2 <- rexp(100000)

mcexp <- mean(exp_f(x2+1)/dexp(x2))
cat("Monte Carlo Integral: ", mcexp)
```

```
## Monte Carlo Integral:  0.0852655
```

```
riexp <- integrate(exp_f, 1, Inf)[[1]]
cat("Real Integral: ", riexp)
```

```
## Real Integral:  0.08546833
```

```
all.equal(mcexp, riexp)
```

```
## [1] "Mean relative difference: 0.002378801"
```

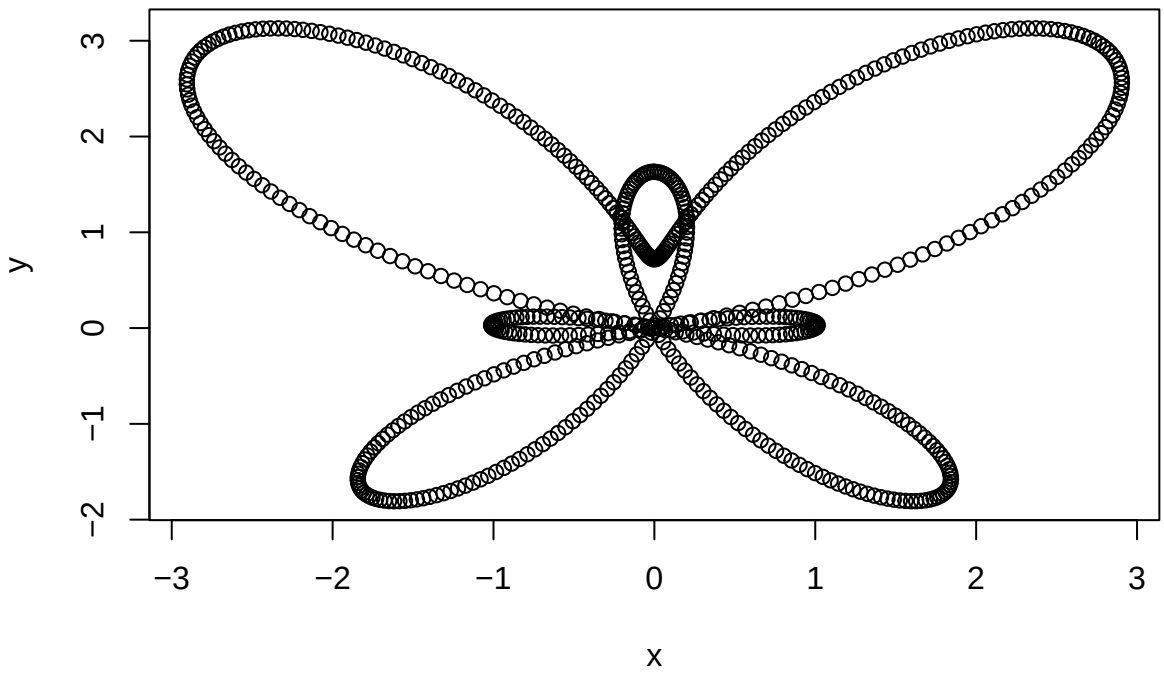
The difference between these two integrals is smaller than in Task 1.1.

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

The second distribution creates more points that fit better, since it is created from exponential distribution, which gives better precision of the integral computation comparing to the samples drawn from the uniform distribution.

2.1 Visualise the function and the area.

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



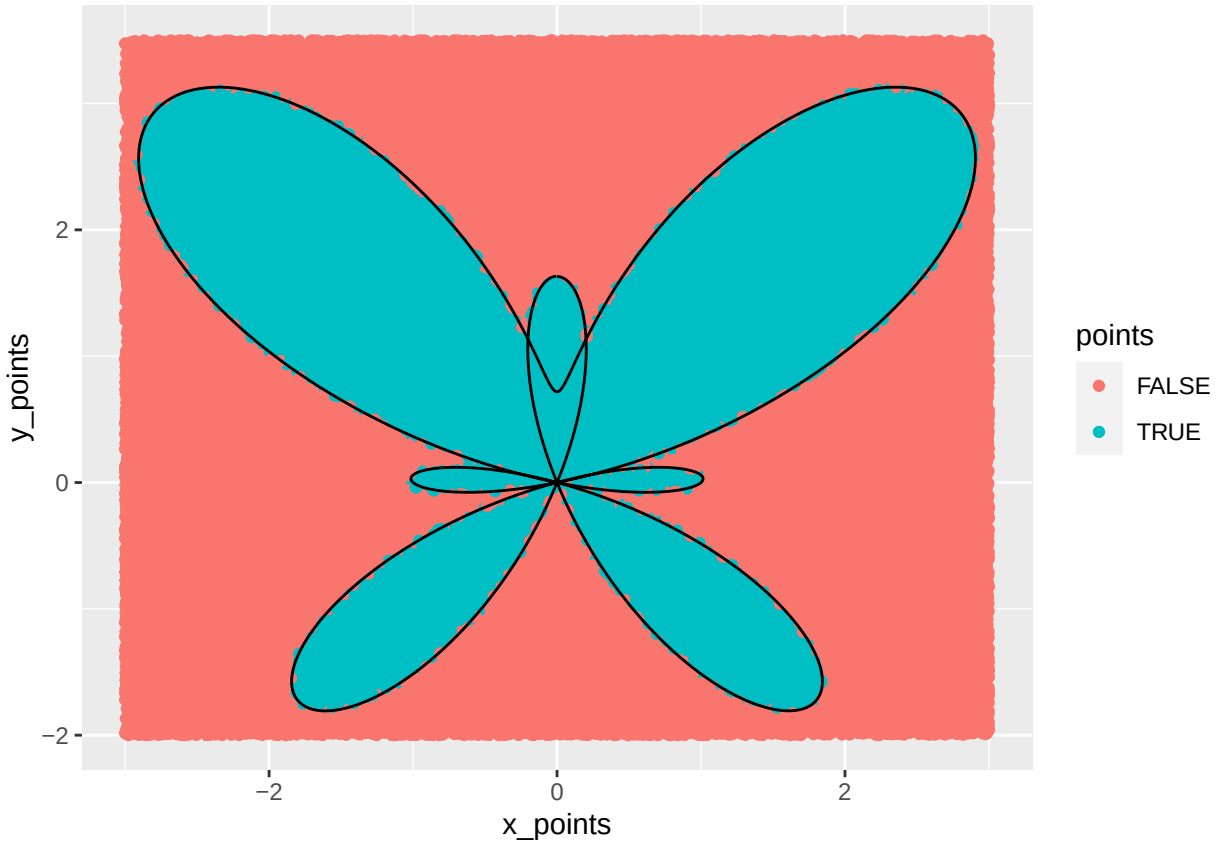
2.2 Generate uniform random coordinates within the rectangle [-3,3]×[-2,3.5] and an indicator whether this point lies within the area in question.

We can do this by converting x and y coordinates to polar ones and check whether they lay inside the are of the graph or not. We add the `pi` value to `teta` to ensure that the points under `y=0` get inside the polygon.

```
get_x_points <- function(n){
  return (runif(n, min = -3, max = 3))
}
get_y_points <- function(n){
  return (runif(n, min = -2, max = 3.5))
}

generate_in_distr <- function(n, x_points, y_points){
  in_polygon <- logical(n)
  for (i in 1:n){
    x <- x_points[i]
    y <- y_points[i]

    if (y > 0){alpha <- 0}
    else {alpha <- pi}

    teta <- atan(x/y) + alpha
```

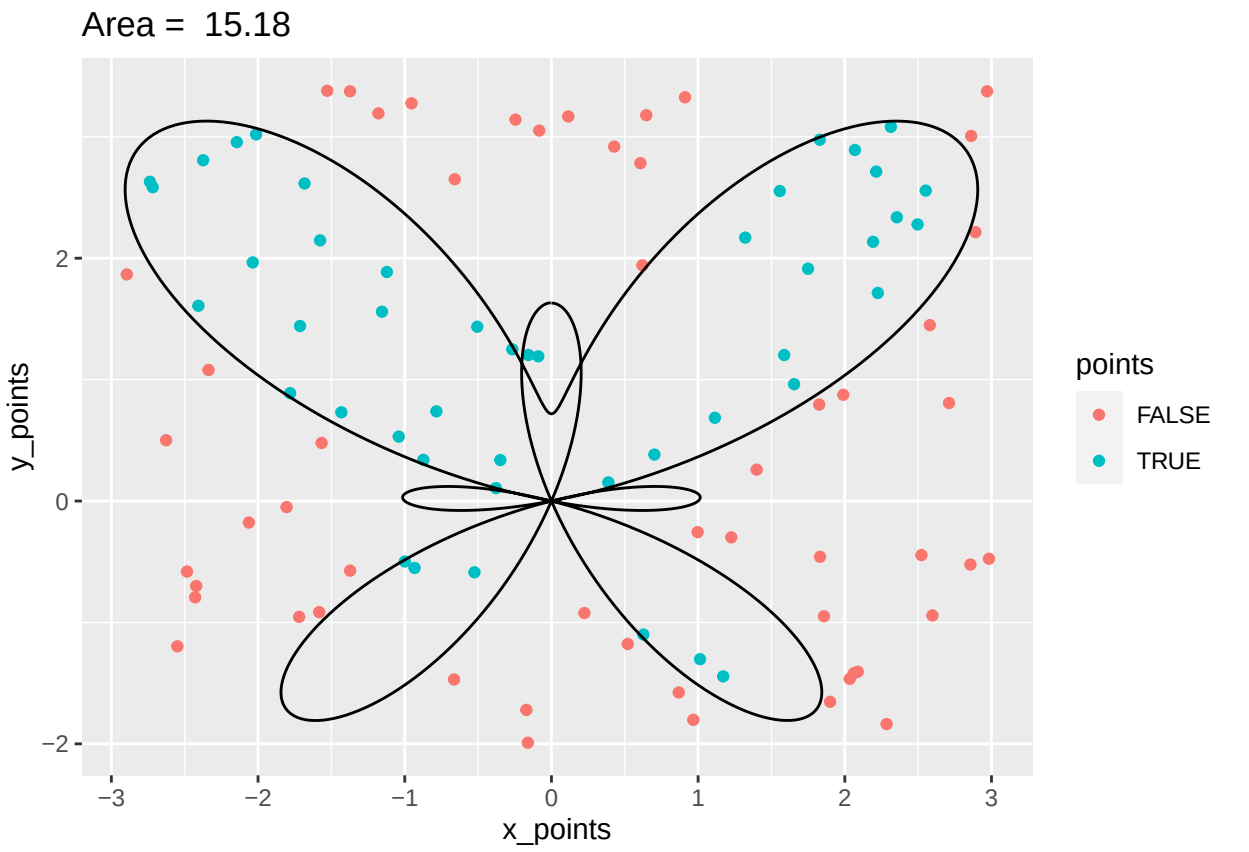
```
    teta_pi <- teta + pi

    rad <- sqrt(x^2 + y^2)

    r_teta <- r(teta)
    r_teta_pi <- r(teta_pi)

    if (r_teta > 0 & (rad < abs(r_teta))){ in_polygon[i] <- TRUE}
    else if (r_teta_pi < 0 & (rad < abs(r_teta_pi))){ in_polygon[i] <- TRUE}
    else { in_polygon[i] <- FALSE }
  }
  return (in_polygon)
}
```

```
set.seed(12208877)

n <- 100000
x_points <- get_x_points(n)
y_points <- get_y_points(n)
points <- generate_in_distr(n, x_points, y_points)

df <- data.frame(x_points, y_points, points)
fig <- ggplot() + geom_point(data = df, aes(x=x_points, y=y_points, color = points))  + geom_path(aes(x=
fig
```

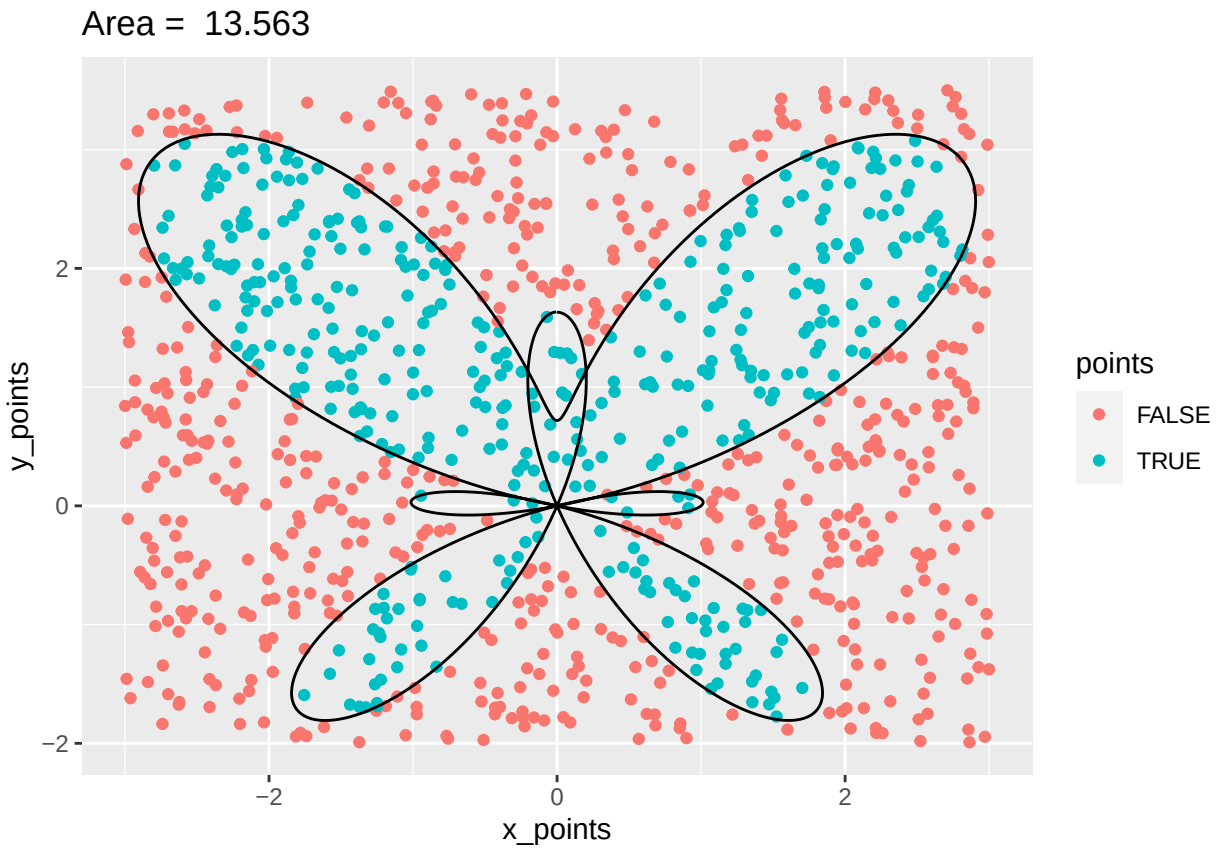


2.3 Simulate 100, 1000, 10000 and 100000 random coordiantes and calculate the percentage of the points within the enclosed area. Based on this information estimate the area of the figure. Summarise those values in a table and visualise them in plots of the function curve and enclosed area.

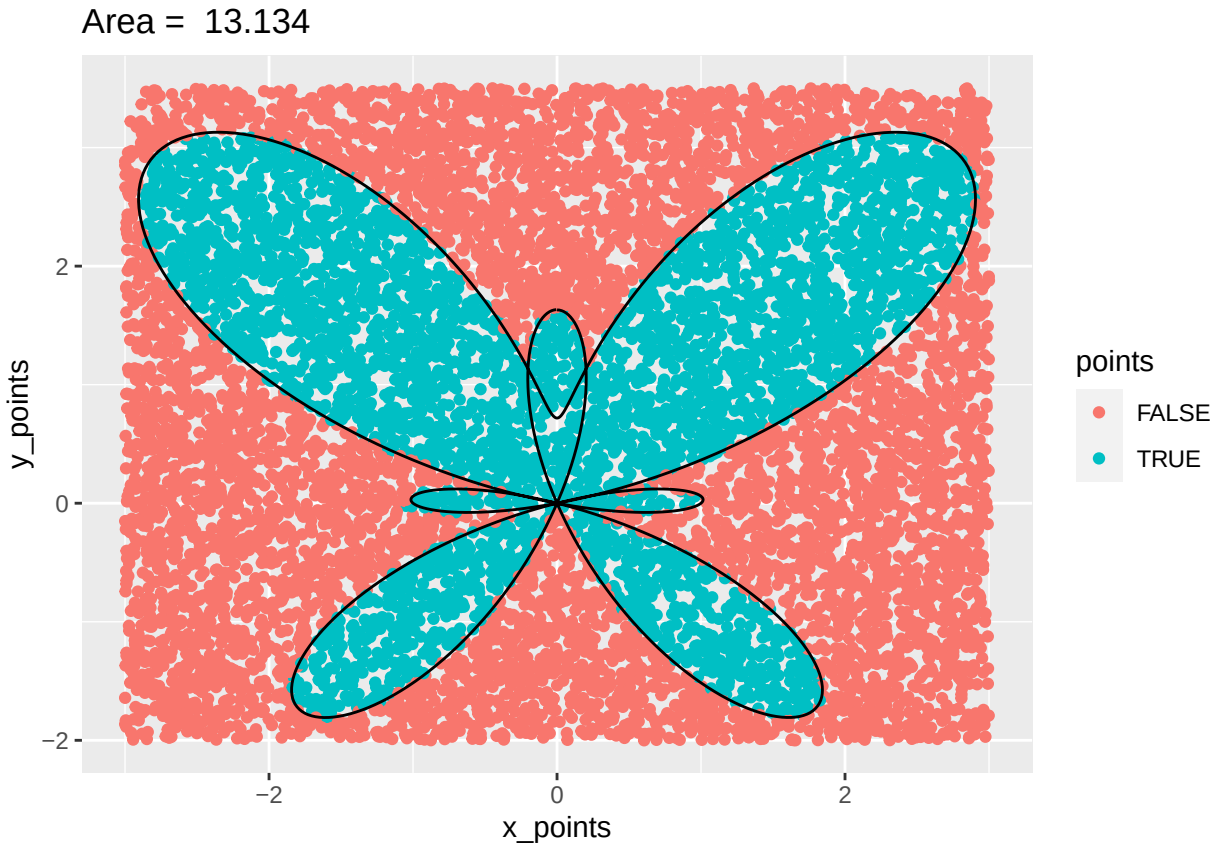
```
run_gen <- function(n){
  set.seed(12208877)
  x_points <- get_x_points(n)
  y_points <- get_y_points(n)
  points <- generate_in_distr(n, x_points, y_points)
  df <- data.frame(x_points, y_points, points)
  pc <- 6*5.5 * sum(points) / n
  print(ggplot() + geom_point(data = df, aes(x=x_points, y=y_points, color = points))  + geom_path(aes(
  return (pc)
}

areas <- c()
areas[1] <- run_gen(100)
```



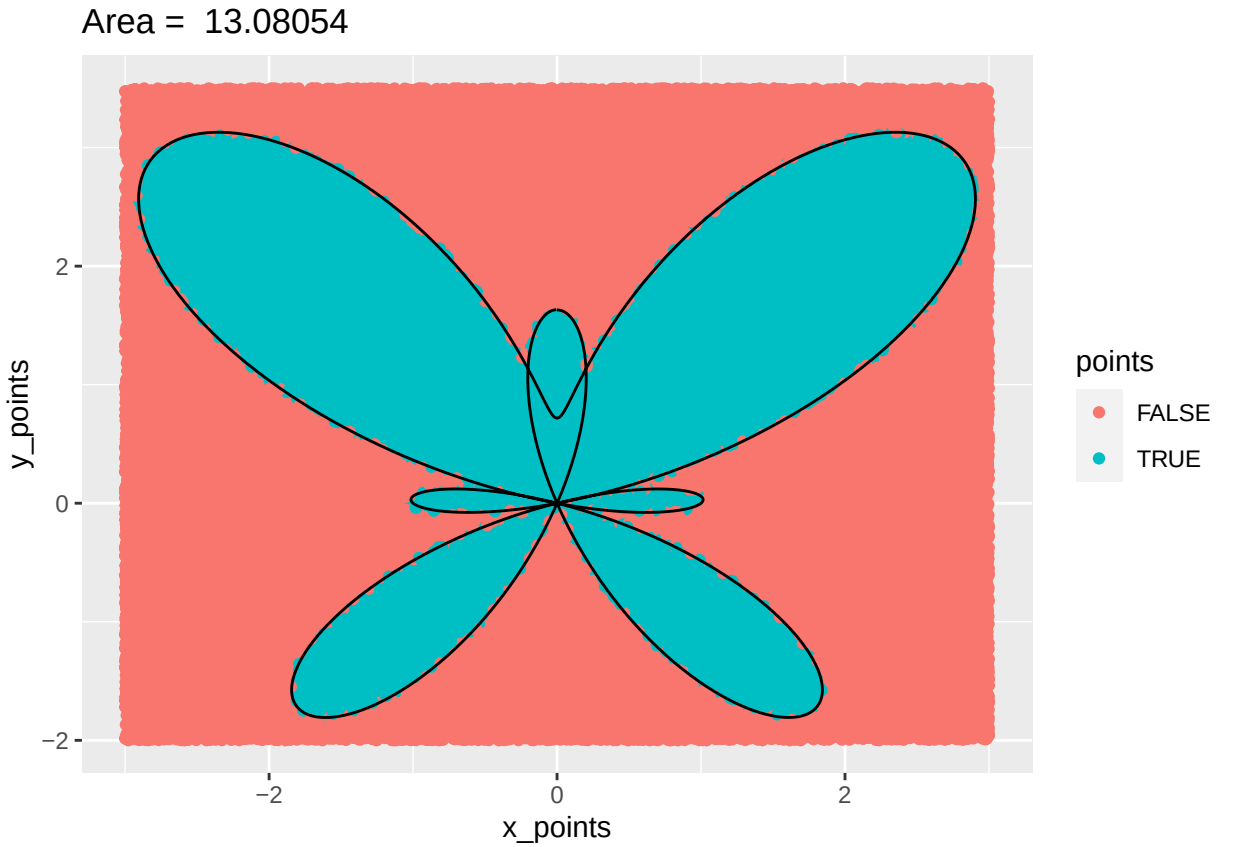

```
areas[2] <- run_gen(1000)
```




```
areas[3] <- run_gen(10000)
```




```
areas[4] <- run_gen(100000)
```




```
df_table <- data.frame (points = c(100, 1000, 10000, 100000), areas = areas)
df_table
```

```
##   points    areas
## 1  1e+02 15.18000
## 2  1e+03 13.56300
## 3  1e+04 13.13400
## 4  1e+05 13.08054
```

We can observe that with sampling larger number of points, the area value gets smaller, since we have more observations and the outcome becomes more precise.

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

Monte Carlo Simulation can be used to estimate the integrals or distinguish the areas of functions, which are too complex to solve otherwise. It uses random sampling to find the outcome. By sampling quite large number, we can depict the output of a function and find out its boundaries, which might be quite hard or impossible to do analytically. Though with the extremely big problems, we would need to trade either accuracy of the solution or it's speed, which we tested in our simulations. Though if the area was -300 to 300 instead of -3 to 3, the 10^5 samples might not be enough to distinguish certain boundaries or be sure of the result.