

WYDZIAŁ NAUK ŚCISŁYCH I TECHNICZNYCH

Symulacje Komputerowe

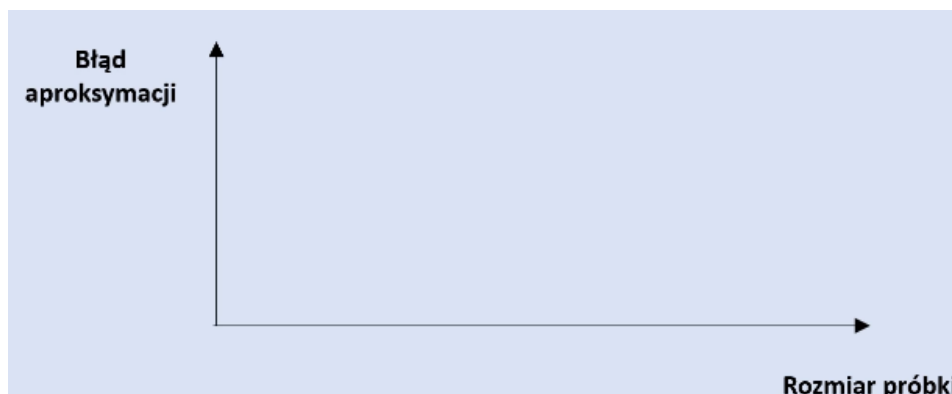
Sprawozdanie “Symulacja Monte Carlo”

Adam Talarczyk, Mateusz Wrzoł

Uniwersytet Śląski, Sosnowiec, 2021

1 Zadanie 1

Należy zmodyfikować kod dla aproksymacji stałej π , aby sprawdzić jak rozmiar próbki wpływa na błąd aproksymacji. Błąd aproksymacji obliczamy jako wartość bezwzględną różnicy, pomiędzy aproksymacją π i wartością rzeczywistą π (3.14159265). Należy przygotować wykres [Rysunek 1].



Rysunek 1: Przykład wykresu

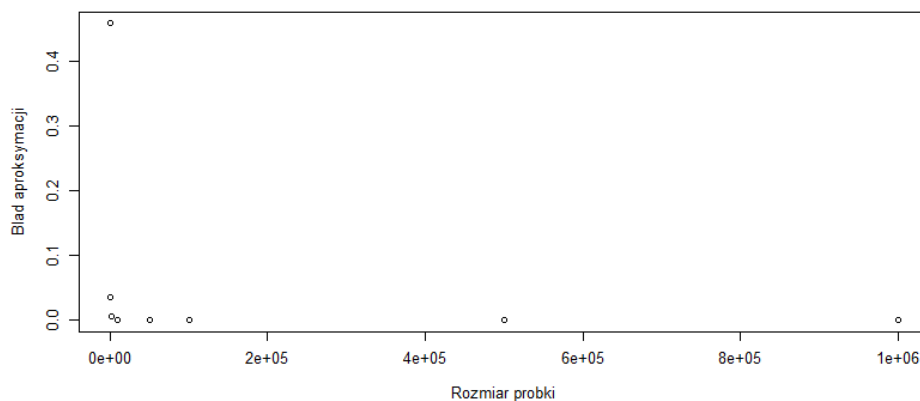
1.1 Rozwiązanie

Do uzyskania odpowiednich wyników konieczne było wyliczenie aproksymacji π dla wielu próbek. W zadaniu wykorzystano próbki z zakresu od 1 do 1000000. Dodatkowo, konieczne było kilkukrotne powtórzenie wykonywanej aproksymacji w celu uśrednienia otrzymanego przybliżenia π . W opisywanym przykładzie zastosowano 10 powtórzeń. Błąd aproksymacji został wyliczony jako różnica bezwzględna pomiędzy zdefiniowaną stałą 3,14159265 oraz uzyskanym uśrednionym przybliżeniem. Wyniki zaprezentowane są w tablicy 1.

| Próbka | Średnie przybliżenie π | Błąd aproksymacji |
|---------|----------------------------|-------------------|
| 1 | 3,6 | 0,45840735 |
| 10 | 3,04 | 0,10159265 |
| 100 | 3,064 | 0,07759265 |
| 1000 | 3,124 | 0,01759265 |
| 10000 | 3,13392 | 0,00767265 |
| 100000 | 3,140084 | 0,00150865 |
| 1000000 | 3,1409996 | 0,00059305 |

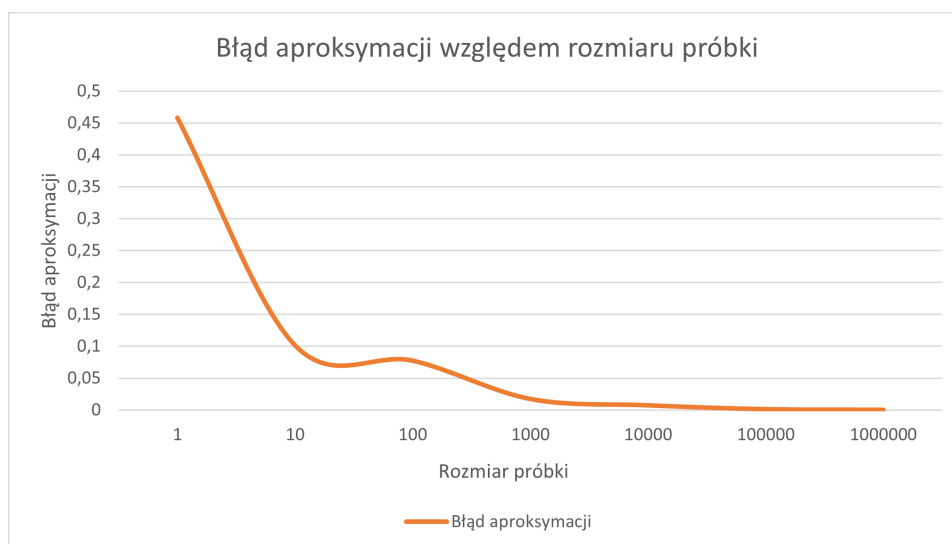
Tablica 1: Tablica z wynikami uzyskanymi w symulacji

Na podstawie powyższych danych wygenerowany został wykres za pomocą funkcji `plot()` (Rysunek 2).

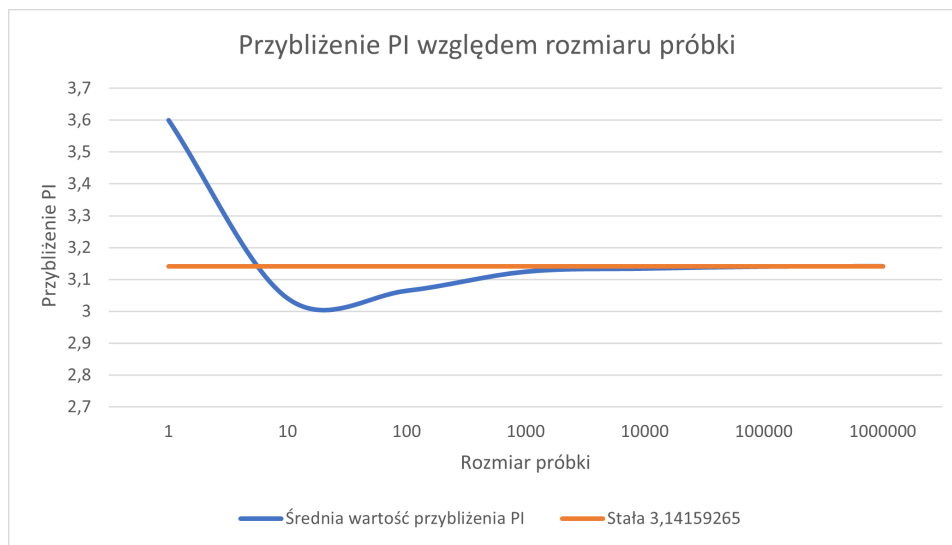


Rysunek 2: Wygenerowany wykres przedstawiający wynik dla aproksymacji π

Przedstawiony na rysunku 2 wykres jest mało czytelny, dlatego wyeksportowano dane do pliku o rozszerzeniu `xlsx` i na ich podstawie utworzono kolejne wykresy: Błąd aproksymacji względem rozmiaru próbki (Rysunek 3) oraz przybliżenie PI względem rozmiaru próbki z zaznaczoną wartością 3,14159265 (Rysunek 4).



Rysunek 3: Wykres przedstawiający błąd aproksymacji względem rozmiaru próbki



Rysunek 4: Wykres przedstawiający przybliżenie PI względem rozmiaru próbki, w porównaniu do wartości 3,14159265

Na podstawie przedstawionej symulacji można stwierdzić, że wielkość błędu aproksymacji zależy od wielkości próbki. Im większa próbka, tym mniejszy jest błąd aproksymacji a przybliżenie liczby π jest bardziej dokładne.

1.2 Kod źródłowy

```

1 # Title      : Monte Carlo Simulation
2 # Objective  :
3 # Created by : Adam Talarczyk, Mateusz Wrzol
4 # Created on : 16.04.2021
5
6 library("xlsx")
7 source('pi/avarage_difference.R')
8
9 calculate <- function(steps, sequences) {
10   diff.vector <- NULL
11   dataset <- data.frame()
12
13   for (runs in steps)
14   {
15     diff <- avarage_difference(runs, sequences)
16     dataset <- rbind(dataset, diff)
17     diff.vector <- append(diff.vector, diff[1, 'diff'])
18   }
19
20   export_dataset(dataset)

```

```

21 draw_plot(steps, diff.vector)
22 }
23
24 export_dataset <- function(dataset){
25   write.xlsx(dataset, file = "pi/export/data.xlsx",
26             sheetName="PI")
27 }
28 draw_plot <- function(steps, results) {
29   plot(steps, results, xlab = 'Rozmiar próbki', ylab = 'Błąd
    aproksymacji', col = 'black')
30 }
31
32 sequence <- c(1,10,100,1000,10000,100000,1000000)
33 calculate(sequence, 10)

```

Listing 1: Plik main.R - wywołanie głównej funkcji

```

1 # Title      : Avarage absolute difference
2 # Objective  :
3 # Created by: Adam Talarczyk, Mateusz Wrzol
4 # Created on: 10.04.2021
5
6 source('pi/approximation.R')
7
8 avarage_difference <- function(runs, sequences = 100) {
9   pi.vector <- NULL
10  for (i in seq(1, sequences, by = 1))
11  {
12    mc.pi <- approximation(runs)
13    pi.vector <- append(pi.vector, mc.pi)
14  }
15
16  avarage_pi <- mean(pi.vector, trim = 0, na.rm = FALSE)
17  difference <- abs(3.14159265 - avarage_pi)
18
19  dataset <- data.frame(
20    step = runs,
21    pi_value = pi.vector,
22    avg_pi = avarage_pi,
23    diff = difference
24  )
25 }

```

Listing 2: Plik avarage_difference.R - odpowiedzialny za wyliczanie błędu aproksymacji PI

```

1 # Title      : PI approximation
2 # Objective  :
3 # Created by: Adam Talarczyk, Mateusz Wrzol

```

```

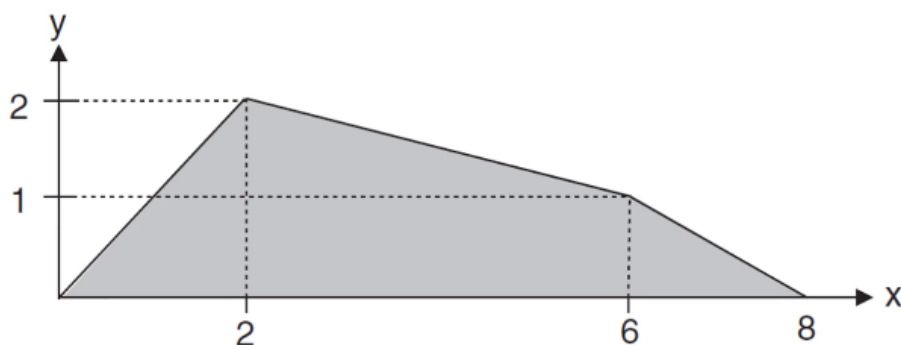
4 # Created on: 10.04.2021
5
6 approximation <- function(runs = 1000) {
7   xs <- runif(runs, min = -0.5, max = 0.5)
8   ys <- runif(runs, min = -0.5, max = 0.5)
9   in.circle <- xs^2 + ys^2 <= 0.5^2
10  mc.pi <- (sum(in.circle) / runs) * 4
11 }

```

Listing 3: Plik approximation.R - odpowiedzialny za obliczanie przybliżenia

2 Zadanie 2

Zaprogramować symulację Monte Carlo (np. w języku R), która pozwoli obliczyć pole powierzchni szarego obszaru, przedstawionego na poniższym rysunku [Rysunek 5]. Obliczyć błąd uzyskanego wyniku.



Rysunek 5: Figura

2.1 Rozwiązanie

Przedział generowania punktów dla współrzędnej x

Przedział generowania punktów dla współrzędnej y

Punkty w trójkącie pierwszym (prawa strona)

Punkty w kwadracie pierwszym (środek)

Punkty w trójkącie drugim (środek)

Punkty w trójkącie trzecim (lewa strona)

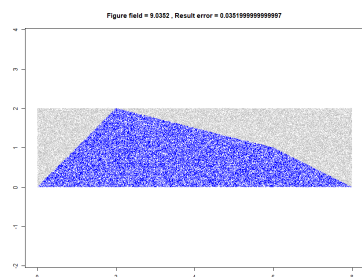
Połączenie wszystkich figur w wielokąt z treści zadania

Iloczyn sumy punktów w figurze i liczby wygenerowanych punktów, został pomnożony przez pole badanego obszaru

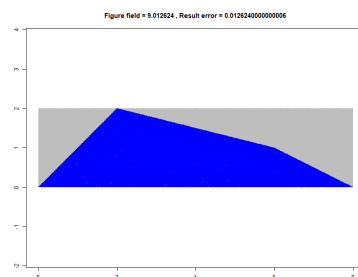
Dokładne pole figury (suma poszczególnych elementów wielokąta)

Błąd uzyskanego wyniku (wartość bezwzględna z różnicy uzyskanego wyniku i dokładnego pola figury)

Tworzenie wykresu wraz z uzyskanym wynikiem i błędem uzyskanego wyniku

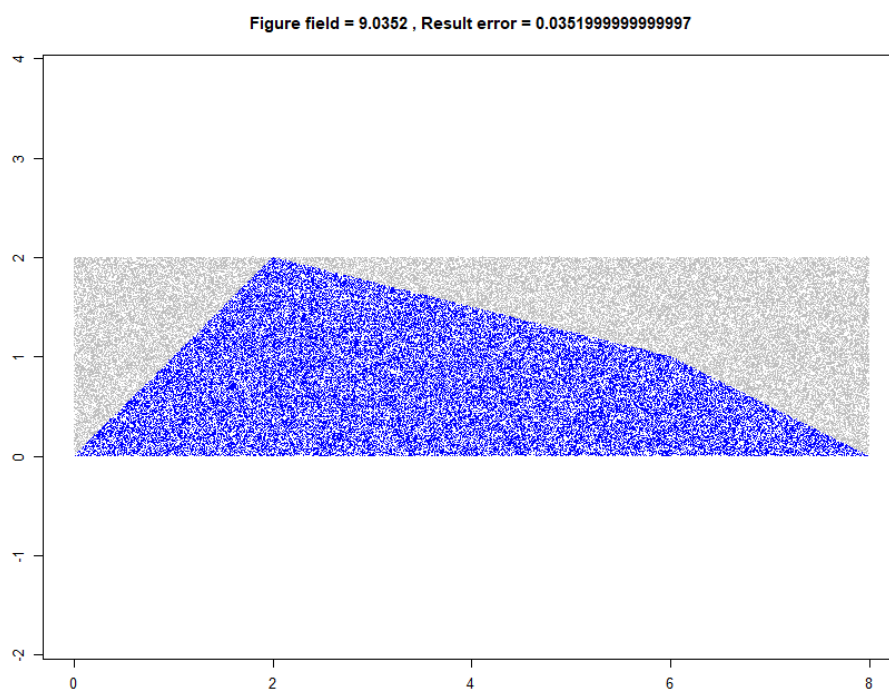


(a) Wycinek z bazy danych

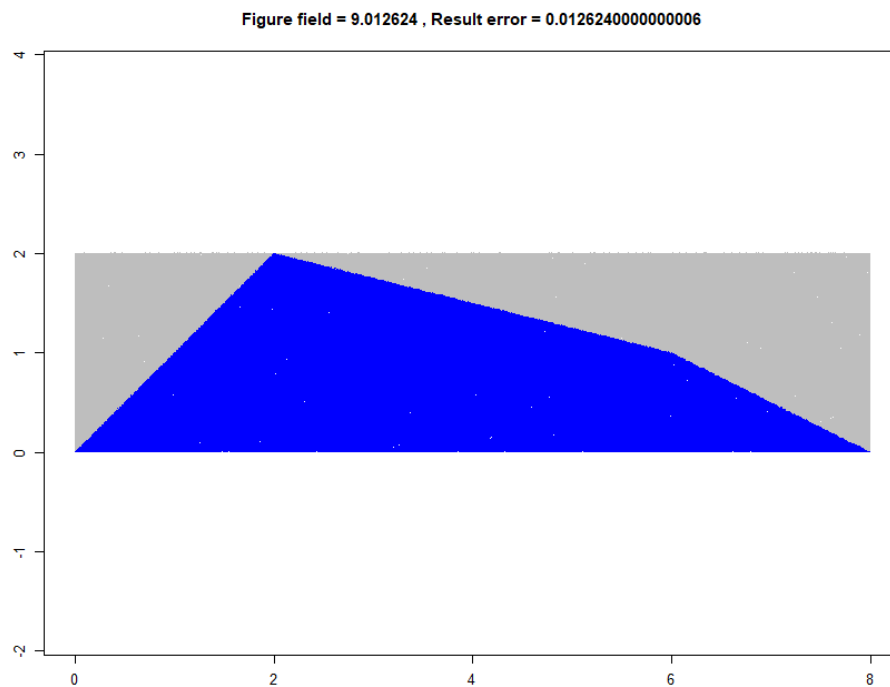


(b) Fragment zwróconej odpowiedzi

Rysunek 6: Działanie funkcji populate()



Rysunek 7: Wykres przedstawiający wynik symulacji dla wartości runs=100000



Rysunek 8: Wykres przedstawiający wynik symulacji dla wartości runs=1000000

2.2 Kod źródłowy

```

1 # Title      : Monte Carlo Simulation
2 # Objective  :
3 # Created by : Adam Talarczyk , Mateusz Wrzol
4 # Created on : 15.04.2021
5
6 figure <- function(runs) {
7   xs <- runif(runs, min = 0, max = 8)
8   ys <- runif(runs, min = -0, max = 2)
9
10  in.r_triangle <- xs <= 2 & ys <= 2 & ys <= xs
11  in.mid_square <- xs <= 6 & xs >= 2 & ys <= 1
12  in.l_triangle <- xs >= 6 &
13    xs <= 8 &
14    ys <= 1 &
15    ys <= (-0.5 * xs + 4)
16  in.mid_triangle <- xs >= 2 &
17    xs <= 6 &
18    ys >= 1 &
19    ys <= (-0.25 * xs + 2.5)

```

```

20
21 in_all_figures <- in.r_triangle +
22   in.mid_square +
23   in.mid_triangle +
24   in.l_triangle
25
26 mc_figure_field <- (sum(in_all_figures) / runs) * 16
27 mc_exact_figure_field <- (2 * 2 / 2) + (1 * 4 / 2) + (1 *
28   2 / 2) + (1 * 4)
29
30 mc.error <- abs(mc_exact_figure_field - mc_figure_field)
31 plot(xs, ys,
32   col = ifelse(in_all_figures, "blue", "grey"),
33   xlab = '', ylab = '', asp = 1, pch = '.',
34   main = paste("Figure field =", mc_figure_field, ",
35     Result error =", mc.error))
36 }
37
38 figure(100000)

```

Listing 4: Plik figure.R - odpowiedzialny za obliczenie pola i wygenerowanie wykresu