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

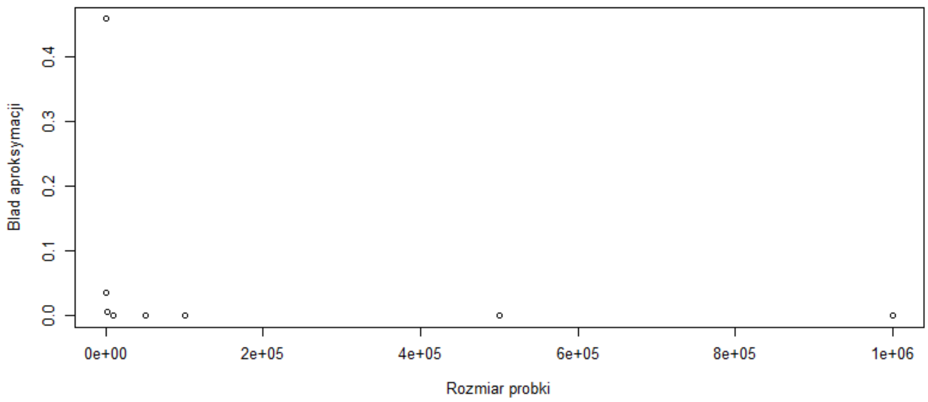
1.1 Rozwiązanie

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

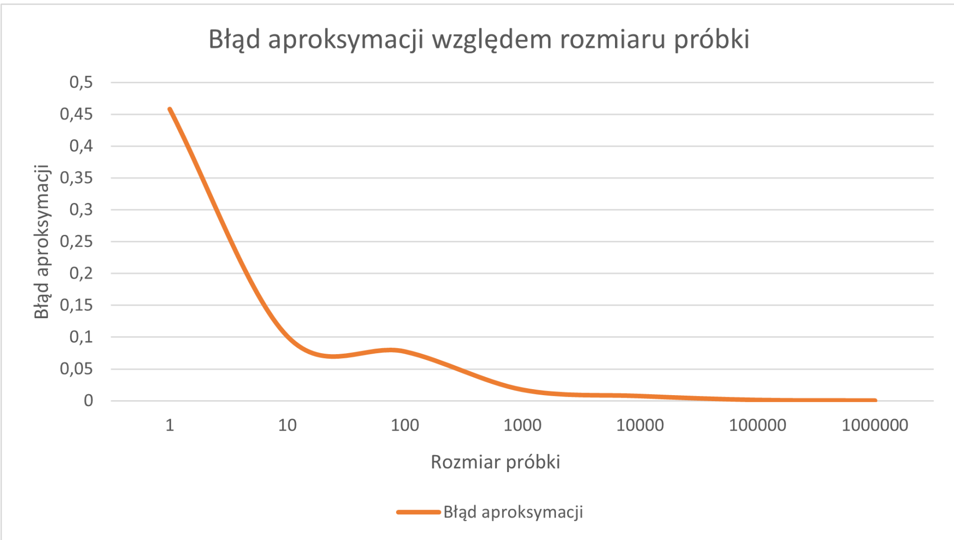
Tablica 1: Tablica z wynikami uzyskanymi w symulacji

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

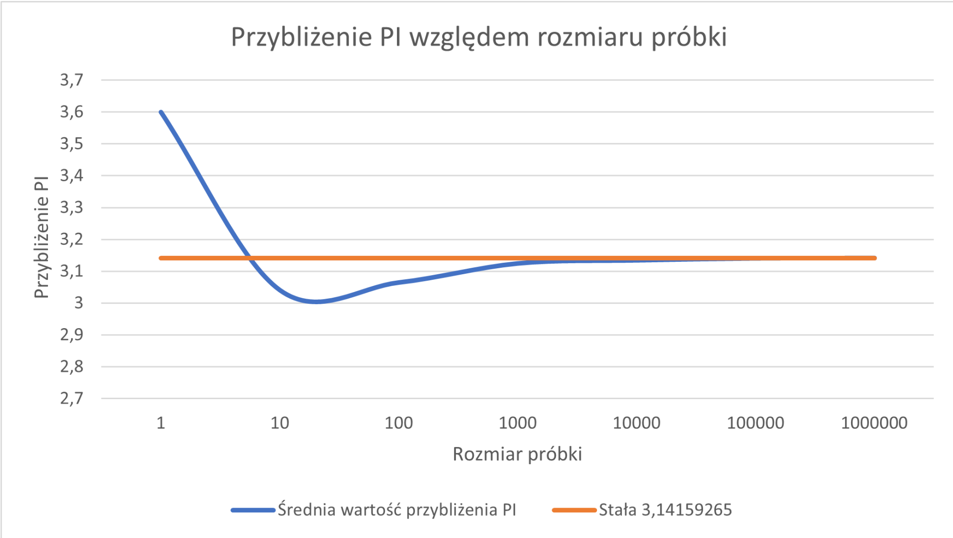


Rysunek 2: Wygenerowany wykres przedstawiający błąd aproksymacji względem rozmiaru próbki

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

```
# Title       : Monte Carlo Simulation
# Objective :
# Created by: Adam Talarczyk, Mateusz Wrzol
# Created on: 16.04.2021

library("xlsx")
source('pi/avarage_difference.R')

calculate <- function(steps, sequences) {
  diff.vector <- NULL
  dataset <- data.frame()

  for (runs in steps)
  {
    diff <- avarage_difference(runs, sequences)
    dataset <- rbind(dataset, diff)
    diff.vector <- append(diff.vector, diff[1, 'diff'])
  }

  export_dataset(dataset)
```

```
  draw_plot(steps, diff.vector)
}

export_dataset <- function(dataset){
  write.xlsx(dataset, file = "pi/export/data.xlsx",
    sheetName="PI")
}

draw_plot <- function(steps, results) {
  plot(steps, results, xlab = 'Rozmiar probki', ylab = 'Blad
    aproksymacji', col = 'black')
}

sequence <- c(1,10,100,1000,10000,100000,1000000)
calculate(sequence, 10)
```

Listing 1: Plik main.R - wywołanie głównej funkcji

```
# Title      : Avarage absolute difference
# Objective  :
# Created by: Adam Talarczyk, Mateusz Wrzol
# Created on: 10.04.2021

source('pi/approximation.R')

avarage_difference <- function(runs, sequences = 100) {
  pi.vector <- NULL
  for (i in seq(1, sequences, by = 1))
  {
    mc.pi <- approximation(runs)
    pi.vector <- append(pi.vector, mc.pi)
  }

  avarage_pi <- mean(pi.vector, trim = 0, na.rm = FALSE)
  difference <- abs(3.14159265 - avarage_pi)

  dataset <- data.frame(
    step = runs,
    pi_value = pi.vector,
    avg_pi = avarage_pi,
    diff = difference
  )
}
```

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

```
# Title      : PI approximation
# Objective  :
# Created by: Adam Talarczyk, Mateusz Wrzol
```

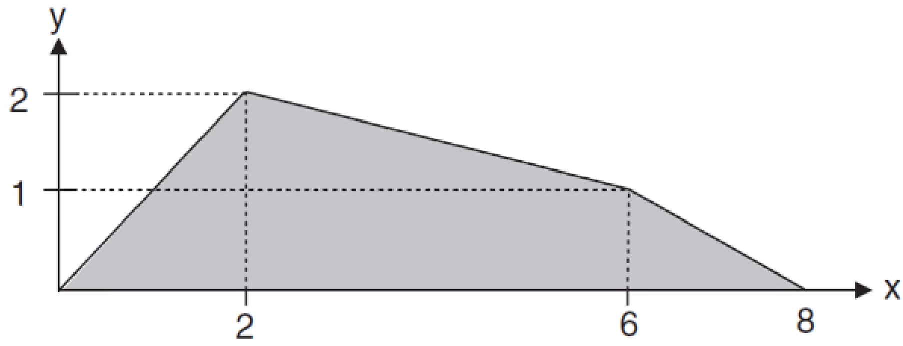
```r
# Created on: 10.04.2021

approximation <- function(runs = 1000) {
  xs <- runif(runs, min = -0.5, max = 0.5)
  ys <- runif(runs, min = -0.5, max = 0.5)
  in.circle <- xs^2 + ys^2 <= 0.5^2
  mc.pi <- (sum(in.circle) / runs) * 4
}
```

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

2 Zadanie 2

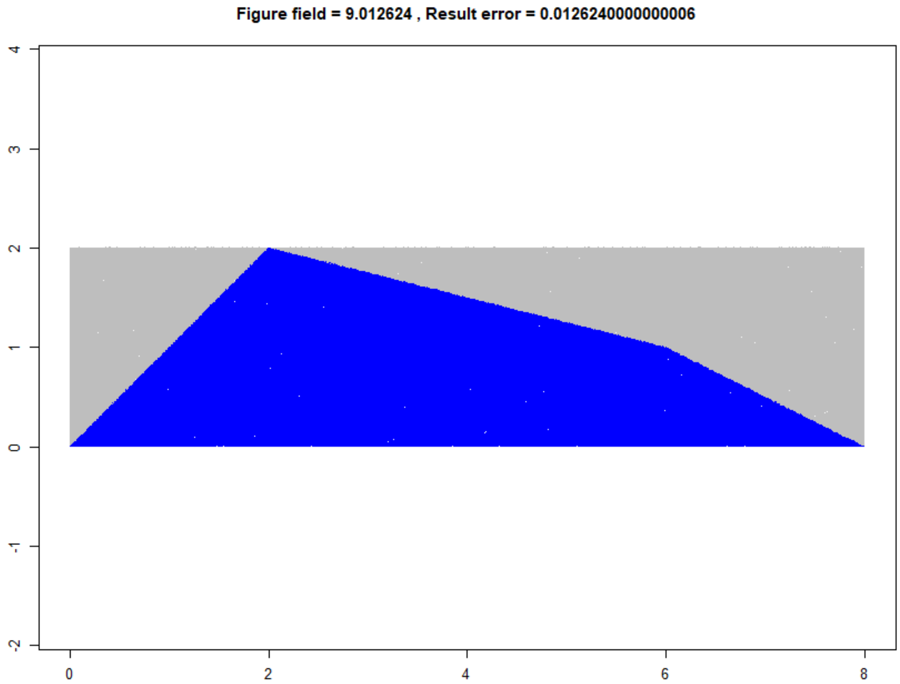
Zaprogramować symulację Monte Carlo (np. w języku R), która pozwoli obliczyć pole powierzchni szarego obszaru, przedstawionego na poniższym rysunku [Rysunek 5]. Obliczyć błąd uzyskanego wyniku.



Rysunek 5: Figura

2.1 Rozwiązanie

W celu uzyskania odpowiedniego wyniku, w pierwszej kolejności generowane są punkty dla współrzednych `x` (od 0 do 8) i `y` (od 0 do 2). Następnie sprawdzane jest, czy punkty znajdują się w przestrzeni odpowiedniej figury (z podziałem na mniejsze figury, takie jak prawy trójkąt, lewy trójkąt, środkowy trójkąt oraz środkowy prostokąt). Kolejnym krokiem jest połączenie wyników dla poszczególnych figur w jeden wielokąt z treści zadania. Dokładne pole figury zostało wyliczone jako suma poszczególnych elementów wielokąta, zastosowano działanie `(2*2/2)+(1*4/2)+(1*2/2)+(1*4)`, którego wynik wynosi 9. Wynik symulacji to iloraz sumy punków w figurze i liczby wygenerowanych punktów, pomnożony przez pole badanego obszaru (`2*8=16`). Błąd uzyskanego wyniku został wyliczony jako wartość bezwzględna z różnicy uzyskanego wyniku i dokładnego pola figury. Na sam koniec wygenerowany został wykres przy użyciu funkcji `plot()` (Resunek 6).

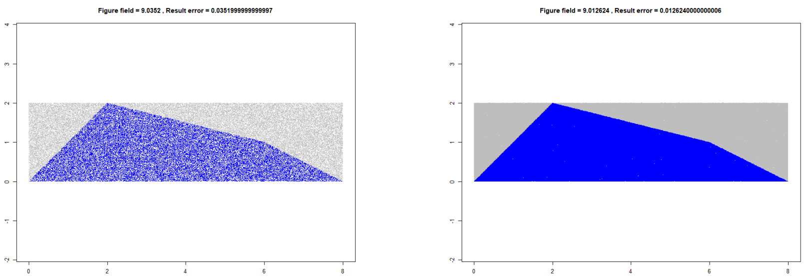


Rysunek 6: Figura utworzona przy użyciu funkcji plot() dla próbki równej 1000000

Wyniki dla poszczególnych próbek zamieszczone są w tablicy 2. Dodatkowo, wygenerowane wykresy przedstawione są na rysunku 7 w formie porównania względem wielkości próbki.

Próbka	Pole figury	Błąd
100000	9.0352	0.0351999999999997
1000000	9.012624	0.0126240000000006

Tablica 2: Tablica z wynikami uzyskanymi w symulacji



(a) Wynik symulacji dla próbki 100000

(b) Wynik symulacji dla próbki 1000000

Rysunek 7: Porównanie wizualizacji figur pod względem wielkości próbki

Na podstawie przedstawionych wyników zauważyć można, że dokładność uzyskanego pola figury rośnie wraz ze zwiększeniem probki.

2.2 Kod źródłowy

```
# Title       : Monte Carlo Simulation
# Objective  :
# Created by: Adam Talarczyk, Mateusz Wrzol
# Created on: 15.04.2021

figure <- function(runs) {
  xs <- runif(runs, min = 0, max = 8)
  ys <- runif(runs, min = 0, max = 2)

  in.r_triangle <- xs <= 2 & ys <= 2 & ys <= xs
  in.mid_rect <- xs <= 6 & xs >= 2 & ys <= 1
  in.l_triangle <- xs >= 6 &
    xs <= 8 &
    ys <= 1 &
    ys <= (-0.5 * xs + 4)
  in.mid_triangle <- xs >= 2 &
    xs <= 6 &
    ys >= 1 &
    ys <= (-0.25 * xs + 2.5)

  in.all_figures <- in.r_triangle +
    in.mid_rect +
    in.mid_triangle +
    in.l_triangle

  mc_exact_figure_field <- (2 * 2 / 2) + (1 * 4 / 2) + (1 *
    2 / 2) + (1 * 4)
  mc_figure_field <- (sum(in.all_figures) / runs) * 16

```

```
  mc.error <- abs(mc_exact_figure_field - mc_figure_field)
  plot(xs, ys,
       col = ifelse(in.all_figures, "blue", "grey"),
       xlab = '', ylab = '', asp = 1, pch = '.',
       main = paste("Figure field =", mc_figure_field, ",
    Result error =", mc.error))
}

figure(100000)
```

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