Laboratorium programistyczne 2
Metoda Monte Carlo

Projekt „Matematyka dla Ciekawych Świata”,
Piotr Morawiecki

kwiecień 2025

Spis treści

1 Powtórzenie wykładu: Metoda Monte Carlo

Metoda Monte Carlo polega na wielokrotnym losowym odgrywaniu danego procesu (np. gry Black Jacka), a następnie na podstawie jego (losowych) wyników oszacowywanie szukanej wartości.

Na wykładzie omówiliśmy dwa przykłady zastosowania Metody Monte Carlo. Jeden przykład dotyczył szacowania pola koła, które będzie dokładniej omówione w Sekcji 2, a drugi dotyczył szacowania prawdopodobieństwa uzyskania 21 punktów w grze Black Jack. W tym celu zasymulowaliśmy wiele rozdań Black Jacka, za każdym razem w losowy sposób dobierając karty. Następnie oszacowaliśmy szukane prawdopodobieństwo, dzieląc liczbę rozdań z wynikiem 21 przez sumę wszystkich rozdań.

Metoda Monte Carlo jest tym dokładniejsza, im więcej razy wykonamy dany proces. Wynika to z *Prawa Wielkich Liczb*. Zgodnie z tym prawem, gdy liczba prób losowego doświadczenia dąży do nieskończoności, średnia arytmetyczna wyników tych prób dąży do ich wartości oczekiwanej.

2 Szacowanie pola koła metodą Monte Carlo

W tej części pracowni spróbujemy oszacować pole koła o promieniu 1 z wykorzystaniem metody Monte Carlo. Jako że pole takiego koła wynosi π, to metoda ta pozwoli nam oszacować wartość π.

Koło o promieniu 1 i środku w (0, 0) to zbiór punktów (x, y), których odległość od środka koła $r = \sqrt{x^2 + y^2}$ jest mniejsza lub równa 1. Zatem:

$$\sqrt{x^2 + y^2} \leq 1,$$

lub po podniesieniu do kwadratu

$$x^2 + y^2 \leq 1.$$

Oszacujemy pole następująco. Wygenerujemy pewną liczbę (n) punktów na kwadracie o wymiarze 2×2, w którego jest wpisane nasze koło. Następnie sprawdzimy jaka część tych punktów znajduje się wewnątrz koła. Pole koła $P_{\text{koło}}$ można oszacować jako:

$$P_{\text{koło}} = \frac{\text{liczba punktów wśrodku koła}}{\text{liczba punktów naniesionych na kwadrat}} \cdot P_{\text{kwadrat}}. \tag{1}$$

gdzie $P_{\text{kwadrat}} = 4$.

Krok 1. Wprowadzamy zmienną n do określania liczby naniesionych punktów oraz zmienną `inside_circle`, w której będziemy zapisywać liczbę punktów, które znalazły się w środku koła.

```
n = 1000
in_circle = 0
```

Krok 2. W pętli **for** generujemy pary punktów x, y z przedziału od -1 do 1. Pamiętaj załączeniu pakietu `random` na początku skryptu za pomocą komendy **import random**.

```
for i in range(n):
  x = random.uniform(-1, 1)
  y = random.uniform(-1, 1)
```

Krok 3. Wewnątrz pętli **for**, dla każdej wygenerowanej pary współrzędnych (x, y) sprawdzamy czy dany punkt leży wewnątrz okręgu, i jeśli tak to zwiększamy wartość zmiennej `in_circle`.

```
  if x ** 2 + y ** 2 <= 1:
    in_circle += 1
```

Krok 4. Po zakończeniu pętli **for**, oszacowujemy pole koła:

```
print(in_circle / n * 4)
```

Końcowy kod powinien wyglądać następująco:

```
import random

n = 1000
in_circle = 0

for i in range(n):
  x = random.uniform(-1, 1)
  y = random.uniform(-1, 1)
  if x ** 2 + y ** 2 <= 1:
    in_circle += 1

print(in_circle / n * 4)
```

Zadanie 2.0.1

Wykorzystując powyższy program odpowiedz na poniższe pytania.

1. Jakie wartości obserwujesz? Czy są one w przybliżone równe powierzchni koła $P_{\text{koło}} = \pi \approx 3.14$?
2. Zwiększ n do 100 000. Jak zwiększenie n wpływa na oszacowane pole koła?
3. Zwiększ n do 10 000 000. Co obserwujesz?

Dla chętnych: Jeśli czujecie się pewnie z programowaniem, to zachęcam was do przeczytania i wykonania przykładów z Sekcji 5, w której zbadacie zbieżność powyżej zaimplementowanej metody.

3 Sekwencje w rzucie monetą

Rozważmy następujący eksperyment. Wykonujemy kolejne rzuty monetą, aż do momentu kiedy wypadną pod rząd dwa orły. Przykładowo sekwencja może wyglądać następująco:

reszka, orzeł, reszka, reszka, **orzeł, orzeł** (6 rzutów)

Chcielibyśmy sprawdzić, średnio po ilu rzutach to nastąpi. Przekonamy się o tym, pisząc prostą symulację Monte Carlo.

Zadanie 3.0.1

Napisz funkcję `test_orzel()`, która wykona pojedynczą symulację tego eksperymentu, a następnie zwróci liczbę rzutów, po których wypadły dwa orły.
Wskazówka 1: Pojedynczy rzut można zasymulować za pomocą komendy `random.randint(0, 1)`. Wygeneruje ona liczbę 0 lub 1. Oznaczmy 0 jako reszkę, a 1 jako orła.
Wskazówka 2: Wykonuj rzuty monetą w pętli `while`, jednocześnie zapisując liczbę wykonanych rzutów w zmiennej `n`. Jeśli w ostatnich dwóch rzutach wypadną orły to zwróć tą wartość `return n`.

Zadanie 3.0.2

Wykonaj funkcję `test_orzel()` 10 000 razy i oblicz średnią liczbę rzutów.

Wynik powinien być w przybliżeniu równy 6 - jest wartość oczekiwana liczby rzutów aż do momentu uzyskania dwóch orłów.

Zadanie 3.0.3

Rozważmy teraz drugi eksperyment, w którym będziemy rzucać monetą aż do momentu kiedy po wyrzuceniu orła wyrzucimy reszkę, np:

reszka, reszka, orzeł, orzeł, **orzeł, reszka** (6 rzutów)

1. Jakie jest prawdopodobieństwo, że po orle wypadnie orzeł, a jakie że wypadnie po nim reszka? Czy spodziewasz się, że czas do momentu uzyskania sekwencji (orzeł, reszka) jest ten sam co w przypadku sekwencji (orzeł, orzeł) z pierwszego eksperymentu?
2. Zaimplementuj funkcję `test_reszka()`, która zasymuluje rzut monetą aż do wypadnięcia sekwencji (orzeł, reszka), a następnie zwróci liczbę rzutów. *Wskazówka: Wystarczy w niewielkim stopniu zmodyfikować funkcję* ***`test_orzel()`***.
3. Wykonaj doświadczenie 10 000 razy i na tej podstawie oszacuj wartość oczekiwaną liczby rzutów.
4. Jak można uzasadnić wynik tej symulacji?

Jeśli zainteresował Cię ten przykład to polecam film na ten temat na kanale Numberphile:
https://www.youtube.com/watch?v=SDw2Pu0-H4g&ab_channel=Numberphile

4 Zadanie dodatkowe: Black Jack

W grze *Black Jack* kartom przypisuje się następujące wartości:

- karty 2-10 są warte tyle samo punktów, ile wynosi ich wartość,
- walet, królowa i król są warte po 10 punktów,
- as jest wart 1 lub 11 punktów, w zależności co jest korzystniejsze dla gracza.

Na wykładzie omówiliśmy jak za pomocą metody Monte Carlo określić prawdopodobieństwo uzyskanie 21 puntków, przy założeniu, że dobieramy karty aż do momentu uzyskania conajmniej 21 punktów. Implementację z wykładu przedstawiam poniżej.

1. Funkcja do obliczania wartości kart:

```
def score(picked_cards):
    score = 0
    n_aces = 0

    for card in picked_cards:
        if card == "J" or card == "Q" or card == "K":
            score += 10
        elif card == "A":
            score += 1
            n_aces += 1
        else:
            score += card

    for i in range(n_aces):
        if (score <= 11):
            score += 10

    return score
```

2. Symulacja Monte Carlo:

```
import random

blackjack = 0
n = 10000

for i in range(n):
    cards = [2, 3, 4, 5, 6, 7, 8, 9, 10, "J", "Q", "K", "A"]
    cards = cards * 4
    picked_cards = []

    while score(picked_cards) < 21:
        random_card = random.choice(cards)
        picked_cards.append(random_card)
        cards.remove(random_card)

    if score(picked_cards) == 21:
        blackjack += 1

print(blackjack / n)
```

Zadanie 4.0.1

Krupier w kasynie dobiera karty aż do momentu kiedy liczba punktów na jego kartach będzie większa lub równa 17. Wykonaj symulację Monte Carlo, żeby odpowiedzieć na następujące pytania:

1. Jakie jest prawdopodobieństwo, że krupier będzie miał 21 punktów?
2. Jakie jest prawdopodobieństwo, że krupier przekroczy 21 punktów?
3. Jakie jest prawdopodobieństwo, że krupier osiągnie 21 punktów dobierając jedynie dwie karty?
4. Jakie jest prawdopodobieństwo, że krupier dobierze co najmniej 5 kart?

5 Badanie zbieżności metody Monte Carlo (dla chętnych)

Zgodnie z *prawem wielkich liczb* oszacowana wartość pola powierzchni powinna zbiegać do wartości oczekiwanej wraz ze wzrostem liczby wygenerowanych punktów. W tym ćwiczeniu sprawdźmy jak szybko oszacowana wartość pola ta zbiega do wartości π.

Zacznijmy od przepisania programu z sekcji 2, w taki sposób, żeby wartość π była oszacowywana po wylosowaniu każdego punktu. Napiszmy funkcję `estimatePi(n)`, która po kolei będzie losować n punktów, a po wygenerowaniu każdego z nich oszacuje wartość π za pomocą wzoru (1):

```
import random

def estimatePi(n):
  in_circle = 0
  number_of_points = 0
  estimatedPi = []

  for i in range(n):
    x = random.uniform(-1, 1)
    y = random.uniform(-1, 1)

    number_of_points += 1
    if x ** 2 + y ** 2 <= 1:
      in_circle += 1

    estimatedPi.append(in_circle / number_of_points * 4)
  return estimatedPi
```

Funkcja ta na wyjściu zwróci listę oszacowanych wartości π. Na przykład wywołanie funkcji

```
estimatePi(8)
```

powinno zwrócić 8 oszacowanych wartości π, np.:

$$[4.0, 4.0, 2.6666666666666665, 3.0, 2.4, 2.6666666666666665, 2.857142857142857, 3.0]$$

gdzie pierwsza wartość została oszacowana na podstawie pierwszego wylosowanego punktu, druga wartość na podstawie dwóch pierwszych wylosowanych punktów, itd.

Teraz wykonajmy kilka razy funkcję `estimatePi(1000)`, przedstawiając uzyskane wartości na wykresie:

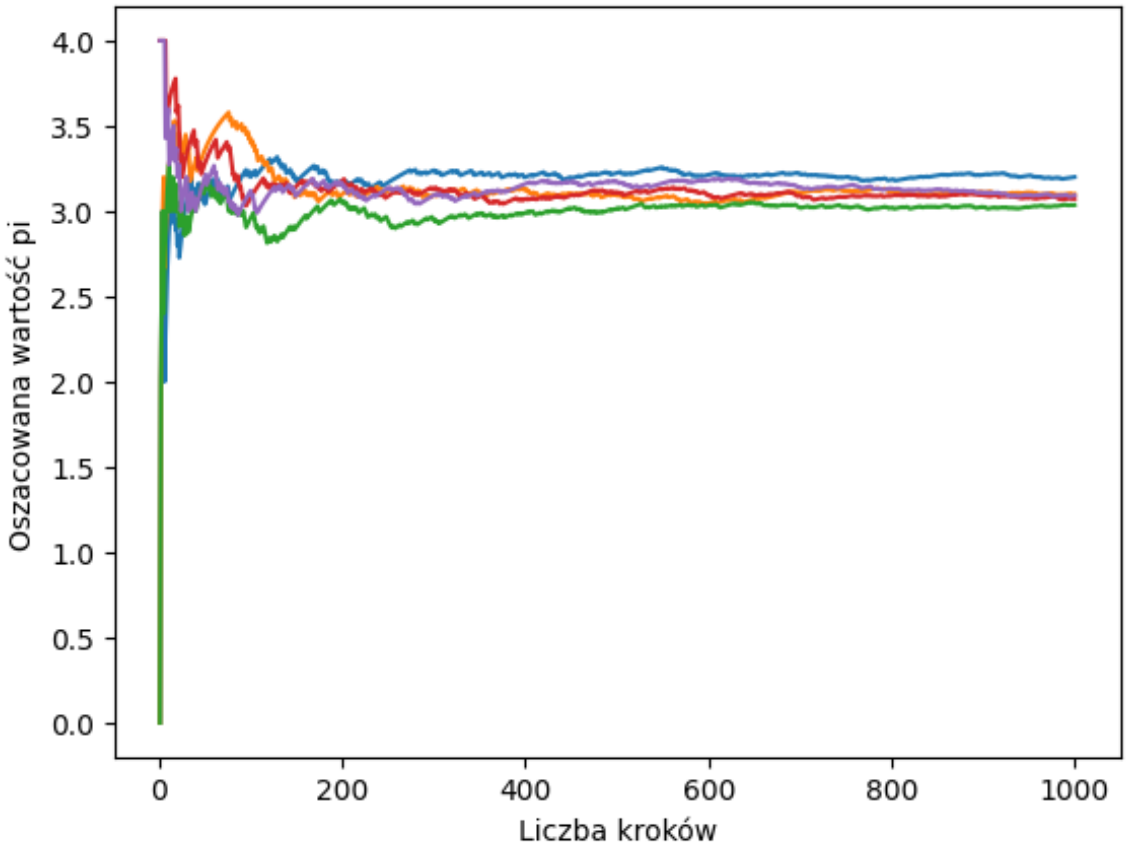
```
import matplotlib.pyplot as plt

n = range(1,1001)

plt.plot(n, estimatePi(1000))
plt.plot(n, estimatePi(1000))
plt.plot(n, estimatePi(1000))
plt.plot(n, estimatePi(1000))
plt.plot(n, estimatePi(1000))

plt.xlabel("Liczba kroków")
plt.ylabel("Oszacowana wartość pi")
plt.show()
```

Wykres powinien wyglądać podobnie do poniższego.



Tak jak oczekiwaliśmy, Wraz ze wzrostem liczby wylosowanych punktów oszacowana wartość zbiega do wartości π. Spróbujmy dokładniej zbadać tę zbieżność. W tym celu wykonamy funkcję `estimatePi(1000)` 100-krotnie, a następnie zbadamy rozrzut oszacowanych wartości π dla kolejnych kroków $1, 2, 3, \ldots, 1000$. W matematyce do określenia rozrzutu wartości stosuje się tzw. odchylenie standardowe (po angielsku *standard deviation, std*). Odchylenia standardowe z wartości $x_1, x_2, \ldots, x_n$ definiuje się jako:

$$\text{std} = \sqrt{\frac{(x_1 - \bar{x})^2 + (x_2 - \bar{x})^2 + \cdots + (x_n - \bar{x})^2}{n}}, \tag{2}$$

gdzie $\bar{x}$ to wartość średnia x. Odchylenie standardowe można obliczyć za pomocą funkcji `std` z pakietu `numpy`.

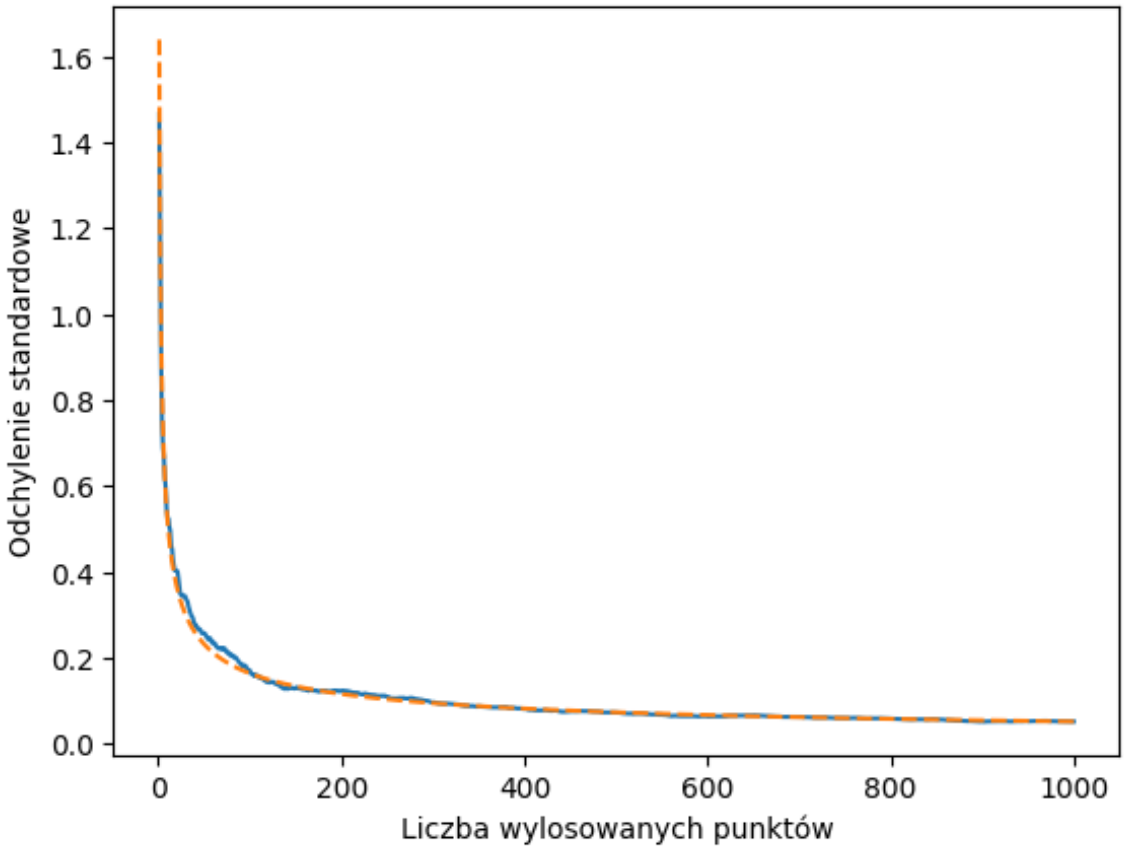
```
import numpy as np

data = np.array([estimatePi(1000) for i in range (100)])
std = np.std(data, axis=0)
```

Program ten 100 razy wykonuje funkcję `estimatePi`, a następnie zapisuje ją w formie tablicy `np.array`, który to format jest potrzebny do wykonania funkcje *np.std* obliczającej odchylenie standardowe z każdej kolumny.

Przedstawmy uzyskane wartości odchylenia standardowego na wykresie:

```
plt.plot(n, std)
plt.xlabel("Liczba wylosowanych punktów")
plt.ylabel("Odchylenie standardowe")
```



Wyraźnie widać, że wraz ze wzrostem liczby wygenerowanych punktów (n) rozrzut uzyskanych wartości maleje. W teorii powinien on maleć proporcjonalnie do $\frac{1}{\sqrt{n}}$. Sprawdźmy to, zaznaczając na naszym wykresie funkcję:

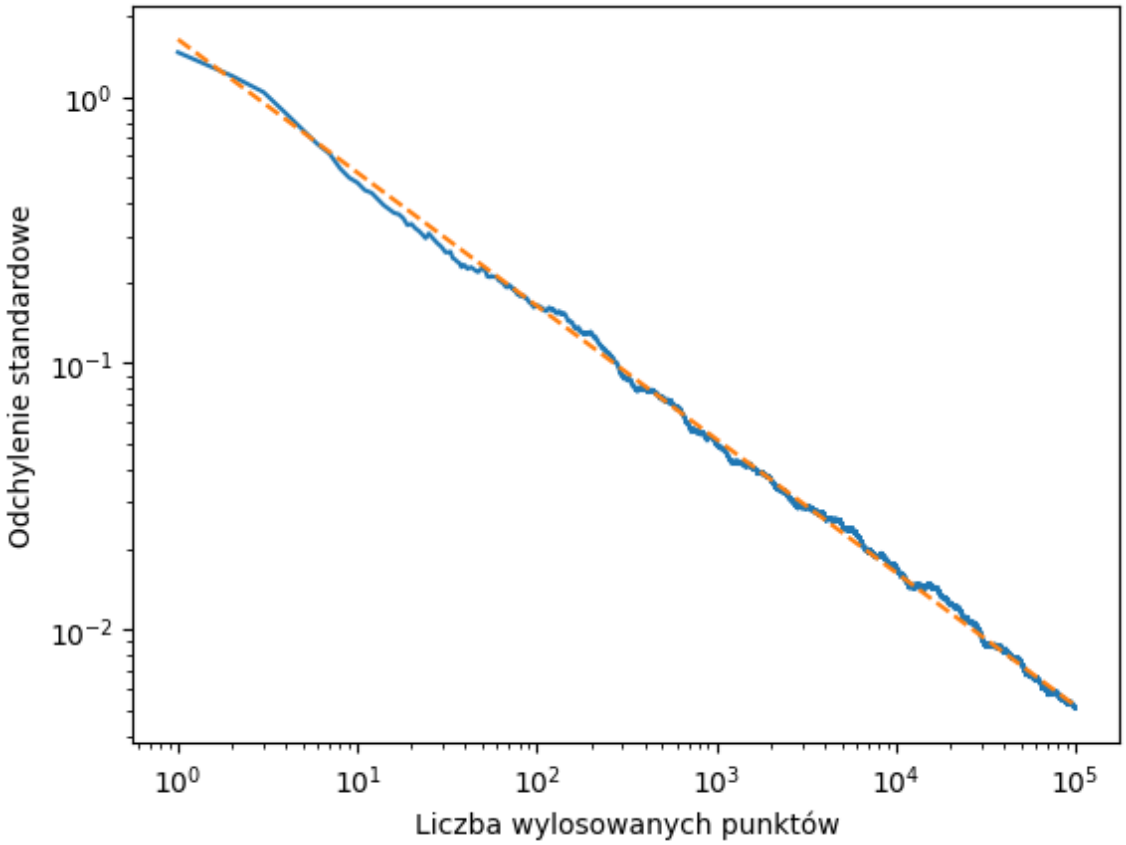
```
plt.plot(n, 1.64 / np.sqrt(n), '--')
```

Wartość 1.64 odpowiada tutaj wartości oczekiwanej odchylenia standardowego dla jednego wylosowanego punktu ($n = 1$). Wygenerowany wykres powinien wyglądać następująco:

$$\text{std} = \frac{1.64}{\sqrt{n}}. \tag{3}$$

Jeszcze lepiej można przedstawić tę zależność na wykresie w skali logarytmicznej. W tej skali zależność std $\propto \frac{1}{\sqrt{n}}$ ma kształt linii prostej. Rysujemy ten wykres wykorzystując zamiast komendy `plt.plot`, komendę `loglog`:

```
n = range(1, 1001)
plt.loglog(n, std)
plt.plot(n, 1.64 / np.sqrt(n), '--')
plt.xlabel("Liczba wylosowanych punktów")
plt.ylabel("Odchylenie standardowe")
```



Zadanie 5.0.1

Sprawdź czy zależność (3) będzie dobrze opisywać odchylenie standardowe dla dużo większych wartości n, np. $n = 10\ 000$ i $n = 100\ 000$.

6 Praca domowa nr 2

Rozwiązania zadań domowych należy przesłać do czasu następnych ćwiczeń. Prowadzący może również zmienić ostateczny termin przesłania pracy domowej, np. na inną godzinę lub dzień.

Za pracę domową można maksymalnie dostać 6 punktów. Można wybierać zarówno łatwiejsze zadania, jak i trudniejsze, za większą liczbę punktów. Można również rozwiązać zadania za większą liczbę punktów, np. za 10. Jeśli wtedy poprawnie będą rozwiązane zadania za 5 punktów, to zostanie przydzielone 5 punktów. Jeśli będą poprawnie rozwiązane zadania za więcej niż 6 punktów, np. 8 czy 10, to taka osoba otrzyma maksymalnie 6 punktów.

Pamiętaj, żeby notebook był czytelny – każde zadanie powinno być umieszczone w osobnym bloku tekstowym oraz poprzedzone numerem (i opcjonalnie opisem) zadania. Komentarze są mile widziane.

Zadanie 6.0.1 — 2 pkt (najwyższa z trzech liczb)

1. Napisz funkcję, która wygeneruje dwie liczby rzeczywiste z przedziału od 0 do 1, a następnie zwróci w wyniku wartość wyższej liczby.
2. Sprawdź z wykorzystaniem metody Monte Carlo, jaka jest wartość oczekiwana wyniku tego losowania.

Zadanie 6.0.2 — 2 pkt (objętość kuli)

Zmodyfikuj program z sekcji 2, tak żeby wykorzystywał on metodę Monte Carlo do oszacowania objętości **kuli** o promieniu 1.
Wskazówka: Odległość punktu (x, y, z) od środka kuli w punkcie $(0, 0, 0)$ wynosi:

$$d = \sqrt{x^2 + y^2 + z^2}.$$

Zadanie 6.0.3 — 2 pkt (pole elipsy)

Rozważmy elipsę o pół osiach $a = 1$ i $b = 2$. Punkt (x, y) należy do elipsy, kiedy spełniony jest warunek:

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$$

1. Napisz program, który oszacuje pole powierzchni tej elipsy z wykorzystaniem metody Monte Carlo.
2. Porównaj oszacowaną wartość pola do pola elipsy obliczonego ze wzoru $P = \pi ab$.

Zadanie 6.0.4 — 3 pkt (wizualizacja metody Monte Carlo)

Narysuj na wykresie x-y 100 losowo wygenerowanych punktów w obrębie kwadratu o wierzchołkach w punktach $(-1, -1)$, $(1, -1)$, $(1, 1)$ i $(-1, 1)$. Punkty znajdujące się wewnątrz koła o środku w punkcie $(0, 0)$ i promieniu 1 zaznacz innym kolorem niż punkty znajdujące się na zewnątrz tego koła.

Zadanie 6.0.5 — 3 pkt (rzut monetą)

Rozważmy następujące doświadczenie. Wykonujemy kolejne rzuty uczciwą monetą, aż do momentu kiedy otrzymamy sekwencję (orzeł, reszka, reszka).

1. Napisz funkcję, która wykona symulację tego doświadczenia, a następnie zwróci na wyjściu łączną liczbę rzutów monetą. *Orła oznacz jako 1, a reszkę jako 0.*
2. Powtarzając to doświadczenie 10 000 razy, oszacuj wartość oczekiwaną tej liczby rzutów.
3. Rozważ doświadczenie, w którym wykonujemy kolejne rzuty uczciwą monetą, aż do uzyskania sekwencji (orzeł, reszka, orzeł). Sprawdź, czy wartość oczekiwana liczby rzutów jest ta sama co poprzednio, czy inna.