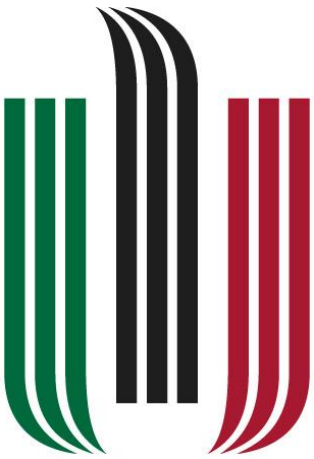
Laboratorium XI

Całkowanie Monte Carlo

Dominik Marek

10 czerwca 2024



AGH

AKADEMIA GÓRNICZO-HUTNICZA
IM. STANISŁAWA STASZICA
W KRAKOWIE

1.Zadania

Tematem zadania będzie obliczanie metodą Monte Carlo całki funkcji:

1) $x^2 + x + 1$

2) $\sqrt{1 - x^2}$

3) $\frac{1}{\sqrt{x}}$

w przedziale (0,1).

Proszę dla tych funkcji:

1. Napisać funkcję liczącą całkę metodą "hit-and-miss". Czy będzie ona dobrze działać dla funkcji $\frac{1}{\sqrt{x}}$?
2. Policzyć całkę przy użyciu napisanej funkcji. Jak zmienia się błąd wraz ze wzrostem liczby prób?
3. Policzyć wartość całki korzystając ze wzoru prostokątów dla dokładności (1e-3, 1e-4, 1e-5 i 1e-6). Porównać czas obliczenia całki metodą Monte Carlo i przy pomocy wzoru prostokątów dla tej samej dokładności, narysować wykres. Zinterpretować wyniki.

2.Rozwiązania

2.1

W metodzie hit and miss losujemy N punktów o współrzędnych z przedziałów: x - (a, b), y - (0, h), gdzie h jest pewną ustaloną przez nas wielkością. Jeżeli punkt leży poniżej wykresu to dodajemy go do zbioru końcowego. Wartość całki estymujemy jako:

$$P \cdot \frac{|N|}{|S|}$$

gdzie S to zbiór punktów, które znalazły się pod wykresem, N to zbiór wszystkich badanych punktów, zaś P jest polem prostokąta o rozmiarach $(b - a) \times h$.

```python
from typing import Callable
from math import sqrt
from random import uniform


def fun_1(x:float) -> float:
    return x**2 + x + 1


def fun_2(x:float) -> float:
    return sqrt(1-x**2)


def fun_3(x:float) -> float:
    return 1/sqrt(x)


def hit_and_miss(a:float, b:float, n:int, h:float, fun:Callable) -> float:
    S = 0
    for i in range(n):
        x = uniform(a, b)
        y = uniform(a, b)

        if y < fun(x):
            S += 1
    return ((b-a)*h)*S/n
```

Należy zauważyć, iż dla $f(x) = \frac{1}{\sqrt{x}}$, gdy x dąży do 0 to wartość tej funkcji dąży do ∞, więc ma tam asymptota, co oznacza, ze nie jesteśmy w stanie dobrze dobrać górnej granicy przedziału, z którego będziemy losować współrzędną y danego punktu. Funkcja może też działać źle dla $f(x) = \frac{1}{\sqrt{x}}$ ze względu na relatywnie duże błędy obliczeniowe pojawiające się przy obliczenia pierwiastka oraz dzieleniu. Są to operacje gubiące precyzję, podczas gdy pozostałych funkcjach nie występuje operacje obarczone takim błędem jak operacja dzielenia

2.2

Początkowo wyznaczam rzeczywiste wartości całek dla podanych funkcji, które będą potrzebne przy wyznaczaniu błędu przy zastosowaniu metody „hit-and-miss”.

$$\int_0^1 (x^2 + x + 1)\,dx = \left[\frac{x^3}{3} + \frac{x^2}{2} + x\right]_0^1 = \frac{1}{3} + \frac{1}{2} + 1 = \frac{11}{6}$$

$$\int_0^1 \sqrt{1 - x^2}\,dx = \frac{1}{2}\left(x\sqrt{1 - x^2} + cos\,sin(x)\right)\Big|_0^1 = \frac{1}{2}\left[\frac{\pi}{2}\right] \cdot = \frac{\pi}{4}$$

$$\int_0^1 \frac{1}{\sqrt{x}}dx = \left[2\sqrt{x}\right]_0^1 = 2$$

Liczba iteracji	*Błąd bezwzględny dla* $f(x) = x^2 + x + 1$	*Błąd bezwzględny dla* $f(x) = \sqrt{1 - x^2}$	*Błąd bezwzględny dla* $f(x) = \frac{1}{\sqrt{x}}$
100	0.003333	0.004602	0.0
1000	0.038667	0.003398	0.11
10000	0.002733	0.002598	0.139
100000	0.000243	0.002268	0.1011
1000000	0.000516	0.000465	0.10593
10000000	0.000607	0.000128	0.099529

Powyższe wyniki zostały otrzymane po wykonaniu kodu rozwiązania zadania pierwszego z odpowiednio przyjętymi stałymi:

```
def main() ->None:
    A:Final = 0
    B:Final = 1
    INTEGRAL_1:Final = 11/6
    INTEGRAL_2: Final = pi/4
    INTEGRAL_3: Final = 2

    num_of_interation = [10**2, 10**3,10**4, 10**5, 10**6, 10**7]

    # Maximum values of functions on [0, 1]
    H_1 = 3
    H_2 = 1
    H_3 = 10  # Arbitrary lagę value sinice fun_3 is problematic as x
approaches 0


    for n in num_of_interation:
        error_1 = round(abs(INTEGRAL_1 - hit_and_miss(A, B, n, H_1,
fun_1)), 6)
        error_2 = round(abs(INTEGRAL_2 - hit_and_miss(A, B, n, H_2,
fun_2)), 6)
        error_3 = round(abs(INTEGRAL_3 - hit_and_miss(A, B, n, H_3,
fun_3)), 6)

        print(f'n={n}, error_1={error_1}, error_2={error_2},
```

```
error_3={error_3}')


if __name__ == '__main__':
    main()
```

Wnioski:

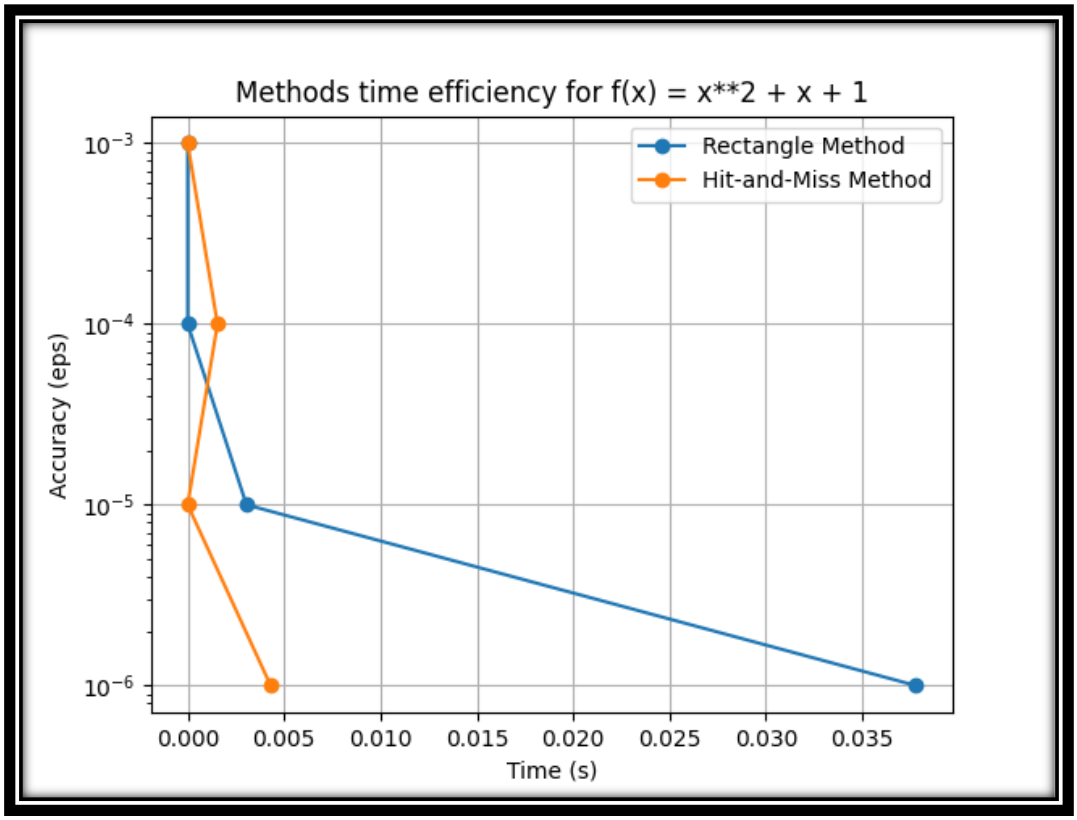
Łatwo zauważyć, że estymacja f(x) = $\frac{1}{\sqrt{x}}$ choć nie daje bardzo dalekich od prawidłowych wyników, to jednak wbrew temu do czego się przyzwyczailiśmy wzrost liczby badanych próbek nie wpływa tu na jakość przybliżenia, tj. błąd bezwzględny nie staje się mniejszy. Inaczej sprawa ma się w przypadku drugiej funkcji, czyli f(x)= $x^2 + x + 1$, gdzie widać wyraźny spadek wartości błędu bezwzględnego wraz ze wzrostem ilości badanych punktów, co jest oczywiście zjawiskiem pozytywnym. Trzeba nam jednak zwrócić uwagę na fakt, że od ilości próbek rzędu 10^5 jakość badania się stabilizuje, a nawet można zauważyć delikatny wzrost błędu. Fakt ten pokazuje, że metoda Monte Carlo choć łatwa w implementacji i jakże prosta w zrozumieniu może nie być wystarczająco dobra w sytuacjach, gdzie precyzja obliczeń jest na przysłowiową wagę złota, czyli w między innymi medycynie, lotnictwie czy nowoczesnym budownictwie.

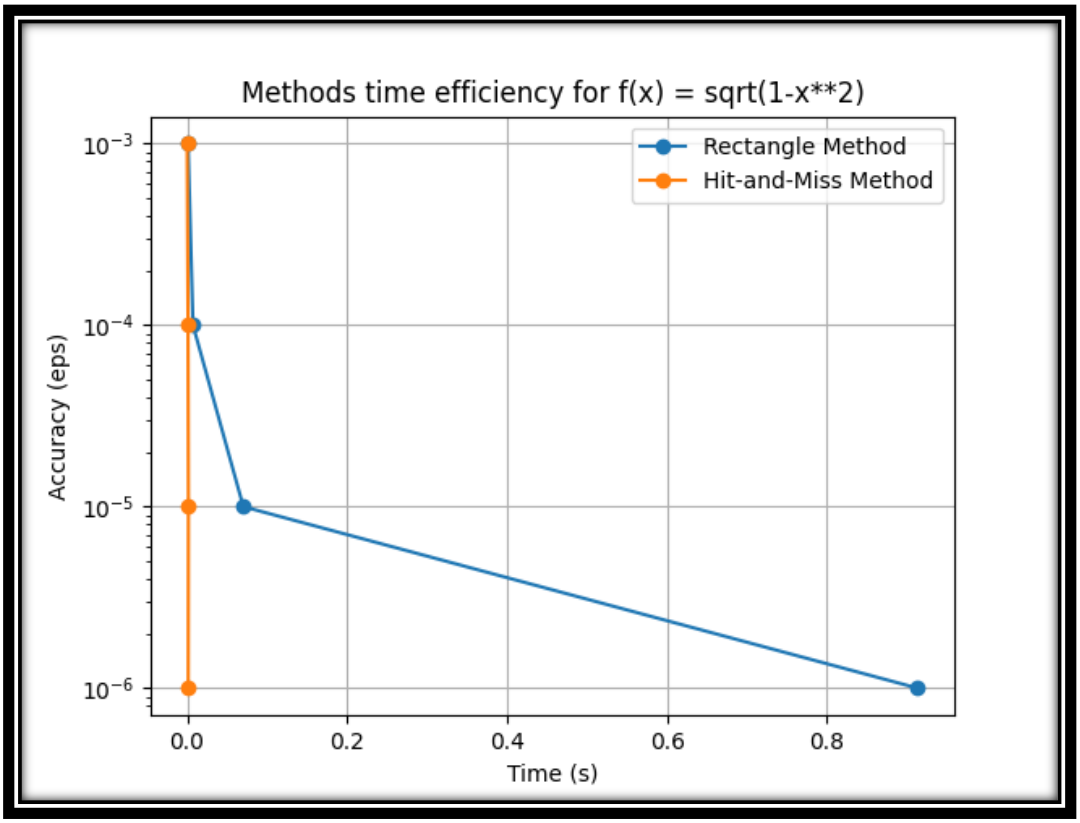
2.3

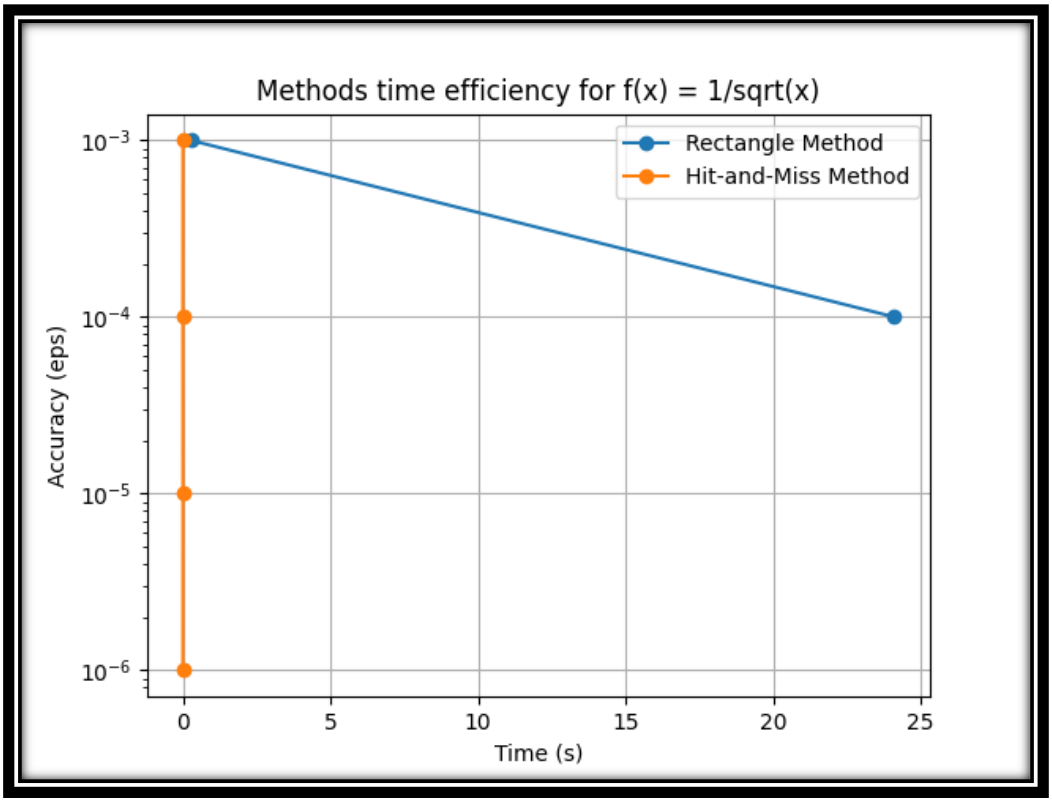
Dokładność	**Czas obliczeń dla $f(x) = x^2 + x + 1$**	
	Metoda prostokątów	**Metoda hit-and-miss**
1e-3	0.0	0.0
1e-4	0.0	0.001546
1e-5	0.003051	0.0
1e-6	0.037795	0.004306

Dokładność	**Czas obliczeń dla $f(x) = \sqrt{1 - x^2}$**	
	Metoda prostokątów	**Metoda hit-and-miss**
1e-3	0.001186	0.0
1e-4	0.007301	0.001066
1e-5	0.070255	0.001593
1e-6	0.913646	0.001328

Dokładność	**Czas obliczeń dla $f(x) = \frac{1}{\sqrt{x}}$**	
	Metoda prostokątów	**Metoda hit-and-miss**
1e-3	0.268121	0.003624
1e-4	24.111992	0.0
1e-5	-	0.003297
1e-6	-	0.001002


Methods time efficiency for f(x) = x**2 + x + 1
Rectangle Method
Hit-and-Miss Method
Accuracy (eps)
10^{-3}
10^{-4}
10^{-5}
10^{-6}
0.000
0.005
0.010
0.015
0.020
0.025
0.030
0.035
Time (s)


Methods time efficiency for f(x) = sqrt(1-x**2)
Rectangle Method
Hit-and-Miss Method
Accuracy (eps)
10^{-3}
10^{-4}
10^{-5}
10^{-6}
0.0
0.2
0.4
0.6
0.8
Time (s)



Powyższe wyniki oraz wykresy zostały otrzymane po wykonaniu następującego programu napisanego w języku Python:

```
from time import time
import numpy as np
from typing import Callable, Final
from math import sqrt, pi,inf
from random import uniform
from matplotlib import pyplot as plt


def fun_1(x:float) -> float:
    return x**2 + x + 1


def fun_2(x:float) -> float:
    return sqrt(1-x**2)


def fun_3(x:float) -> float:
    return 1/sqrt(x)


def hit_and_miss_approximate(a:float, b:float, h:float,
eps:float,exact_value:float, fun:Callable[[float], float]):
    S = 0
    P = (b-a)*h
    n = 1
    while abs(exact_value -P*S/n) > eps:
        x = uniform(a, b)
        y = uniform(0, h)
```

```
        if y < fun(x):

            S += 1
        n += 1


    return P*S/n


def rectangle_approximate(r_min:float, r_max:float, eps:float,
exact_res:float,fun:Callable):
    n = 1
    integral_approx = 0
    while not abs(integral_approx - exact_res) <= eps:
        width = (r_max - r_min) / n
        pieces = np.linspace(r_min, r_max - width, n)
        pieces += width / 2

        integral_approx = sum(fun(x) for x in pieces)*width
        n  += 1
    return integral_approx


def get_times(r_min:float, r_max:float,
h:int,accuracies,exact_res,function):
    times = [[0.0]*len(accuracies) for _ in range(2)]
    res = [[0.0]*len(accuracies) for _ in range(2)]

    for eps in accuracies:
        start_time = time()
        res[0][accuracies.index(eps)] = rectangle_approximate(r_min, r_max,
eps, exact_res, function)
        end_time = time()
        times[0][accuracies.index(eps)] = round(end_time-start_time, 6)
        start_time_1 = time()
        res[1][accuracies.index(eps)] = hit_and_miss_approximate(r_min,
r_max,h, eps, exact_res, function)
        end_time_1 = time()
        times[1][accuracies.index(eps)] = round(end_time_1 - start_time_1,
6)

    return res, times


def plot_times(accuracies, times_rec, times_mc, title):
    plt.figure()
    plt.plot(times_rec,accuracies, label="Rectangle Method", marker='o')
    plt.plot(times_mc, accuracies, label="Hit-and-Miss Method", marker='o')
    plt.yscale('log')
    plt.ylabel('Accuracy (eps)')
    plt.xlabel('Time (s)')
    plt.title(title)
    plt.legend()
    plt.grid(True)
    plt.show()


def main() -> None:
    A: Final = 0
    B: Final = 1
    INTEGRAL_1: Final = 11 / 6
```

```
    INTEGRAL_2: Final = pi / 4
    INTEGRAL_3: Final = 2


    # Maximum values of functions on [0, 1]
    H_1 = 3
    H_2 = 1
    H_3 = 10  # Arbitrary large value since fun_3 is problematic as x
approaches 0

    accuracies = [1e-3, 1e-4, 1e-5, 1e-6]


    res_1, times_1 =get_times(A, B, H_1, accuracies, INTEGRAL_1, fun_1)
    res_2, times_2 = get_times(A, B, H_2, accuracies, INTEGRAL_2, fun_2)
    res_3,times_3 =  get_times(A, B, H_3, accuracies, INTEGRAL_3, fun_3)


    plot_times(accuracies,times_1[0], times_1[1], "Methods time efficiency
for f(x) = x**2 + x + 1")
    plot_times(accuracies, times_2[0], times_2[1], "Methods time efficiency
for f(x) = sqrt(1-x**2)")
    plot_times(accuracies, times_3[0], times_3[1], "Methods time efficiency
for f(x) = 1/sqrt(x)")



if __name__ == '__main__':
    main()
```

Wnioski:

Analizując otrzymane wyniki oraz wykresy można zauważyć, iż metoda „hit-and-miss” w znaczącej liczbie przypadku jest zdecydowanie bardziej wydajna niż metoda prostokątów. Jest to szczególnie zauważalne podczas obliczania całki dla funkcji f(x)= $\frac{1}{\sqrt{x}}$, gdzie dla dokładności 1e-5 oraz 1e-6 obliczenia dla metody prostokątów musiały zostać przerwana , gdyż po czasie oczekiwania przekraczającym 10 minut metoda nie skończyła obliczeń. Ponadto, można dostrzec , iż w przypadku metody „hit-and-miss” czas obliczeń nie zawsze rośnie wraz ze wzrostem wymaganej dokładności obliczeń, dzieje się tak, gdyż w owej metodzie wzrost liczby próbek nie gwarantuje wzrostu dokładności.

3 Bibliografia

- https://www.integral-calculator.com/
- Marian Bubak PhD
- http://wazniak.mimuw.edu.pl/index.php?title=Pr_m06_lab
- https://en.wikipedia.org/wiki/Monte_Carlo_integration