Laboratorium 11

Całkowanie Monte Carlo

Bartłomiej Szubiak

04.06.2024

Tematem zadania będzie obliczanie metodą Monte Carlo całki funkcji:
1) x^2 + x + 1,
2) sqrt(1-x^2)
3) 1/sqrt(*x*)
w przedziale (0,1). Proszę dla tych funkcji:

Metoda "hit-and-miss" polega na losowaniu punktów w określonym obszarze i liczeniu, jaka część z tych punktów znajduje się pod wykresem funkcji. Wartość całki estymujemy jako:

$$P * S / N$$

Gdzie:
P – pole prostokąta o rozmiarach (b − a) × h.
S – to liczba punktów które znalazły się pod wykresem
N – to liczba wszystkich losowanych punktów

Zad1:

Napisać funkcję liczącą całkę metodą "hit-and-miss". Czy będzie ona dobrze działać dla funkcji 1/sqrt(x)?

Kod python implementujący metodę:

```
import random
import math

def hit_and_miss_integration(func, a, b, num_points, y_max):
    count_under_curve = 0

    for _ in range(num_points):
        x = random.uniform(a, b)
        y = random.uniform(0, y_max)

        if y <= func(x):
            count_under_curve += 1

    area_rectangle = (b - a) * y_max
    integral = area_rectangle * (count_under_curve / num_points)
    return integral
```

Przy funkcji $1/\sqrt{x}$ jeśli będziemy losować punkty z przedziału $x \epsilon [0,1]$ istnieje możliwość że będziemy dzielić przez 0 obliczając wartość funkcji (wylosowując punkt o współrzędnej x = 0) dodatkowo nie możemy dobrze określić wysokości prostokąta poprzez asymptotę w tym punkcie.

Zad2:

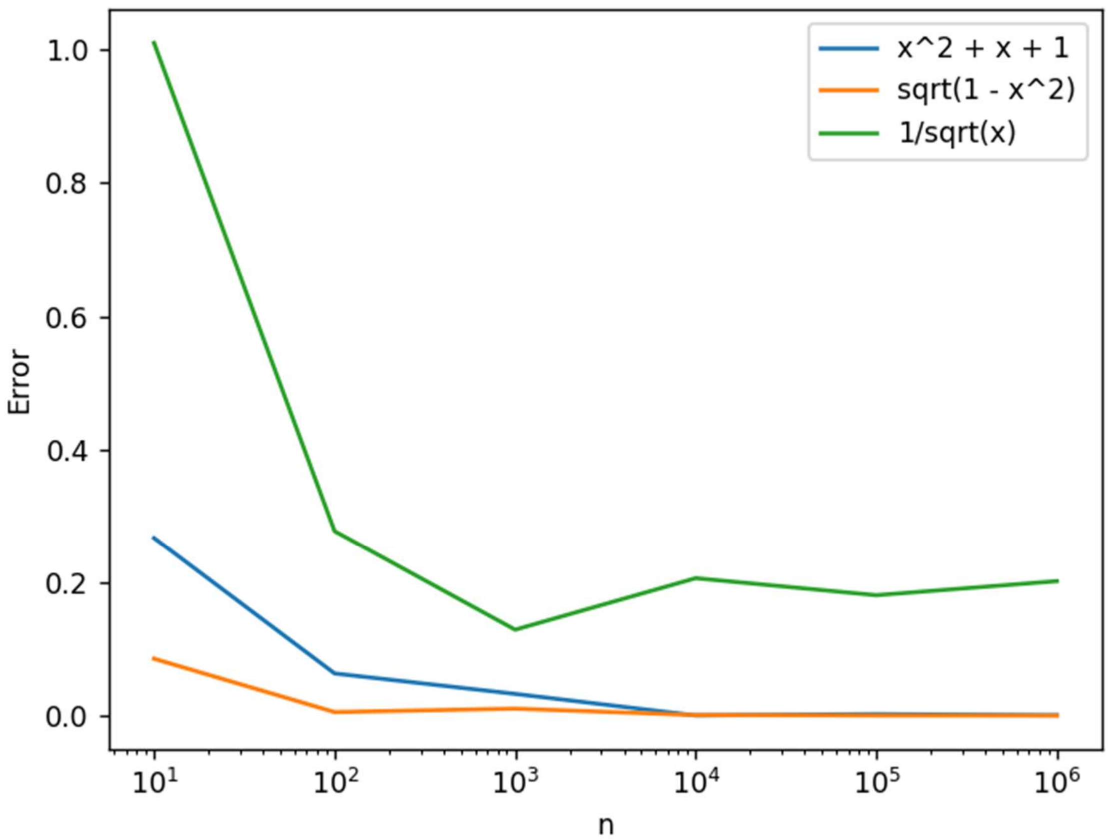
Policzyć całkę przy użyciu napisanej funkcji. Jak zmienia się błąd wraz ze wzrostem liczby prób?

Kod python:

```
print("n = ", n)
print()

def f(x): return x**2 + x + 1
res = hit_and_miss_integration(f, 0, 1, n, f(1))
print("x^2 + x + 1: ", res)
print("error: ", abs(res - 11/6))
print()

def f(x): return math.sqrt(1 - x**2)
res = hit_and_miss_integration(f, 0, 1, n, f(0))
print("sqrt(1 - x^2): ", res)
print("error: ", abs(res - math.pi/4))
print()

def f(x): return 1/math.sqrt(x)
# aby nie dzielić przez zero, zaczynamy od 0.0001
res = hit_and_miss_integration(f, 0.0001, 1, n, f(0.0001))
print("1/sqrt(x): ", res)
print("error: ", abs(res - 2))
print()
```

Wyniki programu:

```
n =  1000000

x^2 + x + 1:  1.8320159999999999
error:  0.0013173333333333925

sqrt(1 - x^2):  0.785749
error:  0.00035083660255175175

1/sqrt(x):  2.00719926
error:  0.007199260000000152
```



Rys 1: Zależność błędu bezwzględnego od ilości próbek n dla testowanych funkcji

Wnioski:

Nie da się jednoznacznie stwierdzić czy większa ilość iteracji wpływa na zmniejszenie błędu. Metoda nie jest deterministyczna wyniki mogą się różnić przy uruchomieniach. 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 bardzo ważna

Zad3:

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.

Kod python:

```
import random
import math
import time
import numpy as np
import matplotlib.pyplot as plt
```

```
def rectangle_integration(func, a, b, epsilon):
    n = int((b - a) / epsilon)
    total_area = 0.0
    for i in range(n):
        x = a + i * epsilon
        total_area += func(x) * epsilon
    return total_area

def hit_and_miss_integration(func, a, b, num_points, y_max):
    count_under_curve = 0

    for _ in range(num_points):
        x = random.uniform(a, b)
        y = random.uniform(0, y_max)
        if y <= func(x):
            count_under_curve += 1

    area_rectangle = (b - a) * y_max
    integral = area_rectangle * (count_under_curve / num_points)
    return integral

def measure_time(method, *args):
    start_time = time.time()
    result = method(*args)
    elapsed_time = time.time() - start_time
    return result, elapsed_time

def f1(x): return x**2 + x + 1
def f2(x): return math.sqrt(1 - x**2)
def f3(x): return 1/math.sqrt(x)



for integrand , y_max, f_name in zip([f1, f2, f3], [f1(1) , f2(0) , f3(0.0001)] , ["x^2 + x + 1", "sqrt(1 - x^2)", "1/sqrt(x)"]):

    # Define integration limits
    a = 0.01  # To avoid division by zero
    b = 1

    # Define accuracy levels
    epsilons = [1e-3, 1e-4, 1e-5, 1e-6]
    num_points_list = [int(1 / epsilon) for epsilon in epsilons]

    # Store results
    rect_results = []
    monte_carlo_results = []

    # Perform calculations and measure times
    for epsilon, num_points in zip(epsilons, num_points_list):
        rect_result, rect_time = measure_time(rectangle_integration, integrand, a, b, epsilon)
```

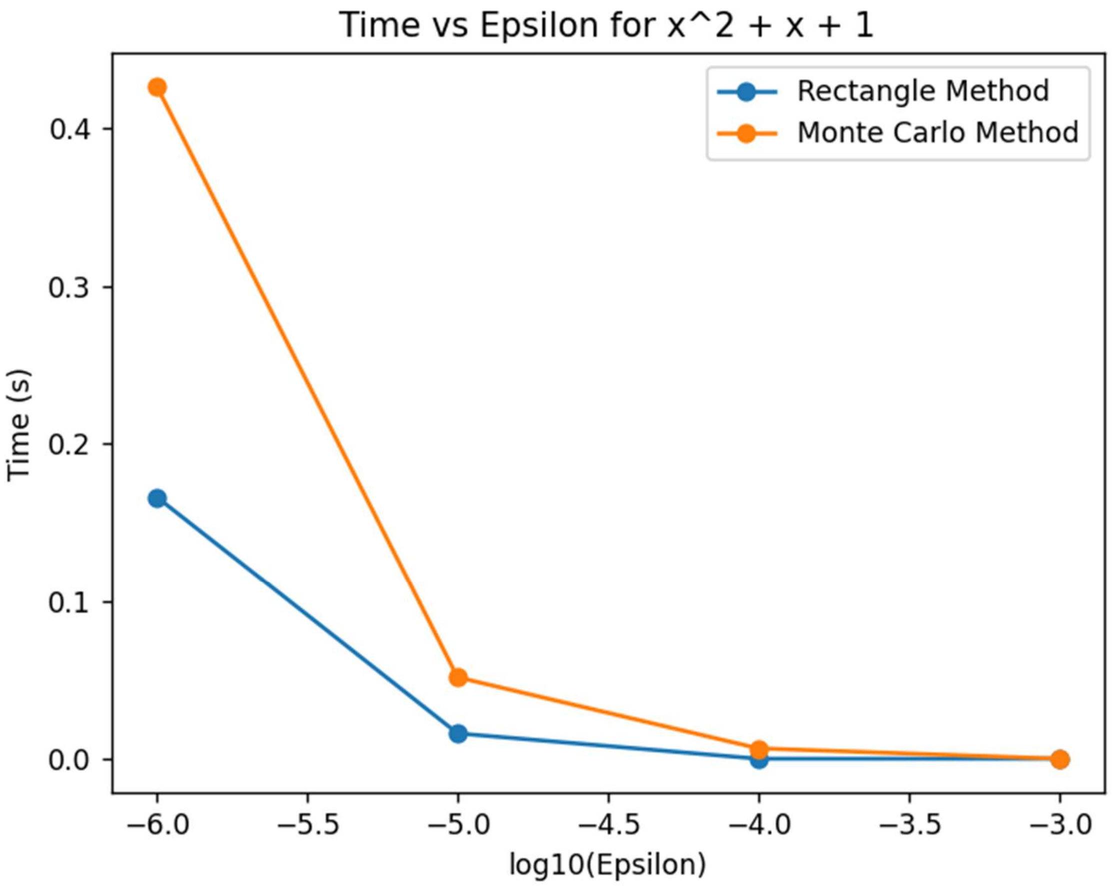
```
        monte_carlo_result, monte_carlo_time = measure_time(hit_and_miss_integration, integrand, a,
b, num_points, y_max)

        rect_results.append((epsilon, rect_result, rect_time))
        monte_carlo_results.append((epsilon, monte_carlo_result, monte_carlo_time))

    # Print results
    print("Rectangle Method Results:")
    for epsilon, result, time_taken in rect_results:
        print(f"Epsilon: {epsilon}, Integral: {result}, Time: {time_taken}s")

    print("\nMonte Carlo Method Results:")
    for epsilon, result, time_taken in monte_carlo_results:
        print(f"Epsilon: {epsilon}, Integral: {result}, Time: {time_taken}s")

    # Plot results
    epsilons_log = [math.log10(eps) for eps in epsilons]
    rect_times = [res[2] for res in rect_results]
    monte_carlo_times = [res[2] for res in monte_carlo_results]

    plt.plot(epsilons_log, rect_times, label="Rectangle Method", marker='o')
    plt.plot(epsilons_log, monte_carlo_times, label="Monte Carlo Method", marker='o')

    plt.xlabel("log10(Epsilon)")
    plt.ylabel("Time (s)")
    plt.legend()
    plt.title(f"Time vs Epsilon for {f_name}")
    plt.show()
```

Wyniki:

Dla funkcji x^2 + x + 1:

Rectangle Method Results:
Epsilon: 0.001, Integral: 1.8222882150000004, Time: 0.0s
Epsilon: 0.0001, Integral: 1.8231835066500013, Time: 0.0s
Epsilon: 1e-05, Integral: 1.8232430508164954, Time: 0.016008377075195312s
Epsilon: 1e-06, Integral: 1.8232820050501835, Time: 0.16588211059570312s

Monte Carlo Method Results:
Epsilon: 0.001, Integral: 1.8057599999999998, Time: 0.0s
Epsilon: 0.0001, Integral: 1.8191249999999999, Time: 0.006555795669555664s
Epsilon: 1e-05, Integral: 1.8185788857888578, Time: 0.05146479606628418s
Epsilon: 1e-06, Integral: 1.8220652999999998, Time: 0.4264812469482422s

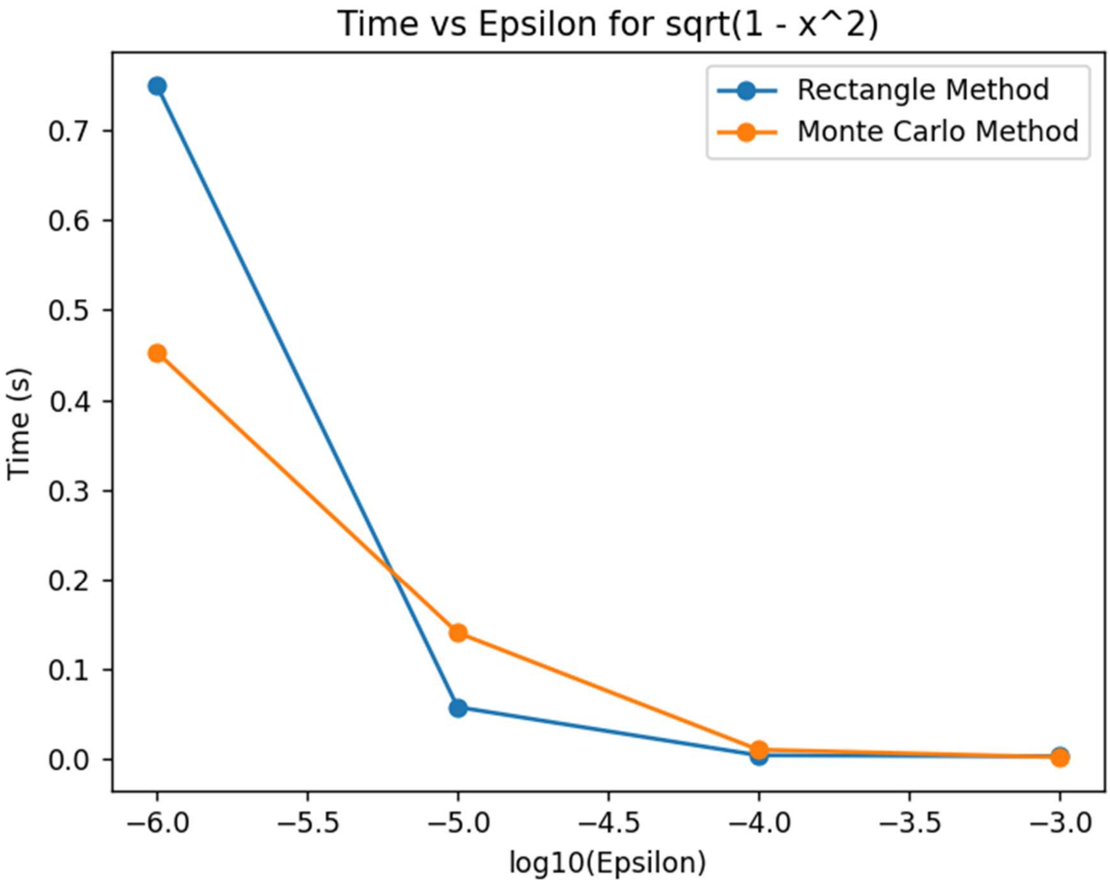


Rys 3.1 : Porównanie czasu obliczenia wartości całki z funkcji **x^2 + x + 1** przy dokładności Epsilon

Dla funkcji $\sqrt{1 - x^2}$:

Rectangle Method Results:
Epsilon: 0.001, Integral: 0.7758890092296736, Time: 0.003173828125s
Epsilon: 0.0001, Integral: 0.7754480335802675, Time: 0.00434422492980957s
Epsilon: 1e-05, Integral: 0.7754032757984833, Time: 0.05791306495666504s
Epsilon: 1e-06, Integral: 0.7753988297476143, Time: 0.7500910758972168s

Monte Carlo Method Results:
Epsilon: 0.001, Integral: 0.77715, Time: 0.002237081527709961s
Epsilon: 0.0001, Integral: 0.777447, Time: 0.010619878768920898s
Epsilon: 1e-05, Integral: 0.7764647646476465, Time: 0.14056897163391113s
Epsilon: 1e-06, Integral: 0.77546799, Time: 0.4531114101409912s

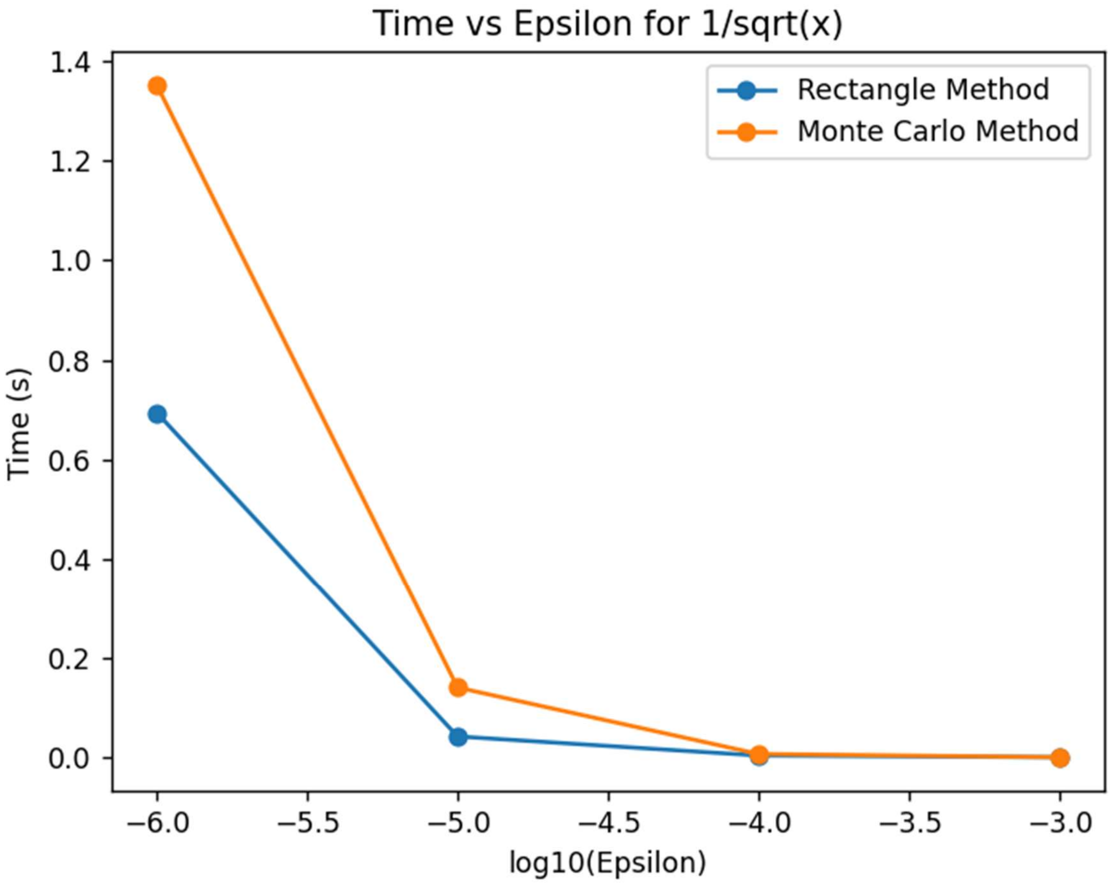


Rys 3.1 : Porównanie czasu obliczenia wartości całki z funkcji $\sqrt{1-x^2}$ przy dokładności Epsilon

Dla funkcji $1/\sqrt{x}$:

Rectangle Method Results:
Epsilon: 0.001, Integral: 1.8045415990551328, Time: 0.0s
Epsilon: 0.0001, Integral: 1.8004504162473935, Time: 0.0035071372985839844s
Epsilon: 1e-05, Integral: 1.8000350041124744, Time: 0.04179883003234863s
Epsilon: 1e-06, Integral: 1.800004500041614, Time: 0.6938409805297852s

Monte Carlo Method Results:
Epsilon: 0.001, Integral: 2.178, Time: 0.0s
Epsilon: 0.0001, Integral: 1.584, Time: 0.006615161895751953s
Epsilon: 1e-05, Integral: 1.7938979389793897, Time: 0.14021539688110352s
Epsilon: 1e-06, Integral: 1.792494, Time: 1.352302074432373s



Rys 3.1 : Porównanie czasu obliczenia wartości całki z funkcji $\mathbf{1}/\sqrt{\boldsymbol{x}}$ przy dokładności Epsilon

Wnioski:

Obie metody mają bardzo podobny czas wykonania do dokładności 10^-4. Dla dużych dokładności dla funkcji $x^2 + x + 1$ oraz dla $1/\sqrt{x}$ metoda Monte Carlo okazała się być wolniejsza.

Bibliografia:

wykład dr inż. Katarzyna Rycerz
materiały podane na zajęciach
https://pl.wikipedia.org/wiki/Metoda_Monte_Carlo