Laboratorium 12

Całkowanie metodą Monte-Carlo

Krzysztof Solecki

9 Czerwca 2023

1 Treści zadań

Tematem zadania będzie obliczanie metodami Monte Carlo całki funkcji 1) x^2+x+1, 2) $\sqrt{1-x^2}$ oraz 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? Narysować wykres tej zależności przy pomocy Gnuplota. Przydatne będzie skala logarytmiczna.
3. Policzyć wartość całki korzystając z funkcji Monte Carlo z GSL. Narysować wykres zależności błędu od ilości wywołań funkcji dla różnych metod (PLAIN, MISER, VEGAS).

2 Rozwiązania zadań

2.1 Zadanie 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:

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

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) × h.

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


def fun1(x: float) -> float:
    return x ** 2 + x + 1


def fun2(x: float) -> float:
    return sqrt(1-x**2)


def fun3(x: float) -> float:
    return 1 / sqrt(x)


def hit_and_miss(a: float, b: float, n: int, h: float, fun: Callable [[float], float]):
    S: int = 0
    for i in range(0, n):
        x = uniform(a, b)
        y = uniform(0, h)
        if y < fun(x):
            S += 1
    return ((b-a)*h)*S/n
```

Zauważmy, że 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 funkcja f(x) = $x^2 + x + 1$ jedyne operacje jakie przeprowadza to mnożenie, które nie jest aż tak obarczone błędem jak działania dzielenia czy pierwiastkowania.

2.2 Zadanie 2

Aby obliczyć błąd funkcji musimy policzyć zadane całki w sposób analityczny, a następnie od dokładnej wartości odjąć wartość wyliczoną przez naszą funkcję.

$$I_1 = \int_0^1 x^2 + x + 1 dx = \frac{x^3}{3} + \frac{x^2}{2} + x|_0^1 \approx 1.8333$$
$$\mathrm{I}_2 = \int_0^1 \sqrt{1 - x^2} dx = \frac{\pi}{4} \approx 0.78540$$
$$\mathrm{I}_3 = \int_0^1 \frac{1}{\sqrt{x}} dx = 2\sqrt{x}|_0^1 = 2$$

Przyjąłem $n = [10^2, 10^3, 10^4, 10^5, 10^6, 10^7]$. Wyniki dla zadanych obu punktów przedstawiają tabelki oraz wykresy wykonane w programie Gnuplot (jako 'error' rozumiemy błąd bezwzględny):

n	error
100	0.14666666666666672
1000	0.07766666666666677
10000	0.020066666666666677
100000	0.002126666666666832
1000000	0.000827666666666671
10000000	0.0010109666666666683

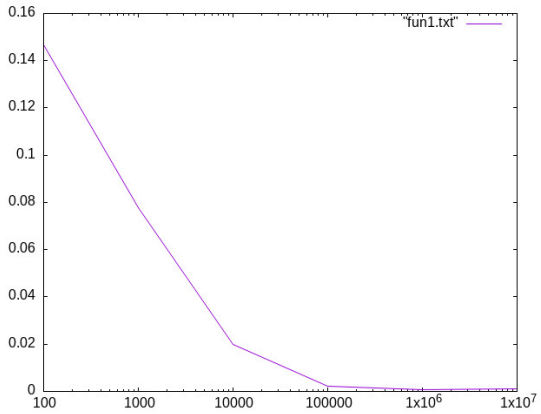
Table 1: Funkcja 1: $f(x) = x^2 + x + 1$

n	error
100	0.06459999999999999
1000	0.00539999999999996
10000	0.010099999999999998
100000	0.0005300000000000304
1000000	0.00029199999999995896
10000000	0.0001782999999999646

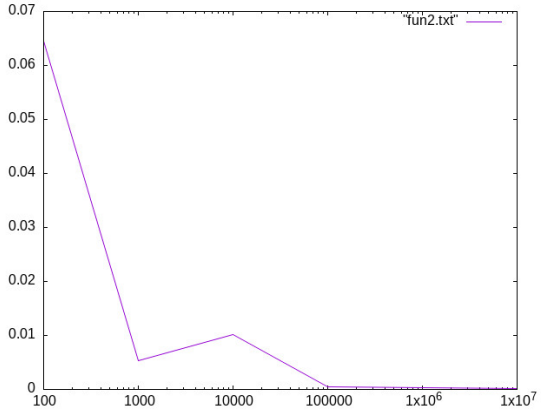
Table 2: Funkcja 2: $f(x) = \sqrt{1 - x^2}$

n	error
100	1.0
1000	0.0
10000	0.3400000000000001
100000	0.007000000000000117
1000000	0.023700000000000054
10000000	0.005109999999999948

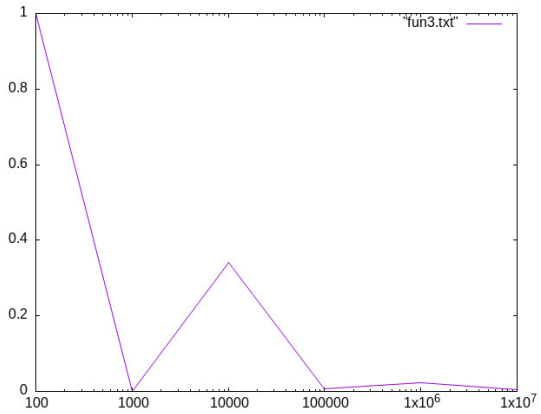
Table 3: Funkcja 3: $f(x) = \frac{1}{\sqrt{x}}$



Rys. 1: Wykres błędu bezwzględnego estymacji $\int_0^1 x^2 + x + 1 dx metod Monte - Carlo$



Rys. 2: Wykres błędu bezwzględnego estymacji $\int_0^1 \sqrt{1-x^2} dx metod Monte - Carlo$



Rys. 3: Wykres błędu bezwzględnego estymacji $\int_0^1 \frac{1}{\sqrt{x}} dx metod Monte - Carlo$

Wnioski: możemy 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 pozostałych funkcji , 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 Zadanie 3

Kod obliczający wszystkie trzy funkcje został napisany z użyciem bibliotek gsl_monte.h oraz gsl_monte_plain.h:

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <gsl/gsl_rng.h>
#include <gsl/gsl_monte_plain.h>
#include <gsl/gsl_monte.h>

double fun1(double *k, size_t dim, void *params) {
    double x = k[0];
    return x*x+x+1;
}

double fun2(double *k, size_t dim, void *params) {
    double x = k[0];
    return sqrt(1-x*x);
}

double fun3(double *k, size_t dim, void *params) {
    double x = k[0];
    return 1/sqrt(x);
}

int main(int argc, char* argv[]) {
    int n = atoi(argv[1]);
    size_t dim = 1;
    double res, err;
    double xl[1] = {0};
    double xu[1] = {1};
    gsl_rng *r = gsl_rng_alloc(gsl_rng_taus);
    gsl_monte_plain_state *s = gsl_monte_plain_alloc(dim);

    //f(x) = x^2+x+1
    gsl_monte_function F = {&fun1, dim, NULL};
    gsl_monte_plain_integrate(&F, xl, xu, dim, n, r, s, &res, &err);
```

```
    printf("x^2+x+1\n");
    printf("res = %f\n", res);
    printf("err = %f\n", err);

    //f(x) = sqrt(1-x*x)
    gsl_monte_function G = {&fun2, dim, NULL};
    gsl_monte_plain_integrate(&G, xl, xu, dim, n, r, s, &res, &err);
    printf("sqrt(1-x*x)\n");
    printf("res = %f\n", res);
    printf("err = %f\n", err);

    //f(x) = 1/sqrt(x)
    gsl_monte_function H = {&fun3, dim, NULL};
    gsl_monte_plain_integrate(&H, xl, xu, dim, n, r, s, &res, &err);
    printf("1/sqrt(x)\n");
    printf("res = %f\n", res);
    printf("err = %f\n", err);

    gsl_monte_plain_free(s);
    gsl_rng_free(r);
    return 0;
}
```

Obliczyłem jakie wyniki (wartości całki oraz błędu) dał powyższy program dla zbioru takiego jak w zadaniu 2.

n	error
100	0.063409
1000	0.018823
10000	0.005797
100000	0.001840
1000000	0.000582
10000000	0.000184

Table 4: Funkcja 1: $f(x) = x^2 + x + 1$

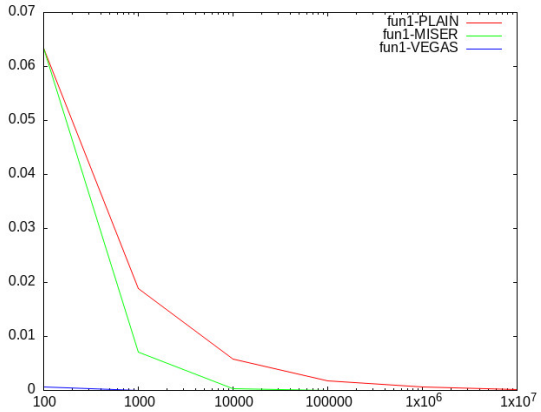
n	error
100	0.017926
1000	0.007051
10000	0.002258
100000	0.000704
1000000	0.000223
10000000	0.000071

Table 5: Funkcja 2: $f(x) = \sqrt{1 - x^2}$

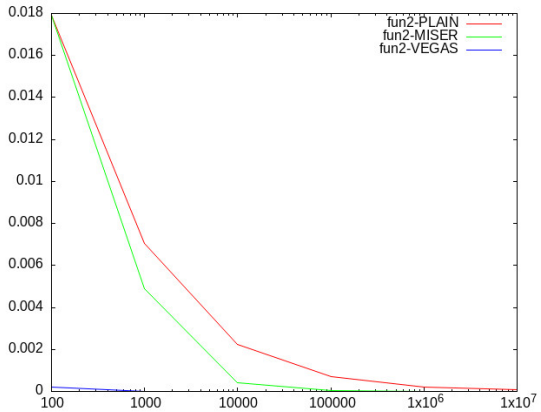
n	error
100	0.127855
1000	0.108698
10000	0.028629
100000	0.009787
1000000	0.003559
10000000	0.001214

Table 6: Funkcja 3: $f(x) = \frac{1}{\sqrt{x}}$

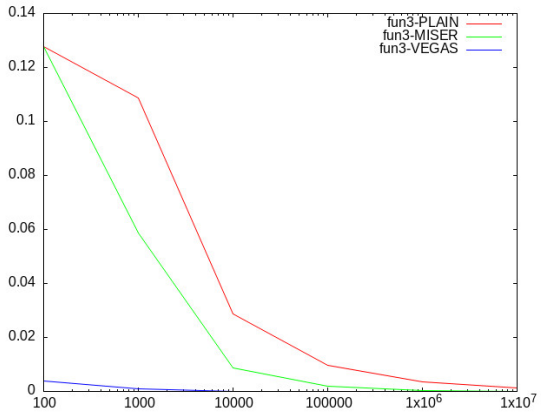
Poniżej zamieściłem wykresy porównujące skuteczność estymacji całek z każdej z trzech funkcji na zadanym przedziale dla metod: PLAIN, MISER i VEGAS:



Rys. 4: Wykres błędu bezwzględnego estymacji $\int_0^1 x^2 + x + 1 dx$ metodą Monte-Carlo metodami PLAIN, MISER i VEGAS



Rys. 5: Wykres błędu bezwzględnego estymacji $\int_0^1 \sqrt{1 - x^2} dx$ metodą Monte-Carlo metodami PLAIN, MISER i VEGAS



Rys. 6: Wykres błędu bezwzględnego estymacji $\int_0^1 \frac{1}{\sqrt{x}} dx$ metodą Monte-Carlo metodami PLAIN, MISER i VEGAS

Wnioski: jak widzimy program napisany w języku C również daje dobre rezultaty. Warto zwrócić uwagę na dosyć ciekawe rozwiązania zastosowane w funkcji gsl_monte_plain_integrate, gdzie ilość argumentów potrzebnych do jej wywołania jest bardzo duża. Staje to w ciekawym kontraście do tego co oferuje język Python, gdzie nawet samodzielna implementacja metody Monte Carlo jest bardzo łatwa. To co jednak przemawia za ponadczasowym językiem C to przede wszystkim prędkość wykonywania obliczeń, która jest wielokrotnie większa niż w Python'ie. Co do precyzji tychże działań również nie będzie nad wyraz stwierdzenie, że oba języki są pod tym względem porównywalne. W kwestii porównania metod PLAIN, MISER i VEGAS zauważalne jest, że trzecia z nich osiąga najmniejsze błędy dla danej ilości podprzedziałów, przy czym metody PLAIN i MISER można stwierdzić, że osiągają zbliżoną dokładność (z uwzględnieniem niewielkiej przewagi dla metody MISER).

3 Bibliografia

1. Katarzyna Rycerz: Materiały wykładowe z przedmiotu Metody Obliczeniowe w Nauce i Technice
2. https://www.gnu.org/software/gsl/doc/html/montecarlo.html
3. http://wazniak.mimuw.edu.pl/index.php?title=Pr_m06_lab
4. https://en.wikipedia.org/wiki/Monte_Carlo_integration