Laboratorium 9
Całkowanie Monte Carlo

Jan Rajczyk

15 maja 2021

1 Treści zadań

Tematem zadania będzie obliczanie metodami Monte Carlo całki funkcji x^2 oraz $\frac{1}{\sqrt{x}}$ w przedziale $(0, 1)$.
Proszę dla obydwu 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 podprzedziałów? Narysować wykres tej zależności przy pomocy *Gnuplot*. Przydatna będzie skala logarytmiczna.

3. Policzyć wartość całki korzystając z funkcji Monte Carlo z *GSL*.

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:

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

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

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


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


def fun2(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 $f(x) = x^2$ 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.

$$I_1 = \int_0^1 x^2\,dx = \frac{x^3}{3}\Big|_0^1 = \frac{1}{3}$$

$$I_2 = \int_0^1 \frac{1}{\sqrt{x}}\,dx = 2\sqrt{x}\Big|_0^1 = 2$$

Przyjąłem $n = [10^2, 10^3, 10^4, 10^5, 10^6, 10^7]$.

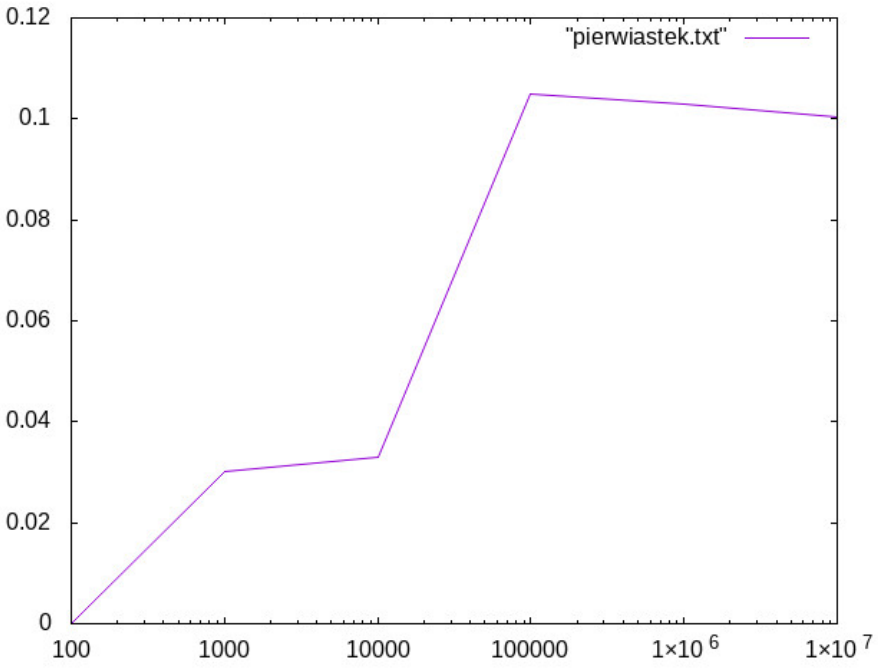
Wyniki dla zadanych obu punktów przedstawiają tabelki oraz wykresy wykonane w programie *Gnuplot* (jako $\overline{y}$ rozumiemy błąd bezwzględny):

N	$\overline{y}$
10	0.0
1000	0.03
10000	0.033
100000	0.1049
1000000	0.10305
10000000	0.1004

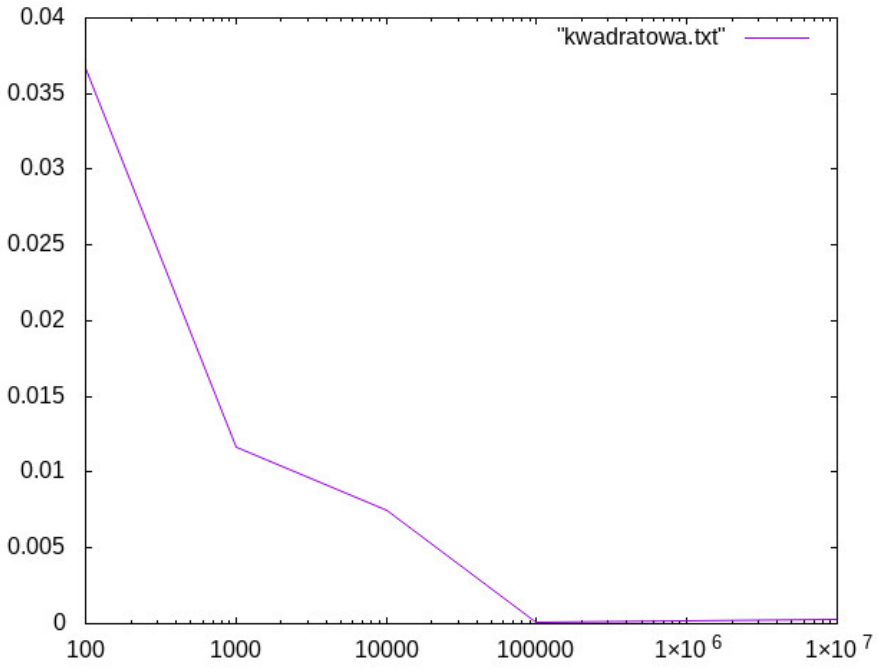
Tablica 1: Wyniki dla $f(x) = \frac{1}{\sqrt{x}}$

N	$\overline{y}$
100	0.037
1000	0.012
10000	0.0075
100000	0.00012
1000000	0.00016
10000000	0.00032

Tablica 2: Wyniki dla $f(x) = x^2$



Rysunek 1: Wykres błędu bezwzględnego estymacji $\int_0^1 \frac{1}{\sqrt{x}}\,dx$ metodą *Monte Carlo*



Rysunek 2: Wykres błędu bezwzględnego estymacji $\int_0^1 x^2\,dx$ metodą *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 drugiej funkcji, czyli $f(x) = x^2$, 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 obie 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_monte_plain.h>
#include <gsl/gsl_monte.h>
#include <gsl/gsl_rng.h>
double fun1 (double* x, size_t dim, void * p) {
    return x[0]*x[0];
}

double fun2 (double* x, size_t dim, void * p) {
    return 1/sqrt(x[0]);
}
int main(int argc, char* argv[]){
    int n = atoi(argv[1]);
    size_t dim = 1;
    gsl_monte_plain_state* monte_carlo = gsl_monte_plain_alloc(dim);
    gsl_rng *rng = gsl_rng_alloc(gsl_rng_taus);
    gsl_monte_function F;
    F.f = &fun1;
    F.dim = 1;
    F.params = NULL;
    const double xl[1] = {0};
    const double xu[1] = {1};
    double result, abserr;
    gsl_monte_plain_integrate(&F, xl, xu, 1, n, rng,
    monte_carlo, &result, &abserr);
    printf("x^2 %lf %lf\n", result, abserr);
    F.f = fun2;
    gsl_monte_plain_integrate(&F, xl, xu, 1, n, rng,
    monte_carlo, &result, &abserr);
    printf("1/sqrt(x) %lf %lf\n", result, abserr);


    gsl_monte_plain_free(monte_carlo);
    gsl_rng_free(rng);
}
```

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

N	I	$\overline{y}$
100	1.907205	0.154359
1000	1.966270	0.052612
10000	2.058049	0.050873
100000	2.006941	0.011589
1000000	2.003176	0.003074
10000000	2.000578	0.001194

Tablica 3: Wyniki dla $f(x) = \frac{1}{\sqrt{x}}$

N	I	$\overline{y}$
100	0.363798	0.032529
1000	0.348344	0.009688
10000	0.330995	0.002965
100000	0.332514	0.000942
1000000	0.332827	0.000298
10000000	0.333302	0.000094

Tablica 4: Wyniki dla $f(x) = x^2$

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.

3 Bibliografia

- `https://www.gnu.org/software/gsl/doc/html/montecarlo.html`
- `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`