Kacper Korta,

30.05.2023

Laboratorium 12

Całkowanie Monte Carlo

1. Treść zadań

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

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

a) 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 znajduję się pod wykresem to dodajemy go do naszego zbioru końcowego.
Wartość całki określamy wzorem:

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

Gdzie N to zbiór wszystkich badanych punktów, P jest polem prostokąta o rozmiarach (b – a) * h, a S to zbiór punktów które znajdują się pod wykresem.

Program napisany w pythonie:

```
from math import sqrt
from random import uniform


def fun1(x):
    return x**2 + x + 1

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

def fun3(x):
    return 1 / sqrt(x)
```

```python
def hit_and_miss(a, b, n, h, f):
    s = 0
    for i in range(n):
        x = uniform(a, b)
        y = uniform(0, h)
        if y < f(x):
            s += 1

    return ((b - a) * h) * s / n
```

Wnioski: metoda "hit-and-miss" może nie działać dobrze dla niektórych funkcji, w tym dla funkcji 1/sqrt(x). Problemem jest to, że funkcja 1/sqrt(x) ma osobliwość w punkcie x=0, a metoda "hit-and-miss" nie bierze pod uwagę struktury funkcji. W przypadku tej funkcji, większość punktów losowych wygenerowanych w obszarze całkowania może "trafić" w pobliże x=0, gdzie wartość funkcji jest bardzo duża, co zniekształca przybliżenie całki.

Zamiast metody "hit-and-miss", lepszym podejściem do całkowania funkcji 1/sqrt(x) byłoby użycie metody numerycznej, takiej jak metoda prostokątów, trapezów lub Simpsona. Te metody biorą pod uwagę strukturę funkcji i działają lepiej dla funkcji z osobliwościami lub innymi cechami, które utrudniają zastosowanie prostej metody "hit-and-miss".

b) Aby obliczyć błąd funkcji musimy najpierw policzyć wynik całki w sposób analityczny:

$$I_1 = \int_0^1 x^2 + x + 1 = \frac{11}{6}$$

$$I_2 = \sqrt{1 - x^2} = \frac{\pi}{4}$$

$$I_3 = \frac{1}{\sqrt{x}} = 2$$

N	Błąd bezwględny
10	0.8666666666666669
100	0.2666666666666683
1000	0.024333333333333318
10000	0.01766666666666672
100000	0.00030333333333332213
1000000	0.00017133333333330114

Tablica 1: Wynik dla f(x) = x^2+x+1

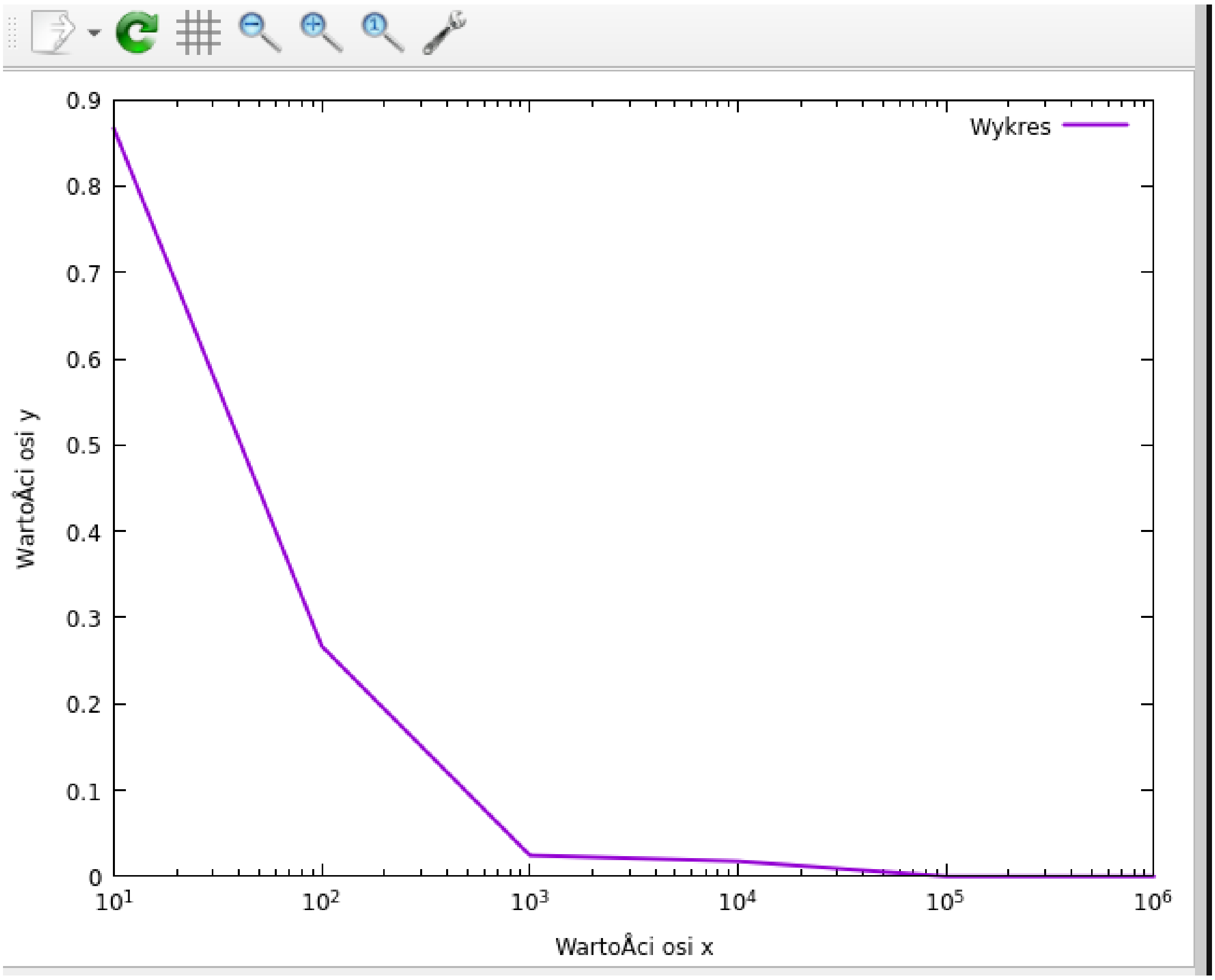
N	Błąd bezwględny
10	0.11460183660255174
100	0.015398163397448261
1000	0.00860183660255176
10000	0.007798163397448321
100000	0.0020518366025517043
1000000	1.6339744823845592e-07

Tablica 1: Wynik dla f(x) = sqrt(1-x^2)

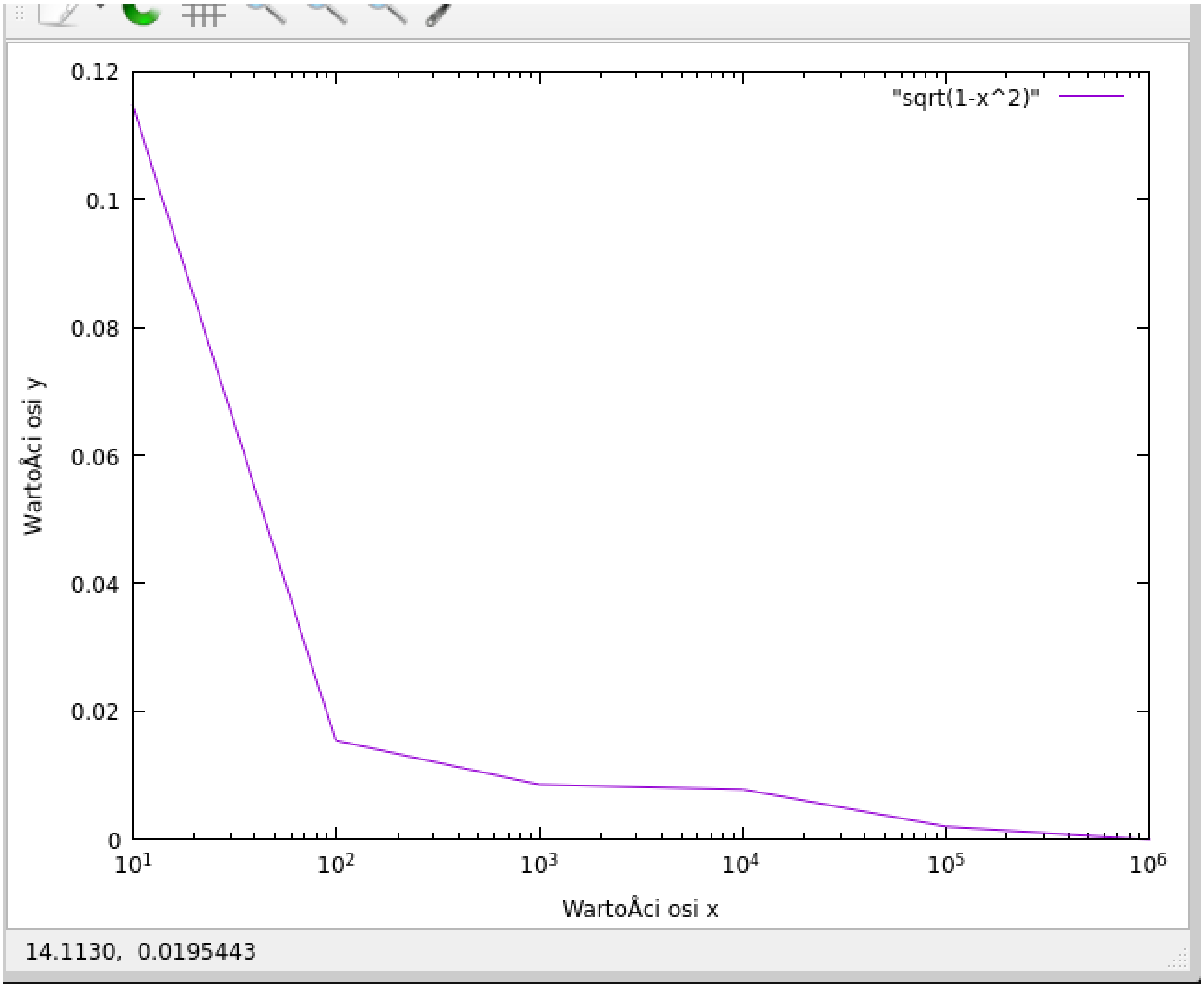
N	Błąd bezwględny
10	0.0
100	0.6000000000000001
1000	0.48
10000	0.5089999999999999
100000	0.4976799999999999
1000000	0.4995099999999999

Tablica 1: Wynik dla f(x) = 1/sqrt(x)

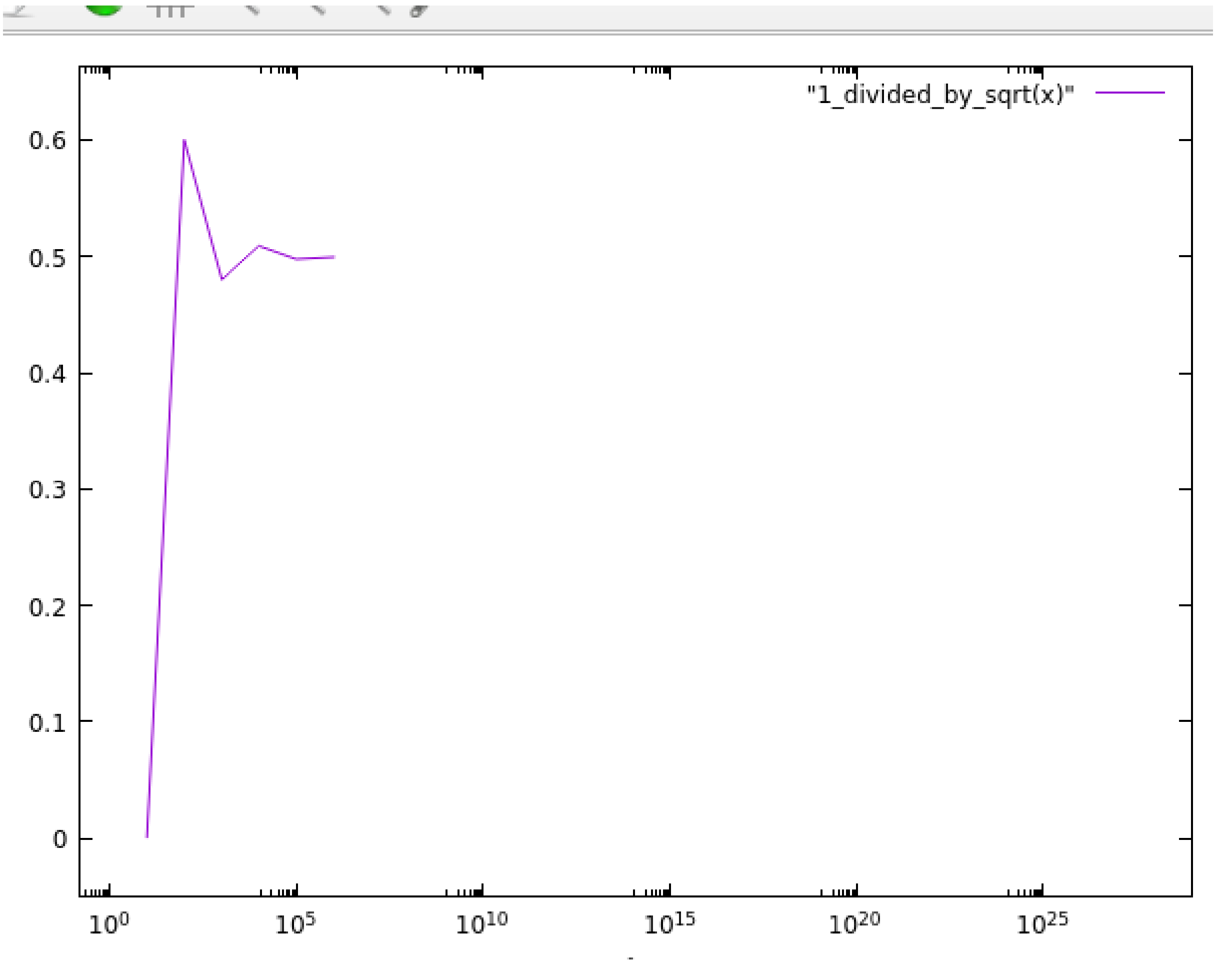
Dla funkcji x^2 + x + 1



Dla sqrt(1-x^2)



Dla funkcji 1/sqrt(x)



Wnioski:
Dla funkcji x^2+x+1:

W miarę wzrostu liczby punktów, błąd bezwzględny maleje dla wszystkich wartości N. Oznacza to, że wyniki obliczone przez algorytm są bardziej zbliżone do wartości rzeczywistej im więcej punktów jest używanych. Można stwierdzić, że zwiększanie liczby punktów prowadzi do bardziej dokładnych wyników dla tej funkcji.

Dla funkcji sqrt(1-x2):

Podobnie jak w przypadku poprzedniej funkcji, dla wszystkich wartości N błąd bezwzględny maleje wraz z wzrostem liczby punktów.

Dla funkcji 1/sqrt(x):

Dla niektórych wartości N, błąd bezwzględny jest dokładnie równy 0.0. Oznacza to, że wyniki obliczone przez algorytm są identyczne z wartościami rzeczywistymi.

Wnioskujemy, że dla tej funkcji, zwiększanie liczby punktów może prowadzić zarówno do bardziej dokładnych wyników (błąd bezwzględny bliski 0.0), jak i do wyników bardziej oddalonych od wartości rzeczywistej

c) Kod obliczający funkcje napisany w C

```c
#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(const double *x, size_t dim, void *p){
   return x[0] * x[0] + x[0] + 1;
}

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

double fun3(const 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 x1[1] = {0};
    const double xu[1] = {1};
    double result, abserr;
    gsl_monte_plain_integrate(&F, x1, xu, 1, n, rng, monte_carlo,
&result, &abserr);
    printf("Funkcja x^2 + x + 1: %f, abserr: %f\n", result, abserr);
    F.f = &fun2;
    gsl_monte_plain_integrate(&F, x1, xu, 1, n, rng, monte_carlo,
&result, &abserr);
    printf("Funkcja sqrt(1 - x^2): %f, abserr: %f\n", result,
abserr);
    F.f = &fun3;
    gsl_monte_plain_integrate(&F, x1, xu, 1, n, rng, monte_carlo,
&result, &abserr);
    printf("Funkcja 1/sqrt(x): %f, abserr: %f\n", result, abserr);

    gsl_monte_plain_free(monte_carlo);
    gsl_rng_free(rng);

    return 0;
}
```

Przy pomocy programu obliczyłem wyniki danych funkcji:

N	I	y
100	1.880072	0.063409
1000	1.860448	0.018823
10000	1.829156	0.005797
100000	1.831719	0.001840
1000000	1.832334	0.000582

Wyniki dla funkcji x^2 + x + 1

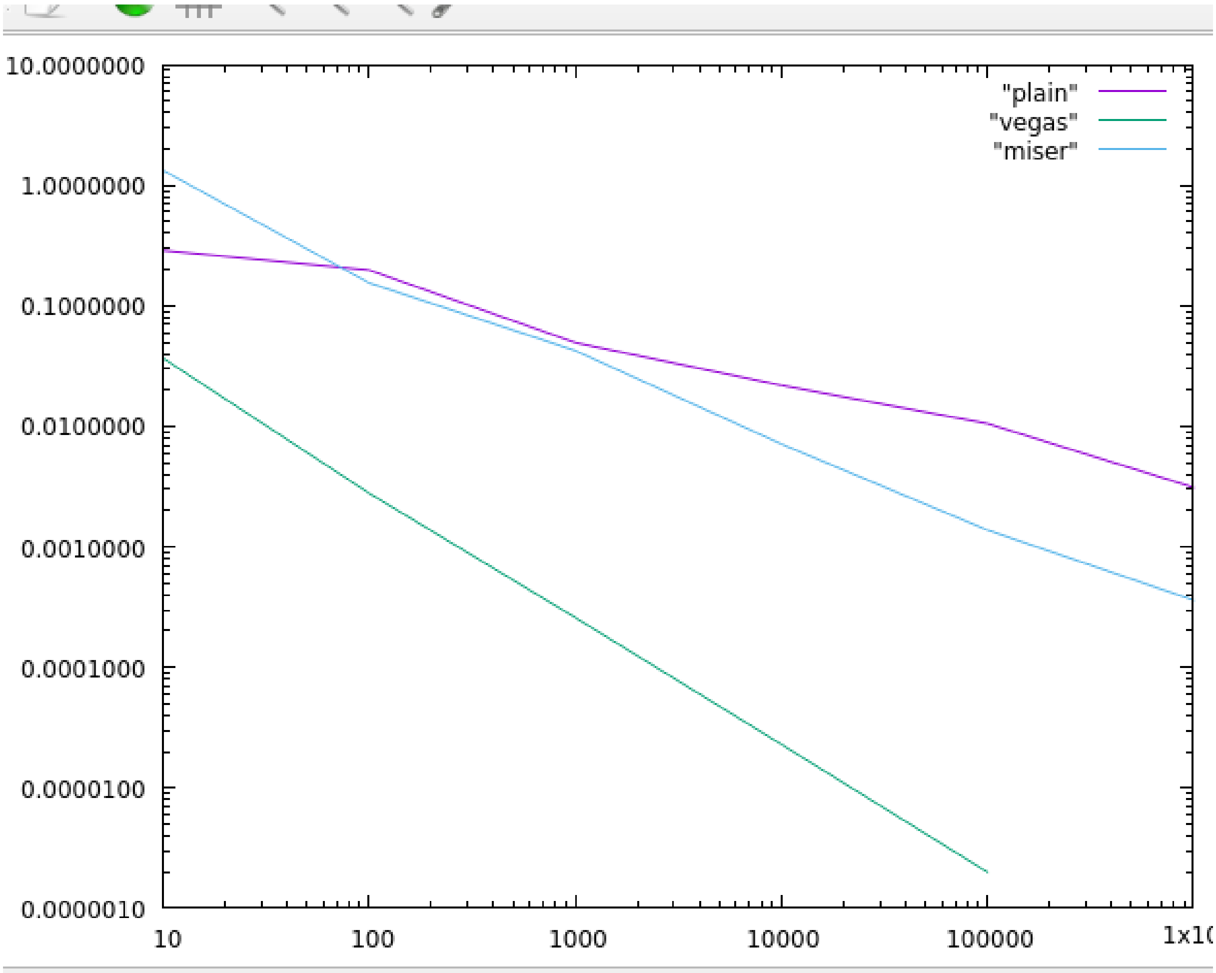
N	I	y
100	0.820559	0.017926
1000	0.790186	0.007051
10000	0.782252	0.002258
100000	0.786108	0.000704
1000000	0.785998	0.000223

Wyniki dla funkcji: sqrt(1 – x^2)

N	I	y
100	1.797653	0.127855
1000	2.115896	0.108698
10000	1.992300	0.028629
100000	1.997697	0.009787
1000000	1.998331	0.003559

Wyniki dla funkcji: 1/sqrt(x)

wykres zależności błędu od ilości wywołań funkcji dla różnych metod (PLAIN, MISER, VEGAS).



Wnioski: Dla wszystkich algorytmów, błąd bezwzględny maleje wraz z wzrostem liczby punktów. Oznacza to, że im więcej punktów jest używanych w obliczeniach, tym wyniki są bardziej zbliżone do wartości rzeczywistej. Algorytm "Vegas" wydaje się dawać najdokładniejsze wyniki, ponieważ ma najmniejszy błąd bezwzględny we wszystkich przypadkach. Oznacza to, że wartości obliczone przez ten algorytm są najbliższe wartościom rzeczywistym.

Algorytm "Miser" jest również skuteczny, ale ma nieco większy błąd bezwzględny niż "Vegas". Wciąż jednak osiąga znacznie mniejszy błąd niż "Plain".

Algorytm "Plain" ma największy błąd bezwzględny we wszystkich przypadkach, co sugeruje, że może być mniej precyzyjny od pozostałych dwóch algorytmów.

Wszystkie trzy algorytmy wykazują skalowalność w stosunku do liczby punktów. Im większa liczba punktów, tym mniejszy błąd bezwzględny. Jest to korzystne, ponieważ większa liczba punktów oznacza dokładniejsze wyniki.

3. Bibliografia

Wykład

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

http://mathworld.wolfram.com/MonteCarloMethod.html
http://www.taygeta.com/rwalks/node3.html