MOWNIT - Zestaw 5 Całkowanie Monte Carlo

Opracował: Mateusz Woś

Większość zadań została zaimplementowana w języku Python. W ostatnim zadaniu ze względu na brak wymienionych metod w standardowych bibliotekach Pythona skorzystałem z biblioteki gsl, natomiast wykresy z wygenerowanych już danych rysowałem w matplotlibie.

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

Zaimportowanie bibliotek:

```python
import numpy as np
from matplotlib import pyplot as plt
import random
```

Następnie zdefiniowałem pierwszą i drugą funkcję:

```python
def function1(x):
    return x*x
```

```python
def function2(x):
    return 1/np.sqrt(x)
```

Funkcja sprawdzająca czy punkt leży pod wykresem:

```python
def ifin(function, x, y):
    if y > 0 and y <= function(x):
        return 1
    return 0
```

Funkcja generująca losowy punkt z wyznaczonego obszaru:

```python
def randomPoint(a, b):
    return random.random() * (b-a) + a
```

Funkcja hit and miss:

```python
from math import ceil
def intVal(x_s, x_e, n, function):
    y_s, points_in = 0, 0
    y_e = ceil(max(function(x_s), function(x_e)))
    for i in range(n):
        points_in += ifin(function, randomPoint(x_s, x_e), randomPoint(y_s, y_e))

    integral_val = (points_in * ((x_e - x_s) * (y_e - y_s))) / float(n)
    return integral_val
```

Funkcja hit and miss nie poradzi sobie z drugą funkcją: $f(x) = \frac{1}{\sqrt{x}}$, ponieważ nie da się poprawnie wyznaczyć wartości zakresów wysokość w tym przypadku. Funkcja dla wartości x = 0 będzie dążyć do nieskończoności.

2. *Policzyć całkę przy użyciu napisanej funkcji. Jak zmienia się błąd wraz ze wzrostem liczby podprzedziałów? Narysować wykres tej zależności przy pomocy Gnuplota. Przydatne będzie skala logarytmiczna.*

Wartość pierwszej funkcji, używając wcześniej napisanej funkcji. Przedział [0,1] dla 100000 ilość prób trafień.

```
intVal(0, 1, 100000, function1)
```

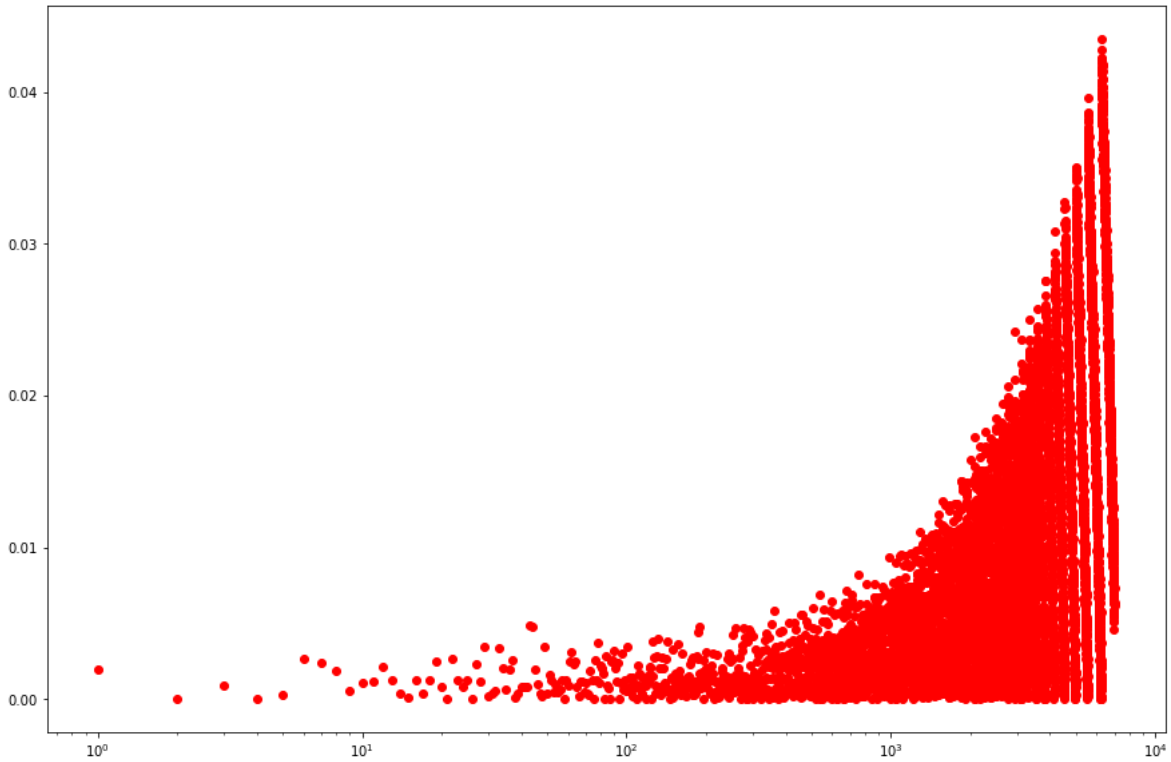
```
0.33194
```

Do obliczenia błędów napisałem, poniższą funkcję:

```
data = {}
n = 50000

def data_log(function, x_s, x_e):
    pointsIn, calka, y_s = 0, 0, 0
    y_e = ceil(max(function(x_s), function(x_e)))
    for i in range(1,7000):
        for j in range(i):
            for k in range(int(n/i)):
                pointsIn += ifin(function, randomPoint(x_s, x_e), randomPoint(y_s, y_e))

            calka += (pointsIn * ((x_e - x_s) * (y_e - y_s))) / float(n/i)
            x_s = x_e
            x_e = (1/i) + x_s
            y_e = ceil(max(function(x_s), function(x_e)))
            pointsIn = 0

        x_s, x_e, y_s = 0, (1/(i+1)), 0
        y_e = ceil(max(function(x_s), function(x_e)))
        val = abs((1/3) - calka)
        data.update({i:val})
        calka, pointsIn = 0, 0
```

Obliczenia wykonałem dla 7000 przedziałów i dla N = 50000. Nie wiem czy słusznie, ale ilość możliwych trafień zawsze dzieliłem pomiędzy podprzedziały.

Z otrzymanych danych wyrysowałem wykres:



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).*

Posłużyłem się teraz biblioteką gsl. Wygenerowałem dane z wszystkich metod do plików .txt i narysowałem wykresy w Pythonie.

Wartość pierwszej funkcji przy użyciu metod z gsl'a była bardzo podbna do tego co otrzymałem z mojej implementacji hit and miss.

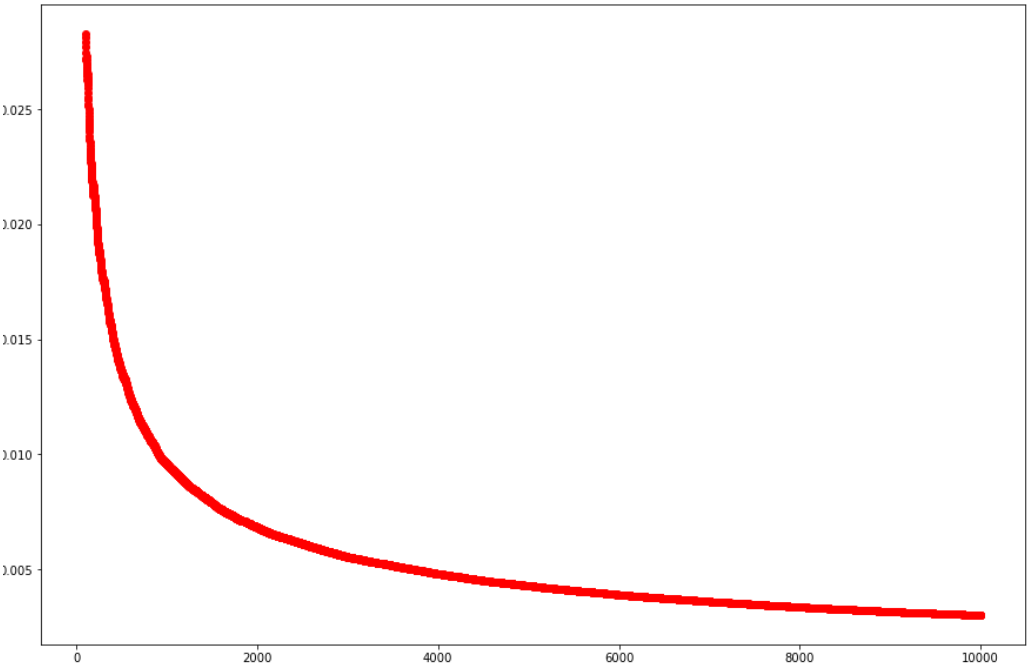
Kod generujący wartości błędów metod plain, miser i vegas:

```
for(int i=100; i<10000;i++) {
    size_t calls = i;
    gsl_rng_env_setup();
    T = gsl_rng_default;
    r = gsl_rng_alloc(T);
    {
        gsl_monte_plain_state *s = gsl_monte_plain_alloc(2);
        gsl_monte_plain_integrate(&G, xl, xu, 2, calls, r, s, &res, &err);
        gsl_monte_plain_free(s);
        fprintf(plain, "%d\t\t\t", calls);
        fprintf(plain, "%f \n", err);
    }
    {
        gsl_monte_miser_state *s = gsl_monte_miser_alloc(2);
        gsl_monte_miser_integrate(&G, xl, xu, 2, calls, r, s, &res, &err);
        gsl_monte_miser_free(s);
        fprintf(miser, "%d\t\t", calls);
        fprintf(miser, "%f \n", err);
    }
    {   gsl_monte_vegas_state *s = gsl_monte_vegas_alloc(2);
        gsl_monte_vegas_integrate(&G, xl, xu, 2, calls , r, s,&res, &err);
        fprintf(vegas, "%d\t\t", calls);
        fprintf(vegas, "%f \n", err);
        gsl_monte_vegas_free(s);
    }
}
gsl_rng_free (r);
```

Metoda plain:

```
plain_x, plain_y = [], []
with open('..//resources//plain.txt') as p:
    for line in p:
        a ,b = " ".join(line.split()).split(" ")
        plain_x.append(a)
        plain_y.append(b)
```

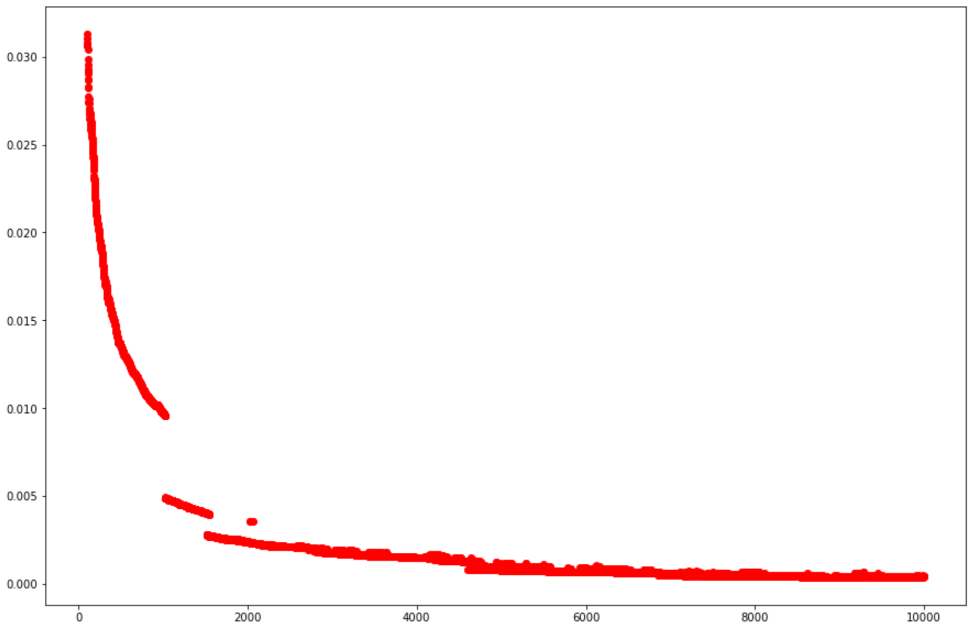
```
plt.figure(figsize=(15,10))
plt.plot(plain_x,plain_y, 'ro')
plt.show()
```



Metoda miser:

```
miser_x, miser_y = [], []
with open('..//resources//miser.txt') as p:
    for line in p:
        a ,b = " ".join(line.split()).split(" ")
        miser_x.append(a)
        miser_y.append(b)
```

```
plt.figure(figsize=(15,10))
plt.plot(miser_x,miser_y, 'ro')
plt.show()
```



Metoda vegas:

```
vegas_x, vegas_y = [], []
with open('..//resources//vegas.txt') as p:
    for line in p:
        a ,b = " ".join(line.split()).split(" ")
        vegas_x.append(a)
        vegas_y.append(b)
```

```
plt.figure(figsize=(15,10))
plt.plot(vegas_x,vegas_y, 'ro')
plt.show()
```