Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра автоматики та управління в технічних системах

Лабораторна робота №1
Моделювання методом Монте-Карло

Виконала

студентка групи ІТ-91: Перевірила:

Луцай Катерина Сокульський О. Є.

Київ 2022

Мета: Ознайомлення з методикою вирішення задач моделювання методом Монте-Карло.

1. Визначення площі фігури.

А) Використовуючи будь-який програмний математичний пакет за допомогою методу Монте-Карло обчислити площу фігури (15) ромб a=4 α=30°

Б) Результати експерименту необхідно виразити у вигляді довірчих інтервалів, що вказують величину відхилення від точного значення

2. Обчислення одномірних інтегралів.

А) Використовуючи будь-який програмний математичний пакет за допомогою методу Монте-Карло обчислити одномірний інтеграл

$$15 \left| \int_0^3 \frac{\sqrt[3]{-7x^4 - 3x + 11}}{\sqrt[4]{-2x^6 - 14x - 8}} \right.$$

Б) Обчислити точне значення інтегралу за допомогою засобів символьної математики будь-якого програмного математичного пакету та порівняйте значення.

3. Обчислення багатомірних інтегралів.

А) Використовуючи будь-який програмний математичний пакет за допомогою методу Монте-Карло обчислити багатомірний інтеграл

$$\left| 15 \right| \int_0^2 \int_0^1 7x_1^2 x_2^5 dx_1 dx_2$$

Б) Обчислити точне значення інтегралу за допомогою засобів символьної математики будь-якого програмного математичного пакету та порівняйте значення

Лістинг програми:

```
import numpy as np
import pandas as pd
from scipy import integrate

# area of romb, side = 4, angle = 30

xmin, xmax = -4 * np.sin(np.deg2rad(15)), 4 *
np.sin(np.deg2rad(15))  # x domain
ymin, ymax = -4 * np.sin(np.deg2rad(75)), 4 *
np.sin(np.deg2rad(75))  # y domain
t = 2.2622  # t-crit

n_random = [100, 200, 500, 1000, 2000, 5000, 10000, 20000]
indexes = [*range(1, 11, 1)] + ["M", "D"]
results = pd.DataFrame(columns=n_random, index=indexes)

# get n random points from domains
def random_points(n):
    x_random = (xmax - xmin) * np.random.random_sample(n) + xmin
    y_random = (ymax - ymin) * np.random.random_sample(n) + ymin
    points = np.concatenate((x_random.reshape(n, 1),
y_random.reshape(n, 1)), axis=1)
    return points.reshape(n, 2)  # array of n*2

# get number of points inside the figure
def inside_count(points):
    shift = 4 * np.sin(np.deg2rad(75))  # y shift
    coef = np.tan(np.deg2rad(75))  # angle
    inside = 0
    for p in points:
        cur_shift = -shift if p[1] < 0 else shift  # y pos/neg
check
        cur_coef = -coef if p[0] * p[1] > 0 else coef  # y and x
both pos/neg
        cur_val = cur_shift + cur_coef * p[0]  # function of one
side
        inside += int(p[1] < cur_val) if p[1] > 0 else int(p[1] >
cur_val)  # above/below side line
    return inside


print("Trust intervals for n random points:")
for n in n_random:
    for r in range(1, 11):
        results.loc[r][n] = (xmax-xmin) * (ymax-ymin) *
inside_count(random_points(n))/n  # area
    mean = results[n].mean()
```

```
    std = results[n].std()
    results.loc['M'][n] = mean
    results.loc['D'][n] = std
    print(f"{n}:\t{round(mean - np.sqrt(std) * t / np.sqrt(n), 4)}
< S < {round(mean + np.sqrt(std) * t / np.sqrt(n), 4)}")

print(results)

# integral

a, b = 0, 3
n = 1000000
x = np.random.uniform(a, b, n)  # random points
f_x = np.power(np.abs(np.abs(-7*x)**4 - 3*x + 11),
(1/3))/np.power(np.abs(np.abs(-2*x)**6 - 14*x - 8), (1/4))  # func
print(f"Monte Carlo:\t{np.mean(f_x)*(b-a)}")

res = integrate.quad(lambda x: np.power(np.abs(np.abs(-7*x)**4 -
3*x + 11), (1/3))/np.power(np.abs(np.abs(-2*x)**6 - 14*x - 8),
(1/4)), a, b)
print(f"Scipy:\t\t{res[0]}")

# double integral

a, b = 0, 2
c, d = 0, 1
n = 10000000
x, y = np.random.uniform(c, d, n), np.random.uniform(a, b, n)  #
random points
f_xy = 7 * np.power(x, 2) * np.power(y, 5)  # func
print(f"Monte Carlo:\t{np.mean(f_xy)*(b-a)*(d-c)}")

res = integrate.dblquad(lambda x, y: 7 * np.power(x, 2) *
np.power(y, 5), a, b, c, d)
print(f"Scipy:\t\t{res[0]}")
```

Результат:

```
Trust intervals for n random points:
100:    7.9776 < S < 8.3104
200:    7.6765 < S < 7.9075
500:    7.7983 < S < 7.9329
1000:   8.0181 < S < 8.0843
2000:   7.957 < S < 7.9982
5000:   8.0085 < S < 8.0273
10000:  8.0128 < S < 8.0243
20000:  7.97 < S < 7.9771
```

	100	200	500	...	5000	10000	20000
1	8.48	7.36	8.352	...	8.0288	8.0304	7.9232
2	7.84	7.92	7.552	...	7.9808	8.048	8.0112
3	7.68	7.6	8.0	...	8.0608	7.9616	7.9128
4	7.52	8.32	8.224	...	8.048	8.0224	7.9912
5	8.16	7.84	6.912	...	8.0864	8.0	7.9352
6	8.48	8.32	7.84	...	7.9744	7.992	7.9656
7	7.68	6.72	7.712	...	8.0352	8.104	7.9488
8	9.28	7.68	7.68	...	8.1568	8.136	7.9608
9	8.48	7.68	7.968	...	7.8304	7.9536	8.0752
10	7.84	8.48	8.416	...	7.9776	7.9376	8.0112
M	8.144	7.792	7.8656	...	8.01792	8.01856	7.97352
D	0.541011	0.521468	0.442583	...	0.086409	0.064276	0.049457

```
Monte Carlo:    12.174645816321108
Scipy:      12.167463543418773
Monte Carlo:    24.878665152598664
Scipy:      24.88888888888889
```

Висновки: було використано метод Монте-Карло для визначення площі ромба, інтегралу та подвійного інтегралу засобами мови Python.