МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

Кафедра компьютерных технологий и систем

ИМИТАЦИОННОЕ И СТАТИСТИЧЕСКОЕ МОДЕЛИРОВАНИЕ

ОТЧЁТ

ПО ЛАБОРАТОРНОЙ РАБОТЕ № 3

Савицкой Елизаветы Дмитриевны

студентки 4 курса, 4 группы

Минск, 2021

Задание №7, пункт 3

Задание

Вычислить интеграл $\boldsymbol{I = \int_0^4 \frac{sin(x)}{x} dx}$ по методу Монте-Карло, используя простейший метод и метод симметризации подынтегральной функции.

Вычисление определенного интеграла методом Монте-Карло

Пусть требуется вычислить интеграл $I_0 = \int_a^b f(x)dx$ по конечному интервалу $a < x < b$. Рассмотрим случайную величину ξ, равномерно распределенную в этом интервале, и величину $Z = (b - a)f(x)$. Так как $MZ = I_0$, то простейший метод Монте-Карло приводит к оценке интеграла $\theta_N = \frac{b-a}{N}\sum_{i=0}^{N} f(\xi_i)$, где $\xi_1, \dots, \xi_N$ – независимые значения ξ. Рассмотрим теперь симметризованную функцию $f^{(1)}(x) = \frac{1}{2}[f(x) + f(a + b - x)]$, интеграл который по-прежнему равен I_0, и пусть $Z^{(1)} = (a - b)f^{(1)}(\xi)$. Ввиду того, что $MZ^{(1)} = I_0$, можно записать симметризованную оценку интеграла $\theta_N^{(1)} = \frac{b-a}{2N}\sum_{i=1}^{N}[f(\xi_i) + f(a + b - \xi_i)]$.

Реализация алгоритма

```
lower_limit = 0.0  # нижний предел интергрирования
upper_limit = 4.0  # верхний предел интергрирования

number_of_iterations = [100, 1000, 10000, 10000, 100000]  # чило итераций
number_of_experiments = 5  # число экспериментов для каждого числа итераций

monte_carlo_results = {}  # результаты полученные при различном числе итераций методом Монте-Карло
symmetric_function_results = {}  # результаты полученные при различном числе итераций симметризации
# подынтегральной функции
```

Вычисление значения подынтегральной функции.

```
def integrand_function(value):  # вычисление подынтегральной функции
    return mpf(np.sin(value) / value)
```

Вычисление значения симметризованной подынтегральной функции.

```
# вычисление симметризованной подынтегральной функции
def symmetric_integrand_function(value, lower_limit_, upper_limit_):
    symmetric_value = lower_limit_ + upper_limit_ - value
    return mpf(mpf(1/2) * (mpf(np.sin(value) / value) + mpf(np.sin(symmetric_value) / symmetric_value)))
```

Вычисление определенного интеграла методом Монте-Карло, используя простейший метод и метод симметризации подынтегральной функции.

```
# вычисление определенного интеграла методом Монте-Карло
def monte_carlo_and_symmetric_i_function_methods(results_m, results_s, iterations, experiments, lower_lim, upper_lim):
    for iter_number_counter in range(len(iterations)):  # для каждого числа итераций проводим 5 экспериметов
        monte_carlo_experiments_results = []  # результаты пяти экспериметов для метода Монте-Карло
        symmetric_function_experiments_results = []  # результаты пяти экспериметов для метода симметризации
        # подынтегральной функции
        for experiment in range(experiments):
            integral_monte_carlo = mpf(0.0)
```

```python
            integral_symmetric_function = mpf(0.0)
            for counter in range(iterations[iter_number_counter]):
                value = lower_lim + (upper_lim - lower_lim) * random.uniform()  # произвольная случайная величина
                integral_monte_carlo += mpf(integrand_function(value))
                integral_symmetric_function += mpf(symmetric_integrand_function(value, lower_lim, upper_lim))
            monte_carlo_experiments_results.append((upper_lim - lower_lim) / mpf(float(iterations[iter_number_counter]))
                                                   * integral_monte_carlo)
            symmetric_function_experiments_results.append((upper_lim - lower_lim) /
                                                          mpf(float(iterations[iter_number_counter]))
                                                          * integral_symmetric_function)
        results_m[iterations[iter_number_counter]] = monte_carlo_experiments_results
        results_s[iterations[iter_number_counter]] = symmetric_function_experiments_results

    results_m_table = pd.DataFrame.from_dict(results_m)
    results_s_table = pd.DataFrame.from_dict(results_s)
    print(results_m_table)
    print(results_s_table)


monte_carlo_and_symmetric_i_function_methods(monte_carlo_results, symmetric_function_results, number_of_iterations,
                                             number_of_experiments, lower_limit, upper_limit)
```

Результаты

Результаты полученные по методу Монте-Карло, используя простейший метод.

	100	1000	10000	100000
0	1.8814187363644	1.81910096006267	1.76543138609309	1.76920808059835
1	1.67501598524961	1.70208953626256	1.72976347998091	1.75120811467847
2	1.90612932975528	1.75844348823162	1.7600421583726	1.75084099445627
3	1.64067855738449	1.74276447724759	1.76966504837534	1.75809537769676
4	2.02331453098253	1.7866351108014	1.7706202223293	1.75843207327694

Результаты, полученные используя симметризацию подынтегральной функции.

	100	1000	10000	100000
0	1.75724192061251	1.75867810442802	1.75835601891941	1.75819472793032
1	1.75679158848215	1.76039882872492	1.7588825176721	1.75840957187377
2	1.76130704989091	1.75870700363036	1.75856279096617	1.75815062571566
3	1.75226585125657	1.76027068755497	1.75736383187602	1.75786632662025
4	1.74672335879347	1.75738183935619	1.75864082471418	1.75821057640303

Для анализа точности вычислений я также нашла значение интеграла с помощью WolframAlpha.

$$\int_0^4 \frac{\sin(x)}{x}\,dx = \mathrm{Si}(4) \approx 1.7582$$

Выводы

Анализируя данные полученные, проведением пяти экспериментов для различного числа итераций (100, 1000, 10000, 100000), можно заключить, что метод симметризации функции дает более точный результат, чем простейший метод Монте-Карло, так как уже при числе итераций $n = 100$ он даёт точность вычислений до сотых, при том что простейший метод Монте-Карло достигает её лишь при $n = 1000$. Точность увеличивается с увеличением числа итераций, наиболее близкий к полученному при помощи WolframAlpha результат был достигнут методом симметризации при $n = 1000$.