**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный университет**

**Факультет прикладной математики-процессов управления**

**Кафедра “фундаментальная информатика и информационные технологии”**

**отчет**

**по лабораторной работе №4**

**по дисциплине «Алгоритмы и структуры данных»**

**на тему «Приближенное вычисление площади фигуры методом Монте-Карло»**

**Вариант 4**

|  |  |  |
| --- | --- | --- |
| Студент гр. 22Б15-пу |  | Добренкова Л.С. |
| Преподаватель |  | Дик А.Г. |

**Санкт-Петербург**

**2024 г**

Оглавление

[Цель работы 3](#__RefHeading___Toc9662_524586906)

[Задачи 3](#__RefHeading___Toc9664_524586906)

[Теоретическая часть 3](#__RefHeading___Toc9666_524586906)

[Описание алгоритма 4](#__RefHeading___Toc9668_524586906)

[Описание программы 6](#__RefHeading___Toc9670_524586906)

[Рекомендации для пользователя 8](#__RefHeading___Toc9672_524586906)

[Контрольный пример 9](#__RefHeading___Toc8166_812848710)

[Рекомендации для программиста 9](#__RefHeading___Toc9676_524586906)

[Анализ результатов работы алгоритма 9](#__RefHeading___Toc9680_524586906)

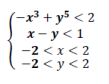
[Вывод 13](#__RefHeading___Toc9682_524586906)

[Листинг 13](#__RefHeading___Toc8168_812848710)

# ****Цель работы****

Изучение метода Монте-Карло (метода статистических испытаний) на примере вычисления площади фигуры.

# ****Задачи****

1. Составить и отладить программу определения площади фигуры методом Монте-Карло в соответствии с индивидуальным заданием.
2. Вычислить методом Монте-Карло определенный интеграл. Сравнить результат со значением, полученным аналитическим путем при значениях N=10, 100, 1000, 10000, 100000, 1000000. Выразите относительную погрешность метода Монте-Карло при каждом значении N.

# ****Теоретическая часть****

Применим метод статических испытаний или метод Монте-Карло к задаче вычисления площади геометрической фигуры на плоскости.

Метод заключается в следующем. Поместим данную фигуру в квадрат и будем наугад бросать точки в этот квадрат. Будем исходить из того, что чем больше площадь фигуры, тем чаще в нее будут попадать точки. Таким образом, при большом числе N точек, наугад выбранных внутри квадрата, доля точек, содержащихся в данной фигуре k, приближенно равна отношению площади этой фигуры и площади квадрата:

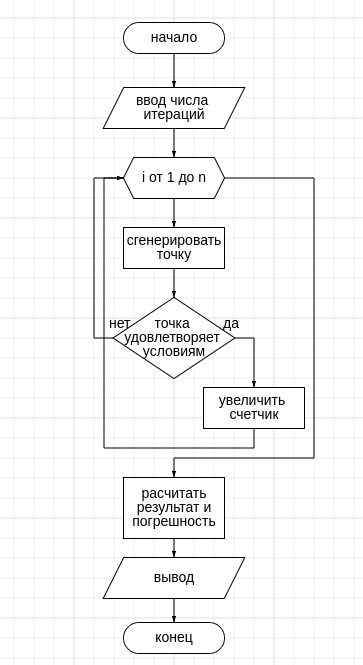
Если площадь квадрата равна S0 (4х4=16 отрезки [-2;2]) и в результате N испытаний, из которых при K исходах случайные точки оказались внутри фигуры, то площадь фигуры S будет определяться выражением:

Относительная погрешность метода Монте-Карло определяться выражением:

𝜎 =

# ****Описание алгоритма****

1. Генерируем случайные числа x и y равномерно распределенные на отрезке [-2;2] Это будут координаты случайной точки в квадрате, в которую заключена фигура, площадь которой требуется найти.
2. Проверяем принадлежность точки к исследуемой фигуре. Если попадания нет, т.е. не выполняется хотя бы одно из неравенств системы, то переходим к пункту 1 и генерируем координаты новой точки.
3. Пункты 1 и 2 следует повторить в цикле достаточно большое число N раз. От этого, в конечном итоге, зависит точность вычислений. После проведения N повторов площадь фигуры найдем по формуле

Рисунок 4.1. Основной алгоритм программы

# ****Описание программы****

Программа реализована при помощи языка python 3.8.

Использованные модули: tkinter, random.

В таблицах 5.1, 5.2 представлены описания функций и методов программы.

Таблица 5.1. Описание методов класса App

|  |  |  |
| --- | --- | --- |
| **Метод** | **описание** | **Параметры** |
| create\_widgets | Инициализация gui | self |
| calculate\_area | Запуск метода Монте-Карло | self |
| calculate\_integral | Запуск метода Монте-Карло | self |
| get\_count | Парсинг количества итераций | self |

Таблица 5.2. Описание методов lib

|  |  |  |
| --- | --- | --- |
| **Метод** | **Описание** | **Параметры** |
| monte\_carlo\_area | Расчет площади | N: int |
| monte\_carlo\_integral | Расчет интеграла | N: int |
| area\_condition | Проверка принадлежности точки фигуре | x: int,  y: int |
| f | Функция под интегралом | x: int |
| int\_f | Первообразная | x: int |

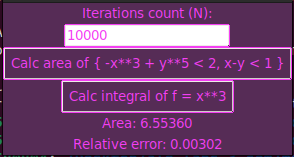
# ****Рекомендации для пользователя****

Для запуска программы необходим интерпретатор python 3.8, пакетный менеджер pip. Необходимо установить библиотеку tkinter.

Системные требования 200мб ОЗУ, 1мб ПЗУ.

Запуск файла main.py выдаст интуитивно понятный графический интерфейс. Нужно указать число итераций и нажать на соответствующую кнопку.

# Контрольный пример

Рисунок 7.1. Пример работы программы

# ****Рекомендации для программиста****

Для внесения изменений необходима ide для python. Код доступен по ссылке: <https://github.com/v131v/alg_labs_3_sem/tree/main/monte_carlo>

# Сравнительный анализ

*Таблица 9.1. Сравнительный анализ по количеству итераций*

|  |  |  |
| --- | --- | --- |
| Количество  итераций | Значение интеграла | Относительная  погрешность |
| 10 | 6.3 | 0.575 |
| 100 | 4.09 | 0.024 |
| 1000 | 3.911 | 0.022 |
| 10000 | 4.06 | 0.017 |
| 100000 | 3.987 | 0.003 |
| 1000000 | 4.003 | 0.0009 |

С увеличением количества итераций точность в среднем растет.

# ****Вывод****

В рамках данной работы был исследован метод Монте-Карло и способы его применения для вычисления площадей и определенных интегралов. Реализована программа, которая, используя этот алгоритм вычисляет площадь заданной фигуры и значение определенного интеграла.

# Листинг

Файл main.py

import tkinter as tk

from tkinter import ttk

from lib import \*

class App(tk.Frame):

def \_\_init\_\_(self, master=None):

super().\_\_init\_\_(master)

self.master = master

self.pack(fill="both", expand=True)

self.create\_widgets()

def create\_widgets(self):

light = "#f33ff3"

dark = "#562c56"

self.light = light

self.dark = dark

styleBtn = ttk.Style()

styleBtn.theme\_use("clam")

styleBtn = styleBtn.configure(

"TButton", foreground=light, background=dark, bordercolor=light

)

styleEntry = ttk.Style()

styleEntry.theme\_use("clam")

styleEntry = styleEntry.configure(

"TEntry", foreground=light, background=dark, bordercolor=light

)

styleLabel = ttk.Style()

styleLabel.theme\_use("clam")

styleLabel = styleLabel.configure("TLabel", foreground=light, background=dark)

self.configure(bg=dark)

ttk.Label(self, text="Iterations count (N):").pack(expand=True)

self.entry\_n = ttk.Entry(self)

self.entry\_n.pack(expand=True)

self.calculate\_button = ttk.Button(

self,

text="Calc area of { -x\*\*3 + y\*\*5 < 2, x-y < 1 }",

command=self.calculate\_area,

)

self.calculate\_button.pack(expand=True)

self.calculate\_button = ttk.Button(

self, text="Calc integral of f = x\*\*3", command=self.calculate\_integral

)

self.calculate\_button.pack(expand=True)

self.result\_label = ttk.Label(self, text="")

self.result\_label.pack(expand=True)

self.error\_label = ttk.Label(self, text="")

self.error\_label.pack(expand=True)

def calculate\_area(self):

n = self.get\_count()

result, relative\_error = monte\_carlo\_area(n)

self.result\_label.config(text=f"Area: {result:.5f}")

self.error\_label.config(text=f"Relative error: {relative\_error:.5f}")

def calculate\_integral(self):

n = self.get\_count()

result, relative\_error = monte\_carlo\_integral(n)

self.result\_label.config(text=f"Integral: {result:.5f}")

self.error\_label.config(text=f"Relative error: {relative\_error:.5f}")

def get\_count(self):

try:

n = int(self.entry\_n.get())

if n < 0:

raise ValueError()

return n

except ValueError:

self.result\_label.config(text="Error: enter integer count > 0")

if \_\_name\_\_ == "\_\_main\_\_":

root = tk.Tk()

app = App(root)

root.mainloop()

Файл lib.py

from random import uniform

def monte\_carlo\_area(n):

s0 = 4 \* 4

s\_t = 6.57343

cnt = 0

for \_ in range(n):

x = uniform(-2, 2)

y = uniform(-2, 2)

if area\_condition(x, y):

cnt += 1

result = (cnt / n) \* s0

relative\_error = abs(result - s\_t) / s\_t

return (result, relative\_error)

def area\_condition(x, y):

return -(x\*\*3) + y\*\*5 < 2 and x - y < 1

def monte\_carlo\_integral(n):

a = 0

b = 2

analytic\_result = int\_f(b) - int\_f(a)

integral\_sum = 0

for \_ in range(n):

x = uniform(a, b)

integral\_sum += f(x)

result = (integral\_sum / n) \* (b - a)

relative\_error = abs(result - analytic\_result) / analytic\_result

return (result, relative\_error)

def f(x):

return x\*\*3

def int\_f(x):

return x\*\*4 / 4