

Министерство науки и высшего образования Российской Федерации  
Федеральное государственное автономное образовательное учреждение высшего образования «Уральский федеральный университет имени первого Президента России Б.Н. Ельцина» (УрФУ)

Физико-технологический институт

Кафедра «Технической физики»

Оценка

Преподаватель

Кашин И.В.

**ЧИСЛЕННЫЙ РАСЧЕТ ОБЪЕМА СЛОЖНОГО ТЕЛА**

ОТЧЕТ

по лабораторной работе №2

|  |  |  |
| --- | --- | --- |
| Студент | Черняков Матвей Сергеевич | ФИО студента |

|  |
| --- |
| Специальность (направление подготовки) |
| 09.03.02 Информационные системы и технологии | |

|  |  |
| --- | --- |
| Группа | Фт-420008 |

Екатеринбург

2025

**СОДЕРЖАНИЕ**

[ОПИСАНИЕ ЗАДАЧИ 3](#_Toc209510410)

[ПРИНЦИП ПАРАЛЛЕЛИЗАЦИИ 7](#_Toc209510411)

[РЕШЕНИЕ ЗАДАЧИ 8](#_Toc209510412)

[ОЦЕНКА ЭФФЕКТИВНОСТИ ПАРАЛЛЕЛИЗАЦИИ 10](#_Toc209510413)

[ПРИЛОЖЕНИЕ А – ЛИСТИНГ КОДА 12](#_Toc209510414)

[ПРИЛОЖЕНИЕ Б – ТАБЛИЦА 2 16](#_Toc209510418)

[ПРИЛОЖЕНИЕ В – ТАБЛИЦА 3 17](#_Toc209510419)

ОПИСАНИЕ ЗАДАЧИ

В задаче рассматривается численный расчет объема сложного тела. Целью является сравнение точности и производительности двух способов вычислений — однопоточного и многопоточного. Для примера была задана функция в цилиндрических координатах, описывающая трехмерное тело:

Параметры были подобраны так, чтобы изображение было похоже на объемную звезду с полостью внутри (рисунок 1). Такой эффект достигается при параметрах:

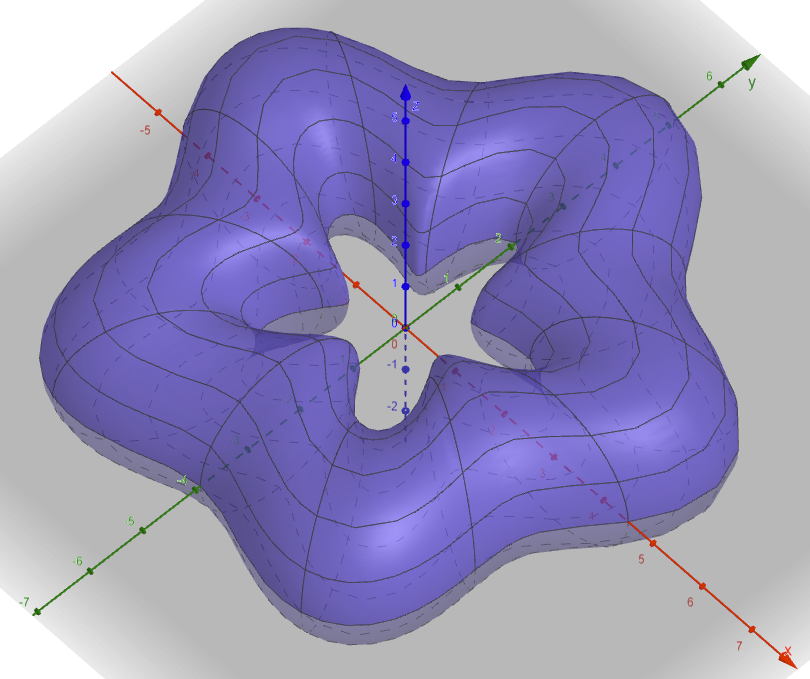


Рисунок 1 – Исходное трехмерное тело

При переводе в сферическую систему координат получим:

И если выразить :

Для проверки была построена фигура по формулам сферической системы координат на языке Python (рисунок 2).

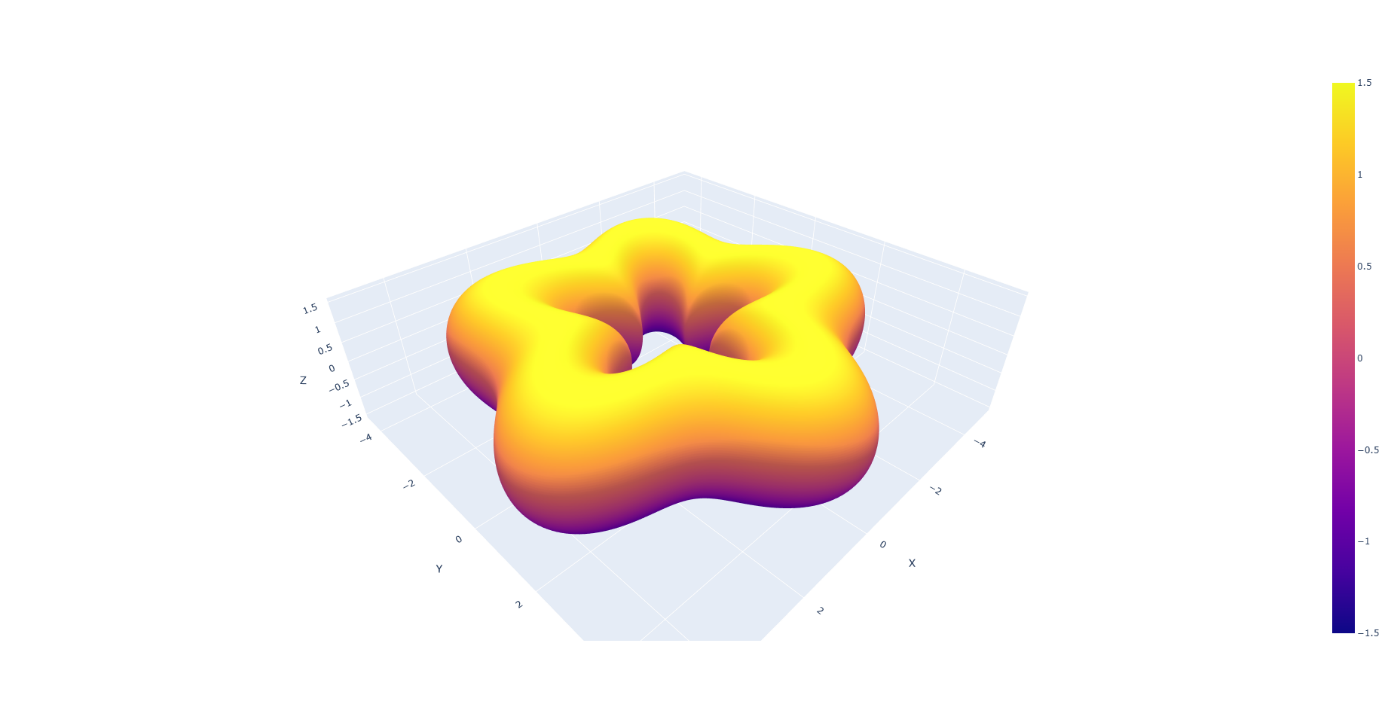


Рисунок 2 – Исходное тело, построенное в сферической системе координат

Объем можно найти аналитическими методами. Одним из примеров является метод вычисления через тройной интеграл. В сферической системе координат тройной элемент объёма выражается как:

где

Отсюда тройной интеграл для объёма записывается как:

где ρ′ – переменная интегрирования по радиусу

После ряда сокращений и подстановки радиуса получаем:

Решение данного интеграла можно получить аналитически или численно с помощью сторонних калькуляторов или Python-библиотек. Численное значение объема составляет 148.157509543295 (c точностью до 12 знаков после запятой).

В рамках лабораторной работы требуется сравнить полученный результат и объем, полученный методом Монте-Карло – статистический метод, который используется для оценки различных величин, включая объем фигур. Для вычисления объема фигуры с помощью метода Монте-Карло можно следовать следующим шагам:

1. Определение фигуры и границ, а также куб, в который вписана эта фигура;

2. Генерация случайных точек внутри куба;

3. Проверка попадания в фигуру;

4. Подсчет точек – общее количество сгенерированных точек (N) и количество точек, попавших внутрь фигуры (M);

5. Вычисление объема – объем фигуры можно оценить по формуле:

V ≈ M / N ⋅ V\_(куба)

где V\_(куба) – объем куба, в который вписана фигура.

ПРИНЦИП ПАРАЛЛЕЛИЗАЦИИ

Задача разбивается на части, которые распределяются между несколькими процессами. Каждый процесс выполняет свой участок работы, а затем результаты собираются и объединяются в итоговый ответ.

В рамках поставленной задачи это сводится к разделению количества генерируемых точек на равные части, где каждый поток забирает часть точек и проверяет их на вхождение в данное тело. Затем количество входящих точек суммируется и делится на общее количество точек, домножив на объем выбранного параллелепипеда мы получим примерный объем фигуры методом Монте-Карло.

РЕШЕНИЕ ЗАДАЧИ

В коде реализуются две основные функции и ключевой стартовый блок кода.

Функция is\_dot\_inside\_shape(x, y, z) – проверяет точку на вхождение в данную фигуру.

def is\_dot\_inside\_shape(x, y, z):  
 rho = np.sqrt(x \*\* 2 + y \*\* 2 + z \*\* 2)  
 theta = np.arccos(z / rho) if rho != 0 else 0.0  
 phi = np.arctan2(y, x)  
 factor = 3 + 1.5 \* np.cos(phi) + 0.6 \* np.sin(5 \* theta)  
 r\_max = factor  
  
 return rho <= r\_max

Функция worker\_generate\_points(point\_amount) – отдельный воркер, который генерирует заданное количество точек и вызывает функцию проверки вхождения. Возвращает количество входящих в фигуру точек.

def worker\_generate\_points(point\_amount):  
 *"""  
 Генерирует N точек в пределах заданного параллелепипеда и проверяет их на вхождение в фигуру.  
 :param N: int, количество точек  
 :return: int, количество точек внутри фигуры  
 """* count\_inside = 0  
 start = time.perf\_counter()  
 for \_ in range(point\_amount):  
 x = np.round(np.random.uniform(X\_MIN, X\_MAX), 12)  
 y = np.round(np.random.uniform(Y\_MIN, Y\_MAX), 12)  
 z = np.round(np.random.uniform(Z\_MIN, Z\_MAX), 12)  
 if is\_dot\_inside\_shape(x, y, z):  
 count\_inside += 1  
 end = time.perf\_counter()  
 return count\_inside, (end - start)

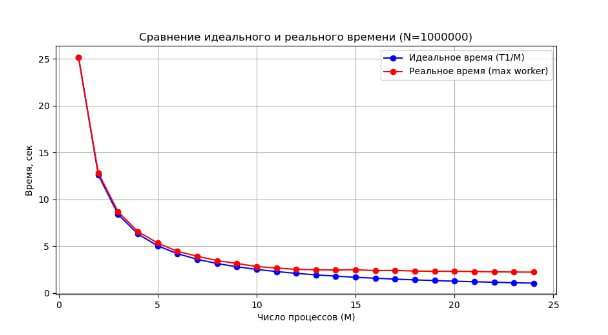
Далее для экспериментов используется вызов этих функций с помощью pool из библиотеки multiprocessing.

chunk\_size = N // M  
task\_sizes = [chunk\_size] \* M  
task\_sizes[-1] += N % M *# остаток в последний процесс*with multiprocessing.Pool(processes=M) as pool:  
 results = pool.map(worker\_generate\_points, task\_sizes)  
  
total\_inside = sum(r[0] for r in results)  
max\_time = max(r[1] for r in results) *# берём самое долгое время*

ОЦЕНКА ЭФФЕКТИВНОСТИ ПАРАЛЛЕЛИЗАЦИИ

Ключевыми параметрами является количество ядер (12) и количество потоков (24). Для данных характеристик было проведено два тестирования:

1. Сравнение идеального и условного времени вычисления при M от 1 до 24. Путем перебора значения N было выявлено, что результат превышает 20 секунд при значениях N>1000000. Результаты тестирования при N=1000000 представлены на верхнем графике (рисунок 4). Точные значения представлены в таблице 1 (приложение Б);
2. Зависимость ошибки от N при M=24. Перебор N осуществлялся от 24 до 1152 с шагом 24, чтобы достичь равномерного распределения объема вычислений площадей. Результаты представлены на нижнем графике (рисунок 3). Точные значения представлены в таблице 2 (приложение В).



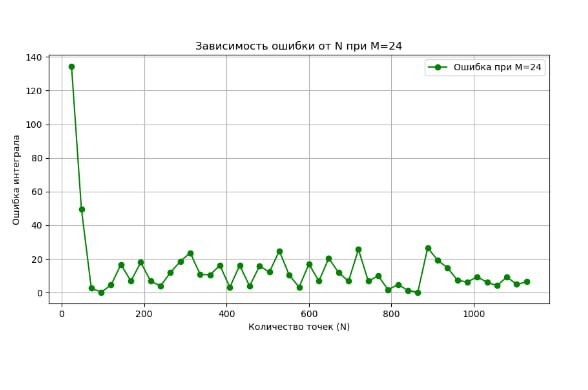


Рисунок 3 – Графики экспериментов

По графику для эксперимента 1 видно, что реальное время достаточно близко к идеальному и отклоняется на большом количестве потоков меньше чем на 1 секунду.

Второй график демонстрирует зависимость погрешности от числа N – чем больше делений участка функции, тем точнее результат.

ПРИЛОЖЕНИЕ А – ЛИСТИНГ КОДА

import time  
import multiprocessing  
import matplotlib.pyplot as plt  
import numpy as np  
  
*# Global variables*X\_MIN, X\_MAX = -4.95, 4.95  
Y\_MIN, Y\_MAX = -5.13, 5.13  
Z\_MIN, Z\_MAX = -3.34, 3.34  
  
  
def is\_dot\_inside\_shape(x, y, z):  
 rho = np.sqrt(x \*\* 2 + y \*\* 2 + z \*\* 2)  
 theta = np.arccos(z / rho) if rho != 0 else 0.0  
 phi = np.arctan2(y, x)  
 factor = 3 + 1.5 \* np.cos(phi) + 0.6 \* np.sin(5 \* theta)  
 r\_max = factor  
  
 *# Если расстояние rho меньше или равно r\_max — точка внутри фигуры* return rho <= r\_max  
  
  
def worker\_generate\_points(point\_amount):  
 *"""  
 Генерирует N точек в пределах заданного параллелепипеда и проверяет их на вхождение в фигуру.  
 :param N: int, количество точек  
 :return: int, количество точек внутри фигуры  
 """* count\_inside = 0  
 start = time.perf\_counter()  
 for \_ in range(point\_amount):  
 x = np.round(np.random.uniform(X\_MIN, X\_MAX), 12)  
 y = np.round(np.random.uniform(Y\_MIN, Y\_MAX), 12)  
 z = np.round(np.random.uniform(Z\_MIN, Z\_MAX), 12)

ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А – ЛИСТИНГ КОДА

if is\_dot\_inside\_shape(x, y, z):  
 count\_inside += 1  
 end = time.perf\_counter()  
 return count\_inside, (end - start)  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 analytical\_result = 148.157509543295  
 V\_parallelepiped = (abs(X\_MAX) + abs(X\_MIN)) \* (abs(Y\_MAX) + abs(Y\_MIN)) \* (abs(Z\_MAX) + abs(Z\_MIN))  
  
 R = 3  
 r = 1.5  
 k = 0.6  
 n = 5  
  
 N = 1\_000\_000 *# Количество точек  
  
 # Сначала замерим время для одного потока (T1)* print(f"Замер времени для одного процесса...")  
 single\_result, single\_time = worker\_generate\_points(N)  
 print(f"T1 (1 процесс) = {single\_time:.6f} sec")  
 print(V\_parallelepiped \* (single\_result / N))  
  
 M\_values = []  
 real\_times = []  
 ideal\_times = []  
 print(f"\nЗамер времени для M процессов от 1 до 24:")  
  
 for M in range(1, 25): *# от 1 до 24 процессов* if M == 1:  
 M\_values.append(1)  
 real\_times.append(single\_time) *# берём время одного воркера* ideal\_times.append(single\_time / M)  
 print(f"M={M}, Real={single\_time:.6f}, Ideal={single\_time / M:.6f}, "

ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А – ЛИСТИНГ КОДА

f"Result={V\_parallelepiped \* (single\_result / N):.6f}")  
 continue  
  
 *# делим количество точек на чанки* chunk\_size = N // M  
 task\_sizes = [chunk\_size] \* M  
 task\_sizes[-1] += N % M *# остаток в последний процесс* with multiprocessing.Pool(processes=M) as pool:  
 results = pool.map(worker\_generate\_points, task\_sizes)  
  
 total\_inside = sum(r[0] for r in results)  
 max\_time = max(r[1] for r in results) *# берём самое долгое время* M\_values.append(M)  
 real\_times.append(max\_time)  
 ideal\_times.append(single\_time / M)  
  
 print(f"M={M}, Real={max\_time:.6f}, Ideal={single\_time / M:.6f}, "  
 f"Result={V\_parallelepiped \* (total\_inside / N):.6f}")  
  
 *# ---------------- ГРАФИК №1 ----------------* plt.figure(figsize=(10, 5))  
 plt.plot(M\_values, ideal\_times, "bo-", label="Идеальное время (T1/M)")  
 plt.plot(M\_values, real\_times, "ro-", label="Реальное время (max worker)")  
 plt.xlabel("Число процессов (M)")  
 plt.ylabel("Время, сек")  
 plt.title(f"Сравнение идеального и реального времени (N={N})")  
 plt.legend()  
 plt.grid(True)  
 plt.show()

ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А – ЛИСТИНГ КОДА

*# ---------------- ГРАФИК №2 ----------------* print("\nРасчет ошибки для разных N при M=24:")  
 N\_values = list(range(24, 24 \* 24 \* 2, 24))  
 errors\_24 = []  
  
 for n\_points in N\_values:  
 chunk\_size = n\_points // 24  
 task\_sizes = [chunk\_size] \* 24  
 task\_sizes[-1] += n\_points % 24  
  
 with multiprocessing.Pool(processes=24) as pool:  
 results = pool.map(worker\_generate\_points, task\_sizes)  
  
 total\_inside = sum(r[0] for r in results)  
 volume\_est = V\_parallelepiped \* (total\_inside / n\_points)  
 error = abs(analytical\_result - volume\_est)  
 errors\_24.append(error)  
  
 print(f"N={n\_points}, Volume={volume\_est:.6f}, Error={error:.6e}")  
  
 plt.figure(figsize=(10, 5))  
 plt.plot(N\_values, errors\_24, "go-", label="Ошибка при M=24")  
 plt.xlabel("Количество точек (N)")  
 plt.ylabel("Ошибка интеграла")  
 plt.title("Зависимость ошибки от N при M=24")  
 plt.legend()  
 plt.grid(True)  
 plt.show()

ПРИЛОЖЕНИЕ Б – ТАБЛИЦА 1

Таблица 2. Время и ошибки для разного числа процессов M.

|  |  |  |  |
| --- | --- | --- | --- |
| **M** | **Real (сек)** | **Ideal (сек)** | **Error** |
| 1 | 21.953083 | 21.953088 | 1.828761e-08 |
| 2 | 11.261112 | 10.976544 | 8.353204e-09 |
| 3 | 7.480686 | 7.317696 | 5.002950e-06 |
| 4 | 5.578242 | 5.488272 | 8.870394e-09 |
| 5 | 4.538055 | 4.390618 | 7.203745e-09 |
| 6 | 3.917484 | 3.658848 | 5.007518e-06 |
| 7 | 3.407021 | 3.136155 | 1.002153e-05 |
| 8 | 3.054618 | 2.744136 | 3.855924e-09 |
| 9 | 2.840623 | 2.439232 | 1.252888e-05 |
| 10 | 2.500210 | 2.195309 | 3.230234e-09 |
| 11 | 2.953146 | 1.995735 | 2.506200e-05 |
| 12 | 2.440295 | 1.829424 | 2.004931e-05 |
| 13 | 2.373145 | 1.688699 | 1.503654e-05 |
| 14 | 2.355782 | 1.568078 | 1.002371e-05 |
| 15 | 2.326788 | 1.463539 | 1.253035e-05 |
| 16 | 2.245960 | 1.372068 | 1.933998e-09 |
| 17 | 2.185620 | 1.291358 | 4.010202e-05 |
| 18 | 2.208825 | 1.219616 | 3.508915e-05 |
| 19 | 2.129643 | 1.155426 | 2.506326e-05 |
| 20 | 2.198139 | 1.097654 | 1.542510e-09 |
| 21 | 2.204646 | 1.045385 | 2.756992e-05 |
| 22 | 2.189144 | 0.997868 | 2.506351e-05 |
| 23 | 2.064238 | 0.954482 | 2.005058e-05 |
| 24 | 2.123062 | 0.914712 | 2.005063e-05 |

ПРИЛОЖЕНИЕ В – ТАБЛИЦА 2

Таблица 3. Погрешности для 24 процессов, при разных N.

|  |  |
| --- | --- |
| **N** | **Error** |
| 24 | 3.240510e-02 |
| 48 | 8.087008e-03 |
| 72 | 3.593055e-03 |
| 96 | 2.020863e-03 |
| 120 | 1.293284e-03 |
| 144 | 8.980883e-04 |
| 168 | 6.598086e-04 |
| 192 | 5.051603e-04 |
| 216 | 3.991359e-04 |
| 240 | 3.232983e-04 |
| 264 | 2.671876e-04 |
| 288 | 2.245111e-04 |
| 312 | 1.912990e-04 |
| 336 | 1.649462e-04 |
| 360 | 1.436863e-04 |
| 384 | 1.262866e-04 |
| 408 | 1.118662e-04 |
| 432 | 9.978182e-05 |
| 456 | 8.955480e-05 |
| 480 | 8.082316e-05 |
| 504 | 7.330895e-05 |
| 528 | 6.679593e-05 |
| 552 | 6.111383e-05 |