Компьютерные технологии

Понур К.А.

6 июня 2021 г.

Задание 20 Используя метод прямоугольников и метод выборочного среднего, вычислите моменты инерции тела сложного объема с неоднородной плотностью при его вращении вокруг трех перпендикулярных осей. Оцените точность и время вычислений в обоих случаях в зависимости от схемы интегрирования.

Оглавление

1 Постановка задачи

Пусть плотность исследуемого тела задается в декартовой системе координат некоторой функцией $\rho(x, y, z)$

1.1 Метод прямоугольников

Наиболее простой метод численного интегрирования заключается в аппроксимации подынтегральной функции $f(x)$ многочленом нулевой степени на каждом элементарном отрезке. Для n элементарных отрезков подобная составная квадратурная формула на линейной координатной сетке с шагом $h = \frac{b-a}{n}$ примет вид

$$\int\limits_a^b f(x)\mathrm{d}x \approx h \sum_{i=1}^{n} f(x_i) \tag{1}$$

Для N-мерного интеграла стоит рассматривать (1) как взвешенную сумму по всем имеющимся размерностям.

1.2 Метод Монте-Карло

Сущность метода Монте-Карло состоит в следующем: необходимо найти значение a некоторой изучаемой величины. Для этого выбирают величину x, математическое ожидание которой равно a, т.е.

$$M(X) = a$$

На практике это означает проведение n независимых испытаний и вычисление среднего значения величины x по полученному ансамблю данных

$$\langle x \rangle = \frac{1}{n} \sum_{i=1}^{n} x_i.$$

Величину $\langle x \rangle$ принимают в качестве оценки a^* исходной величины a. Пусть теперь J значение интеграла на интервале $[a, b]$ функции $f(x)$

$$J = \int\limits_a^b f(x)\mathrm{d}x$$

Тогда

$$\langle f(x) \rangle \approx \frac{1}{n} \sum_{i=1}^{n} f(x_i)$$

И сам интеграл

$$J \approx \frac{b-a}{n} \sum_{i=1}^{n} f(x_i) \tag{2}$$

Уравнение (2) обобщается на случай N-мерного интеграла

$$J \approx \prod_{j=1}^{N} (b_j - a_j) \cdot \frac{1}{n} \sum_{i=1}^{m} f(x_i),$$

где $\prod\limits_{j=1}^{N} (b_j - a_j)$ – объем N-мерного параллелепипеда, m – точки N-мерного пространства, которые принадлежат области интегрирования.

2 Тестирование алгоритмов на тестовом объеме

В качестве тестового тела был выбран тороид, располагаемый в начале координат и задаваемый параметрическим уравнением в сферической системе координат

$$\begin{cases} x(\varphi, \psi) = (R + r\cos\psi)\cos\varphi \\ y(\varphi, \psi) = (R + r\cos\psi)\sin\varphi \qquad \varphi \in [0, 2\pi), \psi \in [-\pi, \pi), \\ z(\varphi, \psi) = r\sin\psi \end{cases}$$

где φ – азимутальный угол в плоскости xy, ψ – угол элевации, отсчитываемый от оси z. Характерная форма поверхности изображена на рис. 1.

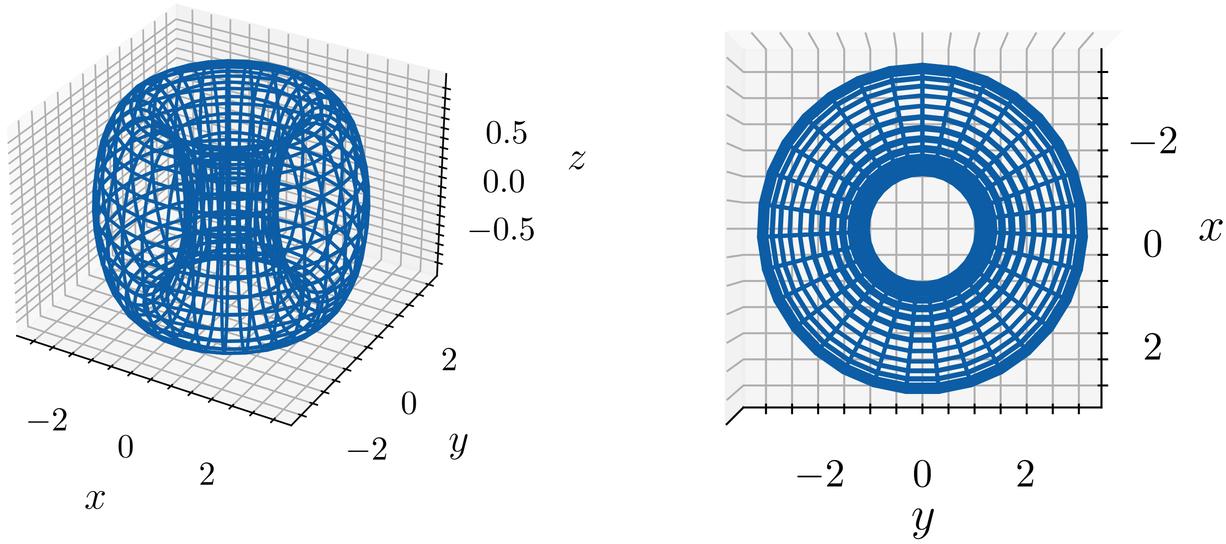

Рис. 1: Тороид с внутренним радиусом образующей $r = 1$ и радиусом до образующей $R = 2$

Вычисление объема тела предполагает вычисление трехмерного интеграла по занимаемой телом области с подынтегральной функцией $f(x, y, z) = 1$. Несмотря на нетривиальность интегрирования данного тела, его объем аналитически вычисляется

$$V = 2\pi^2 R r^2,$$

что позволит нам оценивать точность методов интегрирования сравнивая их относительную ошибку

$$\varepsilon = \frac{\left|V - \tilde{V}\right|}{V},$$

где $\tilde{V}$ – оцененный объем.

Для реализации вышеперечисленных методов был выбран Python с библиотекой NumPy для численных расчетов и Matplotlib для визуализации данных. Базовые методы и объекты класса представлены в лист. 1.

На рис. 2a представлено сравнение относительной ошибки вычисления объема тороида методами Монте-Карло и прямоугольниками при различном количестве точек N. На рис. 2b представлено сравнение времени выполнения программы[1].

[1]Выбросы на графике времени выполнения программы возникли из-за компиляции на фоне LaTeX-документа :)

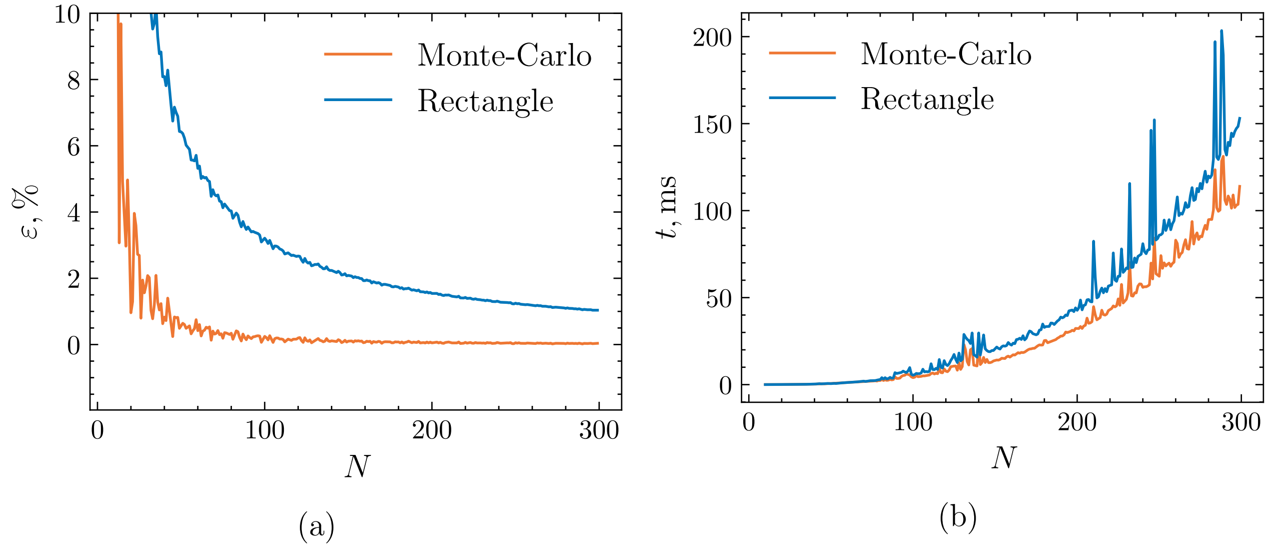

Рис. 2: Сравнение (а) относительной ошибки вычисления интеграла и (б) времени вычисления для различного количества точек. N – количество точек вдоль одной размерности (реальное количество точек N^3).

Стоит заметить, что метод прямоугольников проиграл методу Монте-Карло как по времени выполнения, так и по результативной точности.

3 Исходный код

```python
import numpy as np
from numpy import random


class Tor(object):
    def __init__(self, a: tuple) -> None:
        # R is the distance from the center of the tube to the center of the torus,
        self.R = a[0]
        # r is the radius of the tube.
        self.r = a[1]

    def mask(self, r: float) -> bool:

        r0 = self.r
        R0 = self.R

        # XY-axis radius projection
        rho2 = np.sum(r[0:2]**2, axis=0)
        # Radius absolute value
        R2 = rho2 + r[2]**2
        # Surface equation
        # https://en.wikipedia.org/wiki/Torus
        mask = (R2 + R0**2 - r0**2)**2 - 4*R0**2*(rho2) <= 0
        return mask

    @property
    def volume(self) -> float:
        # Analytical expression for volume
        return 2*np.pi**2*self.R*self.r**2

    def grid(self, num: tuple, dtype="linear"):
        R = self.R
        r = self.r

        if dtype == "linear":
            return cartesian_meshgrid([-R-r, -R-r, -r], [R+r, R+r, +r], num)
        elif dtype == "random":
            return cartesian_random([-R-r, -R-r, -r], [R+r, R+r, +r], np.prod(num))


def monte_carlo(mask: bool, r: float):
    axis = 1
    # Volume of an N-dimensional cube describing the area of integration
    area = np.prod(r.max(axis) - r.min(axis), dtype=np.float64)
    return np.mean(mask) * area
```

```python


def rect(mask: bool, dr: float):
    # Volume of elementary N-dimensional cubes dV inside area of integration
    dV = np.prod(dr, axis=0, where=mask)
    return np.sum(dV, where=mask)


def cartesian_meshgrid(start, stop, num):
    x = np.linspace(start[0], stop[0], num[0])
    y = np.linspace(start[1], stop[1], num[1])
    z = np.linspace(start[2], stop[2], num[2])

    dx = np.zeros_like(x)
    dy = np.zeros_like(y)
    dz = np.zeros_like(z)

    dx[1:] = np.diff(x)
    dy[1:] = np.diff(y)
    dz[1:] = np.diff(z)

    x, y, z = np.meshgrid(x, y, z)
    dx, dy, dz = np.meshgrid(dx, dy, dz)

    return np.array([x, y, z]), np.array([dx, dy, dz])


def cartesian_random(start, stop, size):
    x = random.uniform(start[0], stop[0], size)
    y = random.uniform(start[1], stop[1], size)
    z = random.uniform(start[2], stop[2], size)
    return np.array([x, y, z])
```

Листинг 1: Исходный код задания