Компьютерные технологии

Понур К.А. *

5 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

исходники (.py, *.tex): https://github.com/kannab98/computer-technologies 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

$$\int\limits_a^b f(x)\mathrm{d}x \approx h\frac{f(x_0)}{2} + h\frac{f(x_n)}{2} + h\sum_{i=1}^{n-1} f(x_i)$$

Погрешность вычисления интеграла на i-ом элементарном отрезке в таком случае определяется формулой

$$\varepsilon(f) = \frac{\mathrm{d}^2 f(\xi)}{\mathrm{d}\xi^2} \cdot \frac{h^2}{24}(x_i - x_{i-1}),$$

где $\xi_i = \frac{x_{i-1}+x_i}{2}$ – середина рассматриваемого элементарного отрезка.

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

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

2 Реализации на языке Python

Прежде чем переходить к телу достаточно сложного объема, программа отлаживалась на простейшем теле для интегрирование – кубе (см. рис. **??**).

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

где φ – азимутальный угол в плоскости xy, ψ – угол элевации, отсчитываемый от оси z. На рис. 1

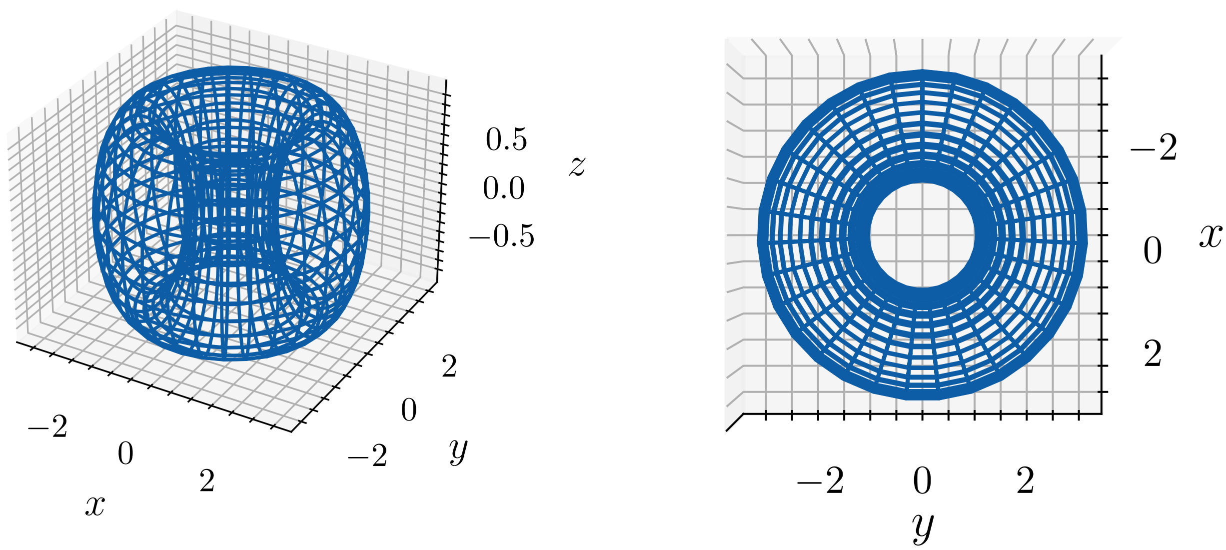


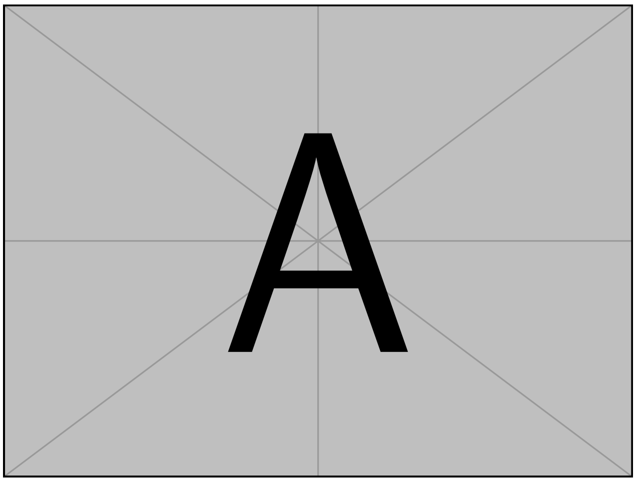
Рис. 1: Тороид с внутренним радиусом $r = 1$ и внешним радиусом $R = 2$

```
import numpy as np
from numpy import float64, random, where
import matplotlib.pyplot as plt
```

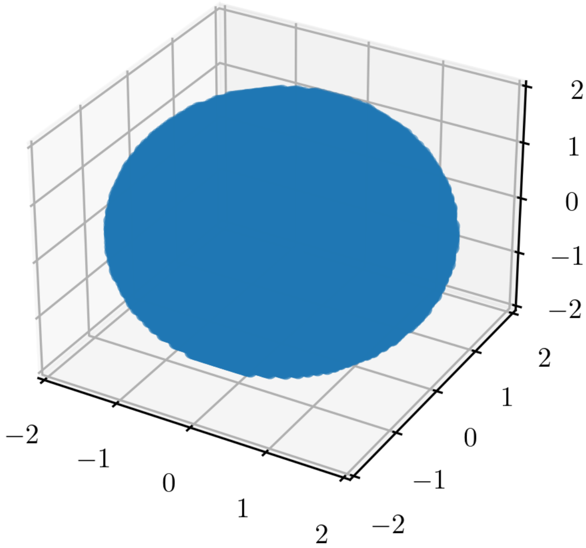
```
from numpy.linalg import norm
from scipy import integrate

def cart2sphere(r):
    eps = 1e-8
    R =  norm(r, axis=0)
    El = np.arccos(r[2]/(R+eps))
    Az = np.arctan2(r[1], (r[0]+eps))
    return R, El, Az


def sphere2cart(r):
    X = r[0] * np.sin(r[1]) * np.cos(r[2])
    Y = r[0] * np.sin(r[1]) * np.sin(r[2])
    Z = r[0] * np.cos(r[1])
    return X, Y, Z

class Cube(object):
    def __init__(self, a: float) -> None:
        self.a = a

    def mask(self, r):
        a = self.a
        mask = (-a <= r) & (r <= a)
        mask = np.prod(mask, axis=0, dtype=bool)
        return mask

    def volume(self):
        return (2*self.a)**3

    def grid(self, num, dtype="linear"):
        a = self.a
        if dtype == "linear":
            return cartesian_meshgrid([-a, -a, -a], [a, a, a], num)
        elif dtype == "random":
            return cartesian_random([-a, -a, -a], [a, a, a], np.prod(num))


class Sphere(object):
    def __init__(self, a: float) -> None:
        self.a = a

    def mask(self, r):
        a = self.a
        rho = norm(r, axis=0)
        mask = (rho <= a)
        return mask

```

```
    def volume(self):
        return 4/3*np.pi*self.a**3

    def grid(self, num, dtype="linear"):
        a = self.a
        if dtype == "linear":
            return cartesian_meshgrid([-a, -a, -a], [a, a, a], num)
        elif dtype == "random":
            return cartesian_random([-a, -a, -a], [a, a, a], np.prod(num))


class Tor(object):
    def __init__(self, a: tuple) -> None:
        self.R = a[0]
        self.r = a[1]

    def mask(self, r):

        r0 = self.r
        R0 = self.R

        rho2 = np.sum(r[0:2]**2, axis=0)
        R2 = np.sum(r**2, axis=0)
        mask = (R2 + R0**2 - r0**2)**2 - 4*R0**2*(rho2) <= 0
        return mask



    def volume(self):
        return 2*np.pi**2*self.R*self.r**2

    def grid(self, num, 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(func: np.ndarray, r: np.ndarray):
    axis = 1
    area = np.prod(r.max(axis) - r.min(axis), dtype=np.float64)
    return np.mean(func) * area

def rect(mask: np.ndarray, dr: np.ndarray):
    dR = np.prod(dr, axis=0, where=mask)
```

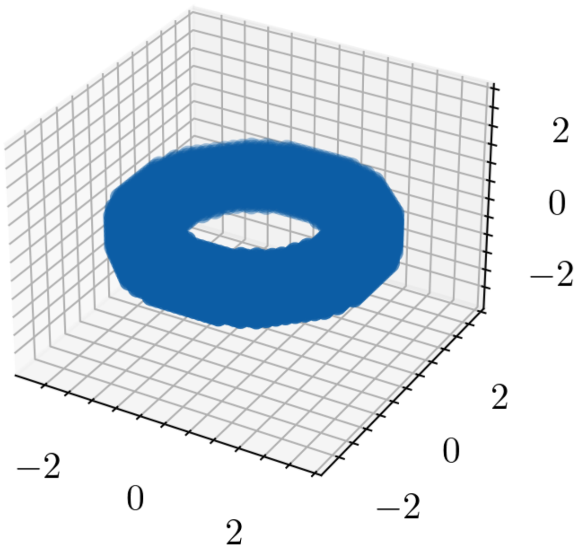
```
    return np.sum(dR, where=mask)

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])
```



(a)



(b)



(c)

Рис. 2: Точки в декартовых координатах, принадлежащие поверхностям (a) куба, (b) сферы, (c) тороида и участвующие в интегрировании