

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	1
1.1	Метод прямоугольников	1
1.2	Метод Монте-Карло	2
2	Реализации на языке Python	3

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

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

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

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

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 $f(x)$

$$J = \int_a^b f(x)dx$$

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 интегрирования – кубе (см. рис. ??).

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

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

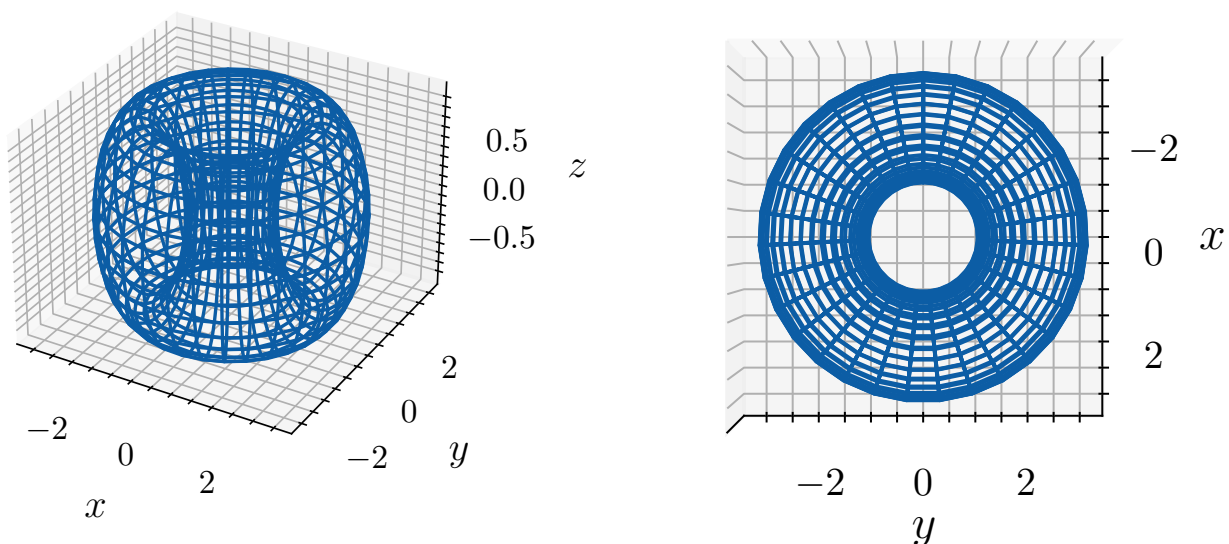


Рис. 1: Тороид с внутренним радиусом $r = 1$ и внешним радиусом $R = 2$

```

1 import numpy as np
2 from numpy import float64, random, where
3 import matplotlib.pyplot as plt

```

```

4  from numpy.linalg import norm
5  from scipy import integrate
6
7  def cart2sphere(r):
8      eps = 1e-8
9      R = norm(r, axis=0)
10     El = np.arccos(r[2]/(R+eps))
11     Az = np.arctan2(r[1], (r[0]+eps))
12     return R, El, Az
13
14
15 def sphere2cart(r):
16     X = r[0] * np.sin(r[1]) * np.cos(r[2])
17     Y = r[0] * np.sin(r[1]) * np.sin(r[2])
18     Z = r[0] * np.cos(r[1])
19     return X, Y, Z
20
21 class Cube(object):
22     def __init__(self, a: float) -> None:
23         self.a = a
24
25     def mask(self, r):
26         a = self.a
27         mask = (-a <= r) & (r <= a)
28         mask = np.prod(mask, axis=0, dtype=bool)
29         return mask
30
31     def volume(self):
32         return (2*self.a)**3
33
34     def grid(self, num, dtype="linear"):
35         a = self.a
36         if dtype == "linear":
37             return cartesian_meshgrid([-a, -a, -a], [a, a, a], num)
38         elif dtype == "random":
39             return cartesian_random([-a, -a, -a], [a, a, a], np.prod(num))
40
41
42 class Sphere(object):
43     def __init__(self, a: float) -> None:
44         self.a = a
45
46     def mask(self, r):
47         a = self.a
48         rho = norm(r, axis=0)
49         mask = (rho <= a)
50         return mask
51

```

```

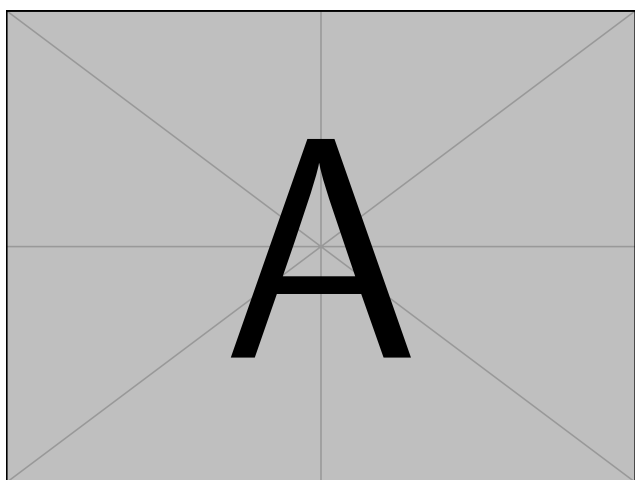
52     def volume(self):
53         return 4/3*np.pi*self.a**3
54
55     def grid(self, num, dtype="linear"):
56         a = self.a
57         if dtype == "linear":
58             return cartesian_meshgrid([-a, -a, -a], [a, a, a], num)
59         elif dtype == "random":
60             return cartesian_random([-a, -a, -a], [a, a, a], np.prod(num))
61
62
63 class Tor(object):
64     def __init__(self, a: tuple) -> None:
65         self.R = a[0]
66         self.r = a[1]
67
68     def mask(self, r):
69
70         r0 = self.r
71         R0 = self.R
72
73         rho2 = np.sum(r[0:2]**2, axis=0)
74         R2 = np.sum(r**2, axis=0)
75         mask = (R2 + R0**2 - r0**2)**2 - 4*R0**2*(rho2) <= 0
76         return mask
77
78
79
80     def volume(self):
81         return 2*np.pi**2*self.R*self.r**2
82
83     def grid(self, num, dtype="linear"):
84         R = self.R
85         r = self.r
86
87         if dtype == "linear":
88             return cartesian_meshgrid([-R-r, -R-r, -r], [R+r, R+r, +r], num)
89         elif dtype == "random":
90             return cartesian_random([-R-r, -R-r, -r], [R+r, R+r, +r], np.prod(num))
91
92
93     def monte_carlo(func: np.ndarray, r: np.ndarray):
94         axis = 1
95         area = np.prod(r.max(axis) - r.min(axis), dtype=np.float64)
96         return np.mean(func) * area
97
98     def rect(mask: np.ndarray, dr: np.ndarray):
99         dR = np.prod(dr, axis=0, where=mask)

```

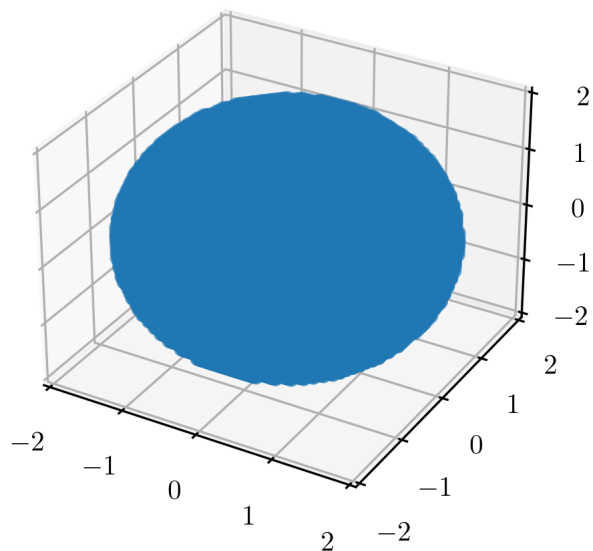
```

100     return np.sum(dR, where=mask)
101
102 def cartesian_meshgrid(start, stop, num):
103     x = np.linspace(start[0], stop[0], num[0])
104     y = np.linspace(start[1], stop[1], num[1])
105     z = np.linspace(start[2], stop[2], num[2])
106
107     dx = np.zeros_like(x)
108     dy = np.zeros_like(y)
109     dz = np.zeros_like(z)
110
111     dx[1:] = np.diff(x)
112     dy[1:] = np.diff(y)
113     dz[1:] = np.diff(z)
114
115     x,y,z = np.meshgrid(x,y,z)
116     dx,dy,dz = np.meshgrid(dx,dy,dz)
117
118     return np.array([x,y,z]), np.array([dx,dy,dz])
119
120 def cartesian_random(start, stop, size):
121     x = random.uniform(start[0], stop[0], size)
122     y = random.uniform(start[1], stop[1], size)
123     z = random.uniform(start[2], stop[2], size)
124     return np.array([x,y,z])

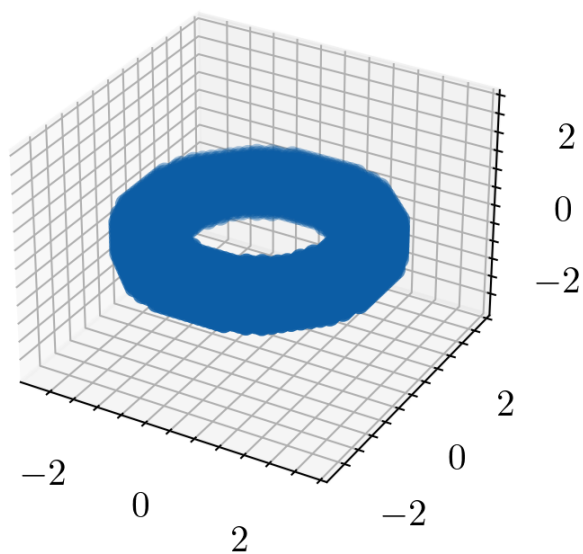
```



(a)



(b)



(c)

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