目录

1. 问题阐述与分析

转动惯量（Moment of inertia）是经典力学中，刚体绕轴转动时惯性（回转物体保持其匀速圆周运动或静止的特性）的量度。转动惯量通常用字母 I 或 J 表示，其国际单位制单位为 $kg \cdot m^2$。转动惯量具有重要的物理意义，在转动力学中被用于方便地描述角动量、角速度、力矩和角加速度等数个量之间的关系，扮演着相当于质量在线性动力学中的角色。其计算方法如下所示：

$$I = \int\int\int \rho(x,y,z) r_\perp^2(x,y,z)\,\mathrm{d}x\,\mathrm{d}y\,\mathrm{d}z \tag{1}$$

其中 $r_\perp(x,y,z)$ 表示该点到转轴的垂直距离。

如图1所示，有一个半径为 r_1 的球体，它由两部分组成：密度为 ρ_2 半径为 r_2 的阴影部分圆柱状体和密度为 ρ_1 的其余部分。现在我们需要求解这个球体关于 z 轴的转动惯量 I_z 和关于 x 轴的转动惯量 I_x。在计算 I_z 时，根据式1，$r_\perp^2(x,y,z)$ 应当取

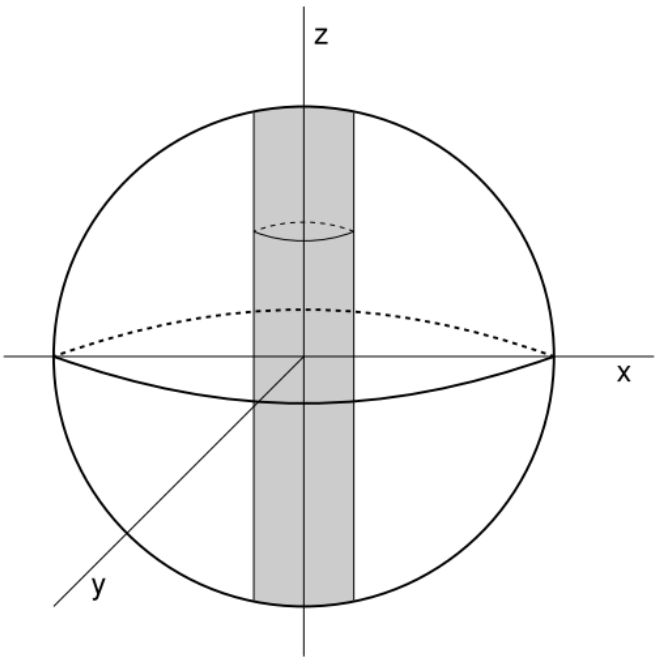


图 1 一个半径为 r_1 的球内嵌着一个半径为 r_2 的圆柱状体，圆柱状体部分的密度为 ρ_2，其余部分的密度为 ρ_1。

x^2+y^2。相应地，在计算 I_x 时，$r_\perp^2(x,y,z)$ 应当取 y^2+z^2。

在本文中会用一个内部圆柱状体半径为 $1cm$ 且材料为金 ($8930kg/m^3$)，其余部分材料为铜 ($19320kg/m^3$) 且球体半径为 $5cm$ 的例子进行具体的计算。

2. 问题中涉及到的变量

- 球体的半径 r_1，单位为米 (m)

- 圆柱部分的半径 r_2，单位为米 (m)，要求 $r_2 \leqslant r_1$
- 球体其余部分的密度 ρ_1，单位为 kg/m^3
- 圆柱形部分的密度 ρ_1，单位为 kg/m^3
- 蒙特卡洛采样点数 `npt`
- 不同的迭代次数 `nbi`

3. 计算模拟原理

3.1 转动惯量的计算

使用蒙特卡洛方法，在以圆点为中心，边长 $L = 2r_1$ 的正方体中随机均匀撒点，并计算撒出的点对应的函数值 $\rho(x, y, z)$：若该点落入圆柱形部分，即：

$$\begin{cases} x^2 + y^2 + z^2 \leqslant r_1^2 \\ x^2 + y^2 \leqslant r_2^2 \end{cases} \tag{2}$$

此时计函数值为 ρ_2；若该点落入球体的其余部分，即：

$$\begin{cases} x^2 + y^2 + z^2 \leqslant r_1^2 \\ x^2 + y^2 \geqslant r_2^2 \end{cases} \tag{3}$$

此时计函数值为 ρ_1；若该点落入正方体的其余部分，则计函数值为 0。

每次撒出的点的数目为 N，将这些点对应的函数值乘以其与转轴的垂直距离的平方并求出平均值，最后乘以撒点区间的体积即可得到式1积分的蒙特卡洛积分值，即：

$$I = \frac{(2r_1)^3}{N} \sum_i \rho(x_i, y_i, z_i) r_\perp^2(x_i, y_i, z_i) = \frac{V}{N} \sum_i f(x_i, y_i, z_i) \tag{4}$$

其中 $f(x, y, z) = \rho(x, y, z) r_\perp^2(x, y, z)$。

3.2 标准误差的计算

接下来，还可进一步计算求得的积分的标准误差，即：

$$\sigma_f = \sqrt{\langle f^2 \rangle - \langle f \rangle^2} \tag{5}$$

其中 $\langle f^2 \rangle = \frac{1}{N} \sum f^2$，$\langle f \rangle = \frac{1}{N} \sum f$，从而可以进一步得到蒙特卡洛积分的一些稳定性分析和统计分析。

4. 程序设计

本程序的流程图如下所示：

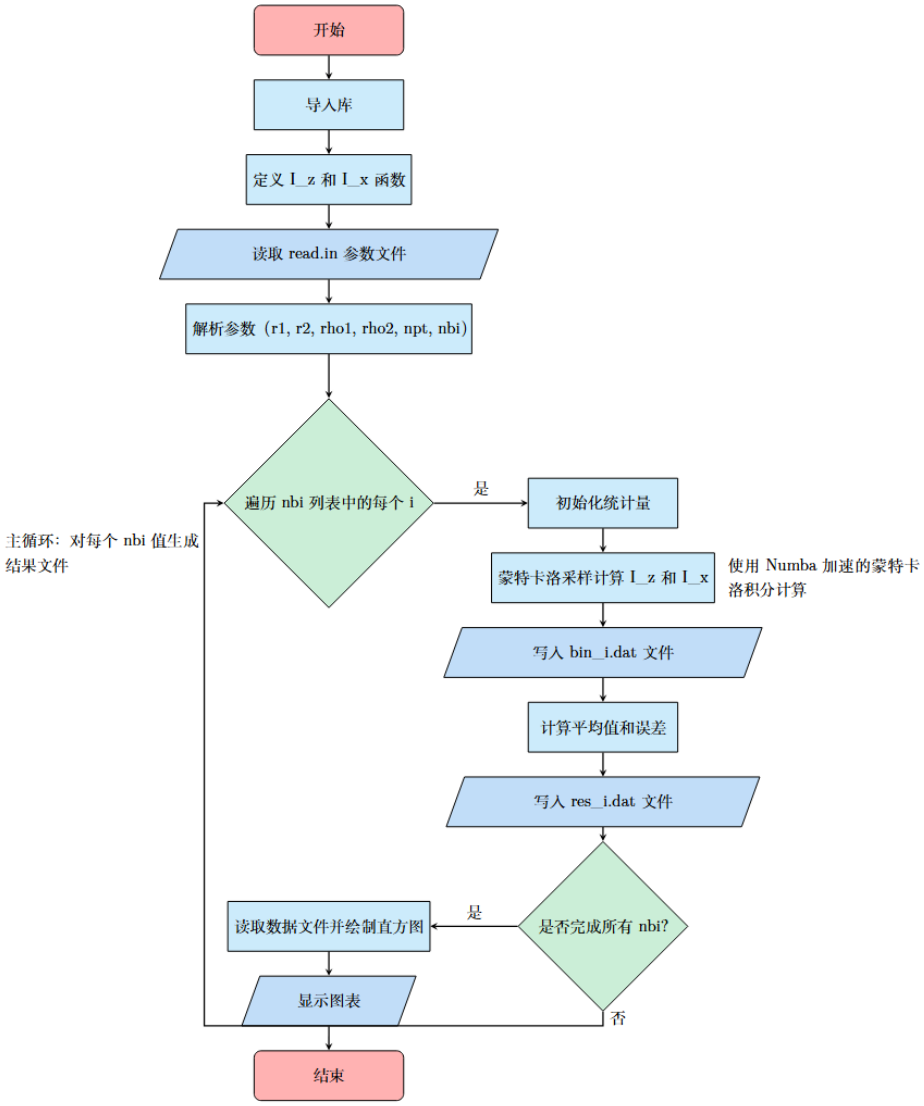


图 2 程序流程图

4.1 库的调用与函数的定义

程序需要使用 `numpy` 库便于随机数据的生成，使用 `matplotlib` 库进行绘图。`numba` 库用于加速 Python 函数的调用和计算，`re` 则用于文本处理。

```
# 导入必要的库
import numpy as np
import matplotlib.pyplot as plt
from numba import jit  # 用于加速 Python 函数
import re  # 正则表达式库，用于文本处理
```

定义使用蒙特卡洛方法计算绕 z 轴的转动惯量 I_z 和绕 x 轴的转动惯量 I_x 的函

数 `I_z(npt, r1, r2, rho1, rho2)` 和 `I_x(npt, r1, r2, rho1, rho2)`。其中 `npt` 是撒点的个数，`r1` 和 `rho1` 是球体的半径和非圆柱区域的密度，`r2` 和 `rho2` 是圆柱区域的半径和密度。函数中按照计算模拟原理部分所给的方法进行撒点和积分值计算，在计算 I_z 时，$r_\perp^2(x,y,z)$ 取 x^2+y^2；在计算 I_x 时，$r_\perp^2(x,y,z)$ 取 y^2+z^2。最后，函数会输出本次蒙特卡洛积分的积分值。

```
# 使用 numba 的 jit 加速装饰器，提升计算性能
@jit(nopython=True)
def I_z(npt, r1, r2, rho1, rho2):
    """ 计算绕 Z 轴的转动惯量（蒙特卡洛方法）"""
    total = 0
    for _ in range(npt):
        # 在立方体范围内生成随机点
        x = np.random.uniform(-r1, r1)
        y = np.random.uniform(-r1, r1)
        z = np.random.uniform(-r1, r1)
        # 判断点是否在球体内且在圆柱体内
        if x*x + y*y + z*z <= r1*r1 and x*x + y*y <= r2*r2:
            total += rho2 * (x*x + y*y)  # 圆柱区域使用 rho2 密度
        elif x*x + y*y + z*z <= r1*r1:
            total += rho1 * (x*x + y*y)  # 球体非圆柱区域使用 rho1
            ↪ 密度
    # 计算积分结果（体积乘以平均值）
    return (2*r1)**3 * total / npt

@jit(nopython=True)
def I_x(npt, r1, r2, rho1, rho2):
    """ 计算绕 X 轴的转动惯量（蒙特卡洛方法）"""
    total = 0
    for _ in range(npt):
        x = np.random.uniform(-r1, r1)
        y = np.random.uniform(-r1, r1)
        z = np.random.uniform(-r1, r1)
        if x*x + y*y + z*z <= r1*r1 and x*x + y*y <= r2*r2:
            total += rho2 * (y*y + z*z)  # 注意绕 X 轴时的分量计算
        elif x*x + y*y + z*z <= r1*r1:
            total += rho1 * (y*y + z*z)
```

```
    return (2*r1)**3 * total / npt
```

4.2 从 read.in 文件读取参数

在 `read.in` 文件中写入了本次要计算的例子的所有参数。注意程序当中的半径采用的单位是国际单位制中的米 (m)，因此，若给出的数据中的半径是其他的单位，则需要先将数据单位换算成米后再写入 `read.in` 文件从而继续进行后续运算。

```
r1=0.05
r2=0.01
rho1=8930
rho2=19320
npt=1000000
nbi=50 500 5000
```

接下来从已经写好的 `read.in` 文件中读取数据。在这里使用 `params` 方法将文件中的键值对形式的数据读取出来并以字典的形式储存在程序中。然后再对字典进行索引找出相应的值并将其转换为后续需要的数值形式赋给相对应的变量。

```
# 从输入文件读取参数
params = {}
with open("read.in", "r") as f:
    for line in f:
        line = line.strip()  # 去除首尾空白
        if line:  # 跳过空行
            # 分割键值对（以第一个等号为分隔符）
            key, value = line.split("=", 1)
            key = key.strip()
            params[key] = value.strip()
            # 特殊处理 nbi 参数为整数列表
            if key == "nbi":
                params["nbi"] = list(map(int,
                ↪  value.strip().split()))

# 参数类型转换
r1 = float(params["r1"])          # 半径 r1
r2 = float(params["r2"])          # 半径 r2
```

```
rho1 = float(params["rho1"])      # 密度 1
rho2 = float(params["rho2"])      # 密度 2
npt = int(params["npt"])          # 蒙特卡洛采样点数
nbi = np.array(params["nbi"])     # 不同迭代次数列表
```

4.3 主计算循环并输出文件

接下来是本程序的主要计算过程。题目中要求对计算次数`nbi=50, 500, 5000`三种情况各做一次运算，本程序采用`for`循环的方式让`i`作为蒙特卡洛积分的计算次数遍历`nbi`数组。

```
# 主计算循环: 对每个 nbi 值生成结果文件
for i in nbi:
    # 定义输出文件名
    bin_file = f"bin_{i}.dat"  # 原始数据文件
    res_file = f"res_{i}.dat"  # 统计结果文件

    total_z = 0       # 累计 Iz 值
    total_z_sq = 0    # 累计 Iz 平方值 (用于计算方差)
    total_x = 0       # 累计 Ix 值
    total_x_sq = 0    # 累计 Ix 平方值

    # 写入原始数据文件
    with open(bin_file, "w", encoding="utf-8") as file:
        # 写入表头 (注意对齐格式)
        header = (
            f"Number: {'Num':>5s} | "
            f"{'I_z_val(kg*m2)':>25s} | "
            f"{'I_x_val(kg*m2)':>25s}\n"
        )
        file.write(header)

        # 进行 i 次独立计算
        for num in range(i):
            I_z_val = I_z(npt, r1, r2, rho1, rho2)
            I_x_val = I_x(npt, r1, r2, rho1, rho2)
```

```
            # 累加统计量
            total_z += I_z_val
            total_x += I_x_val
            total_z_sq += I_z_val * I_z_val
            total_x_sq += I_x_val * I_x_val

            # 格式化写入数据行
            line = f"Number: {num+1:5d} | {I_z_val:25.18f} |
            ↪  {I_x_val:25.18f}\n"
            file.write(line)

    # 写入统计结果文件
    with open(res_file, "w", encoding="utf-8") as file:
        # 计算平均值和方差
        I_z_avg = total_z / i
        I_x_avg = total_x / i
        error_z = np.sqrt(total_z_sq/i - I_z_avg**2)  # 标准误差计算
        error_x = np.sqrt(total_x_sq/i - I_x_avg**2)

        # 写入结果
        file.write(f"Iz 的平均值: {I_z_avg}\n")
        file.write(f"Ix 的平均值: {I_x_avg}\n")
        file.write(f"Iz 的标准误差: {error_z}\n")
        file.write(f"Ix 的标准误差: {error_x}")

print(" 所有文件已生成! ")
```

在每一次的循环中，本程序会调用之前定义的 `I_z(npt, r1, r2, rho1, rho2)` 和 `I_x(npt, r1, r2, rho1, rho2)` 函数进行 `i` 次独立计算，得到 `i` 个 I_z 和 I_x 的值，并将每一次的计算结果写入 `bin_i.dat` 文件中。最终程序会输出 `bin_50.dat`，`bin_500.dat` 和 `bin_5000.dat` 三个文件，其中的储存样式如下所示（以运行 50 次的输出文件的前 5 行为例）：

```
Number:   Num |            I_z_val(kg*m2) |            I_x_val(kg*m2)
Number:     1 |     0.004699666006017224 |     0.004954402579106020
Number:     2 |     0.004691372933632076 |     0.004933603959354598
Number:     3 |     0.004702647626134264 |     0.004960162183703706
```

```
Number:      4 |      0.004697967857360657 |      0.004940927433723889
```

每次循环中，除了会输出计算结果，程序还会计算得到的 `i` 个 I_z 和 I_x 的积分值的平均值 `I_z_avg` 和 `I_x_avg`，以及利用式5计算 I_z 和 I_x 的标准误差 `error_z` 和 `error_x`，并将结果输出在 `res_i.dat` 文件，输出结果将会在结论与分析部分展示。

4.4 从 res5000.dat 文件读取数据和可视化结果

题目要求对于计算 5000 次的情况画出 I_z 和 I_x 的数据分布直方图。因此，程序需要先从 `res_5000.dat` 文件中导出这 5000 个 I_z 和 I_x 的数据然后才能进行下一步的统计处理和可视化结果。

程序先定义了一个可以从文件中读取数据并把数据输出为一个数组的函数，代码如下：

```python
def read_data(file_path):
    """ 从数据文件中读取 Iz 和 Ix 的值"""
    i_z_values = []
    i_x_values = []

    with open(file_path, 'r', encoding='utf-8') as file:
        next(file)  # 跳过表头行

        for line_num, line in enumerate(file, 1):  # 从第 2 行开始
            line = line.strip()
            # 使用正则表达式分割竖线（处理可能存在的空格）
            parts = re.split(r'\s*\|\s*', line)

            try:
                # 提取数值并转换类型
                i_z = float(parts[1].strip())
                i_x = float(parts[2].strip())
                i_z_values.append(i_z)
                i_x_values.append(i_x)
            except ValueError as e:
                print(f" 第 {line_num} 行数值转换失败: {str(e)}")
                continue
```

```
    return i_z_values, i_x_values
```

接着调用该函数读取 `res_5000.dat` 文件中的 I_z 和 I_x 的数值并输出为数组 `z_data` 和 `x_data` 便于进一步的处理。

```
input_file = "bin_5000.dat"  # 示例数据文件
z_data, x_data = read_data(input_file)
```

最后用这两个数组进行 I_z 和 I_x 的直方图（20 个区间）的绘制。

```
# 绘制 Iz 直方图
plt.figure(figsize=(6, 6), dpi=300)
plt.hist(z_data, bins=20, color='blue', alpha=0.7,
↪  edgecolor='black')
plt.title('Iz-histogram')
plt.xlabel('Iz (kg · m²)')
plt.ylabel('Frequency')
plt.grid(linestyle='--', alpha=0.5)
plt.tight_layout()
plt.show()

# 绘制 Ix 直方图
plt.figure(figsize=(6, 6), dpi=300)
plt.hist(x_data, bins=20, color='orange', alpha=0.9,
↪  edgecolor='black')
plt.title('Ix-histogram')
plt.xlabel('Ix (kg · m²)')
plt.ylabel('Frequency')
plt.grid(linestyle='--', alpha=0.5)
plt.tight_layout()
plt.show()
```

5. 结论与分析

5.1 I_z 和 I_x 的平均值和标准误差

在计算 50 次的条件下，计算得到的 I_z 和 I_x 的平均值和标准误差如下：

```
Iz 的平均值: 0.004692716867469364
Ix 的平均值: 0.004948174078480777
Iz 的标准误差: 5.89104762438419e-06
Ix 的标准误差: 7.397990557323878e-06
```

在计算 500 次的条件下，计算得到的 I_z 和 I_x 的平均值和标准误差如下：

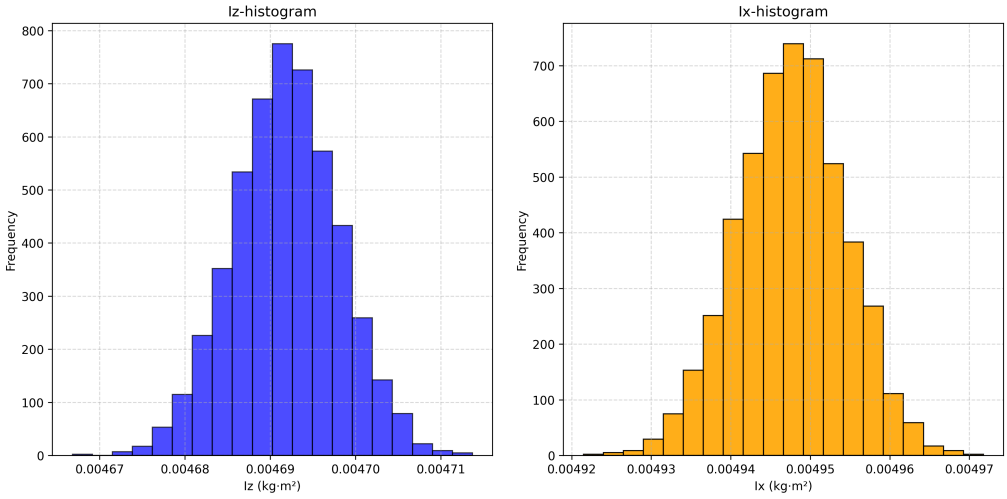
```
Iz 的平均值: 0.0046913604307037655
Ix 的平均值: 0.004947590809498281
Iz 的标准误差: 6.032289038995814e-06
Ix 的标准误差: 6.500963146146622e-06
```

在计算 5000 次的条件下，计算得到的 I_z 和 I_x 的平均值和标准误差如下：

```
Iz 的平均值: 0.004691833601469002
Ix 的平均值: 0.004947629463699874
Iz 的标准误差: 6.181399606557875e-06
Ix 的标准误差: 6.7917144409140565e-06
```

5.2 I_z 和 I_x 的直方图

在计算 5000 次的条件下，程序得到了 5000 个 I_z 和 I_x 的值。其中每一个 I_z 和 I_x 的值都是由 `npt=1000000` 个数据根据式4计算得到的平均值。它们的直方图如下：



a) I_z **的数据分布直方图** **b)** I_x **的数据分布直方图**

图 3 I_z **和** I_x **的直方图**

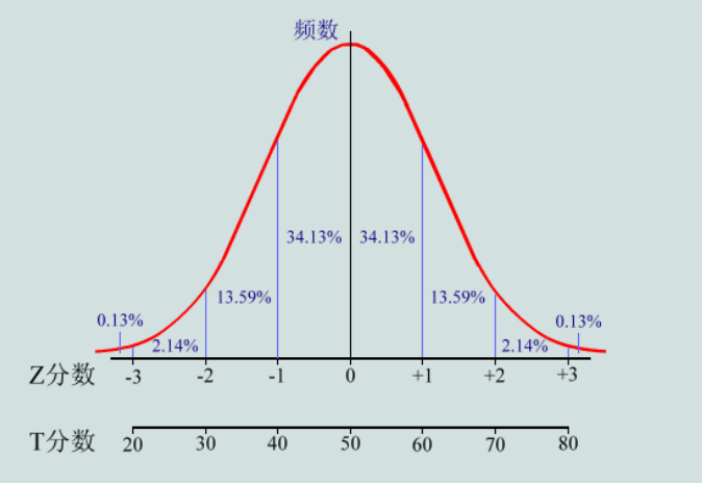


图 4 正态分布图像

定理 5.1. ***（中心极限定理）****样本的平均值约等于总体的平均值。不管总体是什么分布，任意一个总体的样本平均值都会围绕在总体的整体平均值周围，并且呈正态分布。*

从直方图中可以看到，I_z 和 I_x 的分布与正态分布（图4）十分接近，这与中心极限定理（**定理5.1**）是符合的。每一次计算得到的 I_z 和 I_x 数据都是`npt=1000000`个点对应的被积函数的平均值乘以被积空间的体积得到的，即一个样本平均值，程序中进行了 5000 此计算得到了 5000 个样本平均值，这些平均值呈现正态分布的特性与从定理5.1出发得到的预期结果符合地很好。