



兰州大学

结项论文

论文题目（中文） 磁约束下的永磁体棒复摆运动探究

论文题目（英文） Complex Pendulum Motion of permanent

magnet-rod under magnetic field confinement

学生姓名 王佳栋、李石宣、王喜波、姜国文

指导教师 白雪

学 院 物理科学与技术学院

专 业 物理类

年 级 2022 级

兰州大学教务处

诚信责任书

本人郑重声明：本人所呈交的论文（设计），是在导师的指导下独立进行研究所取得的成果。论文（设计）中凡引用他人已经发表或未发表的成果、数据、观点等，均已明确注明出处。除文中已经注明引用的内容外，不包含任何其他个人、集体已经发表或未发表的论文。

本声明的法律责任由本人承担。

论文作者签名： 王琳琳 日期： 3/13/2024

关于论文（设计）使用授权的声明

本人在导师指导下所完成的论文及相关的职务作品，知识产权归属兰州大学。本人完全了解兰州大学有关保存、使用论文（设计）的规定，同意学校保存或向国家有关部门或机构送交论文的纸质版和电子版，允许论文被查阅和借阅；本人授权兰州大学可以将本论文（设计）的全部或部分内容编入有关数据库进行检索，可以采用任何复制手段保存和汇编本论文（设计）。本人离校后发表、使用论文（设计）或与该论文（设计）直接相关的学术论文或成果时，第一署名单位仍然为兰州大学。

本论文（设计）研究内容：

可以公开

不宜公开，已在学位办公室办理保密申请，解密后适用本授权书。

（请在以上选项内选择其中一项打“√”）

论文作者签名： 王琳琳 导师签名： 白雪

日期： 3/13/2024 日期： 3/13/2024

磁约束下的永磁体棒复摆运动探究

中文摘要

磁场的约束是一个非常重要的研究课题，但同时也是一个十分复杂的问题；目前还没有求解复杂磁场系统力学的解析方法，只能通过数值计算来求解，现有的数值积分算法主要有 Monte Carlo 算法、Adaptive complex Simpson 算法、Gauss 积分、Romberg 积分等。它们均存在优势与短板，同时在低算力下也很难对这个问题进行求解。于是如何设计、优化算法就成了不可避免地问题。但通过实验对永磁体在磁场约束下的复摆运动进行分析，可以给出一些定性的实验规律。

关键词：磁场；复摆；数值分析

Complex Pendulum Motion of permanentmagnet-rod under magnetic field confinement

Abstract

Magnetic field confinement is a very important research topic, but it is also a very complicated problem. At present, there is no analytical method to solve the mechanics of complex magnetic field systems, which can only be solved by numerical calculation. The main algorithm to solve this are Monte Carlo algorithm, Adaptive complex Simpson algorithm, Gauss Integral, Romberg Integral and so on. All of them have their goods and shorts, and it is difficult to work when the arithmetic power is low. so how to design the algorithm becomes an inevitable problem. By analyzing the complex pendulum motion of permanent magnet constrained by magnetic field, some useful qualitative experimental rules can be given.

Keywords: magnetic field; complex pendulum; numerical analysis

目 录

中文摘要	I
英文摘要	II
第一章 绪 论	1
第二章 理论分析.....	2
2.1 磁场	2
2.2 平面复摆运动	3
2.2.1 可行性推导	3
2.2.2 顶点位置	4
2.2.3 转动惯量	4
2.2.4 运动方程	5
2.3 圆锥摆.....	7
2.4 其他摆动形式及因素.....	8
2.4.1 进动、章动与过渡	8
2.4.2 微小因素	8
2.4.3 其他	8
第三章 预实验及其结果	9
3.1 实验装置.....	9
3.2 实验方法.....	9
3.3 结论	10
第四章 正式实验以及结果分析.....	11
4.1 实验装置.....	11
4.2 实验方法.....	11
4.3 实验数据以及分析	12
第五章 数值计算与仿真模拟	15

5.1 数值计算.....	15
5.2 COMSOL 模拟	16
第六章 总结与展望.....	18
参考文献	19
附录	20
A.1 实验器材参数	20
A.2 实验数据.....	20
A.3 源代码头文件	26
A.3.1 Romberg.h	26
A.3.2 Gauss.h	41
A.3.3 Simpson.c	47
致谢	51

图 目 录

图 2.1 magneticfield	2
图 2.2 force on currentloop	3
图 2.3 rotation interia.....	4
图 2.4 equation of motion	5
图 2.5 collision.....	5
图 2.6 force	6
图 3.1 magnet.....	9
图 3.2 摄像法（左）和声波法（右）	10
图 3.3 两种方法的实验结果.....	10
图 4.1 实验装置	11
图 4.2 滚动准直法.....	11
图 4.3 重复性实验	12
图 4.4 总体数据拟合	12
图 4.5 改变介质板厚	13
图 4.6 改变磁场分布	13
图 4.7 改变磁棒长度	13
图 5.1 Z-magneticfield	15
图 5.2 Z-24	16
图 5.3 COMSOL 模拟	16
图 5.4 COMSOL 模拟结果.....	17

表 目 录

表 A.1 实验器材参数.....	20
表 A.2 实验数据	20

第一章 絮 论

本课题选自 2023 年中国大学生物理学术竞赛第八题。题目的叙述如下：

Euler's Pendulum 欧拉摆

Take a thick plate of non-magnetic material and fix a neodymium magnet on top of it. Suspend a magnetic rod (which can be assembled from cylindrical neodymium magnets) underneath it. Deflect the rod so that it touches the plate only with highest edge and release it. Study the motion of such a pendulum under various conditions.

取一块厚的、无磁性材料制成的板，并在其上面固定一个钕磁铁。在板的下方悬挂一根磁性杆（可以由多块圆柱形钕磁铁组装制成）。使杆偏转至它只有最顶部的边沿接触到板，然后将其释放。研究这种摆在各种条件下的运动。

项目以此题为基础，探究了磁场约束下永磁体棒的运动，得到了棒的复摆运动周期与磁场分布、棒的长度和介质板厚度的定性关系，提供了大量数据，为经验公式的创建提供了必要支持。

对于复杂系统的求解目前没有解析解法，只能使用数值算法。项目从最基本的 Biot–Savart 定律出发建立微分方程，通过数值积分的方式求解，得到了磁场在空间中的分布。通过 COMSOL 仿真软件模拟物理场，得出了在棒长较短时与实验相符合的结果。

通过文献调研，在此之前没有任何相关文献记载此类问题的研究，项目建立在自主创新的基础上对此问题进行了较为全面的探究。

本文的主要内容分为以下几个方面：

第二章具体推导了永磁体的磁场分布以及受力情况，并给出了数值算法的基本思路。同时也考虑了相关因素对于本实验的影响。

第三章给出了预实验的基本方法以及得到的结果。为正式实验找寻到了更为合适的方式。

第四章介绍了正式实验的过程、数据的具体分析以及从数据及实验过程中可以得到的结论。同时介绍了项目组在实验中发明的一种比较实用的一种准直方法——“滚动准直法”。

第五章根据实验的各类参数设计了数值算法计算了磁场的分布，同时进行了物理场仿真模拟，对问题有了进一步了解。

最后第六章对整个项目进行回顾、总结与展望。

第二章 理论分析

2.1 磁场

Biot-Savart law [1] 是描述恒定电流产生的磁场的方程，这是一个近似方程。因为根据量子力学，磁性起源于轨道电流和自旋电流且自旋电流占有很大一部分。但是其与安培定律和高斯定理是相容的，可以说其是磁场近似计算非常好的方程。本项目以此定律为基础建立了永磁体的磁场分布方程。

总体来说，形状及充磁方向一致的永磁体磁场分布大致相同，不同的是相对强度的大小。对于圆柱体永磁体，假设其磁化电流均匀分布在侧面，磁化电流密度为 i' 。运用毕奥-萨伐尔定律来计算磁场的空间分布。

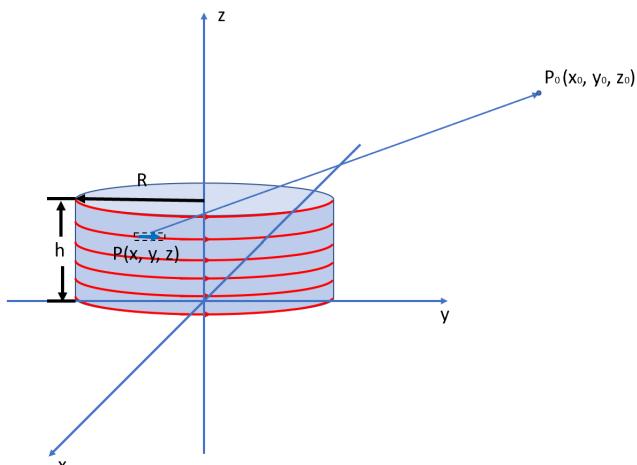


图 2.1 magneticfield

Biot-Savart 定律的微分形式（考虑磁介质）为：

$$d\mathbf{B}(\mathbf{r}) = \frac{\mu}{4\pi} \frac{Id\ell \times \mathbf{r}}{|\mathbf{r}|^3} \quad (1)$$

在图2.1中：

$$\begin{aligned} \mathbf{r} &= (x_0 - x, y_0 - y, z_0 - z) \\ d\ell &= (dx, dy, 0) \end{aligned} \quad (2)$$

因此：

$$d\ell \times \mathbf{r} = \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ dx & dy & 0 \\ x_0 - x & y_0 - y & z_0 - z \end{vmatrix} = (z_0 - z)dy\mathbf{i} - (z_0 - z)dx\mathbf{j} + [(y_0 - y)dx - (x_0 - x)dy]\mathbf{k} \quad (3)$$

对 x, y, z 使用极坐标表示，并带入上式，就可以得到 P_0 点的磁感应分量：

$$\begin{cases} x = R\cos\theta, & 0 \leq \theta \leq 2\pi \\ y = R\sin\theta, & 0 \leq \theta \leq 2\pi \\ z = z, & 0 \leq z \leq h \\ K = [(x_0 - R\cos\theta)^2 + (y_0 - R\sin\theta)^2 + (z_0 - z)^2]^{\frac{3}{2}} \end{cases} \quad (4)$$

$$\begin{cases} B_x = \frac{\mu_0 i'}{4\pi} \int_0^h \int_0^{2\pi} \frac{(z_0 - z)R\cos\theta}{K} d\theta dz \\ B_y = \frac{\mu_0 i'}{4\pi} \int_0^h \int_0^{2\pi} \frac{(z_0 - z)R\sin\theta}{K} d\theta dz \\ B_z = \frac{\mu_0 i'}{4\pi} \int_0^h \int_0^{2\pi} \frac{-(y_0 - R\sin\theta)R\sin\theta - (x_0 - R\cos\theta)R\cos\theta}{K} d\theta dz \end{cases}$$

$$I = i' dz$$

经调查 [2]：

$$i' = \frac{B_\gamma}{\mu_0} \quad (5)$$

其中， B_γ 是剩余磁化强度，这是永磁体的一个参数。

2.2 平面复摆运动

2.2.1 可行性推导

对于项目研究的系统，介质板可以被视为是无限大的，这样磁场就是轴对称的。如图2.2，取任意关于运动平面（假设它是平面运动，只要没有垂直于平面的分力，摆动就是在平面内的，即假设成立）位置对称的电流元 $Id\ell_1 = I(x_0, y_0, z_0)$, $Id\ell_2 = I(x_0, -y_0, -z_0)$ ，这两点处的磁感应强度分别为 $B_1 = (B_x, B_y, B_z)$, $B_2 = (-B_x, B_y, B_z)$ 。可以得到 $Id\ell_1 \times B_1 + Id\ell_2 \times B_2 = 0\mathbf{i} + 2(z_0 B_x - x_0 B_z)\mathbf{j} + 2(x_0 B_y - y_0 B_x)\mathbf{k}$ 。所以棒是可以在平面运动的。

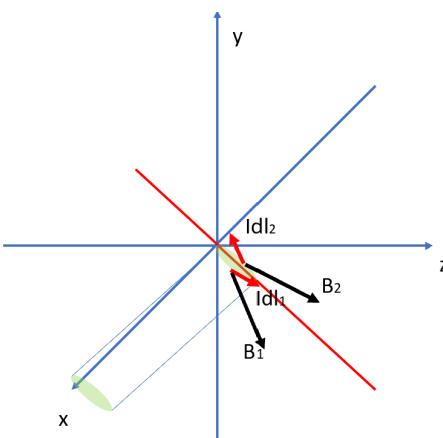


图 2.2 force on currentloop

2.2.2 顶点位置

上面的推导是建立在开始时棒的轴与磁铁的轴重合的基础上的，但这是有条件的。只有当介质板的厚度达到一定程度才可以实现。实验上可以得到这个结果，在第四章中项目组对磁场进行参数绘制，定性给出了这个结果。

2.2.3 转动惯量

转动惯量可用定义及平行轴定理算出，如图2.3：

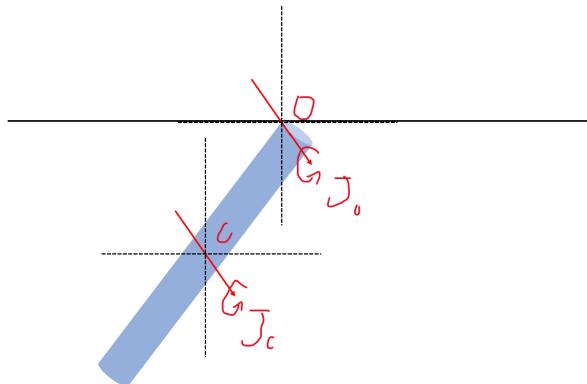


图 2.3 rotation interia

设棒的半径为 R，长为 l，密度均匀且为 ρ ，质量 $m = \rho\pi R^2 l$ 。则：

$$\begin{aligned}
 J_c &= \int \int \int_{\Omega} (x^2 + y^2) \rho \, dx \, dy \, dz \\
 &= \rho \int_{-\frac{l}{2}}^{\frac{l}{2}} \int_{-R}^R \int_{-\sqrt{R^2-x^2}}^{\sqrt{R^2-x^2}} (x^2 + y^2) \, dz \, dx \, dy \\
 &= \rho\pi R^2 l \left(\frac{R^2}{4} + \frac{l^2}{12} \right) \\
 &= m \left(\frac{R^2}{4} + \frac{l^2}{12} \right)
 \end{aligned} \tag{6}$$

于是：

$$J_o = m \left(\frac{5R^2}{4} + \frac{l^2}{3} \right) \tag{7}$$

2.2.4 运动方程

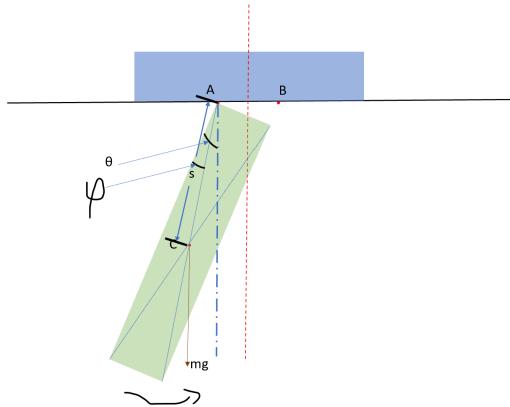


图 2.4 equation of motion

要求运动的微分方程，首先要知道受力，其中比较困难的是阻力和磁场力。先分析阻力，一般的处理方法是认为阻力与速度成正比，这里用角速度，与棒有关的那个常数（由受力面积、形状等决定）计入阻力系数，记为 γ ，则 $f = \gamma \frac{d\theta}{dt}$ 。

对于磁场力，只能做定性分析，给出解析式比较困难。根据对称性，磁场力是 θ 的周期函数。可以设为 $\mu(\theta - \varphi)$ 。加上重力，记为 $F(\theta) = mg \sin(\theta - \varphi) + \mu(\theta - \varphi)$ 。其中 θ 、 φ 的物理意义如图2.4。

考虑 $\frac{1}{4}$ 周期，因为这里涉及到一个碰撞问题，如图2.5：

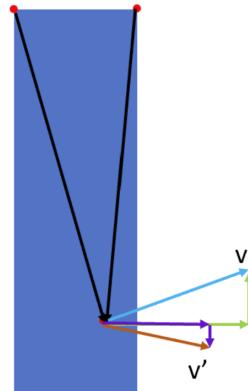


图 2.5 collision

可以预想，在棒在与介质板接触的两个顶点（图2.5中红色的点）切换的瞬间会有一个与介质板的碰撞，假设这个碰撞会使垂直于板的动量分量损失；而水平方向上，也会由于摩擦力的作用而使水平动量发生改变。这是必须的，因为要保证质点与顶点的连线与速度方向垂直。这两者的变化并不是独立的，主导作用是垂直方向，因为摩擦力是约束力，它应响应垂直方向的速度变化（但这也包含了一个摩擦力的临界问题，棒和介质板之间可能会有相对滑动）。于是，假设它可以类比于一维的碰撞，有个恢复系数 e ，这样就可以通过

观察摆幅的变化来判断 e 的大小，当然，前提是得到势场的空间分布，因为是磁场和重力场耦合，所以这会复杂一些。

对于图2.4, 由转动定理：

$$J_0 \frac{d^2\theta}{dt^2} + \gamma \frac{d\theta}{dt} + F(\theta) = 0 \quad (8)$$

对于小角度（摆动的后期），可以近似 $\sin(\theta - \varphi) = \theta - \varphi$ ，同时令 $\theta - \varphi = \phi$ ，得到下列方程：

$$J_0 \frac{d^2\phi}{dt^2} + \gamma \frac{d\phi}{dt} + mgs\phi = -\mu(\phi) \quad (9)$$

解为：

$$\phi = \begin{cases} C_1 e^{r_1 t} + C_2 e^{r_2 t} - e^{r_1 t} \int \frac{U_{21}\mu(\phi)}{\det U} d\phi - e^{r_2 t} \int \frac{U_{22}\mu(\phi)}{\det U} d\phi, & r_1 \neq r_2 \\ (C_1 + C_2 t)e^{rt} - e^{rt} \int \frac{U_{21}\mu(\phi)}{\det U} d\phi - te^{rt} \int \frac{U_{22}\mu(\phi)}{\det U} d\phi, & r_1 = r_2 = r \end{cases} \quad (10)$$

其中， $r_1 = (-P + \sqrt{P^2 - 4Q})/2$, $r_2 = (-P - \sqrt{P^2 - 4Q})/2$, $P = \frac{\gamma}{J_0}$, $Q = \frac{mgs}{J_0}$,

$$U = \begin{cases} \begin{bmatrix} e^{r_1 t} & e^{r_2 t} \\ r_1 e^{r_1 t} & r_2 e^{r_2 t} \end{bmatrix}, & r_1 \neq r_2 \\ \begin{bmatrix} e^{rt} & te^{rt} \\ re^{rt} & (1+rt)e^{rt} \end{bmatrix}, & r_1 = r_2 = r \end{cases}, \quad , U_{ij} \text{ 是 } U \text{ 的代数余子式。}$$

上面的方程用来计算是比较困难的，下面进行简化：

假设能量的损失主要发生在碰撞上面，忽略空气阻力等阻尼，于是给出运动方程：

$$J_0 * \frac{d^2\alpha}{dt^2} + F_L(\alpha) = 0 \quad (11)$$

其中 J_0 已由上面计算给出，为 $m(\frac{5R^2}{4} + \frac{l^2}{3})$ 。

下面将给出 $F_L(\alpha)$ ，如图2.6：

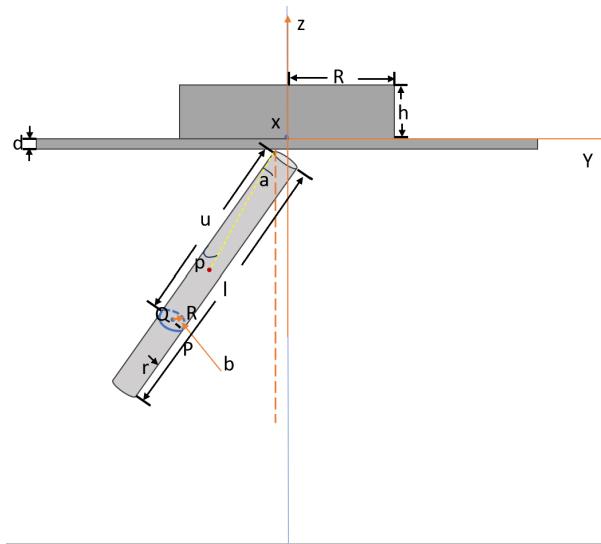


图 2.6 force

用 x,y,z 来代替 x_0, y_0, z_0, z 用 w 代替，则磁场为：

$$\begin{cases} B_x = \frac{\mu_0 i'}{4\pi} \int_0^h \int_0^{2\pi} \frac{(z-w)R\cos\theta}{K} d\theta dw \\ B_y = \frac{\mu_0 i'}{4\pi} \int_0^h \int_0^{2\pi} \frac{(z-w)R\sin\theta}{K} d\theta dw \\ B_z = \frac{\mu_0 i'}{4\pi} \int_0^h \int_0^{2\pi} \frac{-(y-R\sin\theta)R\sin\theta - (x-R\cos\theta)R\cos\theta}{K} d\theta dw \end{cases} \quad (12)$$

其中， $K = [(x-R\cos\theta)^2 + (y-R\sin\theta)^2 + (z-w)^2]^{\frac{3}{2}}$

P 点的坐标： $P(0, r + \mu\sin\alpha, -d - \mu\cos\alpha)$

同理， $Q(0, r + \mu\sin\alpha - r * \cos\alpha, -d - \mu\cos\alpha - r * \sin\alpha)$

$R(r * \sin\alpha, r + \mu\sin\alpha - r * \cos\alpha + r * \cos\beta * \cos\alpha, -d - \mu\cos\alpha - r * \sin\alpha + r * \cos\beta * \sin\alpha)$

$\mathbf{dl} = (r * \cos\beta, -r * \sin\beta * \cos\alpha, -r * \sin\beta * \sin\alpha) * d\beta$

于是，

$$\begin{cases} F_x = i'' * \int_0^l \int_0^{2\pi} (dl_y \times B_z - dl_z \times B_y) d\beta du \\ F_y = i'' * \int_0^l \int_0^{2\pi} (dl_z \times B_x - dl_x \times B_z) d\beta du \\ F_z = i'' * \int_0^l \int_0^{2\pi} (dl_x \times B_y - dl_y \times B_x) d\beta du \end{cases} \quad (13)$$

由上面的对称性分析可以得到 $F_x = 0$ ，于是 $F_L(\alpha) = i'' * \int_0^l \int_0^{2\pi} z * (dl_z \times B_x - dl_x \times B_z) + y * (dl_x \times B_y - dl_y \times B_x) d\beta du - mg(\frac{l}{2} * \cos\alpha + r * \sin\alpha)$

于是角加速度为

$$\frac{d^2\alpha}{dt^2} = -\frac{F_L(\alpha)}{J_0} = -\frac{i' * i''}{J_0} * \frac{F_L(\alpha)}{i' * i''} + \frac{g}{\frac{5R^2}{4} + \frac{l^2}{3}} * (\frac{l}{2} * \cos\alpha + r * \sin\alpha) \quad (14)$$

未知量有恢复系数 e。 $(\alpha_0$ 为 90 度)。

这个积分非常复杂，它是两个二重积分的结合，用数学软件可以给出数值解，但却无法给出运动方程的解，但是可以通过迭代算法来进行周期的求解。离散化可以用下面的方程组表示。根据实验得到的数据，可取 $\psi = 0.001s$ 。

$$\begin{cases} \alpha(t + \psi) = \alpha(t) + \psi * w(t + \psi/2) \\ w(t + \psi/2) = w(t - \psi/2) + \psi * \beta(t) \\ \beta(t) = \beta(\alpha(t)) \\ \alpha(0) = 90 \\ w(\psi/2) = w(0) + \psi/2 * \beta(0) \end{cases} \quad (15)$$

2.3 圆锥摆

除了复摆这种简单的运动形式外，还有圆锥摆这种运动形式。实验上可以观察到两种圆锥摆的摆动形式——1. 接触点不动。2. 接触点做圆周运动。对于这两种圆锥摆动形式，都可以根据向心力公式 $F_c = \int_{\Omega} \rho \omega^2 r d\Omega$ ，其中 Ω 表示棒占据的空间，来计算圆锥摆在稳定时的位置。而这只是在理想条件下的；对接触点做圆周运动的摆动形式，受力得是对称的，这

要求转轴与上方磁铁的轴共线。对于顶点不变的摆动，可能存在也可能不存在这样的位置。由于器材等的限制，本项目未对此问题做进一步的研究。

2.4 其他摆动形式及因素

上述的摆动形式是在理想的情况下才能观察到，实际很难看到；下面将定性分析实际中的复杂摆动。

2.4.1 进动、章动与过渡

由于棒与板之间存在摩擦，摆动的能量就会损耗，这会使圆锥摆向复摆过渡；这种过渡的结果就是产生了进动和章动。因为圆锥摆会退化为平面的摆动，但它很难达到不偏离轴平面（定义通过轴的复摆摆动平面为轴平面），于是根据上面的受力分析，会有一个指向轴的磁场力，这会产生一个力矩，这个力矩的作用效果就是使棒进动。但如果它大了或小了，那么就变成了章动。

2.4.2 微小因素

除了重力、磁力等比较大的影响因素外，还有许多微小的影响因素，这里列举三个。

第一，地磁场的作用。这个在要求精度不高的情况下可以忽略。地磁场的大小为 $0.5 - 0.6 Gs$ ，而钕磁铁一般为 $12200 - 14800 Gs$ ，相比之下，可以忽略地磁。

第二，科氏力。由于地球是非惯性系，棒在运动过程中会受到科氏力的作用。取最大的角度使 $F_k = 2v\omega$ ，这样近似结果约为 $10^{-4} N$ 。实验上，在控制无垂直于振动面的速度情况下，摆顺逆时针进动出现的概率近似相等。说明科氏力的影响比较小。

第三，电磁感应。在棒摆运动的过程中，它产生的磁场会反复磁化介质板和上方磁铁，这会产生电磁波损耗能量。但实际上它非常小，可以忽略其对实验的影响。

2.4.3 其他

上面考虑的是上方圆柱形磁铁的情形，形状不同的磁铁只需重新计算磁场即可，其他可以类比。对于顶点的位置，实际上在摆动过程中其并不是固定的。而像板的厚度，材料，棒的长度等影响因素其实包含在了微分方程中，它们是作为参数的形式起作用的。

第三章 预实验及其结果

3.1 实验装置

如图3.1是实验中用到的磁铁，其中较长的是作为棒的，还有两个与之相同的，可以拼接在一起以改变棒长。

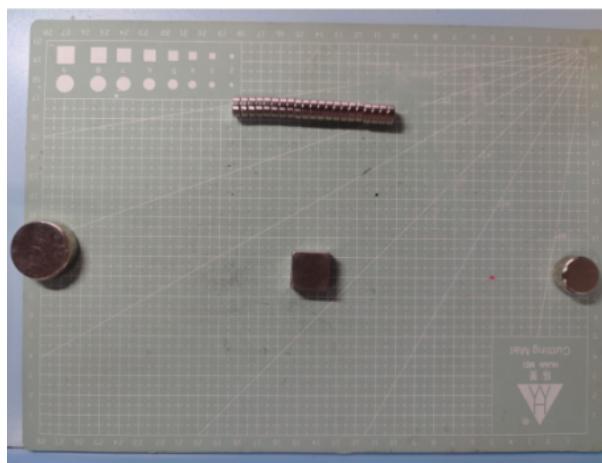


图 3.1 magnet

对于实验中用到的器材的参数请见附录A.1中的表A.1。

预实验的实验平台比较简陋，亚克力板是由两个铁架台夹住的。但这不影响预实验的总体效果。

3.2 实验方法

项目考虑到数据的收集方式对实验结果会有一定的影响，于是在预实验中对比了两种方法——1. 摄像法。以手机摄像功能对实验过程进行录像，后续通过视频处理软件数出两个“周期”的周期的帧数，进而得出周期的大小。2. 声波法。通过手机记录摆动过程中的音频数据，由于棒和板碰撞的过程中响度较大，所以可以通过声波振幅谱中峰值的时间间隔得出周期。

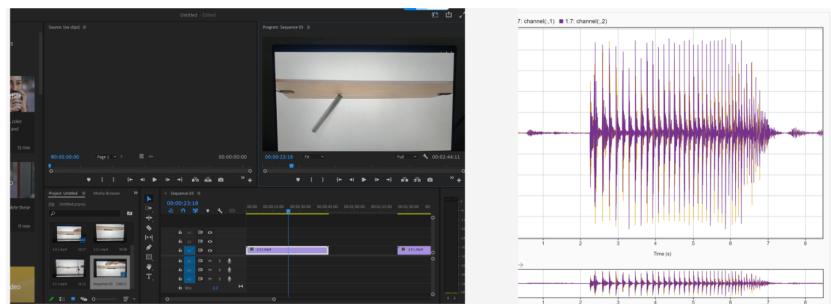


图 3.2 摄像法（左）和声波法（右）

两种方法均有一个共同的缺点——无法准确判断碰撞的时刻。但是通过实验得到声波法的稳定性和可操作性较摄像法更好，如图3.3。因此选择其作为正式实验的实验方法。

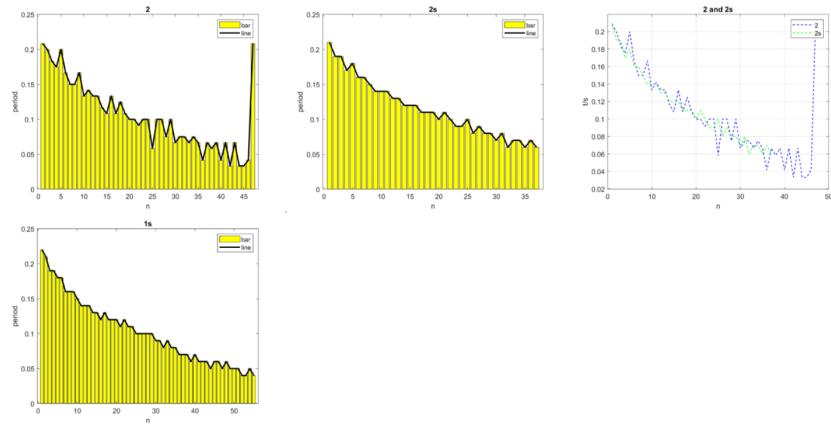


图 3.3 两种方法的实验结果

3.3 结论

- 介质板只有在厚度不是太小时，才能使轴线成为一个稳定的平衡点。
- 比较预实验采用的两种数据处理方法，声波法处理误差更小，实验数据相对稳定。
- 随着倾角的减小，摆的周期递减；当角度小到一定程度时，摆周期将会迅速减小为零。
- 控制相关因素，摆顺逆时针摆动都会出现，说明科氏力影响较小。

第四章 正式实验以及结果分析

4.1 实验装置

实验装置如图4.1, 实验中发现亚克力板晃动较为剧烈, 因此在板的两边加了重物以保证板不会剧烈震动, 图中并未体现。同时图中的电源是用以控制磁铁释放的装置。其连接一个电磁铁。

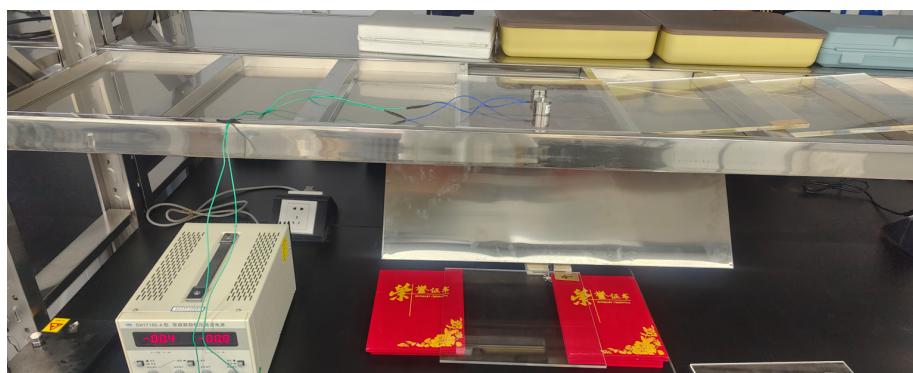


图 4.1 实验装置

4.2 实验方法

实验中发现由小磁铁拼成的磁棒不是严格共轴的, 因此项目组想到了如图4.2方法进行准直。并称为“滚动准直法”。

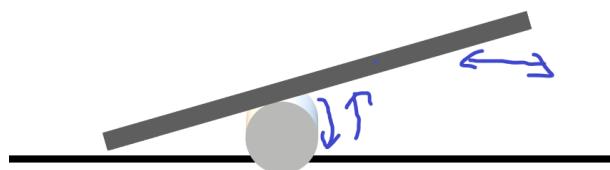


图 4.2 滚动准直法

4.3 实验数据以及分析

具体实验数据以及命名规则见附录A.2中的表A.2。其中数值按列排布，代表两次碰撞之间的时间间隔。

现对数据做如下分析：

实验对同条件下的摆动进行三次测量，测量结果如图4.3。

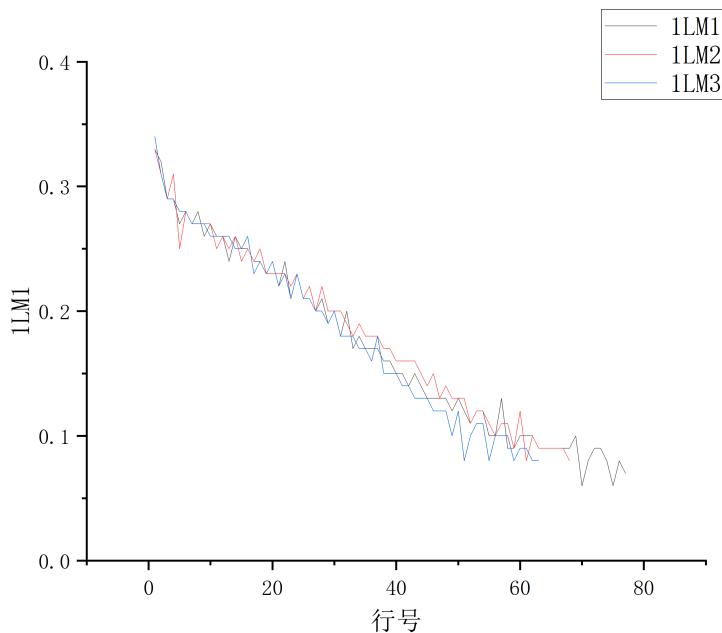


图 4.3 重复性实验

可见，实验具有良好的可重复性。

见图4.4，周期的衰减趋势更加趋向于2次（一次的 $R^2 \approx 0.96$ ，而二次的 $R^2 \approx 0.99$ ）。但如果考虑具体计算的话，线性拟合会比较有优势。

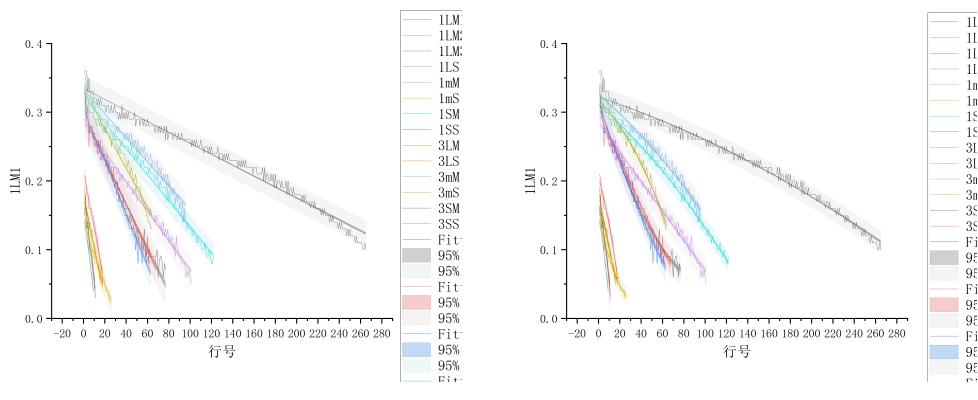


图 4.4 总体数据拟合

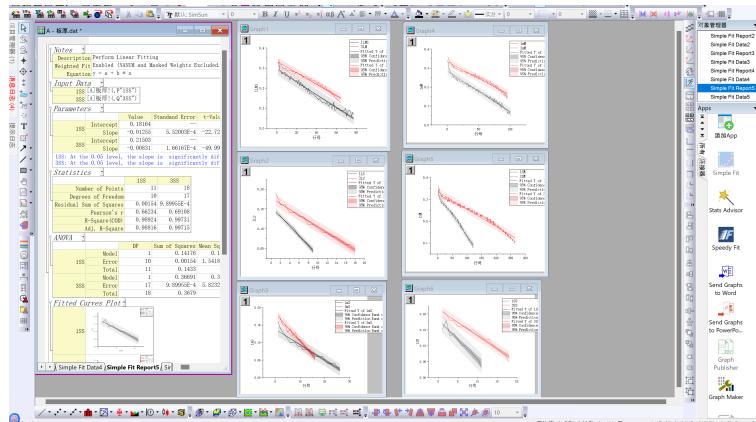


图 4.5 改变介质板厚

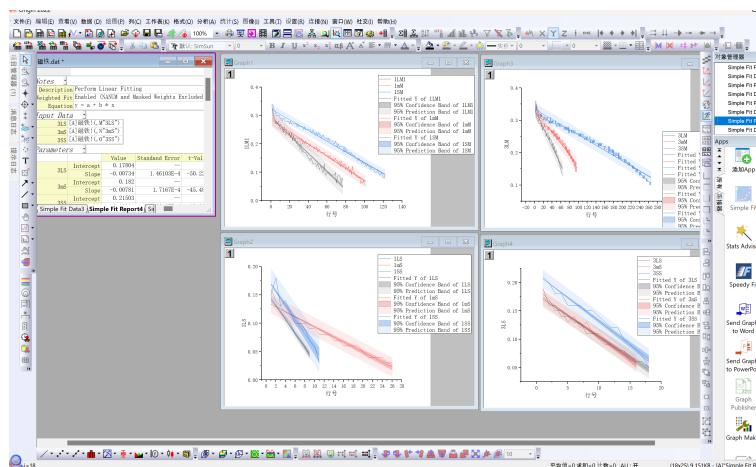


图 4.6 改变磁场分布

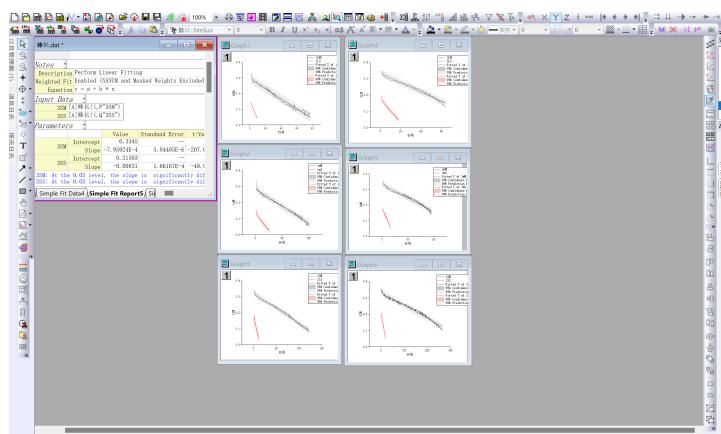


图 4.7 改变磁棒长度

如图4.5、4.6、4.7分别给出控制介质板厚度、磁场分布、棒长的结果，从对比之中可以得到一下结果：

- 介质板越厚，相同倾角，摆动的周期越大；

- 磁铁对磁棒的影响比较复杂，现有的数据无法做出较为严谨的判断；
- 棒越长，周期越大，周期衰减越慢。

第五章 数值计算与仿真模拟

5.1 数值计算

项目通过尝试 Monte Carlo 算法、Adaptive complex Simpson 算法、Gauss 积分、Romberg 积分等 [3]。综合考虑效率、精确度等，可行性最高的是 Romberg 积分。但是个人电脑的算力只能给出小数点后 4-5 位的磁场精度，精度达不到计算磁棒受力的要求。但通过数值计算，可以解决上述问题之一——介质板只有在厚度不是太小时，才能使轴线成为一个稳定的平衡点。

如图5.1和5.2，随着 z 的增大，轴逐渐成为磁场的极大值点。

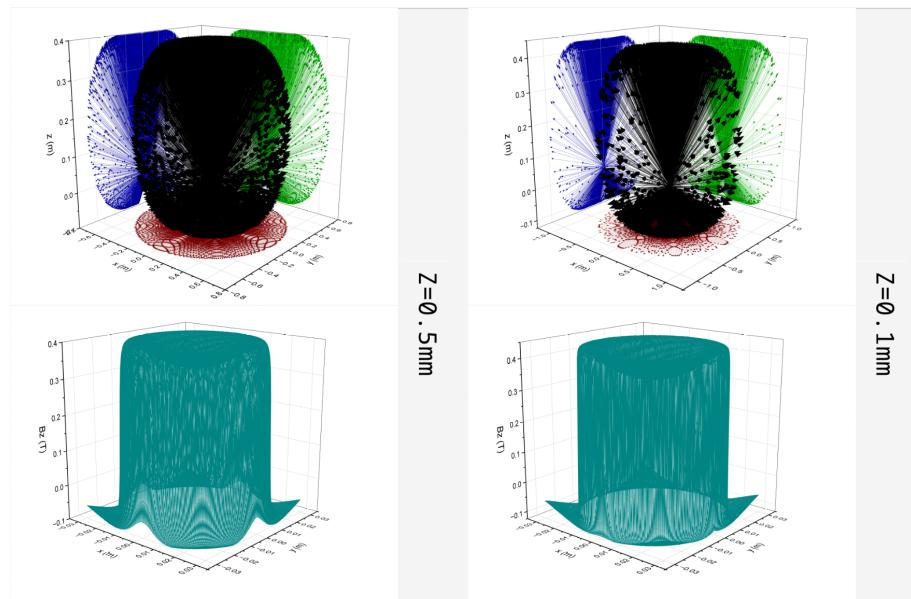


图 5.1 Z-magneticfield

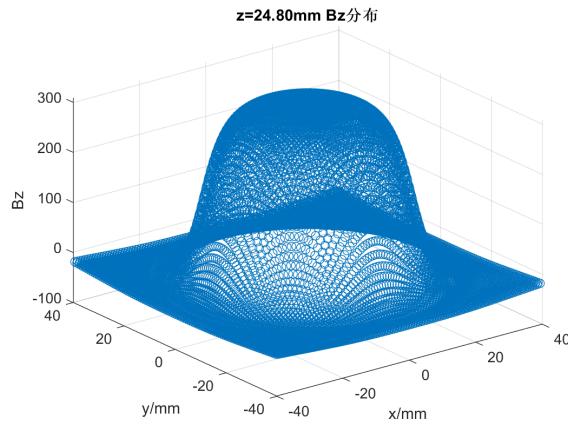


图 5.2 Z-24

5.2 COMSOL 模拟

通过 COMSOL 模拟物理场（图5.3），可以得到棒长与周期的定性关系——周期随棒长增大而增加（如图5.4）。同时在棒长较短时，仿真与实验拟合较好。可能的原因是当棒长较长时，棒与介质板接触不紧密导致的。

但是板厚的变化对周期的影响模拟结果并没有体现。

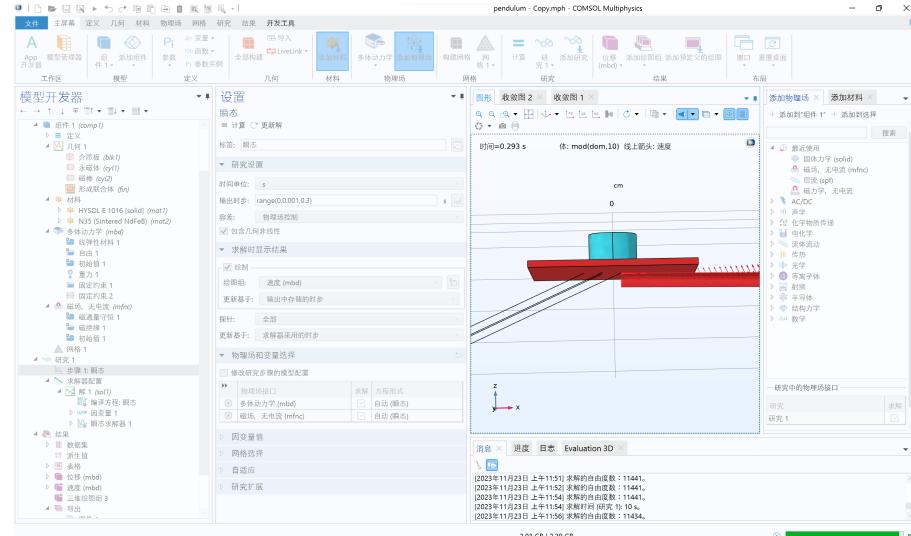


图 5.3 COMSOL 模拟

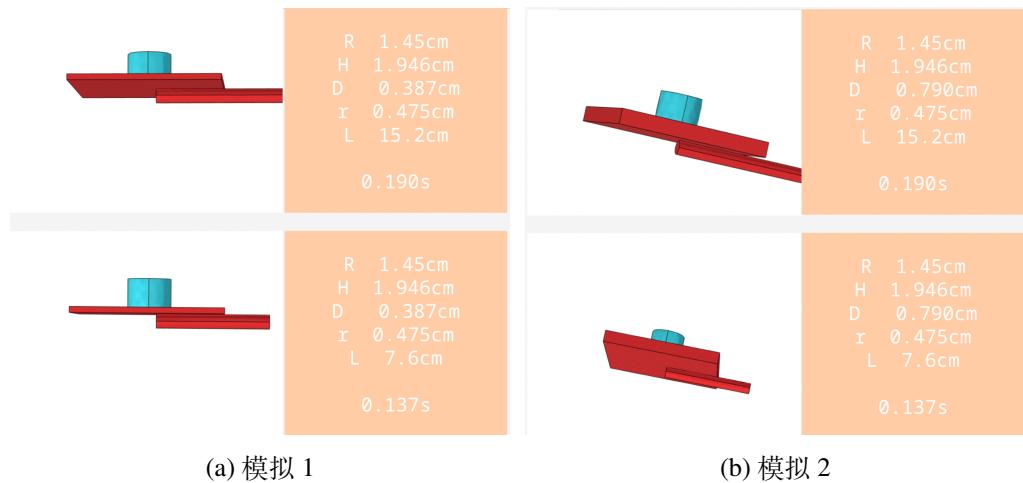


图 5.4 COMSOL 模拟结果

第六章 总结与展望

研究利用多种方法对欧拉摆这个复杂的物理体系进行分析,得到了许多有用的结果。但由于实验器材、个人计算机算力等问题,得到的结果都十分粗糙,理论得不到完备的检验,模拟结果与实验的不符也难以进一步分析、验证。总体来说有以下几个主要问题:

1. 算力不够导致磁场的数值解只能精确 mT 的量级,无法满足力场计算的要求,因此数值上无法对周期进行求解。
2. 对 COMSOL 并不熟练,导致出现了接触点处贯穿的现象。
3. 数值算法本身近似的缺点导致磁场本有的对称性破坏,对结果造成很难预测的结果。
4. 数值算法难以处理奇点问题,毕奥-萨法尔定律的缺陷在数值计算中是存在的,即电流元模型在距离很近时是不成立的。

期待进一步的研究解决它们。

参考文献

- [1] Wikipedia. Biot–savart law[Z], 2024. https://en.wikipedia.org/wiki/Biot%20%80%93Savart_law, last edited on 19 February 2024, at 16:11 (UTC).
- [2] 周恩权, 郑仲桥, 张燕红, 等. 圆柱形永磁体磁场建模及仿真研究 [J]. 河南科技, 2017, 21:139–143.
- [3] Sauer T. Numerical analysis[M]. Pearson, 2019: 253–289.

附录

A.1 实验器材参数

表 A.1 实验器材参数

亚克力板					
序号\厚度 cm	1	2	3	均值/cm	误差/cm
1	0.387	0.386	0.389	0.387	0.001
2	0.792	0.789	0.789	0.790	0.001
3	1.015	1.014	0.984	1.004	0.014

磁铁				
序号	直径/cm	误差/cm	高/cm	误差/cm
L	2.900	0.001	1.946	0.001
m	2.450	0.001	1.916	0.001

单位磁棒 (S 代表一个,M 代表两个的组合)			
次数	质量/g	长/cm	直径/cm
1	44.2968	7.600	0.950
2	44.3012	7.600	0.950
3	44.2959	7.600	0.950
均值	44.2980	7.600	0.950
误差	0.0002	0.001	0.001

A.2 实验数据

编号“ABCDEF”的意义是——A. 数据类型（表中都是声波，即 S）。B. 亚克力板的编号。C. 磁铁编号。D. 磁棒编号。E. 重复次数。F. 摆动类型（均为复摆 C）。

表 A.2 实验数据

S1LMC1	S1LMC2	S1LMC3	S1LSC	S1mMC	S1mSC	S1SMC
0.33	0.33	0.34	0.14	0.34	0.06	0.16
0.32	0.31	0.31	0.13	0.3	0.05	0.14

0.29	0.29	0.29	0.12	0.3	0.13	0.32	0.12
0.29	0.31	0.29	0.11	0.29	0.12	0.32	0.13
0.27	0.25	0.28	0.1	0.28	0.11	0.31	0.12
0.28	0.28	0.28	0.08	0.27	0.1	0.3	0.12
0.27	0.27	0.27	0.07	0.27	0.1	0.3	0.11
0.28	0.27	0.27	0.06	0.27	0.1	0.3	0.12
0.26	0.27	0.27	0.04	0.26	0.09	0.3	0.11
0.27	0.27	0.26		0.25	0.09	0.29	0.11
0.26	0.25	0.26		0.26	0.08	0.29	0.11
0.26	0.26	0.26		0.25	0.08	0.28	0.1
0.24	0.25	0.26		0.25	0.08	0.29	0.1
0.26	0.26	0.25		0.24	0.07	0.29	0.1
0.25	0.24	0.25		0.25	0.07	0.28	0.11
0.25	0.25	0.26		0.23	0.06	0.28	0.1
0.24	0.24	0.23		0.24	0.06	0.28	0.09
0.24	0.25	0.24		0.24	0.06	0.28	0.09
0.23	0.23	0.23		0.24	0.06	0.28	0.1
0.24	0.23	0.24		0.23	0.05	0.27	0.09
0.22	0.23	0.22		0.23	0.04	0.28	0.08
0.24	0.23	0.23		0.23	0.04	0.27	0.08
0.21	0.22	0.21		0.23	0.04	0.27	0.09
0.23	0.23	0.23		0.23	0.04	0.27	
0.21	0.21	0.21		0.22	0.03	0.27	
0.22	0.22	0.21		0.22	0.03	0.27	
0.2	0.2	0.2		0.22		0.27	
0.21	0.22	0.2		0.22		0.26	
0.19	0.2	0.19		0.22		0.27	
0.2	0.2	0.2		0.22		0.26	
0.18	0.2	0.18		0.21		0.26	
0.2	0.19	0.18		0.21		0.26	
0.17	0.18	0.18		0.22		0.26	
0.18	0.19	0.17		0.2		0.26	
0.17	0.18	0.17		0.21		0.25	
0.17	0.18	0.16		0.2		0.26	
0.17	0.18	0.18		0.21		0.25	
0.16	0.17	0.15		0.2		0.25	
0.16	0.17	0.15		0.2		0.25	
0.15	0.16	0.15		0.2		0.25	
0.15	0.16	0.14		0.19		0.25	
0.14	0.16	0.14		0.2		0.25	

0.15	0.16	0.13	0.19	0.24
0.14	0.15	0.13	0.19	0.24
0.13	0.14	0.13	0.19	0.25
0.13	0.15	0.12	0.18	0.24
0.13	0.13	0.12	0.19	0.24
0.13	0.14	0.12	0.18	0.23
0.12	0.13	0.1	0.19	0.24
0.13	0.13	0.12	0.17	0.23
0.12	0.13	0.08	0.18	0.24
0.11	0.11	0.1	0.18	0.23
0.12	0.12	0.11	0.17	0.23
0.12	0.12	0.11	0.17	0.22
0.1	0.11	0.08	0.17	0.23
0.1	0.1	0.1	0.17	0.23
0.13	0.11	0.1	0.17	0.22
0.09	0.11	0.1	0.16	0.22
0.09	0.09	0.08	0.16	0.22
0.1	0.12	0.09	0.16	0.22
0.1	0.08	0.09	0.16	0.21
0.1	0.1	0.08	0.15	0.22
0.09	0.09	0.08	0.16	0.21
0.09	0.09		0.15	0.21
0.09	0.09		0.15	0.21
0.09	0.09		0.14	0.21
0.09	0.09		0.15	0.2
0.09	0.08		0.14	0.2
0.1			0.15	0.21
0.06			0.14	0.2
0.08			0.14	0.19
0.09			0.13	0.2
0.09			0.14	0.19
0.08			0.12	0.19
0.06			0.14	0.19
0.08			0.12	0.19
0.07			0.13	0.19
			0.12	0.17
			0.13	0.19
			0.11	0.17
			0.12	0.18
			0.11	0.17

					0.12		0.17	
					0.1		0.17	
					0.11		0.16	
					0.09		0.17	
					0.12		0.16	
					0.1		0.16	
					0.1		0.16	
					0.09		0.15	
					0.09		0.15	
					0.08		0.15	
					0.08		0.14	
					0.08		0.15	
					0.08		0.14	
					0.06		0.14	
					0.07		0.13	
					0.06		0.14	
					0.07		0.13	
S1SSC	S3LMC	S3LSC	S3mMC	S3mSC	S3SMC			S3SSC
0.15	0.34	0.19	0.36	0.18	0.36	0.26	0.18	0.21
0.18	0.34	0.16	0.35	0.17	0.36	0.25	0.18	0.2
0.13	0.32	0.16	0.34	0.15	0.36	0.26	0.18	0.18
0.14	0.32	0.14	0.33	0.15	0.33	0.25	0.18	0.18
0.12	0.31	0.14	0.33	0.13	0.35	0.26	0.18	0.16
0.1	0.3	0.13	0.33	0.14	0.33	0.25	0.18	0.16
0.1	0.3	0.13	0.32	0.12	0.33	0.26	0.17	0.16
0.09	0.3	0.11	0.31	0.13	0.33	0.25	0.18	0.15
0.07	0.3	0.11	0.32	0.11	0.33	0.25	0.17	0.15
0.06	0.29	0.1	0.3	0.11	0.32	0.25	0.18	0.14
0.03	0.29	0.09	0.31	0.1	0.32	0.26	0.17	0.13
	0.29	0.09	0.3	0.09	0.32	0.25	0.17	0.12
	0.29	0.08	0.3	0.08	0.31	0.25	0.17	0.12
	0.29	0.08	0.3	0.08	0.32	0.24	0.17	0.1
	0.28	0.07	0.3	0.06	0.31	0.25	0.16	0.09
	0.29	0.06	0.29	0.05	0.32	0.25	0.17	0.08
	0.28	0.06	0.29		0.31	0.25	0.17	0.07
	0.27	0.05	0.29		0.31	0.24	0.16	0.05
	0.28		0.29		0.31	0.25	0.17	
	0.28		0.29		0.31	0.24	0.16	
	0.27		0.29		0.3	0.25	0.16	
	0.27		0.28		0.31	0.24	0.16	

0.27	0.29	0.31	0.24	0.16
0.27	0.28	0.3	0.24	0.16
0.27	0.28	0.3	0.24	0.15
0.25	0.28	0.31	0.24	0.16
0.26	0.28	0.3	0.25	0.15
0.26	0.28	0.3	0.23	0.16
0.26	0.28	0.31	0.24	0.15
0.26	0.27	0.3	0.24	0.15
0.25	0.28	0.3	0.24	0.15
0.25	0.27	0.3	0.23	0.15
0.24	0.28	0.29	0.24	0.14
0.25	0.27	0.3	0.23	0.15
0.24	0.27	0.29	0.24	0.14
0.24	0.27	0.31	0.23	0.15
0.24	0.27	0.29	0.24	0.14
0.24	0.26	0.3	0.23	0.14
0.23	0.27	0.29	0.23	0.14
0.23	0.26	0.3	0.23	0.14
0.22	0.27	0.29	0.23	0.13
0.23	0.26	0.3	0.23	0.14
0.22	0.27	0.29	0.23	0.13
0.21	0.25	0.29	0.23	0.14
0.22	0.27	0.29	0.23	0.13
0.21	0.25	0.29	0.22	0.13
0.2	0.26	0.29	0.23	0.13
0.2	0.25	0.3	0.22	0.13
0.21	0.26	0.28	0.23	0.12
0.19	0.24	0.29	0.22	0.13
0.19	0.26	0.29	0.23	0.12
0.19	0.24	0.29	0.22	0.13
0.19	0.25	0.28	0.23	0.12
0.18	0.24	0.29	0.22	0.12
0.17	0.25	0.29	0.22	0.12
0.17	0.24	0.28	0.22	0.12
0.16	0.25	0.29	0.22	0.11
0.16	0.23	0.28	0.22	0.12
0.15	0.24	0.28	0.22	0.11
0.15	0.24	0.29	0.21	0.11
0.14	0.23	0.28	0.22	0.11
0.14	0.23	0.28	0.22	0.11

0.13	0.24	0.28	0.22	0.11
0.13	0.22	0.28	0.21	0.1
	0.23	0.28	0.21	0.1
	0.23	0.28	0.21	0.11
	0.22	0.27	0.21	0.1
	0.22	0.28	0.21	
	0.23	0.28	0.22	
	0.21	0.27	0.2	
	0.22	0.28	0.22	
	0.21	0.27	0.2	
	0.22	0.28	0.21	
	0.2	0.27	0.21	
	0.21	0.27	0.2	
	0.2	0.28	0.2	
	0.21	0.26	0.21	
	0.19	0.28	0.2	
	0.21	0.26	0.21	
	0.19	0.27	0.19	
	0.2	0.27	0.21	
	0.18	0.27	0.19	
	0.19	0.27	0.2	
	0.18	0.26	0.2	
	0.19	0.27	0.2	
	0.17	0.27	0.19	
	0.18	0.26	0.2	
	0.17	0.26	0.19	
	0.16	0.26	0.19	
	0.17	0.26	0.2	
	0.15	0.27	0.18	
	0.17	0.26	0.2	
	0.14	0.26	0.18	
	0.16	0.26	0.19	
	0.14	0.26	0.19	
		0.26	0.18	
		0.26	0.19	
		0.26	0.18	

A.3 源代码头文件

A.3.1 Romberg.h

```

1 #ifndef __MY_EULER_H__
2 #define __MY_EULER_H__
3 /*The above is to prevent the header file from being included multiple
4 times ,
5 it can be omitted, it is best to have, any name, and ensure that it is
6 unique */
7
8 /*The following macro definition is optional, here we define the
9 parameter we need */
10
11 #define PI 3.14159265358979
12 #define E 0.001
13 #define Bg 1.20 //we resume that the neodymium magnet's remanence is
14      the averger value of 1.18-1.22T which is the N35's parameter
15 // #define mu0 4*PI*0.0000001 //the unit is T*m/A to make double
16      available , we chose to use Bg/mu0 which we called Bm.
17
18 #define Bm 3000000/PI
19 #define g 9.80 //N/kg I got it through experiment
20 #define R 0.029 //m 0.029 or 0.0245m
21 #define h 0.01946 //m 0.01946 or 0.01916m
22 #define r 0.0095 //m
23 #define l 0.076 //m
24 #define d 0.00387 //m 0.00387 or 0.00790 or 0.01004m
25 #define m 0.0443 //kg
26
27 /*The following is the definition of functions */
28 /*the functions to calculate the mgnetic field */
29 //the component of x axis
30 double m_bx(double x, double y, double z, double w, double th){
31     double K = pow( (x-R*cos(th))*(x-R*cos(th))+(y-R*sin(th))*(y-R*sin
32           (th))+(z-w)*(z-w) , 1.5 );
33     return (z-w)*R*cos(th)/K;
34 }
35 //The inner layer of the double integral
36 double m_Bx_th_add(double x, double y, double z, double w, int k,
37   double th1 , double t ){

```

```

31     double a = 0;
32     for( int i = 1; i <= (k>2? pow(2,k-2):1); i++){
33         a += m_bx(x, y, z, w, th1+(2*i-1)*t/pow(2,k-1));
34     }
35     return a*t/(k>2? pow(2,k-2):1);
36 }
37 double m_Bx_th( double x, double y, double z, double w){
38     double answer = 0;
39     double th1 = 3*E, th2 = 2*PI;
40     double t = th2-th1;
41     int M = 40;      //the initial M, if it's not enough, we'll give
42                     //you an alert and you need to enlarge M
43     double *Px;
44     Px = (double *)malloc(M*M*sizeof(double));
45
46     Px[0] = ( m_bx(x, y, z, w,th1)+m_bx(x, y, z, w, th2) )*t/2;
47     int i = 1, j = 1;
48     while(i){
49         Px[i*M+0] = (Px[(i-1)*M+0]+m_Bx_th_add(x,y,z,w,i+1,th1,t))/2;
50         while(j){
51             Px[i*M+j] = Px[i*M+j-1]+(Px[i*M+j-1]-Px[(i-1)*M+j-1])/(pow
52                           (4,j)-1);
53             if(j == i){
54                 j = 1;
55                 break;
56             } else {
57                 j++;
58             }
59         }
60         double e = fabs(Px[i*M+i]-Px[(i-1)*M+i-1]);
61         if(e<E&&i!=1){
62             answer = Px[i*M+i];
63             break;
64         } else if(i == M-1){
65             printf("x=%lf, y=%lf, z=%lf\n", x, y, z);
66             printf("Please enlarge Px !!!\n");
67         } else {
68             i++;
69         }
70     }
71 }
```

```
69
70     free(Px);
71     return answer;
72 }
73 //the magnet's x component
74 double m_Bx_add(double x, double y, double z, int k, double w1, double
75 t){
76     double a = 0;
77     for(int i = 1; i <= (k>2? pow(2,k-2):1); i++){
78         a += m_Bx_th(x, y, z, w1+(2*i-1)*t/pow(2,k-1));
79     }
80     return a*t/(k>2? pow(2,k-2):1);
81 }
82 double m_Bx(double x, double y, double z){
83     double answer = 0;
84     double w1 = 0, w2 = h;
85     double t = w2-w1;
86     int M = 40;
87     double *Hx;
88     Hx = (double *)malloc(M*M*sizeof(double));
89
90     Hx[0] = (m_Bx_th(x, y, z, w1)+m_Bx_th(x, y, z, w2))*t/2;
91     int i = 1, j = 1;
92     while(i){
93         Hx[i*M+0] = (Hx[(i-1)*M+0]+m_Bx_add(x,y,z,i+1,w1,t))/2;
94         while(j){
95             Hx[i*M+j] = Hx[i*M+j-1]+(Hx[i*M+j-1]-Hx[(i-1)*M+j-1])/(pow
96             (4,j)-1);
97             if(j == i){
98                 j = 1;
99                 break;
100            } else {
101                j++;
102            }
103        }
104        double e = fabs(Hx[i*M+i]-Hx[(i-1)*M+i-1]);
105        if(e<E&&i!=1){
106            answer = Bg*Hx[i*M+i]/(4*PI);
107            break;
108        } else if(i == M-1){
```

```

107         printf("x=%lf, y=%lf, z=%lf\n", x, y, z);
108         printf("Please enlarge Hx !!!\n");
109     } else {
110         i++;
111     }
112 }
113
114 free(Hx);
115 return answer;
116 }
117 //the component of y axis
118 double m_by(double x, double y, double z, double w, double th){
119     double K = pow((x-R*cos(th))*(x-R*cos(th))+(y-R*sin(th))*(y-R*sin
120 (th))+(z-w)*(z-w), 1.5);
121     //printf("%10.4lf%10.4lf%10.4lf%10.4lf%10.4lf%10.4lf\n", x, y, z,
122     //w, th, (z-w)*R*sin(th)/K);
123     return (z-w)*R*sin(th)/K;
124 }
125 //The inner layer of the double integral
126 double m_By_th_add(double x, double y, double z, double w, int k,
127     double th1, double t){
128     double a = 0;
129     for(int i = 1; i <= (k>2? pow(2,k-2):1); i++){
130         a += m_by(x, y, z, w, th1+(2*i-1)*t/pow(2,k-1));
131     }
132     return a*t/(k>2? pow(2,k-2):1);
133 }
134 double m_By_th(double x, double y, double z, double w){
135     double answer = 0;
136     double th1 = 3*E, th2 = 2*PI;
137     double t = th2-th1;
138     int M = 40;      //the initial M, if it's not enough, we'll give
139     //you an alert and you need to enlarge M
140     double *Py;
141     Py = (double *)malloc(M*M*sizeof(double));
142     Py[0] = (m_by(x, y, z, w, th1)+m_by(x, y, z, w, th2))*t/2;
143     int i = 1, j = 1;
144     while(i){
145         Py[i*M+0] = (Py[(i-1)*M+0]+m_By_th_add(x,y,z,w,i+1,th1,t))/2;

```

```

143     while(j){
144         Py[i*M+j] = Py[i*M+j-1]+(Py[i*M+j-1]-Py[(i-1)*M+j-1])/(pow
145             (4,j)-1);
146         if(j == i){
147             j = 1;
148             break;
149         } else {
150             j++;
151         }
152         //printf("%d,% .8lf\n", i, Py[i*M+i]);
153         double e = fabs(Py[i*M+i]-Py[(i-1)*M+i-1]);
154         //printf("%.10lf\n\n", e);
155         if(e<E&&i!=1){//for the special case
156             answer = Py[i*M+i];
157             break;
158         } else if(i == M-1){
159             printf("x=%lf, y=%lf, z=%lf\n", x, y, z);
160             printf("Please enlarge Py!!!\n");
161         } else {
162             i++;
163         }
164     }
165
166     free(Py);
167     return answer;
168 }
169 //the magnet's y component
170 double m_By_add(double x, double y, double z, int k, double w1, double
171 t){
172     double a = 0;
173     for(int i = 1; i <= (k>2? pow(2,k-2):1); i++){
174         a += m_By_th(x, y, z, w1+(2*i-1)*t/pow(2,k-1));
175     }
176     return a*t/(k>2? pow(2,k-2):1);
177 }
178 double m_By(double x, double y, double z){
179     double answer = 0;
180     double w1 = 0, w2 = h;
181     double t = w2-w1;

```

```

181     int M = 40;
182     double *Hy;
183     Hy = (double *)malloc(M*M*sizeof(double));
184
185     Hy[0] = ( m_By_th(x, y, z, w1)+m_By_th(x, y, z, w2) )*t / 2;
186     int i = 1, j = 1;
187     while(i){
188         //printf("%d\n", i);
189         Hy[i*M+0] = (Hy[(i-1)*M+0]+m_By_add(x,y,z,i+1,w1,t))/2;
190         while(j){
191             Hy[i*M+j] = Hy[i*M+j-1]+(Hy[i*M+j-1]-Hy[(i-1)*M+j-1])/(pow
192                 (4,j)-1);
193             if(j == i){
194                 j = 1;
195                 break;
196             } else {
197                 j++;
198             }
199             double e = fabs(Hy[i*M+i]-Hy[(i-1)*M+i-1]);
200             if(e<E&&i !=1){
201                 answer = Bg*Hy[i*M+i]/(4*PI);
202                 break;
203             } else if(i == M-1){
204                 printf("x=%lf, y=%lf, z=%lf\n", x, y, z);
205                 printf("Please enlarge Hy!!!\n");
206             } else {
207                 i++;
208             }
209         }
210
211         free(Hy);
212         return answer;
213     }
214
215 //the component of z axis
216 double m_bz(double x, double y, double z, double w, double th){
217     double K = pow((x-R*cos(th))*(x-R*cos(th))+(y-R*sin(th))*(y-R*sin
218         (th))+(z-w)*(z-w), 1.5);
219     return(-(y-R*sin(th))*R*sin(th)-(x-R*cos(th))*R*cos(th))/K;

```

```

219 }
220 //The inner layer of the double integral
221 double m_Bz_th_add(double x, double y, double z, double w, int k,
222                     double th1, double t){
223     double a = 0;
224     for(int i = 1; i <= (k>2? pow(2,k-2):1); i++){
225         a += m_bz(x, y, z, w, th1+(2*i-1)*t/pow(2,k-1));
226     }
227     return a*t/(k>2? pow(2,k-2):1);
228 }
229 double m_Bz_th(double x, double y, double z, double w){
230     double answer = 0;
231     double th1 = 3*E, th2 = 2*PI;
232     double t = th2-th1;
233     int M = 40;      //the initial M, if it's not enough, we'll give
234                         //you an alert and you need to enlarge M
235     double *Pz;
236     Pz = (double *)malloc(M*M*sizeof(double));
237
238     Pz[0] = ( m_bz(x, y, z, w,th1)+m_bz(x, y, z, w, th2) )*t/2;
239     int i = 1, j = 1;
240     while(i){
241         Pz[i*M+0] = (Pz[(i-1)*M+0]+m_Bz_th_add(x,y,z,w,i+1,th1,t))/2;
242         while(j){
243             Pz[i*M+j] = Pz[i*M+j-1]+(Pz[i*M+j-1]-Pz[(i-1)*M+j-1])/(pow
244                           (4,j)-1);
245             if(j == i){
246                 j = 1;
247                 break;
248             } else {
249                 j++;
250             }
251         }
252         double e = fabs(Pz[i*M+i]-Pz[(i-1)*M+i-1]);
253         if(e<E&&i!=1){
254             answer = Pz[i*M+i];
255             break;
256         } else if(i == M-1){
257             printf("x=%lf, y=%lf, z=%lf\n", x, y, z);
258             printf("Please enlarge Pz !!!\n");
259         }
260     }
261 }
```

```
256         }  
257         i++;  
258     }  
259 }  
260  
261 free(Pz);  
262 return answer;  
263 }  
264 //the magnet's z component  
265 double m_Bz_add(double x, double y, double z, int k, double w1, double  
t){  
266     double a = 0;  
267     for(int i = 1; i <= (k>2? pow(2,k-2):1); i++){  
268         a += m_Bz_th(x, y, z, w1+(2*i-1)*t/pow(2,k-1));  
269     }  
270     return a*t/(k>2? pow(2,k-2):1);  
271 }  
272 double m_Bz(double x, double y, double z){  
273     double answer = 0;  
274     double w1 = 0, w2 = h;  
275     double t = w2-w1;  
276     int M = 40;  
277     double *Hz;  
278     Hz = (double *)malloc(M*M*sizeof(double));  
279  
280     Hz[0] = ( m_Bz_th(x, y, z, w1)+m_Bz_th(x, y, z, w2) )*t/2;  
281     int i = 1, j = 1;  
282     while(i){  
283         Hz[i*M+0] = (Hz[(i-1)*M+0]+m_Bz_add(x,y,z,i+1,w1,t))/2;  
284         while(j){  
285             Hz[i*M+j] = Hz[i*M+j-1]+(Hz[i*M+j-1]-Hz[(i-1)*M+j-1])/(pow  
                (4,j)-1);  
286             if(j == i){  
287                 j = 1;  
288                 break;  
289             } else {  
290                 j++;  
291             }  
292         }  
293         double e = fabs(Hz[i*M+i]-Hz[(i-1)*M+i-1]);
```

```

294         if (e<E&&i !=1) {
295             answer = Bg*Hz[ i*M+i ]/(4 * PI );
296             break ;
297         } else if (i == M-1){
298             printf( "x=%lf , y=%lf , z=%lf \n" , x , y , z );
299             printf( "Please enlarge Hz !!!\n" );
300         } else {
301             i++;
302         }
303     }
304
305     free(Hz);
306     return answer;
307 }
308
309
310 //Fx
311 float m_fx( float b, float u, float a){
312     return ( -r*sin(b)*cos(a)*m_Bz(-r*sin(b),r*cos(b)*cos(a)+r*sin(a)-
313         r*cos(a),r*cos(b)*sin(a)-d-u*cos(a)-r*sin(a)) \
314         + r*sin(b)*sin(a)*m_By(-r*sin(b),r*cos(b)*cos(a)+r*sin(a)-r*cos(a)
315         ,r*cos(b)*sin(a)-d-u*cos(a)-r*sin(a)));
316 }
317 float m_Fx_b_add( int k, float b1, float t, float u, float a){
318     float ans = 0;
319     for( int i = 1; i <= (k>2? pow(2,k-2):1); i++){
320         ans += m_fx(b1+(2*i-1)*t/pow(2,k-1),u,a);
321     }
322     return ans*t/(k>2? pow(2,k-2):1);
323 }
324 float m_Fx_b( float u, float a){
325     float answer = 0;
326     float b1 = 3*E, b2 = 2*PI;
327     float t = b2-b1;
328     int M = 40;
329     float *Px;
330     Px = ( float *)malloc(M*M* sizeof( float ));
331     Px[0] = ( m_fx(b1,u,a)+m_fx(b2,u,a) )*t/2;
332     int i = 1, j = 1;
333     while(i) {

```

```

332     Px[ i *M+0] = (Px[ ( i -1)*M+0]+m_Fx_b_add( i +1,b1 , t , u , a )) /2 ;
333     while( j ){
334         Px[ i *M+j ] = Px[ i *M+j -1]+(Px[ i *M+j -1]-Px[ ( i -1)*M+j -1]) /( pow
335             (4 , j )-1);
336         if( j ==i ){
337             j =i ;
338             break ;
339         } else {
340             j ++;
341         }
342         float e = fabs( Px[ i *M+i ]-Px[ ( i -1)*M+i -1]);
343         if( e<E&&i !=1 ){
344             answer = Px[ i *M+i ];
345             break ;
346         } else if( i ==M-1){
347             printf( "u = %lf , a = %lf .\n" , u , a );
348             printf( "Please enlarge Px!" );
349         } else {
350             i ++;
351         }
352     }
353     free( Px );
354     return answer ;
355 }
356 float m_Fx_add( int k , float u1 , float t , float a ){
357     float ans = 0;
358     for( int i = 1; i <= (k>2? pow(2 ,k-2):1); i ++){
359         ans += m_Fx_b( u1+(2*i -1)*t /pow(2 ,k-1) ,a );
360     }
361     return ans*t /(k>2? pow(2 ,k-2):1) ;
362 }
363 float m_Fx( float a ){
364     float answer = 0;
365     float u1 = 3*E/10 , u2 = 2*1 ;
366     float t = u2-u1 ;
367     int M = 40 ;
368     float *Hx ;
369     Hx = ( float *) malloc( M*M* sizeof( float ) );
370     Hx[ 0 ] = ( m_Fx_b( u1 ,a )+m_Fx_b( u2 ,a ) )*t /2 ;

```

```

371     int i = 1, j = 1;
372     while(i){
373         Hx[i*M+0] = (Hx[(i-1)*M+0]+m_Fx_add(i+1,u1,t,a))/2;
374         while(j){
375             Hx[i*M+j] = Hx[i*M+j-1]+(Hx[i*M+j-1]-Hx[(i-1)*M+j-1])/(pow
376             (4,j)-1);
377             if(j==i){
378                 j=i;
379                 break;
380             } else {
381                 j++;
382             }
383             float e = fabs(Hx[i*M+i]-Hx[(i-1)*M+i-1]);
384             if(e<E&&i!=1){
385                 answer = Bm*Hx[i*M+i];
386                 break;
387             } else if(i==M-1){
388                 printf("a = %lf.\n", a);
389                 printf("Please enlarge Hx!");
390             } else {
391                 i++;
392             }
393         }
394         free(Hx);
395         return answer;
396     }
397
398 //Fy
399 float m_fy(float b, float u, float a){
400     return (-r*sin(b)*sin(a)*m_Bx(-r*sin(b),r*cos(b)*cos(a)+r*sin(a)-
401         r*cos(a),r*cos(b)*sin(a)-d-u*cos(a)-r*sin(a)) \
402         + r*cos(b)*m_Bz(-r*sin(b),r*cos(b)*cos(a)+r*sin(a)-r*cos(a),r*cos(
403             b)*sin(a)-d-u*cos(a)-r*sin(a)));
404 }
405 float m_Fy_b_add(int k, float b1, float t, float u, float a){
406     float ans = 0;
407     for(int i = 1; i <= (k>2? pow(2,k-2):1); i++){
408         ans += m_fy(b1+(2*i-1)*t/pow(2,k-1),u,a);
409     }

```

```

408         return ans*t/(k>2? pow(2,k-2):1);
409     }
410     float m_Fy_b(float u, float a){
411         float answer = 0;
412         float b1 = 3*E, b2 = 2*PI;
413         float t = b2-b1;
414         int M = 40;
415         float *Py;
416         Py = (float *)malloc(M*M*sizeof(float));
417         Py[0] = (m_fy(b1,u,a)+m_fy(b2,u,a))*t/2;
418         int i = 1, j = 1;
419         while(i){
420             Py[i*M+0] = (Py[(i-1)*M+0]+m_Fy_b_add(i+1,b1,t,u,a))/2;
421             while(j){
422                 Py[i*M+j] = Py[i*M+j-1]+(Py[i*M+j-1]-Py[(i-1)*M+j-1])/(pow
423                     (4,j)-1);
424                 if(j==i){
425                     j=i;
426                     break;
427                 } else {
428                     j++;
429                 }
430                 float e = fabs(Py[i*M+i]-Py[(i-1)*M+i-1]);
431                 if(e<E&&i!=1){
432                     answer = Py[i*M+i];
433                     break;
434                 } else if(i==M-1){
435                     printf("u = %lf, a = %lf.\n", u, a);
436                     printf("Please enlarge Py!");
437                 } else {
438                     i++;
439                 }
440             }
441             free(Py);
442             return answer;
443         }
444         float m_Fy_add(int k, float u1, float t, float a){
445             float ans = 0;
446             for(int i = 1; i <= (k>2? pow(2,k-2):1); i++){

```

```
447         ans += m_Fy_b(u1+(2*i-1)*t/pow(2,k-1),a);
448     }
449     return ans*t/(k>2? pow(2,k-2):1);
450 }
451 float m_Fy(float a){
452     float answer = 0;
453     float u1 = 3*E/10, u2 = 2*l;
454     float t = u2-u1;
455     int M = 40;
456     float *Hy;
457     Hy = (float *)malloc(M*M*sizeof(float));
458     Hy[0] = (m_Fy_b(u1,a)+m_Fy_b(u2,a))*t/2;
459     int i = 1, j = 1;
460     while(i){
461         Hy[i*M+0] = (Hy[(i-1)*M+0]+m_Fy_add(i+1,u1,t,a))/2;
462         while(j){
463             Hy[i*M+j] = Hy[i*M+j-1]+(Hy[i*M+j-1]-Hy[(i-1)*M+j-1])/(pow
464             (4,j)-1);
465             if(j==i){
466                 j=i;
467                 break;
468             } else {
469                 j++;
470             }
471             float e = fabs(Hy[i*M+i]-Hy[(i-1)*M+i-1]);
472             if(e<E&&i!=1){
473                 answer = Bm*Hy[i*M+i];
474                 break;
475             } else if(i==M-1){
476                 printf("a = %lf.\n", a);
477                 printf("Please enlarge Hy!");
478             } else {
479                 i++;
480             }
481         }
482         free(Hy);
483         return answer;
484     }
485 }
```

```

486 //Fz
487 float m_fz( float b, float u, float a){
488     return ( -r*cos(b)*m_By(-r*sin(b),r*cos(b)*cos(a)+r*sin(a)-r*cos(a)
489         ) ,r*cos(b)*sin(a)-d-u*cos(a)-r*sin(a))*100 \
+ r*sin(b)*cos(a)*m_Bx(-r*sin(b),r*cos(b)*cos(a)+r*sin(a)-r*cos(a)
490         ,r*cos(b)*sin(a)-d-u*cos(a)-r*sin(a))*100 );
491 }
491 float m_Fz_b_add( int k, float b1, float t, float u, float a){
492     float ans = 0;
493     for( int i = 1; i <= (k>2? pow(2,k-2):1); i++){
494         ans += m_fz(b1+(2*i-1)*t/pow(2,k-1),u,a)/100;
495     }
496     return ans*t/(k>2? pow(2,k-2):1);
497 }
498 float m_Fz_b( float u, float a){
499     float answer = 0;
500     float b1 = 3*E, b2 = 2*PI;
501     float t = b2-b1;
502     int M = 40;
503     float *Pz;
504     Pz = ( float *) malloc(M*M*sizeof(float));
505     Pz[0] = ( m_fz(b1,u,a)+m_fz(b2,u,a) )*t/200;
506     int i = 1, j = 1;
507     while(i){
508         Pz[ i*M+0 ] = ( Pz[ (i-1)*M+0 ]+m_Fz_b_add( i+1,b1,t,u,a ))/2;
509         while(j){
510             Pz[ i*M+j ] = Pz[ i*M+j-1 ]+(Pz[ i*M+j-1 ]-Pz[ (i-1)*M+j-1 ])/(pow
511                 (4,j)-1);
512             if( j==i ){
513                 j=i;
514                 break;
515             } else {
516                 j++;
517             }
518         }
519         float e = fabs(Pz[ i*M+i ]-Pz[ (i-1)*M+i-1 ]);
520         if( e<E&&i!=1 ){
521             answer = Pz[ i*M+i ];
522             break;
523         } else if( i==M-1 ){

```

```

523         printf("u = %lf, a = %lf.\n", u, a);
524         printf("Please enlarge Pz!");
525     } else {
526         i++;
527     }
528 }
529 free(Pz);
530 return answer;
531 }
532 float m_Fz_add(int k, float u1, float t, float a){
533     float ans = 0;
534     for(int i = 1; i <= (k>2? pow(2,k-2):1); i++){
535         ans += m_Fz_b(u1+(2*i-1)*t/pow(2,k-1),a);
536     }
537     return ans*t/(k>2? pow(2,k-2):1);
538 }
539 float m_Fz(float a){
540     float answer = 0;
541     float u1 = 3*E/10, u2 = 2*u1;
542     float t = u2-u1;
543     int M = 40;
544     float *Hz;
545     Hz = (float *)malloc(M*M*sizeof(float));
546     Hz[0] = (m_Fz_b(u1,a)+m_Fz_b(u2,a))*t/2;
547     int i = 1, j = 1;
548     while(i){
549         Hz[i*M+0] = (Hz[(i-1)*M+0]+m_Fz_add(i+1,u1,t,a))/2;
550         while(j){
551             Hz[i*M+j] = Hz[i*M+j-1]+(Hz[i*M+j-1]-Hz[(i-1)*M+j-1])/(pow
552                 (4,j)-1);
553             if(j==i){
554                 j=i;
555                 break;
556             } else {
557                 j++;
558             }
559             //printf("i = %d,%lf\n", i, Hz[i*M+i]);
560             float e = fabs(Hz[i*M+i]-Hz[(i-1)*M+i-1]);
561             if( e<E&&i!=1 ){

```

```

562             answer = Bm*Hz[ i*M+i ];
563             break;
564         } else if( i==M-1){
565             printf( "a = %lf.\n", a );
566             printf( "Please enlarge Hz!" );
567         } else{
568             i++;
569         }
570     }
571     free(Hz);
572     return answer;
573 }
574 #endif

```

A.3.2 Gauss.h

```

1 #ifndef __MY_GAUSSINTEGRAL_H__
2 #define __MY_GAUSSINTEGRAL_H__
3 #include <stdio.h>
4 #include <stdlib.h>
5 #include <math.h>
6
7 /*The parameter*/
8 #define PI 3.14159265358979
9 #define E 0.001
10 #define Bg 1.20 //we resume that the neodymium magnet's remanence is
11      the averger value of 1.18-1.22T which is the N35's parameter
12 //##define mu0 4*PI*0.0000001 //the unit is T*m/A to make double
13      available, we chose to use Bg/mu0 which we called Bm.
14 #define Bm 3000000/PI
15 #define g 9.80 //N/kg I got it through experiment
16 #define R 0.0145 //m 0.029 or 0.0245m
17 #define h 0.01946 //m 0.01946 or 0.01916m
18 #define r 0.00475 //m
19 #define l 0.076 //m
20 #define d 0.00387 //m 0.00387 or 0.00790 or 0.01004m
21 #define m 0.0443 //kg
22 #define N 20 //the number of GaussIntegral weights and abscissa
23 /*PI, E, Bg, Bm, g, R, h, r, l, d, m are define as macro */

```

```

24  /*Get Gauss Integration's weights*/
25  void Weights(double *weights){
26      FILE *fp;
27      fp = fopen("weights.txt", "r");
28      for(int i = 0; i < N; i++){
29          fscanf(fp, "%lf", &weights[i]);
30          //check
31          //printf("weights[%d] is %.16lf\n", i, weights[i]);
32      }
33  }
34  /*Get Gauss Integratin's abscissa*/
35  void Abscissa(double *abscissa){
36      FILE *fp;
37      fp = fopen("abscissa.txt", "r");
38      for(int i = 0; i < N; i++){
39          fscanf(fp, "%lf", &abscissa[i]);
40          //check
41          //printf("abscissa[%d] is %.16lf\n", i, abscissa[i]);
42      }
43  }
44  /*Bx*/
45  double m_bx(double x, double y, double z, double w, double th){
46      double K = pow((x-R*cos(th))*(x-R*cos(th))+(y-R*sin(th))*(y-R*sin
47          (th))+(z-w)*(z-w), 1.5);
48      return (z-w)*R*cos(th)/K;
49  }
50  double m_Bx_th(double x, double y, double z, double w, double *weights
51      , double *abscissa){
52      double ans = 0;
53      double a = 0, b = 2*PI;
54      for(int i = 0; i < N; i++){
55          ans += weights[i]*m_bx(x, y, z, w, ((b-a)*abscissa[i]+b+a)/2)
56              *(b-a)/2;
57      }
58      //check
59      //printf("Bx_th(%lf, %lf, %lf) is %lf\n", x, y, z, ans);
60      return ans;
61  }
62  double m_Bx(double x, double y, double z, double *weights, double *
63      abscissa){

```

```

60     double ans = 0;
61     double a = 0, b = h;
62     for( int i = 0; i < N; i++){
63         ans += weights[i]*m_Bx_th(x, y, z, ((b-a)*abscissa[i]+b+a)/2,
64             weights, abscissa)*(b-a)/2;
65     }
66     //check
67     //printf("Bx(%lf, %lf, %lf) is %lf\n", x, y, z, ans);
68     return ans;
69 }
70 /*By*/
71 double m_by(double x, double y, double z, double w, double th){
72     double K = pow( (x-R*cos(th))*(x-R*cos(th))+(y-R*sin(th))*(y-R*sin
73         (th))+(z-w)*(z-w) , 1.5);
74     return (z-w)*R*sin(th)/K;
75 }
76 double m_By_th(double x, double y, double z, double w, double *weights
77     , double *abscissa){
78     double ans = 0;
79     double a = 0, b = 2*PI;
80     for( int i = 0; i < N; i++){
81         ans += weights[i]*m_by(x, y, z, w, ((b-a)*abscissa[i]+b+a)/2)
82             *(b-a)/2;
83     }
84     //check
85     //printf("By_th(%lf, %lf, %lf) is %lf\n", x, y, z, ans);
86     return ans;
87 }
88 double m_By(double x, double y, double z, double *weights, double *
89     abscissa){
90     double ans = 0;
91     double a = 0, b = h;
92     for( int i = 0; i < N; i++){
93         ans += weights[i]*m_By_th(x, y, z, ((b-a)*abscissa[i]+b+a)/2,
94             weights, abscissa)*(b-a)/2;
95     }
96     //check
97     //printf("By(%lf, %lf, %lf) is %lf\n", x, y, z, ans);
98     return ans;
99 }
```

```

94  /*Bz*/
95  double m_bz( double x, double y, double z, double w, double th){
96      double K = pow( (x-R*cos(th))*(x-R*cos(th))+(y-R*sin(th))*(y-R*sin
97                      (th))+(z-w)*(z-w) , 1.5 );
98      return ( -(y-R*sin(th))*R*sin(th)-(x-R*cos(th))*R*cos(th) )/K;
99  }
100
101  double m_Bz_th( double x, double y, double z, double w, double *weights
102                  , double *abscissa){
103      double ans = 0;
104      double a = 0, b = 2*PI;
105      for( int i = 0; i < N; i++){
106          ans += weights[i]*m_bz(x, y, z, w, ((b-a)*abscissa[i]+b+a)/2)
107                  *(b-a)/2;
108      }
109      //check
110      //printf("Bz_th(%lf, %lf, %lf) is %lf\n", x, y, z, ans);
111      return ans;
112  }
113  double m_Bz( double x, double y, double z, double *weights , double *
114              abscissa){
115      double ans = 0;
116      double a = 0, b = h;
117      for( int i = 0; i < N; i++){
118          ans += weights[i]*m_Bz_th(x, y, z, ((b-a)*abscissa[i]+b+a)/2 ,
119                                      weights , abscissa)*(b-a)/2;
120      }
121      //check
122      //printf("Bz(%lf, %lf, %lf) is %lf\n", x, y, z, ans);
123      return ans;
124  }
125  /*Fx*/
126  double m_fx( double b, double u, double al , double *weights , double *
127              abscissa){
128      return Bm*(-r*sin(b)*cos(al)*m_Bz(-r*sin(b),r*cos(b)*cos(al)+r*
129                      sin(al)-r*cos(al),r*cos(b)*sin(al)-d-u*cos(al)-r*sin(al),
130                      weights , abscissa) \
131      + r*sin(b)*sin(al)*m_By(-r*sin(b),r*cos(b)*cos(al)+r*sin(al)-r*cos
132                      (al),r*cos(b)*sin(al)-d-u*cos(al)-r*sin(al),weights , abscissa));
133  }

```

```

124     double m_Fx_th( double u, double al, double *weights, double *abscissa)
125     {
126         double ans = 0;
127         double a = 0, b = 2*PI;
128         for( int i = 0; i < N; i++){
129             ans += weights[i]*m_fx(((b-a)*abscissa[i]+b+a)/2, u, al,
130             weights, abscissa)*(b-a)/2;
131         }
132         //check
133         //printf("Fx_th(%lf, %lf, %lf) is %lf\n", x, y, z, ans);
134         return ans;
135     }
136     double m_Fx(double al, double *weights, double *abscissa){
137         double ans = 0;
138         double a = 0, b = 2*1;
139         for( int i = 0; i < N; i++){
140             ans += weights[i]*m_Fx_th(((b-a)*abscissa[i]+b+a)/2, al,
141             weights, abscissa)*(b-a)/2;
142         }
143         //check
144         //printf("Fx(%lf, %lf, %lf) is %lf\n", x, y, z, ans);
145         return ans;
146     }
147     /*Fy*/
148     double m_fy(double b, double u, double al, double *weights, double *
149     abscissa){
150         return Bm*(-r*sin(b)*sin(al)*m_Bx(-r*sin(b), r*cos(b)*cos(al)+r*
151         sin(al)-r*cos(al), r*cos(b)*sin(al)-d-u*cos(al)-r*sin(al),
152         weights, abscissa) \
153         + r*cos(b)*m_Bz(-r*sin(b), r*cos(b)*cos(al)+r*sin(al)-r*cos(al), r*
154         cos(b)*sin(al)-d-u*cos(al)-r*sin(al), weights, abscissa));
155     }
156     double m_Fy_th(double u, double al, double *weights, double *abscissa)
157     {
158         double ans = 0;
159         double a = 0, b = 2*PI;
160         for( int i = 0; i < N; i++){
161             ans += weights[i]*m_fy(((b-a)*abscissa[i]+b+a)/2, u, al,
162             weights, abscissa)*(b-a)/2;
163         }
164     }

```

```

155     // check
156     // printf("Fy_th(%lf, %lf, %lf) is %lf\n", x, y, z, ans);
157     return ans;
158 }
159 double m_Fy(double al, double *weights, double *abscissa){
160     double ans = 0;
161     double a = 0, b = 2*1;
162     for(int i = 0; i < N; i++){
163         ans += weights[i]*m_Fy_th(((b-a)*abscissa[i]+b+a)/2, al,
164             weights, abscissa)*(b-a)/2;
165     }
166     // check
167     // printf("Fy(%lf, %lf, %lf) is %lf\n", x, y, z, ans);
168     return ans;
169 }
170 /*Fz*/
171 double m_fz(double b, double u, double al, double *weights, double *
172     abscissa){
173     return Bm*(-r*cos(b)*m_By(-r*sin(b), r*cos(b)*cos(al)+r*sin(al)-r*
174         cos(al), r*cos(b)*sin(al)-d-u*cos(al)-r*sin(al), weights, abscissa
175         ) \
176         + r*sin(b)*cos(al)*m_Bx(-r*sin(b), r*cos(b)*cos(al)+r*sin(al)-r*cos
177         (al), r*cos(b)*sin(al)-d-u*cos(al)-r*sin(al), weights, abscissa) );
178 }
179 double m_Fz_th(double u, double al, double *weights, double *abscissa)
180 {
181     double ans = 0;
182     double a = 0, b = 2*PI;
183     for(int i = 0; i < N; i++){
184         ans += weights[i]*m_fz(((b-a)*abscissa[i]+b+a)/2, u, al,
185             weights, abscissa)*(b-a)/2;
186     }
187     // check
188     // printf("Fz_th(%lf, %lf, %lf) is %lf\n", x, y, z, ans);
189     return ans;
190 }
191 double m_Fz(double al, double *weights, double *abscissa){
192     double ans = 0;
193     double a = 0, b = 2*1;

```

```

187     for( int i = 0; i < N; i ++){
188         ans += weights[i]*m_Fz_th(((b-a)*abscissa[i]+b+a)/2, al,
189             weights, abscissa)*(b-a)/2;
190     }
191     //check
192     //printf("Fz(%lf, %lf, %lf) is %lf\n", x, y, z, ans);
193     return ans;
194 }
```

weights.txt

```

1      0.15275338713
2      0.15275338713
3      0.14917298647
4      0.14917298647
5      0.14209610931
6      0.14209610931
7      0.13168863844
8      0.13168863844
9      0.11819453196
10     0.11819453196
11     0.10193011981
12     0.10193011981
13     0.08327674157
14     0.08327674157
15     0.06267204833
16     0.06267204833
17     0.04060142980
18     0.04060142980
19     0.01761400713
20     0.01761400713
```

A.3.3 Simpson.c

```

1 #include <stdio.h>
2 #include <math.h>
3 #define Pi 3.14159265358
4 #define r0 25.00
5 #define B_r 1200
6 #define kk B_r/(4*Pi)
```

```
7
8     float f_x( float z0, float z, float theta){
9         return r0*cos(theta)*(z-z0)*kk;
10    }
11    float f_y( float z0, float z, float theta){
12        return r0*sin(theta)*(z-z0)*kk;
13    }
14    float f_z( float x, float y, float z0, float theta){
15        return r0*((r0*sin(theta)-y)*sin(theta)-(x-r0*cos(theta))*cos(
16            theta))*kk;
17    }
18    float K( float x, float y, float z, float z0, float theta){
19        return pow(pow(x-r0*cos(theta),2) + pow(y-r0*sin(theta),2) + pow(z-
20            z0,2),1.5);
21    }
22    float simps( float z0, float x, float y, float z, int f){
23        int i ,m,n;
24        m = 50;n = 2*m;
25        float h = (2*Pi-0)/n;
26        float theta=0;
27        float sum_x , sum_y , sum_z ;
28
29        sum_x = f_x(z0,z,0)/K(x,y,z,z0,0) + f_x(z0,z,2*Pi)/K(x,y,z,z0,2*Pi
30            );
31        sum_y = f_y(z0,z,0)/K(x,y,z,z0,0) + f_y(z0,z,2*Pi)/K(x,y,z,z0,2*Pi
32            );
33        sum_z = f_z(x,y,z0,0)/K(x,y,z,z0,0) + f_z(x,y,z0,2*Pi)/K(x,y,z,z0
34            ,2*Pi);
35
36        for( i = 1;i<n;i++){
37            theta = h*i;
38            if(i%2==0){
39                sum_x += 2*f_x(z0,z,theta)/K(x,y,z,z0,theta);
40                sum_y += 2*f_y(z0,z,theta)/K(x,y,z,z0,theta);
41                sum_z += 2*f_z(x,y,z0,theta)/K(x,y,z,z0,theta);
42            }
43            else{
44                sum_x += 4*f_x(z0,z,theta)/K(x,y,z,z0,theta);
45                sum_y += 4*f_y(z0,z,theta)/K(x,y,z,z0,theta);
46                sum_z += 4*f_z(x,y,z0,theta)/K(x,y,z,z0,theta);
47            }
48        }
49    }
```

```
42         }
43     }
44     sum_x = h*sum_x/3;
45     sum_y = h*sum_y/3;
46     sum_z = h*sum_z/3;
47
48     if (f == 1)
49         return sum_x;
50     else if (f == 2)
51         return sum_y;
52     else if (f == 3)
53         return sum_z;
54 }
55
56 void simpsz(){
57     FILE*fp=fopen("123.txt","w");
58     float z0 = 20.0;
59     float x,y,z,p,q,s,h;
60     int m,n,i,j,k,l;
61     m = 50;
62     n=2*m;
63     h = z0/n;
64     p = 80.0/n;
65     q = 80.0/n;
66     s = 40.0/n;
67     float sum_x,sum_y,sum_z;
68     for(j = 62;j<n;j++){
69         z = j*s;
70         for(k = 0;k<n;k++){
71             x = -40+k*p;
72             for(l = 0;l<n;l++){
73                 /*printf("z = %.8f ",z);
74                 printf("x = %.8f ",x);
75                 printf("y = %.8f ",y); */
76
77                 y = -40+l*q;
78                 sum_x = simps(0,x,y,z,1)+simps(3,x,y,z,1);
79                 sum_y = simps(0,x,y,z,2)+simps(3,x,y,z,2);
80                 sum_z = simps(0,x,y,z,3)+simps(3,x,y,z,3);
81 }
```

```
82         for( i = 1;i<n ; i++) {
83             z0 = i*h;
84             if( i%2==0){
85                 sum_x += 2*simps(z0 ,x ,y ,z ,1 );
86                 sum_y += 2*simps(z0 ,x ,y ,z ,2 );
87                 sum_z += 2*simps(z0 ,x ,y ,z ,3 );
88             }
89             else {
90                 sum_x += 4*simps(z0 ,x ,y ,z ,1 );
91                 sum_y += 4*simps(z0 ,x ,y ,z ,2 );
92                 sum_z += 4*simps(z0 ,x ,y ,z ,3 );
93             }
94         }
95         sum_x = sum_x*h/3;
96         sum_y = sum_y*h/3;
97         sum_z = sum_z*h/3;
98         printf(fp , "%6.2f      %6.2f      %6.2f      %9.6f
99                         %9.6f      %9.6f      \n ",x ,y ,z ,sum_x ,sum_y ,sum_z );
100    }
101    fclose(fp );
102    break ;
103}
104}
105int main(){
106    simpsz();
107    return 0;
108}
```

致 谢

在本次创新创业的实践中,我学到了很多东西。也有很多要感谢的人。

首先是我的指导老师白雪,在我申请项目、中期答辩、结项的过程中,白老师细心地指导我计划书的写作,及时指出我不规范的地方。

其次,感谢学校给予我的支持,包括但不限于实验室的使用,资金支持。这些琐事主要是王心华老师负责,很感谢他的付出。

最后,感谢在我前行过程中帮助我的人,有我的组员,乔亮老师.....