底盘线圈磁场仿真

院系: 电子信息与通信

专业班级: 提高 2101 班

姓名: 杨筠松

学号: U202115980

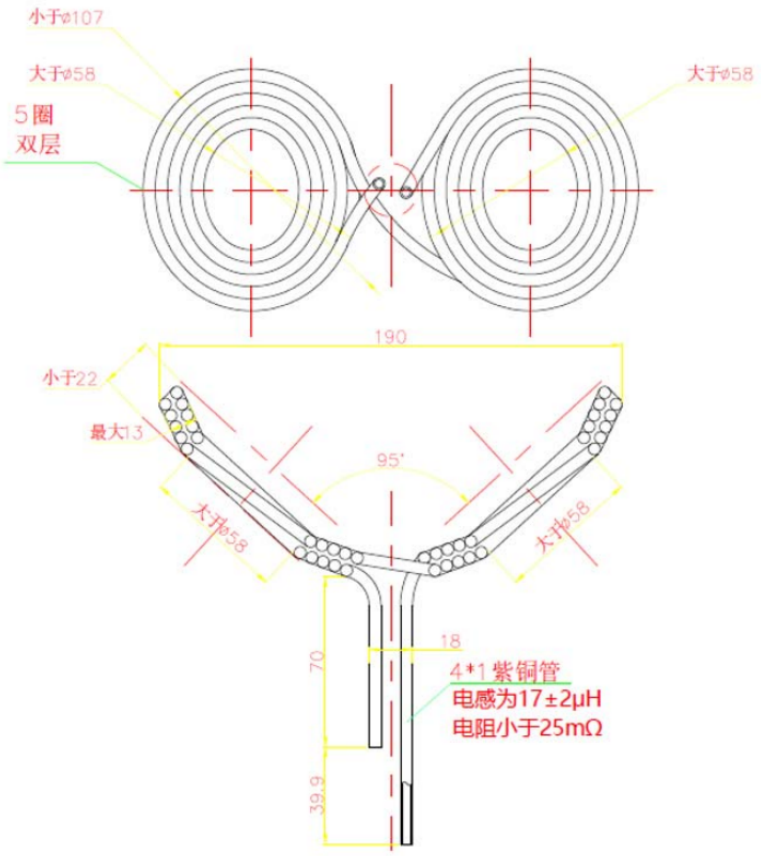
指导老师: 田加胜

时间: 2023 年 12 月

目录

1　线圈模型与结构尺寸

根据学号的顺序，我们需要研究和仿真的是第 8 号线圈——底盘线圈，具体形状如下图所示：



8. 盘底线圈

图 1: 底盘线圈模型与结构尺寸

2 算法表达式与具体计算步骤

2.1 对载流圆环产生的磁场进行数学建模

设载流圆环的半径为 R, 其中通有电流为 I。如下图所示, 设载流圆环位于 $y-z$ 平面上, 圆心与坐标原点重合, 载流圆环中心轴线与 x 轴重合。

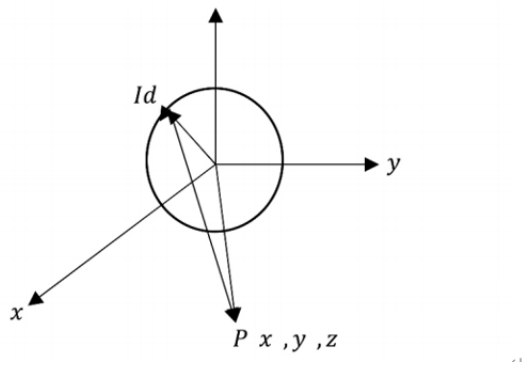


图 2: 载流圆环磁场分析图

在空间任取一点 $P(x_0, y_0, z_0)$, 计算 P 处的磁感应强度。在圆环上任取电流元 $Id\vec{l}$, 由毕奥 - 萨伐尔定律知该电流元在 P 点产生的磁感应强度为

$$d\vec{B} = \frac{\mu_0}{4\pi}\frac{I(d\vec{l}) \times \vec{r}}{r^3}$$

其中 $\vec{r}$ 为电流元到 P 点的距离。由磁场的叠加原理, 则 P 点的磁感应强度为

$$\vec{B} = \oint_C \frac{\mu_0}{4\pi}\frac{I(d\vec{l}) \times \vec{r}}{r^3}$$

因为直接求取该积分计算过程复杂且易出错, 所以为了简化计算过程, 我们在空间直角坐标系中, 首先分别求取在 x 轴, y 轴, z 轴的磁感应强度分量, 然后求矢量和得 P 点的磁感应强度, 最后绘制磁场分布图, 分析载流圆环在全空间的磁场分布特点。

设 (x, y, z) 为电流元的坐标, 则电流元 $Id\vec{l}$ 和电流元到 P 点的距离 $\vec{r}$ 在空间直角坐标系下的表示:

$$d\vec{l} = dx\hat{i} + dy\hat{j} + dz\hat{k} \quad \vec{r} = (x_0 - x)\hat{i} + (y_0 - y)\hat{j} + (z_0 - z)\hat{k}$$

将式（2）（3）代入式（1），其中

$$
\begin{aligned}
d\vec{l}\times\vec{r} &= \begin{vmatrix} \hat{i} & \hat{j} & \hat{k} \\ dx & dy & dz \\ (x_0-x) & (y_0-y) & (z_0-z) \end{vmatrix} \\
&= ((z_0-z)dy-(y_0-y)dz)\hat{i}+((x_0-x)dz-(z_0-z)dx)\hat{j} \\
&\quad +((y_0-y)dx-(x_0-x)dy)\hat{k}
\end{aligned}
$$

所以 P 点的磁感应强度分量为：

$$
\begin{aligned}
B_x &= \oint_C \frac{\mu_0 I}{4\pi}\frac{((z_0-z)dy-(y_0-y)dz)}{r^3}, \\
B_y &= \oint_C \frac{\mu_0 I}{4\pi}\frac{((x_0-x)dz-(z_0-z)dx)}{r^3}, \\
B_z &= \oint_C \frac{\mu_0 I}{4\pi}\frac{((y_0-y)dx-(x_0-x)dy)}{r^3}
\end{aligned}
$$

于是得到了一个载流环的磁场分布。

2.2 对 2D 螺旋线圈产生的磁场进行建模

在上面的一节讨论中，我们清楚了只要给定一个电流元的坐标，那么他对空间中任意一点 $P(x_0, y_0, z_0)$ 的磁感应强度均可计算得到，那么现如今问题在于如何仿真螺旋线圈并且半径正在不断变化的情况呢，如果我们可以确定线圈上任何一点的坐标，那么其产生的磁感应强度便随之迎刃而解了，具体结构如下图所示：

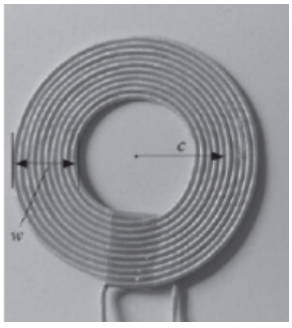


图 3: 螺旋线圈结构

我们有如下假设和约定：考虑线圈圆心位于 $y-z$ 平面上，为 $s_0 = (0,-m,m)$，设 r_0 表示螺旋线圈开始的半径长度，β 为每一圈螺线增加的半径，N 为总共螺线圈数，螺线圈自身材料半径忽略不计，线圈所在平面同 $x-y$ 平面夹角为 α，那么对于给定给的角度 $x(rad)$, 此处 $x \in [0, 10\pi]$, 那么对应的坐标可由下面式子计算得到:

$$
\begin{aligned}
r' &= r_0 + \frac{x\beta}{2\pi} \\
x_{new} &= s_{0_x} + r' sinx \\
y_{new} &= s_{0_y} + r' cosx sin\alpha \\
z_{new} &= s_{0_z} + r' cosx cos\alpha
\end{aligned}
$$

那么此时便可以利用新得到的坐标 $(x_new, y_{new}, z_{new})$ 来计算空间中任何一点 $P(x,y,z)$ 的磁场分布。

3 Python 程序代码运行结果分析

3.1 模型验证

首先我们需要对之前的二维螺线圈进行验证，运行 verify1.py 得到如下图所示：

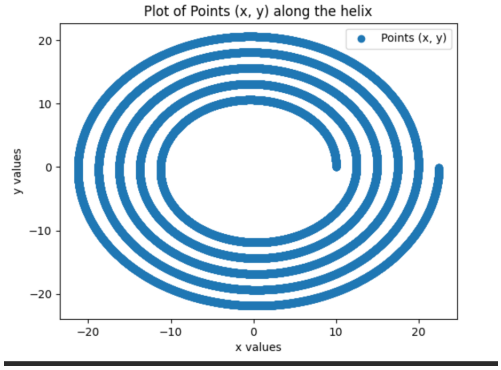


图 4: 2d 螺旋线圈

其中设置参数为 $\beta = 2.5mm/turn$, $N = 5$, 现如今将它放入到三维空间中来保证后续仿真的正确性，运行 verify.py 文件得到下图

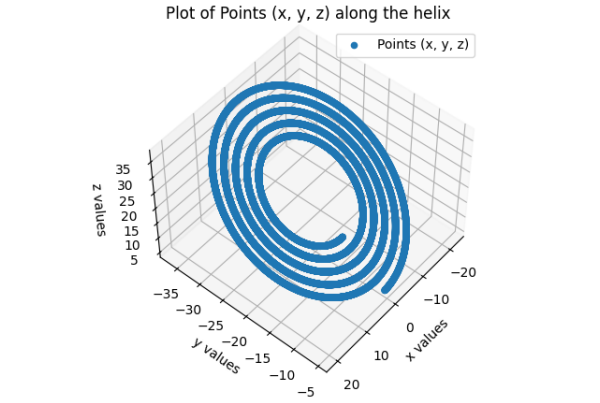


图 5: 3d 螺旋线圈结果

如此可以说明，上述对于螺线线圈的的模型建立是正确的。

3.2 具体仿真结果

运行 main.py 文件可以得到如下的三张图片，分别代表着当 $z = 0$ 时，只展示 xy 平面:

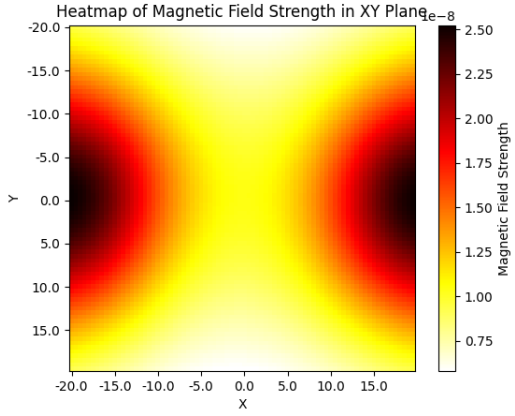


图 6: xy 平面仿真结果

当 $y = 0$, 只展示 xz 平面:

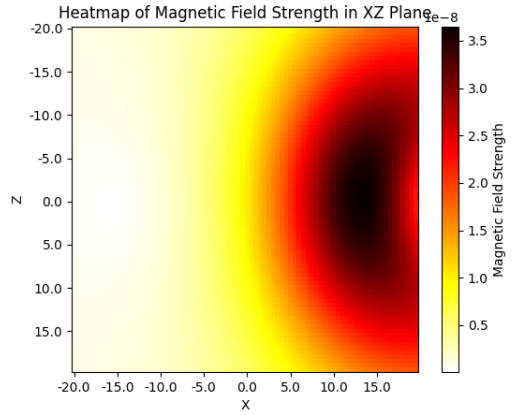


图 7: xz 平面仿真结果

当 $x = 0$ 时，只展示 yz 平面

全部都使用热力图来进行表示，使得结果更加直观，排除了 3d 视觉差效果造成的影响。

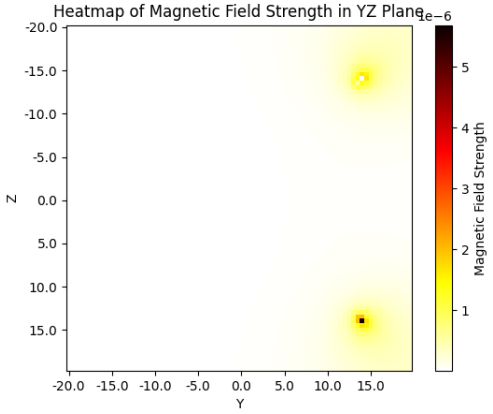


图 8: yz 平面仿真结果

4 Ansys Maxwell 软件仿真结果

4.1 仿真模型建立

观察模型，可以发现把手本身的磁效应并不会对整体结果造成影响，故而可以进行简化，而由于电脑的配置问题，运行两层 5 圈对称的线圈将会使得电脑运行周期过长，并且结果难以进行观察，遂降模型简化成如下图所示，仅有一层但是有良好可分析性：

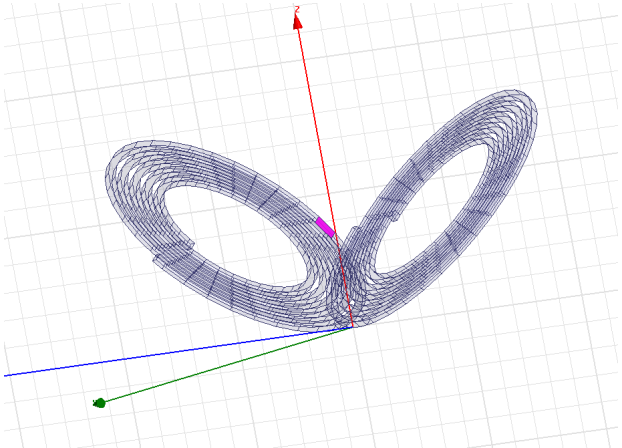


图 9: 模型创建

我们可以进行如下的线圈参数设置：

Properties

Name	Value	Unit	Evaluated V
Command	CreateUserDefinedPart		
Coordinate System	Global		
DLL Name	SegmentedHelix/PolygonHelix		
DLL Location	syslib		
DLL Version	1.0		
PolygonSegments	4		4
PolygonRadius	1	mm	1mm
StartHelixRadius	15	mm	15mm
RadiusChange	2.5	mm	2.5mm
Pitch	0	mm	0mm
Turns	5		5
SegmentsPerTurn	36		36
RightHanded	1		1

图 10: 线圈参数

同时依次设立边界（其余四面不再一一列出），激励电流，以及求解器类型如下：

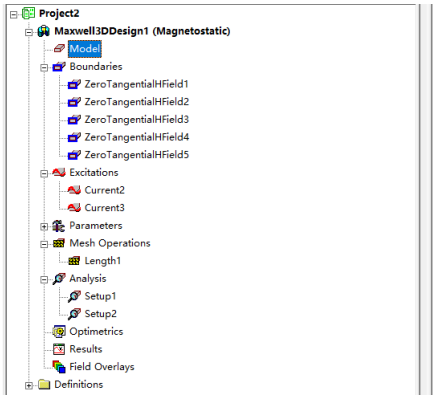


图 11: 项目管理器

选择求解器为静电场即可，此时并不需要考虑边际效应，因为本身的影响较小。

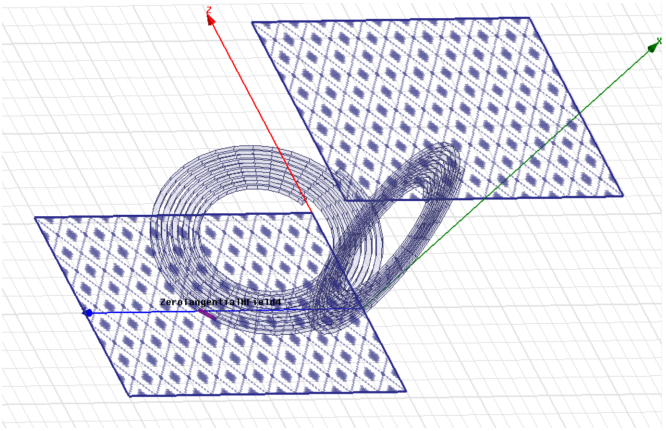


图 12: 边界

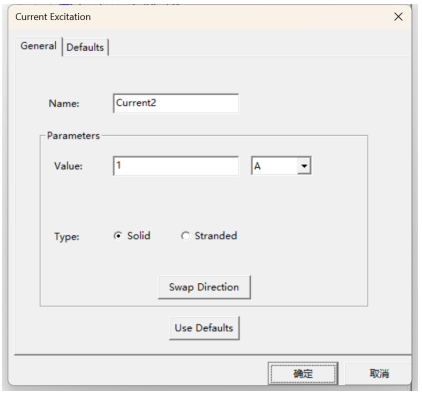


图 13: 激励电流

4.2 仿真结果展示

在仿真之前需要对模型进行认证，防止出现无法运行的错误，需要点击 Validation check, 所有验证均通过后，便可选择 Analyse all, 进行模型求解，得到如下的三个平面上的 B 热力图，如下所示, 注意矩阵的设置和选择，并且执行 Field Overlays 需要选取一个平面 (plane) 进行分析，不能选择物体 (object)，否则将会明显报错。

XY 平面

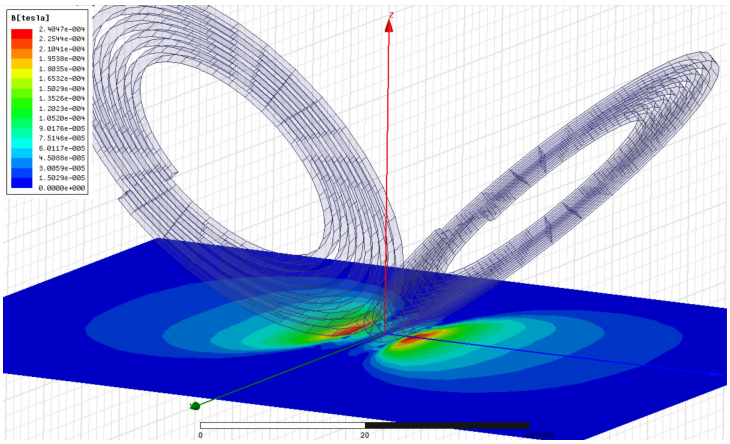


图 14: XY plane

XZ 平面

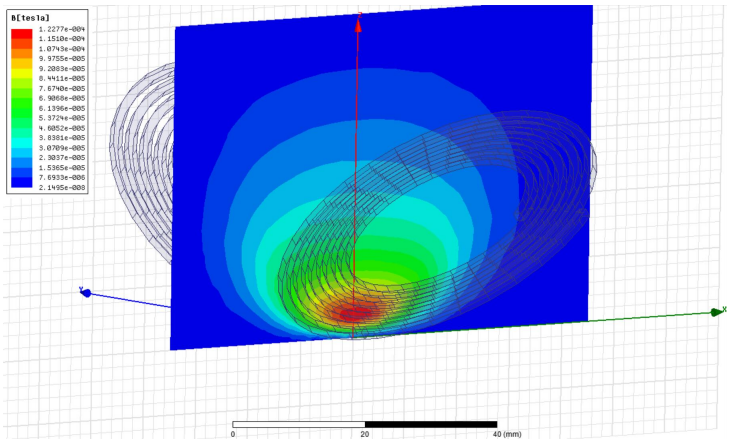


图 15: XZ plane

YZ 平面

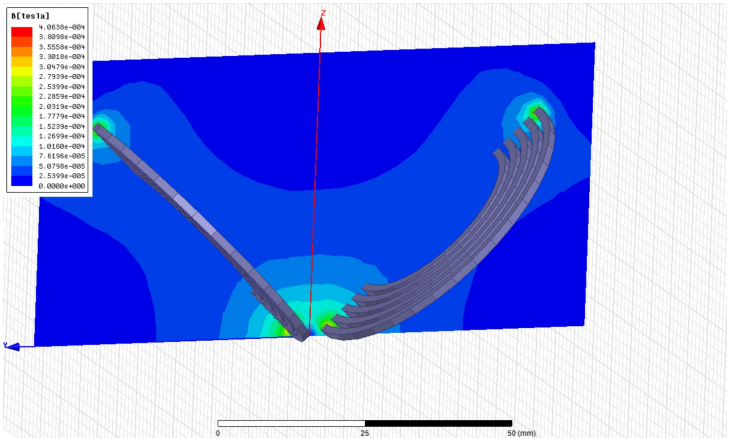


图 16: YZ plane

5 结果分析与对比

观察 python 仿真的热力图和 Ansys Maxwell 仿真结果，对于 xy 平面，可以发现呈现出中轴对称的形式，二者结果一致，对于 xz 平面，可以发现靠近原点的到 X 轴正半轴方向逐渐加大，但是碍于图片大小限制，无法完整展现又最大变到最小的过程，但是基本符合事实。对于 yz 平面，两点型特征极具辨识度，有效的展现了此时的磁场变化特征。

综上所述，本次仿真 python 代码和 Ansys Maxwell 软件基本一致，实验内容较为完善。

A verify.py

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# define the basic variables
m = 27.5 / np.sqrt(2) + 2
pi = 3.1415926
s0, s1 = [0, -m, m], [0, m, m]
alpha0, alpha1 = 3 / 4 * pi, 1 / 4 * pi
```

```
startHelix = 10
radiusChange = 2.5
turns = 5

def calculateB(x, y, z, elev=None, azim=None):
    x_values, y_values, z_values = [], [], []
    for x_val in np.arange(0, 10 * np.pi, 0.001):
        rPrime = calculateRPrime(x_val)
        x_new = s0[0] + rPrime * np.sin(x_val)
        y_new = s0[1] + rPrime * np.cos(x_val) * np.sin(alpha0)
        z_new = s0[2] + rPrime * np.cos(x_val) * np.cos(alpha0)
        x_values.append(x_new)
        y_values.append(y_new)
        z_values.append(z_new)

    # Plot the values for verifying
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')

    if elev is not None and azim is not None:
        ax.view_init(elev=elev, azim=azim)

    ax.scatter(x_values, y_values, z_values, label='Points(x,y,z)')
    ax.set_xlabel('x values')
    ax.set_ylabel('y values')
    ax.set_zlabel('z values')
    ax.set_title('Plot of Points (x, y, z) along the helix')
    plt.legend()
    plt.show()


def calculateRPrime(x):
    return startHelix + x * radiusChange / (2 * np.pi)
```

```
if __name__ == "__main__":
    calculateB(5, 0, 0, elev=50, azim=40)
```

B main.py

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm
from matplotlib import axes
from matplotlib.font_manager import FontProperties
font = FontProperties(fname='/Library/Fonts/Songti.ttc')

# define the basic variables
m = 27.5 / np.sqrt(2) + 2
pi = 3.1415926
s0, s1 = [0, -m, m], [0, m, m]
alpha0, alpha1 = 3 / 4 * pi, 1 / 4 * pi
startHelix = 10
radiusChange = 2.5
turns = 5
mu = 4*np.pi * 1e-7
I = 1


def calculateB(x, y, z):
    tar = np.array([x, y, z])
    sumB = np.array([0.0, 0.0, 0.0])
    pre = None
    # calculate r_prime for all the cases
    r_prime_values = [calculateRPrime(x_val) for x_val
        in np.arange(0, 10 * np.pi, 0.01)]
```

```
    for s in [s0, s1]:
        pre = None
        for x_val, rPrime in zip(np.arange(0, 10 * np.pi, 0.1),
                r_prime_values):
            # calculate the position of new point
            x_new = s[0] + rPrime * np.sin(x_val)
            y_new = s[1] + rPrime * np.cos(x_val)
                * np.sin(alpha0 if s == s0 else alpha1)
            z_new = s[2] + rPrime * np.cos(x_val)
                * np.cos(alpha0 if s == s0 else alpha1)

            if pre is not None:
                cur = np.array([x_new, y_new, z_new])
                # calculate the vector dl
                dl = pre - cur

                # calculate the vector r and /r/^3
                r = cur - tar
                r_norm = np.linalg.norm(r) ** 3

                # calculate dl x r and sum them altogether
                demo_B = np.cross(dl, r)
                sumB += demo_B / r_norm

            pre = np.array([x_new, y_new, z_new])
    # return back
    return mu * I * np.linalg.norm(sumB) / (4 * np.pi)


def calculateRPrime(x):
    return startHelix + x * radiusChange / (2 * np.pi)
```

```
def simulateXY():
    x_values, y_values= np.arange(-20, 20, 0.5),
            np.arange(-20, 20, 0.5)
    B_values = []
    for x in x_values:
        temp = []
        for y in y_values:
            temp.append(calculateB(x, y, 0))
        B_values.append(temp)
        print(x)
    fig = plt.figure()
    ax = fig.add_subplot(111)

    fig, ax = plt.subplots()
    im = ax.imshow(B_values, cmap=plt.cm.hot_r)

    x_ticks = np.arange(0, len(x_values), 10)
    y_ticks = np.arange(0, len(y_values), 10)

    ax.set_xticks(x_ticks)
    ax.set_xticklabels(x_values[x_ticks])
    ax.set_yticks(y_ticks)
    ax.set_yticklabels(y_values[y_ticks])

    cbar = plt.colorbar(im)
    cbar.set_label('Magnetic␣Field␣Strength')

    ax.set_xlabel('X')
    ax.set_ylabel('Y')
    ax.set_title('Heatmap␣of␣Magnetic␣Field␣Strength␣in␣XY␣Plane')
```

```
    plt.show()


def simulateXZ():
    x_values, z_values= np.arange(-20, 20, 0.5),
            np.arange(-20, 20, 0.5)
    B_values = []
    for x in x_values:
        temp = []
        for z in z_values:
            temp.append(calculateB(x, 0, z))
        B_values.append(temp)
        print(x)
    fig = plt.figure()
    ax = fig.add_subplot(111)

    fig, ax = plt.subplots()
    im = ax.imshow(B_values, cmap=plt.cm.hot_r)

    x_ticks = np.arange(0, len(x_values), 10)
    y_ticks = np.arange(0, len(z_values), 10)

    ax.set_xticks(x_ticks)
    ax.set_xticklabels(x_values[x_ticks])
    ax.set_yticks(y_ticks)
    ax.set_yticklabels(z_values[y_ticks])

    cbar = plt.colorbar(im)
    cbar.set_label('Magnetic␣Field␣Strength')

    ax.set_xlabel('X')
    ax.set_ylabel('Z')
    ax.set_title('Heatmap␣of␣Magnetic␣Field␣Strength␣in␣XZ␣Plane')
```

```
    plt.show()


def simulateYZ():
    y_values, z_values= np.arange(-20, 20, 0.5),
    np.arange(-20, 20, 0.5)
    B_values = []
    for y in y_values:
        temp = []
        for z in z_values:
            temp.append(calculateB(0, y, z))
        B_values.append(temp)
        print(y)
    fig = plt.figure()
    ax = fig.add_subplot(111)

    fig, ax = plt.subplots()
    im = ax.imshow(B_values, cmap=plt.cm.hot_r)

    x_ticks = np.arange(0, len(y_values), 10)
    y_ticks = np.arange(0, len(z_values), 10)

    ax.set_xticks(x_ticks)
    ax.set_xticklabels(y_values[x_ticks])
    ax.set_yticks(y_ticks)
    ax.set_yticklabels(z_values[y_ticks])

    cbar = plt.colorbar(im)
    cbar.set_label('Magnetic␣Field␣Strength')

    ax.set_xlabel('Y')
    ax.set_ylabel('Z')
```

```
    ax.set_title('Heatmap␣of␣Magnetic␣Field␣Strength␣in␣YZ␣Plane')

    plt.show()


if __name__ == "__main__":
    simulateXY()
    simulateXZ()
    simulateYZ()
```