等温冷离子模型下
等离子体漂移波的模拟与可视化

1600011319 唐钺
1600011400 陈远微

March 13, 2017

Contents

1. 课题概述

1.1 背景介绍

纵观古今，人类对能源的需求日益增加，从而燃料开采的规模也不断扩大。随着大量开采带来的资源枯竭的问题，若不采取有效的措施，未来能源危机爆发的可能性将极大提高。

核聚变研究是当今世界科技为解决人类未来能源问题而开展的重大的国际合作计划。聚变能具有资源无限，不污染环境，不产生高放射性核废料等优点，是人类未来能源的主导形式之一，也是目前认识到的可以最终解决人类社会能源问题和环境问题，推动人类社会可持续发展的重要途径之一。

ITER（国际热核聚变实验堆计划）是目前全球规模最大的国家科研合作项目之一。ITER 装置是一个能产生大规模核聚变反应的超导托克马克，它的建立是为了验证和平利用聚变能的科技可能性。而在托克马克装置内部，由于强磁场中等离子体漂移波的存在，轻微的密度扰动都会使整个系统逐渐演化形成湍流，若不加以束缚，其后果轻则使能量耗散，输出能低于预期；重则使整个体系崩溃，从而导致实验的失败。

为了研究可控核聚变的可能性，模拟漂移波则是研究很基础的一步，也是很重要的一步。而想要实现漂移波的模拟，必要先实现简化条件下的 2 维模型模拟。我们的课题组对此产生了兴趣，在简化条件的 2 维模型基础上，模拟了等离体漂移波，并在数学图像中实现了可视化。

1.2 课题目标

本课题旨在利用 C 语言通过计算机对简化条件的 2 维模型的托克马克装置中的等离子体漂移波现象进行数值计算，用 Qt 画出 Colormap，并利用 OpenCV 制作视频动画，实现可视化，并分析扰动量级、碰撞频率等参数对演化的影响。验证漂移波的存在，并且验证其具有导致湍流的趋势。

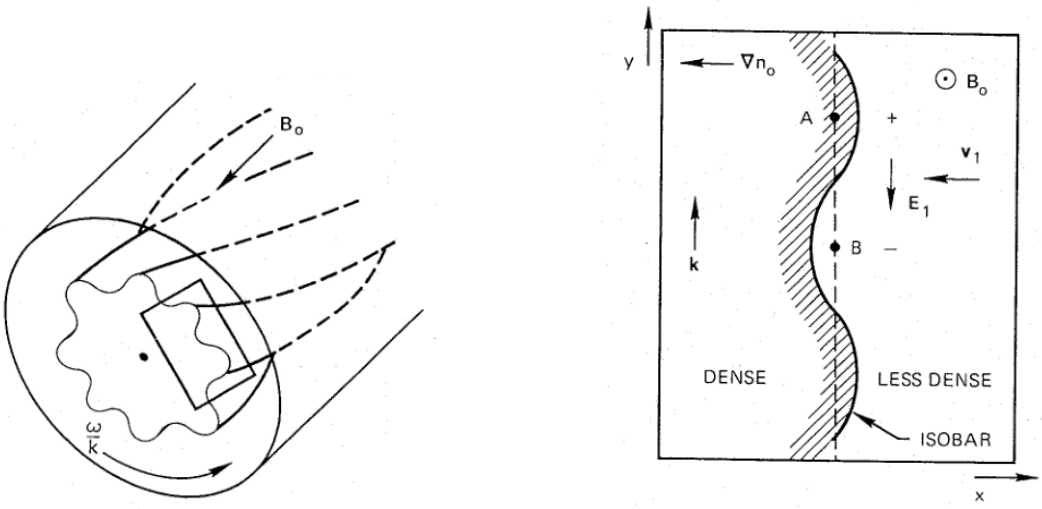
2. 原理

2.1 物理原理

2.1.1 物理模型

本课题的基本模型是流体近似下的二维等温冷离子模型，即 $T_i = 0, T_e = Const$ 条件下离子静止、电子运动的等温模型。

流体模型是等离子体问题的一个简化模型。经实验检验，其预测结果比较可靠，而且复



(a) 直角坐标系 (b) 物理机制

杂程度低于动力理论中模型。流体模型是目前大约 80% 的等离子体问题采用的模型，本课题所讨论的问题也属于这种模型的适用范围，因此我们选用了流体模型。

考虑环形托克马克装置中一小段弧段，由于托克马克装置半径较大，并且为了简化模型，所取的小弧段可近似看作均匀柱体。我们放大柱体中的一小部分，如图 1。方框中框住的小段对应的圆心角很小，可以将图示中框住的部分扩展到笛卡尔几何，如图 2。我们讨论的区域即为图 2 表示出的区域。

沿着柱体轴向有恒定的磁场 B_0。由于托克马克装置内电子数密度分布未知，不妨假设其为高斯分布，这种分布内部集中，边界稀疏，常常符合实际情况。扰动形式则为随机扰动。实现方案如下，其中 n_e 二维数组表示电子数密度分布。

```
void MainWindow:: setne()
{
    srand(time(NULL));
    for(int j=0;j<258;j++){
        for(int i=0;i<194;i++){
            ne[i][j]=1.+pert*(rand()%10000)+exp(-(i/50.)*(i/50.));
            nei[i][j]=ne[i][j];
        }
    }
}
```

由于漂移波的轴向分量相比垂直轴分量可以忽略，问题简化为垂直轴平面上的研究。

2.1.2 物理方程

基本方程为

$$\begin{aligned}\frac{dn_e}{dt} &= -\alpha(\widetilde{n_e}-\widetilde{\phi})+\kappa_\perp\nabla_\perp^2 n_e\\ \frac{dw}{dt} &= -\alpha(\widetilde{n_e}-\widetilde{\phi})+\kappa_\perp\nabla_\perp^2 w\\ w &= \nabla^2\phi\end{aligned}$$

代入 $\frac{d\boldsymbol{A}}{dt}=\frac{\partial \boldsymbol{A}}{\partial t}+\boldsymbol{v}\cdot\nabla\boldsymbol{A}$ 并与 $\boldsymbol{v}=\frac{\boldsymbol{E}\times\boldsymbol{B}}{\boldsymbol{B}^2}$ 联立得方程为

$$\frac{\partial n_e}{\partial t}+v\cdot\nabla n_e = -\alpha(\widetilde{n_e}-\widetilde{\phi})+\kappa_\perp\nabla_\perp^2 n_e \tag{2.1}$$

$$\frac{\partial w}{\partial t}+v\cdot\nabla w = -\alpha(\widetilde{n_e}-\widetilde{\phi})+\kappa_\perp\nabla_\perp^2 w \tag{2.2}$$

$$w=\nabla^2\phi \tag{2.3}$$

$$v_x=-\frac{\partial\phi}{\partial y} \tag{2.4}$$

$$v_y=\frac{\partial\phi}{\partial x} \tag{2.5}$$

其中 $\widetilde{n_e}=n_e-\bar{n_e}$

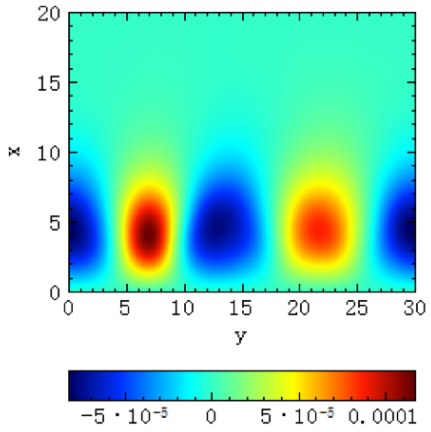
2.2 算法原理

2.2.1 差分法

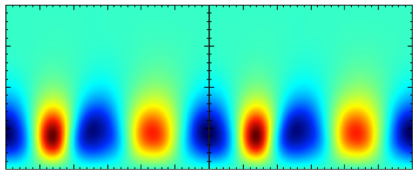
数值模拟偏微分方程的第一步是离散化，用离散元的行为代替连续系统的行为。一般有两种方法，其中一种将空间划分成许多网格，另一种在空间中布点。前者包括有限元法，各类差分法等，后者包括无网格伽辽金法等。由于其他方法具有计算量大、计算时间长、算法复杂的特点，对于本课题，选用正方形网格，通过差分法解决。

对于空间差分，经过阅读相关论文，发现取 194*258 的网格比较理想。参考前辈经验与自己的反复试验，我们发现时间间隔 tau 取 0.005 时展示效果令人满意。空间差分的具体格式为：

$$\nabla^2 u\Big|^{x_{i,j}} = \frac{u\Big|^{x_{i+1,j}}+u\Big|^{x_{i-1,j}}+u\Big|^{x_{i,j+1}}+u\Big|^{x_{i,j-1}}-4u\Big|^{x_{i,j}}}{(\Delta x)^2}$$



(c) 初始电势分布



(d) 周期边界条件演示

2.2.2 周期边界条件

边界条件的设置在本课题中非常重要，直接影响到程序的科学性。一般常用的边界条件有 Dirichlet 条件、Neumann 条件和 Robin 条件。

在 x 方向上，电势的边界应该满足固定边条件，其原因为等离子体近似。一般地，在等离子体中，可以假设 $n_i = n_e$ 和 $\nabla \cdot \boldsymbol{E} \neq 0$ 同时成立。对于本课题的情境，由于扰动微小，整个体系平均而言电势为 0，故边界处可以直接固定电势值。

在 y 方向上，体系应该满足周期边界条件。由于整个体系的对称性，在垂直于轴的平面上，物理量随角度增大呈周期变化。假定我们所取区域即为演化的一个周期，那么形成漂移波时，从图像右端移出画面的部分，必在画面左侧再次出现，如上图所示。

实现代码：

```
template<class T>
void edgey(T*h){
    for(int i=0;i<194;i++)
    {
        h[i][0]=h[i][258-2];
        h[i][258-1]=h[i][1];
    }
}
```

2.2.3 预估校正法

本课题采用的预估校正法为 leapfrog trapezoidal 算法。该算法对本课题所讨论的问题有很强的针对性，能使得计算达到关于时间间隔（delta t）的二次方的精度。其具体体现在：if(f<0.6) transvalue(w,wi); transvalue(wi,work); else transvalue(wi,work);

```
sve();
double f=0.5;
double fi=0.5;
sw(f,fi);
sden(f,fi);
Poisson();
sve();
f=1.0;
fi=0.0;
sw(f,fi);
sden(f,fi);
Poisson();
```

这种算法主要是为了提高计算的精度，特别是为了使得演化的非线性部分更加精确。我们是在李博老师的指导下使用了这种算法，可是算法的原理复杂，我们无法完全了解清楚，但实际结果证明了这种算法的优越性。

2.2.4 矩阵特征值法

$$\psi_{j-1}-\psi_j+\psi_{j+1}+(\frac{\Delta z}{\Delta x})^2\boldsymbol{A}\psi_j=\boldsymbol{H}_j$$

$$\boldsymbol{A}=\begin{bmatrix} -2 & 1 & & & & & \\ 1 & -2 & 1 & & & & \\ & 1 & -2 & 1 & & & \\ & & & \ddots & & & \\ & & & 1 & -2 & 1 & \\ & & & & 1 & -2 & \end{bmatrix}$$

$$\boldsymbol{P}_{j+1}+[(\frac{\Delta z}{\Delta x})^2\Lambda-2]\boldsymbol{P}_j+\boldsymbol{P}_{j-1}=\boldsymbol{R}_j$$

$$\tau \boldsymbol{P_j} = \frac{N}{2}\psi_j \, \boldsymbol{R_j} = \tau \boldsymbol{H_j}$$

$$\boldsymbol{P_{i,j+1}} + [(\frac{\Delta z}{\Delta x})^2 \lambda_i - 2]\boldsymbol{P_{i,j}} + \boldsymbol{P_{i,j-1}} = \boldsymbol{R_{ij}}$$

$$\psi_{ij} = \frac{2}{N}\sum_{k=1}^{N-1} \tau_{ik} \boldsymbol{P_{kj}}$$

$$R_{ij} = \sum_{k=1}^{N-1} \tau_{ik} \boldsymbol{H_{kj}}$$

$$(i = 1,2,\cdots,N-1; j = 1,2,\cdots,M-1)$$

$$\tau_{ik} = sin(\frac{\pi ik}{N}) \; \lambda_i = -4sin^2(\frac{i\pi}{2N})$$

$$Y_{ij} = \sum_{k=1}^{N-1} X_{kj} sin(\frac{2\pi ik}{2N})$$

$$X'_{kj} = \begin{cases} X_k j, & k = 1,2,\cdots,N-1 \\ 0, & k = 0; k = N, N+1, \cdots, 2N-1 \end{cases}$$

$$Y_{ij} = \sum_{K=0}^{2N-1} X'_{kj} sin(\frac{2\pi ik}{2N}), \quad (i = 0,1,\cdots,2N-1; j = 1,2,\cdots,M-1)$$

```
template<class T>
void slinear(int i, double* a, T* d, T*x)
{
    double l[258];
    double u[258-1];
    l[0]=a[0];
    for(int k=0;k<258-1;k++)
    {
        u[k]=1./l[k];
        l[k+1]=a[k+1]-u[k];
    }
    double y[258];
    y[0]=d[i][0]/l[0];
    for(int k=0;k<258-1;k++)  y[k+1]=(d[i][k]-y[k])/l[k+1];
    x[i][258-1]=y[258-1];
    for(int k=258-2;k>=0;k--)  x[i][k]=y[k]-u[k]*x[i][k+1];
}
```

```
void Poisson()
{
    double Rxy[194][258]={};
    double lamda;
    lamda=2*(ky*ky*dx*dx-1);
    for(int j=0;j<258;j++)
    {
        for(int i=0;i<194;i++)
        Rxy[i][j]+=2.*dx*dx*wi[i][j];
    }
    transvalue(phi,zeros);
    for(int i=0;i<=194-1;i++)
    {
        double a[258];
        for(int j=0;j<258;j++)
        a[j]=lamda;
        slinear(i,a,Rxy,phi);
    }
}
```

2.2.5 超松弛迭代法 (S0R)

下面是求解方程近似解的超松弛迭代公式

$$x_i^{k+1}=(1-\omega)x_i^{k+1}\frac{\omega}{a_{ii}}(b_i-\sum_{i<j}a_{ij}x_j^{k+1}-\sum_{i>j}a_{ij}x_j^k) \tag{2.6}$$

所需迭代步数由收敛精度和超松弛因子确定。超松弛因子在不同问题下有不同的最适值。

3. 算法实现

3.1 密度演化方程

$$\frac{\partial n_e}{\partial t} + v \cdot \nabla n_e = -\alpha(\widetilde{n_e} - \widetilde{\phi}) + \kappa_\perp \nabla_\perp^2 n_e$$

主要实现函数为 sden()。实现的基本思想是将各个部分单独编写成计算函数，最后再进行合并。为了提高计算精度，采用了预估校正法。

方程左边第一项由 convect.h 进行计算，并将值赋给一个二维数组，第二项则由 dif3.h 进行推演。具体实现函数如下：

```
convect:
extern  double vx[194][258];
extern  double vy[194][258];
extern double dx;
extern double dy;
extern double tau;
extern double arho; //归一化常数

template<class T>
void convect(T*h,T*g)
{
     for(int i=2; i<=194-1; i++)
     {
          for(int j=2; j<=258-1; j++)
          {
             h[i-1][j-1]=(vx[i][j-1]*g[i][j-1]-vx[i-2][j-1]*g[i-2][j-1])*arho*tau/(2.*dx
             h[i-1][j-1]+=(vy[i-1][j]*g[i-1][j]-vy[i-1][j-2]*g[i-1][j-2])*arho*tau/(2.*d
          }
     }
}

dif3:
extern double dx;
extern double dy;
extern double dif;
extern double tau;
```

```
template<class T>
void dif3(T* h,T* g)
{
    for(int i=2; i<=194-1; i++)
    {
        for(int j=2; j<=258-1; j++)
        {
            h[i-1][j-1]=(g[i][j-1]+g[i-2][j-1]-2*g[i-1][j-1])*dif*tau/(dx*dx);
            h[i-1][j-1]+=(g[i-1][j]+g[i-1][j-2]-2*g[i-1][j-1])*dif*tau/(dy*dy);
        }
    }
}
```

可以看出，在这两个函数中，对边界均未进行处理。但演化的最后我们用 sden() 函数设定了边界。这是为了减少计算量所进行的设计，因为 sden() 函数中只进行矩阵的加减与数乘运算，故边界不会影响整个体系的演化，在函数的最后进行边界设定是合理的。

第三项涉及到 $\widetilde{n_e},\widetilde{\phi}$ 的演算。关于这两个量的演化，我们先规定两个一维数组用来存储 x 方向 n_e,ϕ 的平均值，扰动量则是 $_e,\phi$ 与平均值之差。其中，取平均由 mean.h 进行，给扰动量赋值则由 givevalue.h 负责。

实现的代码为：

```
double meanphi[194];
double meandeni[194];
mean(meanphi,phi);
mean(meandeni,ne);
givevalue(meanphi,phi,delta_phi);
givevalue(meandeni,ne,delta_ne);
```

最终，利用预估校正法完成最后函数的拼合演算。

```
double fxy[194][258]={};
convect(fxy,ne);
double dif2t[194][258]={};
dif3(dif2t,nei);
double work[194][258];
for(int i=1; i<=194; i++)
{
    for(int j=1; j<=258; j++)
```

```
    {
        work[i-1][j-1]=nei[i-1][j-1]*f+ne[i-1][j-1]*fi;
        work[i-1][j-1]+=dif2t[i-1][j-1]-fxy[i-1][j-1];
        work[i-1][j-1]+=alpha0*(delta_phi[i-1][j-1]-delta_ne[i-1][j-1])*tau;
    }
}
    if(f<0.6)
    {
        transvalue(nei,ne);
        transvalue(ne,work);
    }
    else transvalue(ne,work);
    edgey(ne);
```

3.2 涡度演化方程

$$\frac{\partial w}{\partial t}+v\cdot\nabla w=-\alpha(\widetilde{n_e}-\widetilde{\phi})+\kappa_{\perp}\nabla_{\perp}^{2}w$$

实现方法与密度演化方程基本相似，代码如下：

```
#include "givevalue.h"
#include "convect.h"
#include "dif3.h"
#include "mean.h"
#include "transvalue.h"
extern double w[194][258];
extern double wi[194][258];
extern double ne[194][258];
extern double phi[194][258];
extern double tau;
extern double alpha0;
extern double delta_ne[194][258];
extern double delta_phi[194][258];

void sw(double& f,double& fi)
{
    double fxy[194][258]={};
```

```
    convect(fxy,wi);
    double dif2t[194][258]={};
    dif3(dif2t,w);

    double work[194][258];
    double meanphi[194];
    double meandeni[194];
    mean(meanphi,phi);
    mean(meandeni,ne);

    givevalue(meanphi,phi,delta_phi);
    givevalue(meandeni,ne,delta_ne);

    for(int i=1;i<=194;i++)
    {
        for(int j=1;j<=258;j++)
        {
            work[i-1][j-1]=w[i-1][j-1]*f+wi[i-1][j-1]*fi;
            work[i-1][j-1]+=dif2t[i-1][j-1]-fxy[i-1][j-1];
            work[i-1][j-1]+=alpha0*(delta_phi[i-1][j-1]-delta_ne[i-1][j-1])*tau;
        }
    }

    if(f<0.6)
    {
        transvalue(w,wi);
        transvalue(wi,work);
    }
    else transvalue(wi,work);
    edgey(wi);
}
```

关于其 x 方向的边界，由于程序的设计，每一个参与运算的数组均用 0 进行初始化，并且计算过程中只涉及矩阵之间的加法与矩阵自身的数量乘积，x 方向边界条件自动实现，故没有必要再进行设定。

3.3 泊松方程

泊松方程的实现是本课题的关键。为了实现泊松方程，我们先后尝试了 superLU 法，矩阵赋值法，SOR 超松驰迭代法，最终发现 SOR 超松驰迭代法在本问题中具有较强的收敛性与较快的收敛速度，故程序中 Poisson.cpp 采用 SOR 法。

具体实现步骤如下：

```
#include <cmath>
#include "edgey.h"
#include "edgex.h"
extern  double phi[194][258];
extern  double wi[194][258];
extern double dx;
extern double dy;
double eds=1e-10;
double zeros[194][258]={};

void Poisson()
{
    int counts=0;
    double w=1.97175;
    double maxdu,du;
    double temp[194][258];
    maxdu=0.0;
    do
    {
        for(int j=1;j<257;j++)
        {

            for(int i=1;i<193;i++)
            {
                temp[i][j]=phi[i][j];
                phi[i][j]=(1-w)*phi[i][j];
                phi[i][j]+=(w/2.)*(dx*dx*(phi[i][j-1]+phi[i][j+1])+dy*dy*(phi[i-1][j]+p
                du=temp[i][j]-phi[i][j];
                du=abs(du);
```

```
                if(du>maxdu)maxdu=du;
            }
        }
        counts++;
        edgey(phi);
    } while ((maxdu>eds)&&counts<3000);
    edgex(phi);
}
```

函数中，eds 表示迭代的精度，而 counts 用于存储迭代的次数。这两个量的值是循环结束的判据。判据 while ((maxdu>eds)counts<3000) 中出现的数字 3000，是为了满足软件设计的预期而给定的。在默认状态下，即 ntp=5, nts=600, alpha0=0.05, pert=1.e-7, dif=0.01，若只将迭代数作为判据，在处理器为 Intel(R)Core(TM)i7-6500U CPU @2.50GHz 2.60GHz 的计算机上，出一张图的时间大约为 5 分钟，而若想做成视频则大约需 3000 分钟，这超出了软件设计的预期，设计 3000 这个量值正是对计算时间上限的一种粗略调控。不过对于本课题而言，在默认条件下，出一张图的平均时间大约 18 秒，结束循环的仍是迭代的精度。迭代的精度是人为设置的，这也规定了扰动的下限，在 6.1 节中我们将会发现，扰动越小，漂移波的形成速度也越慢，因此为了保证在默认设置的时间参数下能模拟出漂移波，我们固定了 eds 的值。

3.4 运算方程

运算方程将所有的演化方程组合在一起，进行最终的计算。具体实现如下：

```
for( ;t<=nts;t++)
{
    for(int tt=1;tt<=ntp;tt++)
    {
        sve();
        double f=0.5;
        double fi=0.5;
        sw(f,fi);
        sden(f,fi);
        Poisson();
        sve();
```

```
            f=1.0;
            fi=0.0;
            sw(f,fi);
            sden(f,fi);
            Poisson();
            progress.setValue(t*ntp-ntp+tt);
            if(progress.wasCanceled())
            {
                t=1;
                return;
            }
        }

        plot(phi, "phi", customPlot1, colorMap1, colorScale1);
        plot(ne, "ne", customPlot2, colorMap2, colorScale2);
        plot(delta_ne, "delta ne", customPlot3, colorMap3, colorScale3);
        plot(wi, "wi", customPlot4, colorMap4, colorScale4);
        plot(vx, "vx", customPlot5, colorMap5, colorScale5);
        plot(vy, "vy", customPlot6, colorMap6, colorScale6);
    }
```

其中两次设置 f，fi 的值便是预估校正法的体现。

3.5 为提高计算速度所采取的措施

计算速度是科学计算中十分重视的参量。在保证计算精度的前提下提高计算速度，是数值模拟中一项重要工作。在本次课题中，为了提高计算速度，我们主要采取了以下措施：

1. 省略 convect.h 和 dif3.h 中边界条件的设定。相关内容已经在 3.1 中做出了解释。

2. 调整 Poisson.h 中边界条件的作用域。在 Poisson.h 中，为了使迭代结果正确，需要在 SOR 法的迭代中不断设定边界条件，然而，这项工作相当于在循环中插入循环，如果边界条件的作用域设计不尽合理，那么，要么会使得计算结果错误，要么会使得计算速度缓慢，所以 Poisson.h 中边界条件的位置就十分重要。曾经我们出现过由于边界条件位置的不正确，导致 10 分钟才出一张图的惨痛教训；也出现过因为这种原因模拟出假漂移波的情况。在吸取了多次失败教训的总结后，我们对原方程进行了深入地剖析，并最终给出了合理的作用域，

得到了符合预期的模拟结果。
3. 调整 phi 的迭代初值。在 Poisson.h 中，关键在于 phi 的迭代，而迭代从什么值开始，对计算速度的影响非常显著。曾经的代码中，phi 是从 0 开始迭代，于是计算速度可想而知，十分缓慢。后来经过进一步分析，让 phi 保持原来的值，直接迭代。由于时间间隔很短，phi 在经过 Poisson.h 的运算后变化不大，让 phi 保持原来的值直接迭代，计算速度于是得到了显著的提升。由原来的 2 分钟一张图降低到了 25 秒一张。
4. 调整超松驰迭代因子。超松驰迭代因子的改变，会使得 SOR 法的迭代次数发生变化。在本次课题中，当超松驰迭代因子从 1.01 变化到 1.99 的过程中，我们深刻的感受到了计算速度的差异，经过粗略的实验，我们最终选择 1.97。在实验的过程中我们发现，超松驰因子的最价值与 x，y 方向取的格点数密切相关。

4. 数据可视化

4.1 概述

为了直观清晰地演示漂移波现象，我们决定采用 colorMap 的形式进行数据可视化。在 colorMap 中，z 值由小到大对应颜色的色调由冷到暖，因此漂移波表现为一团平行移动的火焰，非常直观。

Qt 有丰富的可视化库，常用来进行图表绘制的是 Qwt 与 QCustomPlot。经过调查，发现 QCustomPlot 的体积更小，使用更方便，也足够满足我们绘制 colorMap 的需要，因此选用 QCustomPlot 库。

为了展示漂移的动态过程，我们决定将计算出的分布图写入视频，并实现在程序里播放。经过调查我们选用 OpenCV 库写入视频，利用 QMediaPlayer 类实现视频的播放功能。

4.2 customPlot 对象的构造与析构

创建对象的代码如下：

```
QCustomPlot* customPlot1 = new QCustomPlot;
QCustomPlot* customPlot2 = new QCustomPlot;
QCustomPlot* customPlot3 = new QCustomPlot;
QCustomPlot* customPlot4 = new QCustomPlot;
QCustomPlot* customPlot5 = new QCustomPlot;
```

```
QCustomPlot* customPlot6 = new QCustomPlot;
```

利用指针和 new，我们一共创建六个 customPlot 对象，分别对应我们即将绘制的六个 colorMap 以及其对应的 colorScale。

所有画图结束后，利用 QCustomPlot() 析构函数进行析构，释放内存。

4.3 colorMap 与 colorScale 的创建

代码如下：

```
QCPColorMap* colorMap1 = new QCPColorMap(customPlot1->xAxis, customPlot1->yAxis);
QCPColorMap* colorMap2 = new QCPColorMap(customPlot2->xAxis, customPlot2->yAxis);
QCPColorMap* colorMap3 = new QCPColorMap(customPlot3->xAxis, customPlot3->yAxis);
QCPColorMap* colorMap4 = new QCPColorMap(customPlot4->xAxis, customPlot4->yAxis);
QCPColorMap* colorMap5 = new QCPColorMap(customPlot5->xAxis, customPlot5->yAxis);
QCPColorMap* colorMap6 = new QCPColorMap(customPlot6->xAxis, customPlot6->yAxis);

QCPColorScale* colorScale1 = new QCPColorScale(customPlot1);
QCPColorScale* colorScale2 = new QCPColorScale(customPlot2);
QCPColorScale* colorScale3 = new QCPColorScale(customPlot3);
QCPColorScale* colorScale4 = new QCPColorScale(customPlot4);
QCPColorScale* colorScale5 = new QCPColorScale(customPlot5);
QCPColorScale* colorScale6 = new QCPColorScale(customPlot6);
```

每个 customPlot 对象对应一个 colorMap 与 colorScale，colorScale 为颜色－数值对应轴。

由代码可见，colorMap 创建时即与 customPlot 的坐标轴绑定了，colorScale 则没有，因此需要另外添加到 customPlot 中。

4.4 customPlot 的初始设定

代码如下：

```
iniplot(customPlot1, colorMap1, colorScale1);
iniplot(customPlot2, colorMap2, colorScale2);
iniplot(customPlot3, colorMap3, colorScale3);
```

```
    iniplot(customPlot4, colorMap4, colorScale4);
    iniplot(customPlot5, colorMap5, colorScale5);
    iniplot(customPlot6, colorMap6, colorScale6);
```

其中 iniplot 函数的定义为：

```
    void MainWindow::iniplot(QCustomPlot *customPlot, QCPColorMap* colorMap, QCPColorS
{
    //坐标轴设定
    customPlot->axisRect()->setupFullAxesBox(true);
    customPlot->xAxis->setLabel("y");
    customPlot->yAxis->setLabel("x");

    //设置colorScale并添加到customPlot中
    colorScale->setType(QCPAxis::atBottom);
    colorScale->axis()->setNumberPrecision(2);
    customPlot->plotLayout()->addElement(1, 0, colorScale);

    //colorMap的初始设定
    colorMap->data()->setSize(258, 194);
    colorMap->data()->setRange(QCPRange(0, 30.), QCPRange(0, 20.));
    colorMap->setColorScale(colorScale);
    colorMap->setGradient(QCPColorGradient::gpJet);
}
```

这样 colorScale 就添加到了 customPlot 中，并且完成初始设定。

这里我们将 colorScale 置于 colorMap 下方，是为了防止范围改变导致 colorScale 右边标签上的数字长度改变，继而引起 colorMap 的水平轴长度改变，影响动画的演示效果。

4.5 plot 函数

函数定义：

```
    void MainWindow::plot(double a[194][258], QString myString, QCustomPlot* customPlot
{
    double x, y, z;
    for (int xIndex=0; xIndex<194; ++xIndex)
    {
```

```
        for (int yIndex=0; yIndex<258; ++yIndex)
        {
          colorMap->data()->cellToCoord(xIndex, yIndex, &x, &y);
          z=a[xIndex][yIndex];
          colorMap->data()->setCell(yIndex, xIndex, z);
        }
      }
      colorMap->rescaleDataRange();
      QCPMarginGroup *marginGroup = new QCPMarginGroup(customPlot);
      customPlot->axisRect()->setMarginGroup(QCP::msLeft|QCP::msRight, marginGroup);
      colorScale->setMarginGroup(QCP::msLeft|QCP::msRight, marginGroup);
      customPlot->rescaleAxes();
      customPlot->savePng("/Applications/.PlasmaCalc/"+myString+QString::number(t)+".png"
}
```

plot 函数需实现的功能是：将二维数组转化为一张带有 colorScale 的 colorMap，并将它保存。

我们将传入量设置为：二维数组，数组名字符串，customPlot，colorMap，colorScale 对象。这样可以避免对 colorMap 与 colorScale 的重复初始化与设置。由于 QCPMarginGroup 的创建比较方便，也不需要特别的设置，我们将 QCPMarginGroup 对象作为函数中创建的临时对象。它的作用是创建图片外侧的白边。

我们的 customPlot 对象并不是可见的，因此一开始考虑过只创建一个 customPlot 对象，即在同一块绘图板上反复绘制六类图像。然而这样我们的 colorScale 就出现了错误，z 坐标的量级有很大偏差。查阅库函数的定义后，发现 colorScale 是根据 colorMap 中 z 的最大和最小值来定标的，而且 colorScale 的标度只能增大不能缩小，也就是说，最大值只能变大，而最小值只能减小。因此，为了如实反映各物理量数量级的变化，我们需要让它们各自有一个 colorScale 和 colorMap，所以最终我们创建了六个 customPlot 对象，即六块画板。

4.5.1 赋值的实现

利用 cellToCoord 函数，我们将 cell 与 data 关联起来。cell 是 colorMap 上的面元，用整数来索引，而 data 是 colorMap 上的实际点，用 double 格式索引。
利用 setCell 函数给每个 cell 赋值，用格点处数值近似表示面元上数值。

4.5.2 范围的改变

上述程序利用了两个函数：rescaleDataRange() 和 rescaleAxes()。前者表示将 colorMap 中的成员变量——最值根据新的数值改变，后者是改变标度必要的。
经过研究这两个函数足以完成任务，因此我们删除了很多冗余的函数。

4.5.3 图像保存

png 格式是一种适用性很广的格式，许多论文插图都采用这种格式，因此我们选择 png 作为图片导出的格式，而 QCustomPlot 库也给导出 png 提供了方便的解决办法：savePng 函数。

为了不在用户的电脑上产生垃圾，我们使用临时创建的文件夹保存图片，并在程序结束后将文件夹中的图片删除（后面的视频也是这样）。

4.6 写入视频

我们采用 OpenCV 库的 VideoWriter 类将图片写入视频，具体是通过 writevideo 函数实现的，代码如下：

```
    void Videoplayer::writevideo(string myString)
{
    VideoWriter writer;
    int codec = CV_FOURCC('M', 'J', 'P', 'G');
    double fps = 25.0;
    string filename0 = "/Applications/.PlasmaCalc/"+myString+".avi";
    writer.open(filename0, codec, fps, Size(300, 300));
    Mat frame;
    for(int i =1; i<=nts; i++)
    {
        string filename = "/Applications/.PlasmaCalc/"+myString + to_string(i) +".png"
        frame = imread(filename);
        writer << frame;
    }
}
```

先创建 VideoWriter 对象，再利用 open 函数初始化视频编码等。利用 Mat 类逐帧读入视频，再传到 VideoWriter 中。视频命名利用传入的字符串。

4.7 计算进度显示

利用 QProgressDialog 类可以方便地创建进度条窗口，代码如下：

```
QProgressDialog progress("Calculating...", "Cancel", 0, nts*ntp, this);
progress.setWindowModality(Qt::WindowModal);
```

进度显示与结束计算功能的实现如下：

```
 for( ;t<=nts;t++)
{
```

```
for(int tt=1;tt<=ntp;tt++)
{
    sve();
    double f=0.5;
    double fi=0.5;
    sw(f,fi);
    sden(f,fi);
    Poisson();
    sve();
    f=1.0;
    fi=0.0;
    sw(f,fi);
    sden(f,fi);
    Poisson();
    progress.setValue(t*ntp-ntp+tt);
    if(progress.wasCanceled())
    {
        t=1;
        return;
    }
}
```

利用 setValue 函数在每个小循环中给 progress 赋值。当按下 cancel 键时，将静态变量 t 重设为初始值 1 并返回，跳出 void MainWindow::on$_r unButton_c licked()$

4.8 播放视频

4.8.1 视频的定位

我们利用 QMediaPlayer 和 QVideoWidget 类实现视频播放功能。

```
player1 = new QMediaPlayer;
player2 = new QMediaPlayer;
player3 = new QMediaPlayer;
player4 = new QMediaPlayer;
player5 = new QMediaPlayer;
player6 = new QMediaPlayer;
```

```
locate("phi", ui->phi, player1);
locate("wi", ui->wi, player2);
locate("delta ne", ui->deltane, player3);
locate("ne", ui->ne, player4);
locate("vx", ui->vx, player5);
locate("vy", ui->vy, player6);
```

先创建 QMediaPlayer 播放器对象，再将 QMediaPlayer 定位到 QVideoWidget 中，并利用传入的字符串读入已生成的视频。

4.8.2 播放控制

```
//将QMediaPlayer状态转变的信号与pause_after_stop槽关联
connect(player1, &QMediaPlayer::stateChanged, this, &Videoplayer::pause_after_stop
connect(player2, &QMediaPlayer::stateChanged, this, &Videoplayer::pause_after_stop
connect(player3, &QMediaPlayer::stateChanged, this, &Videoplayer::pause_after_stop
connect(player4, &QMediaPlayer::stateChanged, this, &Videoplayer::pause_after_stop
connect(player5, &QMediaPlayer::stateChanged, this, &Videoplayer::pause_after_stop
connect(player6, &QMediaPlayer::stateChanged, this, &Videoplayer::pause_after_stop

//QMediaPlayer的时长与QSlider最大值同步
connect(player1, &QMediaPlayer::durationChanged, ui->slider, &QSlider::setMaximum);
connect(player2, &QMediaPlayer::durationChanged, ui->slider, &QSlider::setMaximum);
connect(player3, &QMediaPlayer::durationChanged, ui->slider, &QSlider::setMaximum);
connect(player4, &QMediaPlayer::durationChanged, ui->slider, &QSlider::setMaximum);
connect(player5, &QMediaPlayer::durationChanged, ui->slider, &QSlider::setMaximum);
connect(player6, &QMediaPlayer::durationChanged, ui->slider, &QSlider::setMaximum);

//QMediaPlayer的播放进度与QSlider值同步
connect(player1, &QMediaPlayer::positionChanged, ui->slider, &QSlider::setValue);
connect(player2, &QMediaPlayer::positionChanged, ui->slider, &QSlider::setValue);
connect(player3, &QMediaPlayer::positionChanged, ui->slider, &QSlider::setValue);
connect(player4, &QMediaPlayer::positionChanged, ui->slider, &QSlider::setValue);
connect(player5, &QMediaPlayer::positionChanged, ui->slider, &QSlider::setValue);
connect(player6, &QMediaPlayer::positionChanged, ui->slider, &QSlider::setValue);

//实现QSlider调控播放进度
```

```
    connect(ui->slider, &QSlider::sliderMoved, player1, &QMediaPlayer::setPosition);
    connect(ui->slider, &QSlider::sliderMoved, player2, &QMediaPlayer::setPosition);
    connect(ui->slider, &QSlider::sliderMoved, player3, &QMediaPlayer::setPosition);
    connect(ui->slider, &QSlider::sliderMoved, player4, &QMediaPlayer::setPosition);
    connect(ui->slider, &QSlider::sliderMoved, player5, &QMediaPlayer::setPosition);
    connect(ui->slider, &QSlider::sliderMoved, player6, &QMediaPlayer::setPosition);

    void Videoplayer::on_pauseButton_clicked()
{
    player1->pause();
    player2->pause();
    player3->pause();
    player4->pause();
    player5->pause();
    player6->pause();
}

void Videoplayer::on_playButton_clicked()
{
    player1->play();
    player2->play();
    player3->play();
    player4->play();
    player5->play();
    player6->play();
}

void Videoplayer::on_stopButton_clicked()
{
    player1->stop();
    player2->stop();
    player3->stop();
    player4->stop();
    player5->stop();
    player6->stop();
}
```

利用 Qt 的信号和槽机制可以很方便地实现播放控制。

首先我们发现视频的初态和终态都显示为黑屏，这是视频的终止状态（stoppedstate）导致的，因此我们设置了 pause$_a f t e r_s t o p$

为了实现基本的播放、暂停、结束以及控制播放进度的功能，我们设置了三个按钮和一个 QSlider 滑动条。播放、暂停、结束分别由三个按钮控制。QSlider 与 QMediaPlayer 的播放的关联由上述三个信号、槽关联建立。

5. 模拟结果与分析

5.1 模拟结果与运算速度

最终，我们成功演化出了漂移波，证实了它的存在，发现了漂移波的移动与增长，同时也模拟出了漂移波最终导致湍流的图像。并制作成了视频。

5.1.1 默认情况

在碰撞常数为 0.05，扩散系数为 0.01，扰动大小为 1e-9，时间倍率与视频长度默认的情况下，第 1 张、第 730 张、第 1014 张的模拟结果如图所示。

由第 1014 张结果图可见，系统明显有了形成湍流的趋势。对比数值，不难发现漂移波的增长。

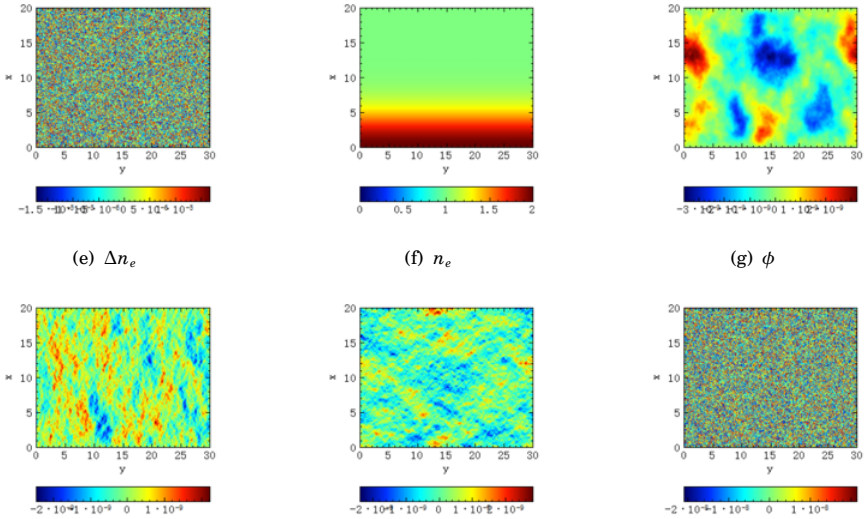
在处理器为 Intel(R)Core(TM)i7-6500U CPU @2.50GHz 2.60GHz 的计算机上，出一张图的时间大约为 25 秒，一共产生了 1014 张图，共耗时约 7 小时。

5.1.2 改变碰撞常数

以下将碰撞常数改为 0.005。

5.1.3 改变扰动量

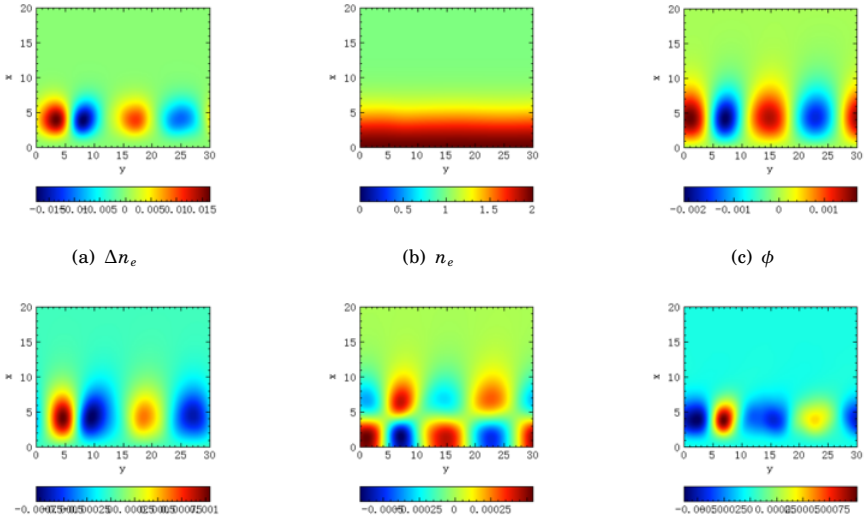
以下将扰动量改为 1e-7。



(e) Δn_e (f) n_e (g) ϕ

(h) v_x (i) v_y (j) w_i

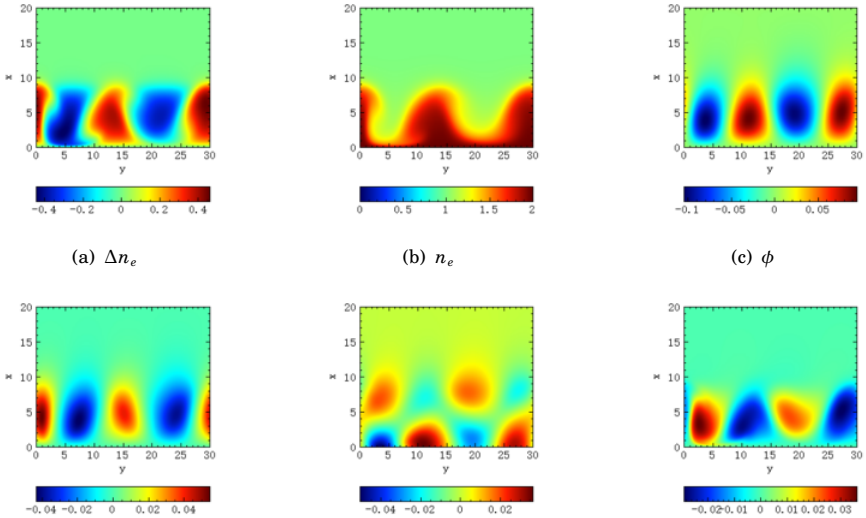
Figure 1: 第 1 张



(a) Δn_e (b) n_e (c) ϕ

(d) v_x (e) v_y (f) w_i

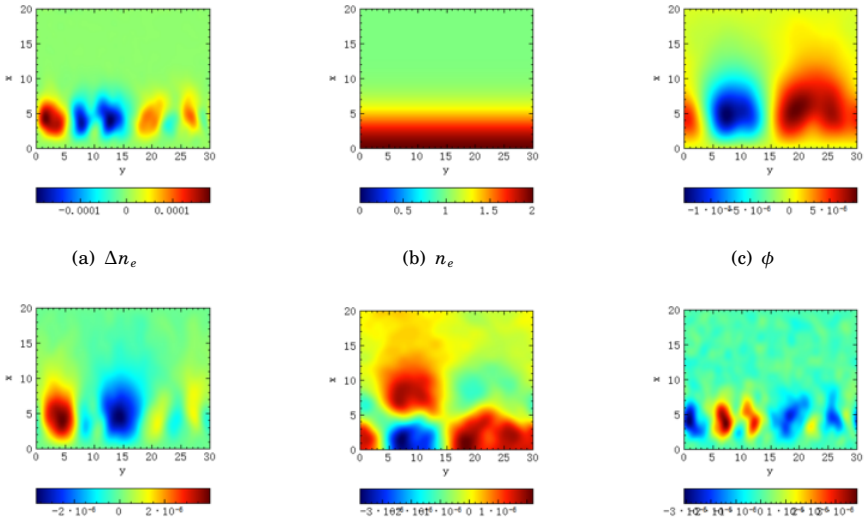
Figure 2: 第 730 张



(a) Δn_e (b) n_e (c) ϕ

(d) v_x (e) v_y (f) w_i

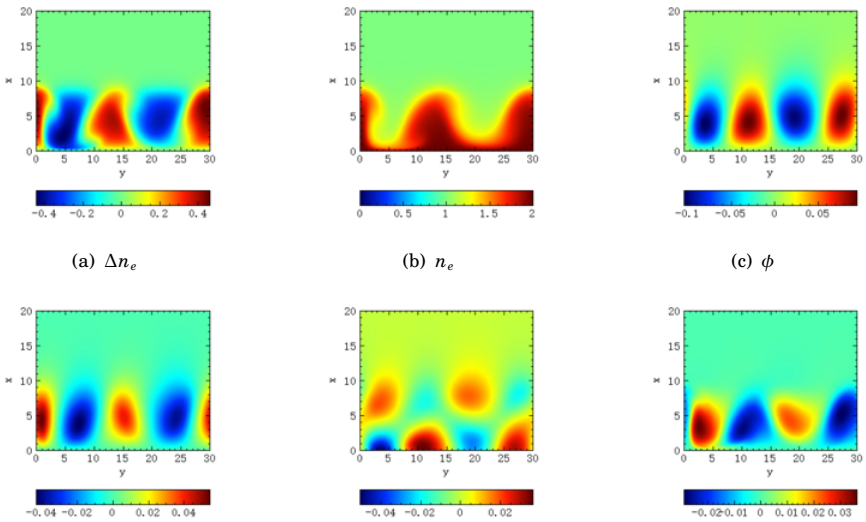
Figure 3: 第 1014 张



(a) Δn_e (b) n_e (c) ϕ

(d) v_x (e) v_y (f) w_i

Figure 4: 改变碰撞常数：第 730 张



(a) Δn_e (b) n_e (c) ϕ

(d) v_x (e) v_y (f) w_i

Figure 5: 改变扰动量：第 730 张

可以看到，体系已进入非线性状态。

5.2 结果分析与物理解释

根据模拟结果我们得出以下结论：

1. 电子数密度的微小扰动最终演化成为了漂移波，数密度等物理量均在漂移波漂移的过程中逐渐增大，并具有形成湍流的趋势。而漂移波存在的物理解释如下：

先不考虑碰撞，则等离子体方程为

$$mn\left[\frac{\partial \boldsymbol{v}}{\partial t}+(\boldsymbol{v}\cdot\nabla)\boldsymbol{v}\right]=qn(\boldsymbol{E}+\boldsymbol{v}\times\boldsymbol{B})-\nabla p$$

$$\left|\frac{mn\dfrac{\partial \boldsymbol{v}}{\partial t}}{qn\boldsymbol{v}\times\boldsymbol{B}}\right|\simeq\left|\frac{mni\omega v_{\perp}}{qnv}\right|\simeq\frac{\omega}{\omega_c}$$

由于在强磁场中，漂移波的漂移速度较为缓慢，故此式近似为 0，$mn\dfrac{\partial \boldsymbol{v}}{\partial t}$ 可以忽略。

根据模型，电子速度分布在垂直轴的平面上。对方程两边同时叉乘 $\boldsymbol{B}$，则等号左侧为 0。

$$\begin{aligned} 0 &= qn[\boldsymbol{E}\times\boldsymbol{B}+(\boldsymbol{v}_{\perp\times\boldsymbol{B})\times\boldsymbol{B})-\nabla p\times\boldsymbol{B}} \\ &= qn[\boldsymbol{E}\times\boldsymbol{B}-\boldsymbol{v_p erp}B^2]-\nabla p\times\boldsymbol{B} \end{aligned}$$

故最终有

$$v_E \equiv \frac{E\times B}{B^2}$$

$$\mathrm{v}_D \equiv -\frac{\nabla p\times B}{qnB^2}$$

由此可以看出，因为抗磁漂移的存在，只要具有 x 方向的密度梯度，则必会演化出漂移波。而漂移波增长的原因，对照图片可得，形成漂移波时，不论哪一种演化，在 Δn_e 极大或极小时，v_x 总是随之处于接近极大或极小的状态，这也导致了密度扰动的不断增长。在物理上，这种情况发生的原因是电子发生的碰撞，使得原本没有相位差的 ϕ 和 Δn_e 产生了一定的相位差，从而使得 $\boldsymbol{v_E}$ 的 x 分量（即 v_x）相位接近于 Δn_e 的相位，产生如此特性。当然，实际情况中，离子的极化漂移也是重要的影响因素，不过在本课题中，由于冷离子模型，不加考虑。

2. 改变碰撞系数，演化过程发生改变。从图中可以看出，相同的相对时间内，碰撞系数变小，漂移波演化变慢。由电势的演化方程：

$$\widetilde{\phi}=\sum_{k_y}\widetilde{\phi_{k_y}}e^{i(k_y y+k_z z-w_r t)+\gamma t}$$

可知若碰撞常数越小，则 γ 越小，漂移波演化得越慢。其实由 2.1.2 中各式可以发现，若碰撞常数趋近于 0，只有扩散项只有扩散项起到了作用，漂移波不再演化，故碰撞系数越小，漂移波越难形成。

3. 改变扰动量，演化过程发生改变。从图中可以看出，扰动量越大，漂移波形成得越快，湍流形成得越早，在物理上这也是显而易见的。不过扰动量不可太大，否则一来不符合微扰的

前提条件，二来使得整个系统来不及形成成型的漂移波便形成了湍流。

6. 结语

6.1 总结与展望

从科学探索的角度，本次课题成功地完成了预设目标，验证了漂移波的存在和增长，完成了对模拟结果的科学分析并且实现了视频的可视化。通过这次课题，我们组的成员们对数值模拟，科学研究有了更清晰的认识，和更深入的了解，实现课题的过程中，培养我们编程能力的同时，也增长了我们的信息搜集能力、快速学习能力和团队合作能力，这些都让我们受益匪浅。

我们的课题还可以使模型更加复杂化，还可以使用户体验更加多样化，使科学模拟更加精确化，还可以与实际的联系更加紧密化，这些都是我们可以继续提升的空间。

算法部分，我们的模型可以变得更加复杂，可以解开等温冷离子的条件限制，提高算法的难度，提高自由度。

界面部分，我们还可以做得更加美观，还可以再思考哪些有意义的参量可以再设计，增加用户体验性。不过，受模型和课题本身的限制，我们没能想出更好更有意义的可调参数，实属可惜。

不过不管怎样，一路走来，我们攻克层层难关，走走停停，时而另辟蹊径，我们共同努力，携手同心，终于完成了最初的目的。这样的经历，确实值得回味。

6.2 课题分工明细

在本次课题研究中，人员分工如下：

唐钺：决定了课题任务和内容，负责了课题的原理分析、公式推导、参数设定、计算方法部分，撰写了相应内容的结题论文，参与并实现了程序的最终整合与优化。制作了结题报告的 powerpoint 文档。

陈远微：负责了课题的界面设计、colorMap 绘制、视频制作、程序发布等部分，完成了图像与视频的结合、界面的美化、图像的流畅处理，撰写了相应内容的结题论文，参与并实现了程序的最终整合与优化。并负责了论文的统稿和 LaTeX 化。

6.3 致谢

首先要感谢雷奕安教授，感谢他一个学期以来的悉心讲解与富有启发性的指导，感谢他给我们提供了一个极好的学习和锻炼的机会。在这次小课题的研究中，我们不仅学到了知识，而且磨练了面对困难的意志，增强了克服问题、解决问题的能力，意识到以严谨的态度对待课题的重要性。这对我们以后的学习乃至人生道路都是很有帮助的。

也要感谢李博老师。若没有李博老师的引导，我们组的课题就不知会从何而来；若没有李博老师的指导和帮助，在课题进展陷入僵局时，我们就不会知道还会在原地徘徊多久。

还要感谢孟聪学长，感谢他的引路，感谢他耐心的指导，感谢他对我们课题研究路程的参与与见证。

还要感谢谢华生博士。感谢谢博士在我们攻克核心程序面临瓶颈时，从百忙之中抽空为我们指点，并给我们提出了决定性的建议。在程序 BUG 一直无法得到解决的情况下，感谢他耐下性子，一点一点的帮助我们分析程序，提供意见。

还要感谢彭良友教授。不曾相见，却只因我一封邮件的请求，便向我推荐了组里的三位富有经验，实力雄厚的学长。感谢肖相如学长，曹睿枭学长，当然也要感谢韩兆宇学长，韩学长无私地给予了我许多的帮助，我很是感动，当然还要感谢李宇星学长..... 感谢这么多曾给予过无私帮助的朋友们，在我们遇到困境时，是他们的陪伴和建议点破了我们的迷津。

感谢聂瑞娟老师，是她将李博老师和彭良友老师推荐与我，是我们课题的重要引路人！

最后感谢我们的同学，是我们的讨论让我们共同进步，是我们的相互帮助让我们共度难关！