Numerical Methods and Two Stream Instability Simulation

PIC and SOR algorithms in Python

Aidan Klemmer

Earlham College
Math Methods and Physics
April 24, 2018

Motivation:

In many fields of Physics the exact solutions to a behavior cannot be found analytically. In these situations, numerical methods are extremely popular and have been used extensively for many decades even before personal computing existed. Specifically, for certain types of partial differential equations, the mathematics given the many parameters of the system are extremely challenging to solve analytically in a reasonable amount of time. In fields such as plasma physics, numerical methods are important as they are the only way to solve and simulate the equations used to described plasma. One of the greatest problems in tracking the vast number of particles. Even in low density "cold" plasmas, the density of electrons present in a plasma can be around one hundred million per cubic centimeter, and in high density "hot" plasmas the density is closer to one trillion electrons per cubic centimeter.[1]

These systems are made even more complex when one understands the collective behavior nature of plasma. Plasma is defined as ionized gas such that the molecules of gas are split (ionized) into their individual ions and electrons. The extent to which a plasma is ionized (α) is of great concern in many plasma equations, but for the scope of this paper and project we will consider constant α with a value of approximately 1 where the plasma is approximately fully ionized. The equations to explain the behavior of system of particles in the plasma are increasingly complicated then as each ion and electron are individually charged, with opposite signs, and possess differing masses. Each ion will be attracted to the electron of the opposite sign and vice versa, along with ions repelling other ions and electrons repelling other electrons.

In this way, the motion of individual electrons throughout the plasma generates forces and motion in the rest of the particles of the plasma giving way to the collective behaviors mentioned. The collective behavior of plasma is often better thought of in terms of fluid dynamics where the fluid is like water for example, but electrically conductive. Further, the net charge of the plasma is usually zero, however the individual ions and electron are not, and thus regions of the plasma may have a net charge negative or positive in various situations.

The challenge of calculating and modelling the overall behavior of a plasma is very great indeed. Many of the techniques and methods to be discussed in this paper were designed

[1] https://www.thierry-corp.com/plasma/knowledge/what-is-plasma-density

specifically for plasma simulations as far back as the 1940's and 50's. Today, these techniques are still used to great success but take advantage of advances in computing power using super computers to run calculations. Computational plasma physics is an incredibly interesting and important field of study for plasma physics, astrophysics, and fusion research.

Plasma Physics:

There are two equations of plasma physics, beyond what have been mentioned previously, that will be discussed in further detail. These equations, the Poisson equation and the Vlasov equation, are used in other fields, but are very important for numerical methods to plasma problems for reasons we will soon see.

The Vlasov-Maxwell system of equations, see Eq. 1, describe the kinetic collision events of a plasma interaction more accurately than the Boltzmann equation for long range Coulomb effects.

$$\frac{\partial f}{\partial t} + \vec{v} \cdot \frac{\partial f}{\partial t} + \frac{q}{m}(\vec{E} + \vec{v} \times \vec{B}) \cdot \frac{\partial f}{\partial \vec{v}} = 0$$

Eq. 1

Where $q, m, E(x,t)$ are the particle electric charge, particle mass, and electric field respectively. The Vlasov-Poisson equation, see Eq. 2, is an equation like the Vlasov-Maxwell system of equations except that the magnetic field term is considered zero, and velocities are non-relativistic. The forms of the equations are the same, except with the terms simplified slightly so that eq. 1 becomes the Vlasov-Poisson equation[2]:

$$\frac{\partial f}{\partial t} + \vec{v} \cdot \frac{\partial f}{\partial x} + \frac{qE}{m} \cdot \frac{\partial f}{\partial \vec{v}} = 0$$

Eq. 2

Next, we will make some assumptions about the system to further simplify our equations. We realize that the particle density $f(x,v,t)$ is constant along the particle's (ion, electron, etc) trajectory. This means that the density derivative with respect to time is constant; $\frac{\partial f}{\partial t} = 0$. We then find our Vlasov equation to be:

[2] http://www.phys.nthu.edu.tw/~thschang/notes/PP06.pdf

$$\frac{\partial x}{\partial t} = \vec{v}$$

Eq. 3

$$\frac{\partial \vec{v}}{\partial t} = \frac{q}{m}(\vec{E})$$

Eq. 4

We will attempt to find $\vec{x}$ and $\vec{v}$ such that they satisfy the Vlasov equation of motion eq. 4 above.

The Poisson Equation is a partial differential equation that is used extensively all throughout mathematics and physics. In plasma physics it is used mainly to calculate the potential field ϕ generated from a charge or charge density ρ. The Poisson Equation is written:

$$\nabla^2\phi + \rho = 0$$

Eq. 5

The ∇^2 is the Laplacian operator acting on the charge density. We can rewrite eq. 5 to show the Laplacian in its derivative form to be:

$$-\frac{\partial \phi^2}{\partial x^2} = \rho$$

Eq. 6

Numerical Methods:

Particle in cell

The Particle In Cell (PIC) method is a numerical method for solving partial differential equations. This method works by approximating the charge potential distributions in space as charge potentials at many nodes within a grid. At the same time, it follows the path of many "super particles", idealized collections of particles, as they move through continuous space. If the number of nodes within a grid space is great enough and other parameters are satisfied, reasonable approximations for the motion of an ion or electron through space may be obtained.

We will try to gain some insight into the physical system we are using for our model. The grid space is divided into many small squares that have a node at each intersection. At each node,

a value for charge is assigned that represents the charge distribution at that location. This grid, along with many nodes, and a particle can be seen in fig. 1.

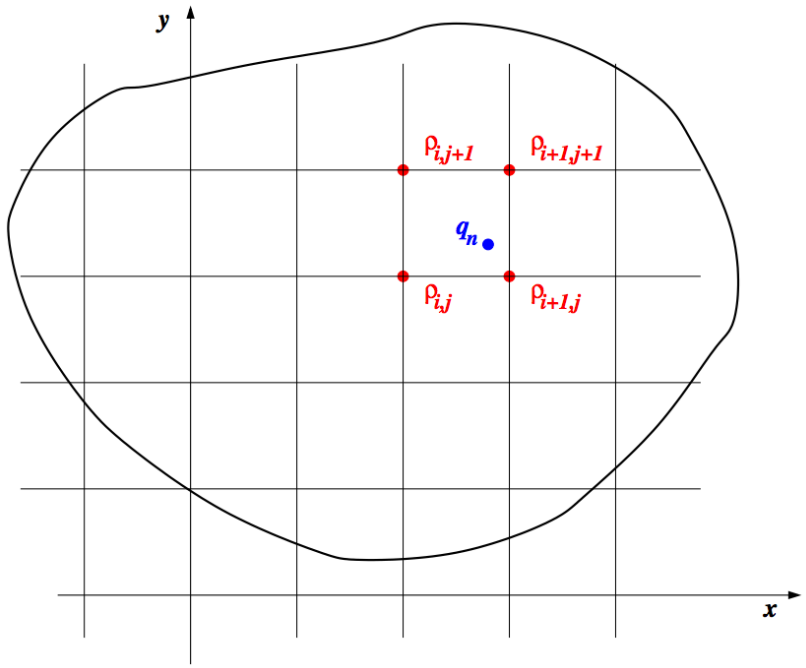


Fig. 1[3]

The PIC method can be implement in many ways, with a wide range of accuracies and complexity. However, the PIC process can be simplified to a process that is characterized by a four-step iteration that is seen all throughout PIC code. The four step PIC iteration is:

1) Integration of equations of motion
2) Assign charge and current values to field grid
3) Compute the fields present on the nodes in grid
4) Assign field values from grid to particle locations

This process is iterative, meaning that this four-step process is a loop that must happen many times over the course of the entire calculation. The length of the simulation, usually described in

[3] Patrick Guio, 2012. Department of Physics and Astronomy, University College London. http://www.ucl.ac.uk/~ucappgu/files/picsimBench.pdf

number of times iterating through these four steps, is divided into small sections of time Δt. The iterative PIC process can be expressed in a diagram as seen in Fig. 2.

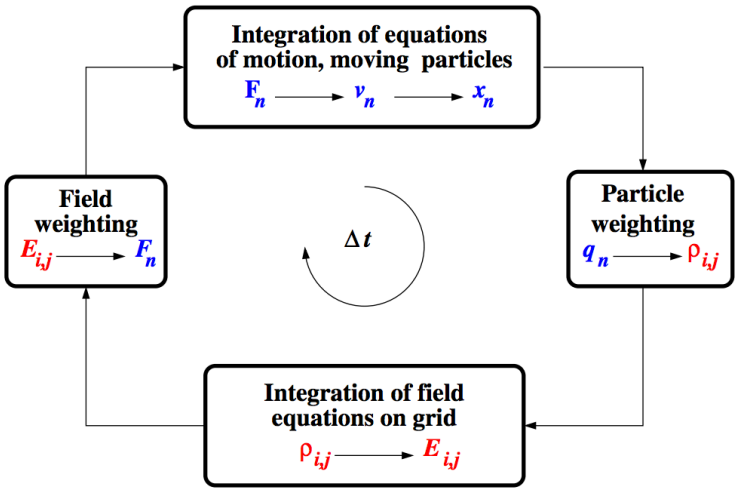


Fig. 2[4]

Within the PIC process, the "integration of field equation step" (see bottom of fig. 2) is of interest. This step often utilizes another very interesting numerical method to find a solution to a linear system of equations. Here, the method discussed is the Successive Over-Relaxation (SOR) method where a partial differential equation such as the Poisson equation, eq. 5, is written as a linear system of equations of the matrix form:

$$Ax = b$$

Eq. 7

A is the Laplacian central difference matrix value, x is the charge potential ϕ, and b is the charge density ρ.

SOR Method

The Successive Over-Relaxation (SOR) method is a numerical method for solving a class of partial differential equations such as the Poisson equation. By expressing the differential

[4] Patrick Guio, 2012. Department of Physics and Astronomy, University College London. http://www.ucl.ac.uk/~ucappgu/files/picsimBench.pdf

equation as a linear system of equations as seen in eq. 7, solutions to the equations may be found through a converging iterative process. To see how one would implement the SOR method, we will start with the Poisson from eq. 5, turn it into the linear system of equations from eq. 7, then manipulate it into something that will be useful in solving the Poisson equation computationally.

Derivation:

We start with the Poisson equation:

$$\nabla^2\phi + \rho = 0$$

Rewrite it into a linear system of equations. Then multiplied by a relaxation constant w to obtain:

$$Ax = b$$

$$(Ax)w = wb$$

Eq. 8

Writing the matrix A as its diagonal (D), upper (U), and lower (L) components:

$$(wD + wU + wL)x = wb$$

Lastly, a series of algebra steps leads to an equation for x:

$$(D + wL + wU + wD - D)x = wb$$

$$[D + wL + wU + (w-1)D]x = wb$$

$$(D + wL)x + [wU + (w-1)D]x = wb$$

$$(D + wL)x = wb - [wU + (w-1)D]x$$

$$x = (D + wL)^{-1} \cdot [wb - [wU + (w-1)D]x]$$

Taking the triangular form of (D + wL) and applying forward substitution (see below):

$$x = (1-w)x + \frac{w}{a}(b - \sigma)$$

$$x_{new} = (1-w)x_{old} + \frac{w}{a}(b - \sigma)$$

$$x_{new} = (1-w)x_{old} + \frac{w}{a_{ii}}(b_i - \sum_{j<i} a_{ij}x_j^{k+1} - \sum_{j>i} a_{ij}x_j^{k})$$

Eq. 9

Sigma σ is the difference in sums between new and old (k+1 and k), where k is the current iteration and time step. The final equation, eq. 9, is useful as it allows us to express a new value of x in terms of its old value and a and b. These values of a and b, or rather matrices as part of a linear system of equations, are equivalent to the A and b of eq. 7 and 8. This is the form that will be useful for numerically solving the Poisson equation. If we substitute in variables for charge potential, charge density, we get:

$$\phi_i^{k+1} = (1 - w)\phi_i^k + \frac{w}{a_{ii}}(b_i - \sum_{j<i} a_{ij}x_j^{k+1} - \sum_{j>i} a_{ij}x_j^k)$$

Eq. 10

Triangular Form and Forward Substitution[5]:

In linear algebra, a matrix (A) may be decomposed into two other matrices, an upper (U) and lower (L). These upper and lower matrices are triangular matrices with only non-zero elements in the upper and lower halves of the matrix respectively.

For example:

$$A = LU$$

Eq. 11

This matrix decomposition of A into L and U can be written:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix}$$

Now, we want to solve the matrix equation, a system of equations, for x, where A is now lower elements of the lower matrix of A. Our matrix equation is:

$$Ax = b$$

Eq. 12

[5] Matrix equations and graphics are taken from:
https://en.wikipedia.org/wiki/Triangular_matrix
https://www.gaussianwaves.com/2013/05/solving-a-triangular-matrix-using-forward-backward-substitution/

$$\begin{bmatrix} a_{11} & 0 & \dots & 0 \\ a_{21} & a_{22} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mm} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}$$

Eq. 13

Once here, using the upper and lower triangular matrices, forward substitution may be used to solve the matrix equation. Starting with the first x and L terms, we may substitute values forward, and eventually we obtain the equation for any x term of the matrix. This process is outlined here:

$$\begin{aligned} x_1 &= b_1/a_{11} \\ x_2 &= (b_2 - a_{21}x_1)/a_{22} \\ x_3 &= (b_3 - a_{31}x_1 - a_{32}x_2)/a_{33} \\ &\vdots \\ x_m &= b_m - a_{m1}x_1 - a_{m2}x_2 - \dots - a_{m,m-1}x_{m-1})/a_{mm} \end{aligned}$$

Eq. 14

This may be written further using a summation as:

$$X_m = \frac{b_m - \sum_{i=1}^{m-1} a_{m,i}x_i}{a_{m,m}}$$

Eq. 15

This is eq. 10, but without the SOR algorithm applied to it.

Finite Difference Method:

Using the finite difference method, we can approximate the Laplace operator. The Laplace operator can be written as a sum of the second partial derivatives of a function, and can be expressed in a matrix. The Laplace operator can be written as a sum:

$$\Delta u(x) = \sum_{i=1}^{n} \partial_i^2 u(x)$$

Eq. 16

Next, in a single dimension the Laplace operator can be approximated in fractional form. The Laplace approximation is written in eq. 17 below. Here u is a function of variable x, and h is an integer.

$$\Delta u(x) = u''(x) \approx \frac{u(x-h) - 2u(x) + u(x+h)}{h^2}$$

Eq. 17

Eq. 17 can be written in a more useful manner if we express the Laplacian in matrix form with the contributing terms across the diagonal[6]:

$$\begin{bmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & -1 & 2 & -1 & \\ & & & \cdots & \\ & & & -1 & 2 \end{bmatrix}$$

Eq. 18

The matrix form holds significance as the value for a_{ii} in eq. 10 above is found from this finite difference approximation. This is perhaps more obvious if we write the matrix in stencil form:

$$\frac{1}{h^2}\begin{bmatrix} 1 & -2 & 1 \end{bmatrix}$$

Eq. 19

This leads to the constant value of -2 seen in the Python implementation.

Application for PIC and SOR methods:

Two Stream Instability

One of the basic problems of plasma physics is the two-stream instability. It describes the dynamics of two streams of plasma passing close to each other in opposite directions. The

[6] The background provided here is extremely brief and is only meant as an introduction to the respective numerical methods. The following link provides a far more detailed explanation. http://www.mathematik.uni-dortmund.de/~kuzmin/cfdintro/lecture4.pdf

dynamics include a fluctuation, vortices, and can generate an oscillating instability over time.

The plasma vortices generated can be thought of as, and indeed behave like, fluid vortices seen commonly in water. The following analog is very helpful in understanding the system and its dynamics, so indulge me with your attention. As water in a river passes around objects such as rocks, it creates vortices where water is flowing in both directions. More accurately, the rock blocks the river in that spot, causing water to divert its original path, and flow around it. When water taking either path around reaches the end of the rock, it has the option to continue its path or "fill in" the space blocked by the rock. Some of the water from each path around the rock will in this space behind the rock until the space is filled, and the system is in equilibrium. The rotation of the water when it fills in this space creates an oscillation and vortices between the outside water path and the water filling in space behind the rock.

In a PIC simulation like we have discussed, the plasma is a single "body" in space. But this is of no interest and as such we will explore the two-stream instability problem. As with water in my analogy, plasma streams travelling in opposite directions will interact as seen in fig. 3.

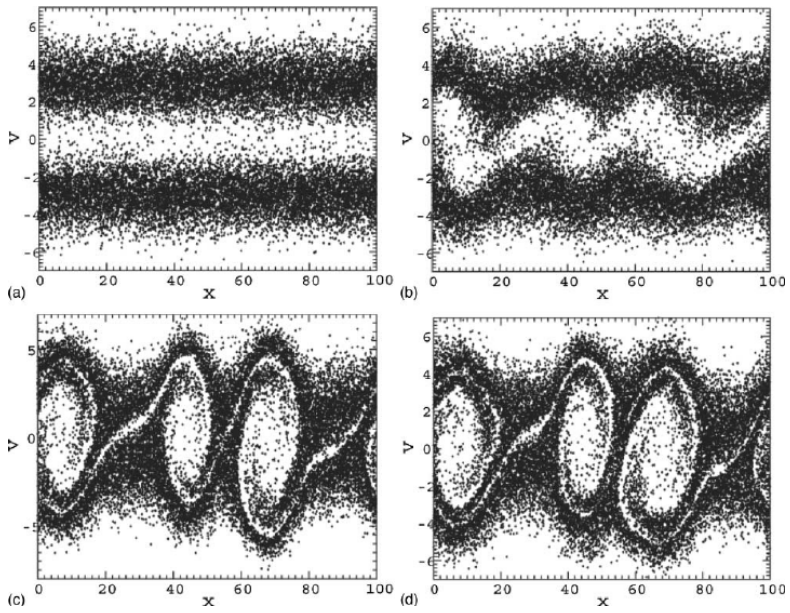


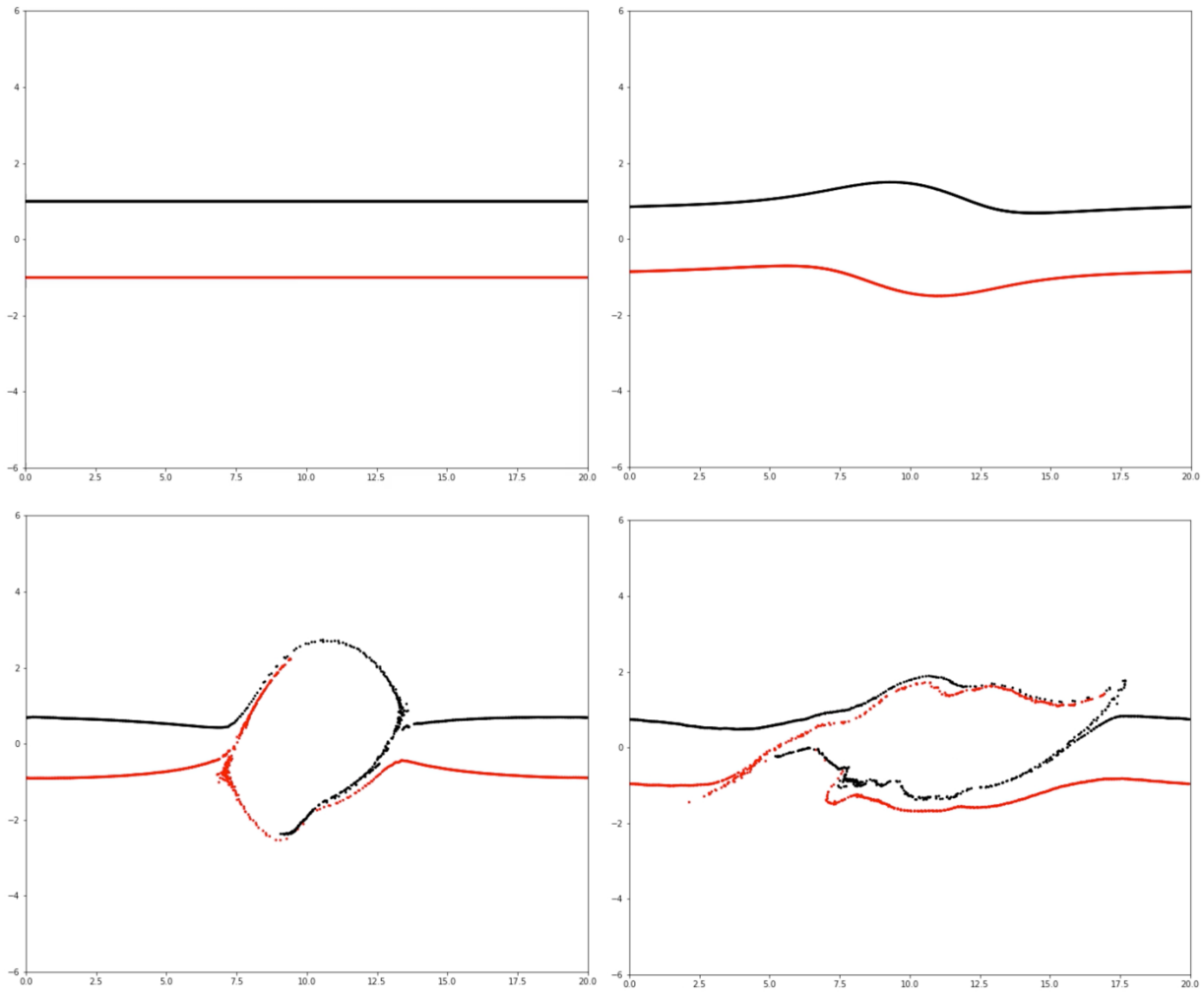
Fig. 3[7]

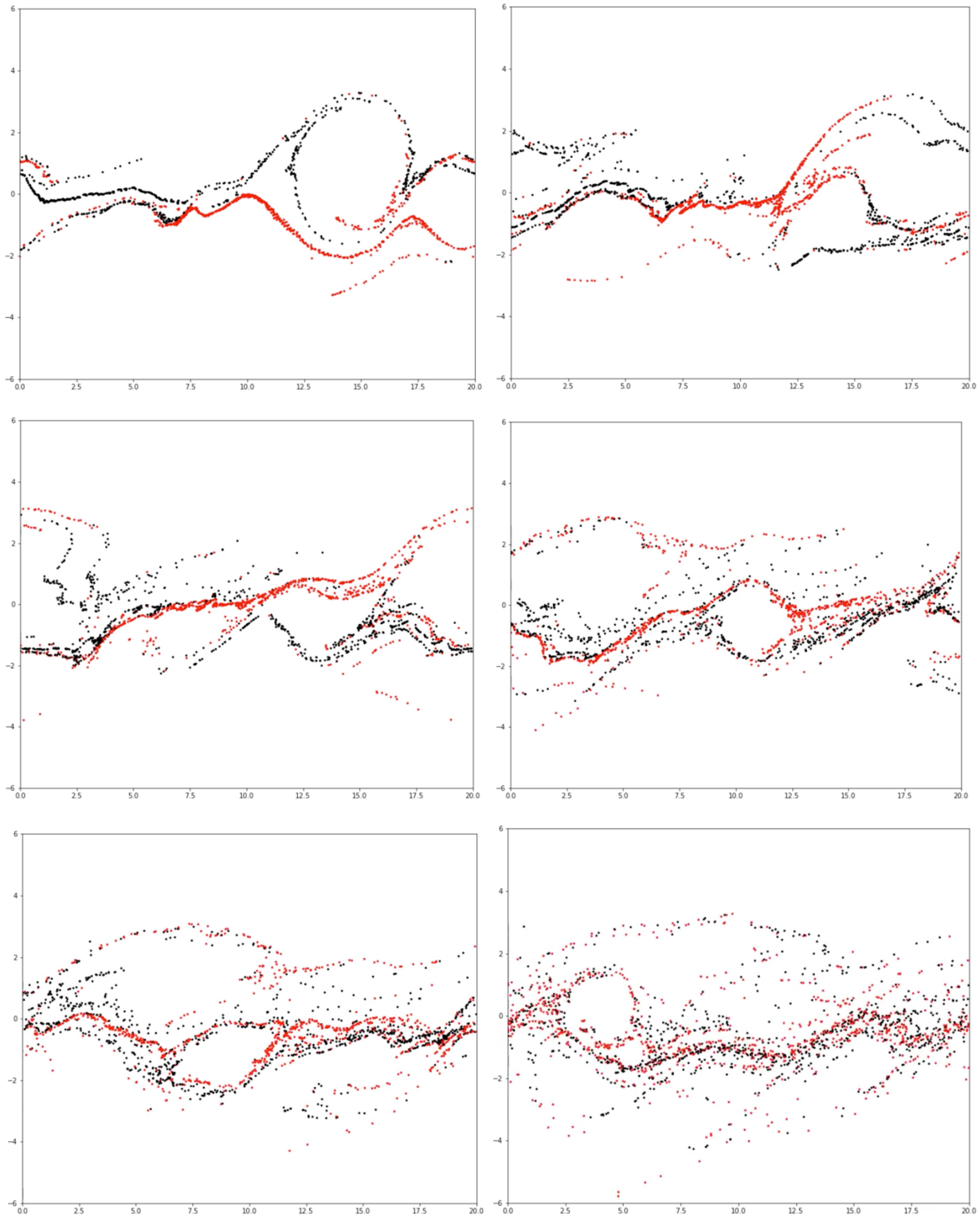
[7] Ha, Seung Yeal & Ha, Taeyoung & Hwang, Chi-Ok & Lee, Ho. (2011). Nonlinear instability of the one-dimensional Vlasov-Yukawa system. Journal of Mathematical.

The simulation seen in fig. 3 was done with very sophisticated and complex code and serious computing power. This can be seen in that the number of particles tracked is very high, and resolution of the dynamics is also high.

However, for our purposes we wanted simplicity while maintaining an accurate physical model. We chose to limit the number of particles to 2000 and the limit the number of modelled interactions. We chose to use Python 3 running NumPy, mathplotlib, and ffmpeg for our computation, plotting, and animation. Following the PIC method, and implementing the SOR iterative method, we created a meaningful 1D simulation of a basic two stream instability. The short-term behavior was found to be match predictions.

The images below, taken from an animation, show the results using these processes. The process of making the calculations and recording them in files presented fewer difficulties than reading and plotting the data. The images are taken at roughly 1.5 second intervals.





These images are from a 17 second animation generated from the 500 step PIC with SOR computations. The 500 step PIC code, with 2000 particles, and 100 nodes produced a 17 second

animation at 30 fps. To store the data needed for this animation we made one file for each step. Within each file, we recorded the position and velocity of each of the 2000 particles.

The code used can be found at: https://github.com/aidank03/Projects

The Jupyter file found there presents a condensed explanation of the background and derivations along with basic version of the PIC code.

Special Considerations:

w Relaxation Constant:

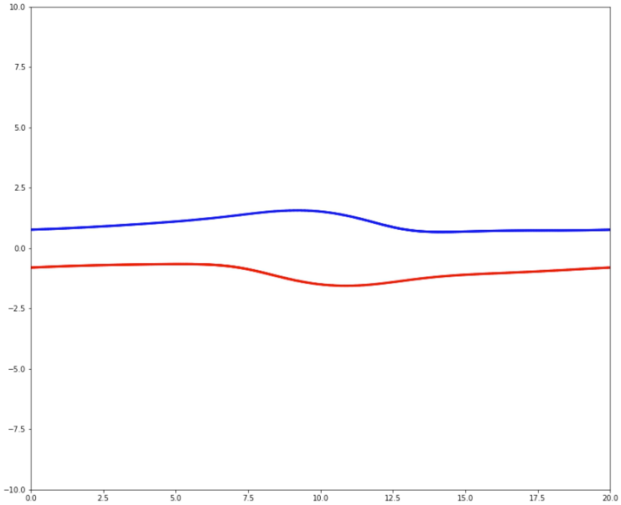
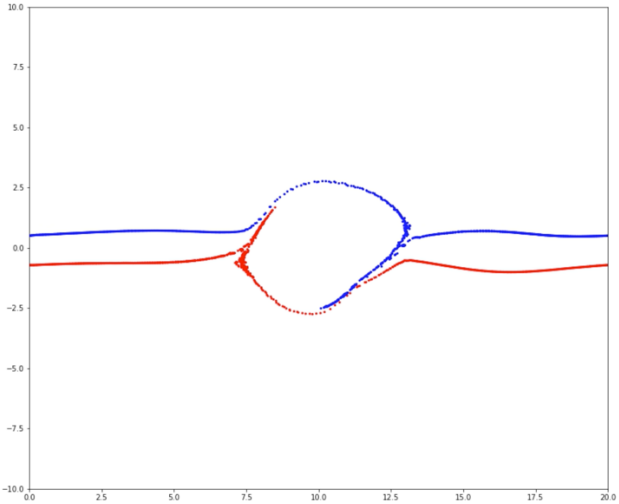
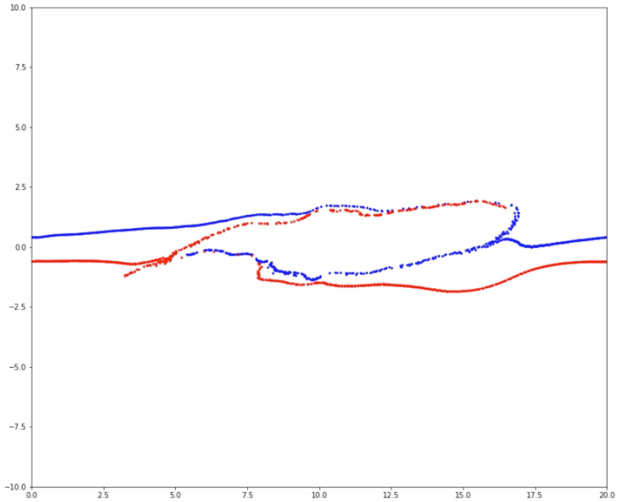
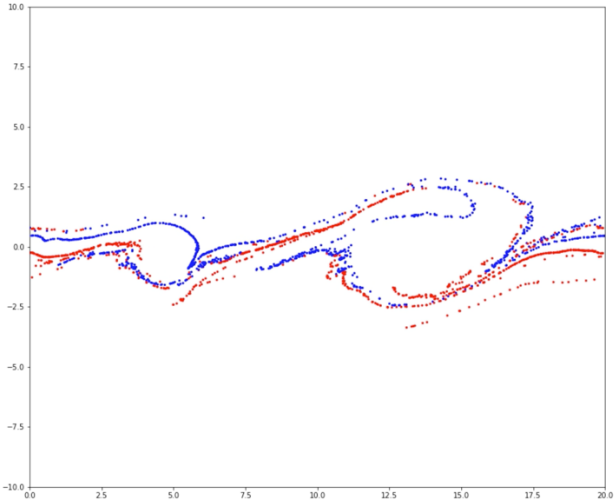
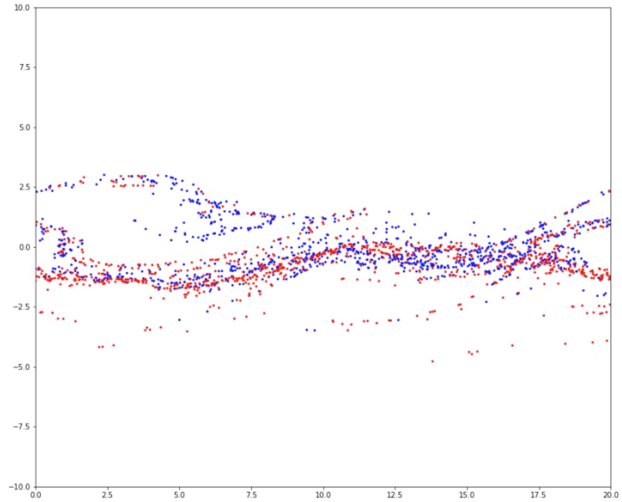
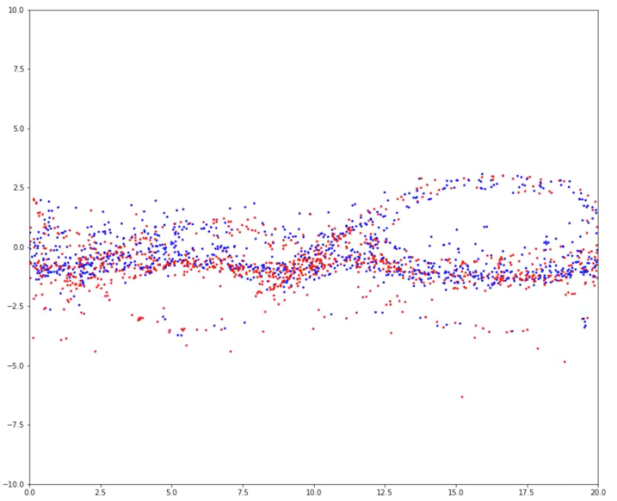
Within the code there are many factors that may be studied further. Of some interest is the relaxation factor w that is the constant used within the SOR method for solving the Poisson Equation. The range of possible w is:

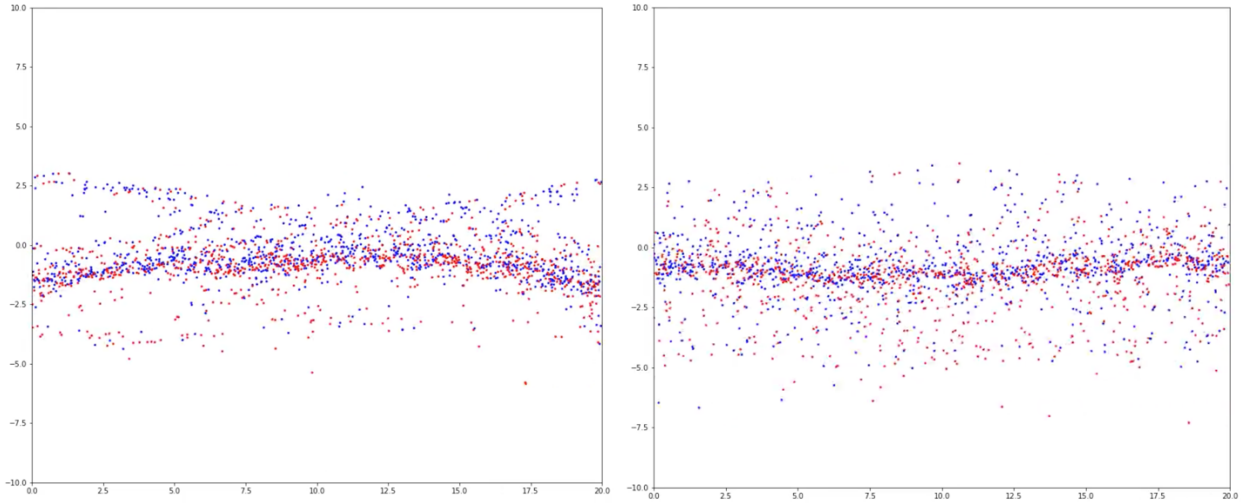
$$0 < w < 2$$

Determining the optimal w value is important in that it makes a very significant difference in the convergence speed of the iteration. For this code, a maximum of 10,000 iterations were used to determine charge new potentials for each step. Experiments were done with varying values of w to determine experimentally the optimal value and if it would result in differing behaviors in the animation. The results were largely inconclusive as calculation times varied by too small a percentage to accurately calculate. In near future, I will be experimenting with having the system calculate time elapsed for calculation to gain more accurate insight into the possible small speed differences.

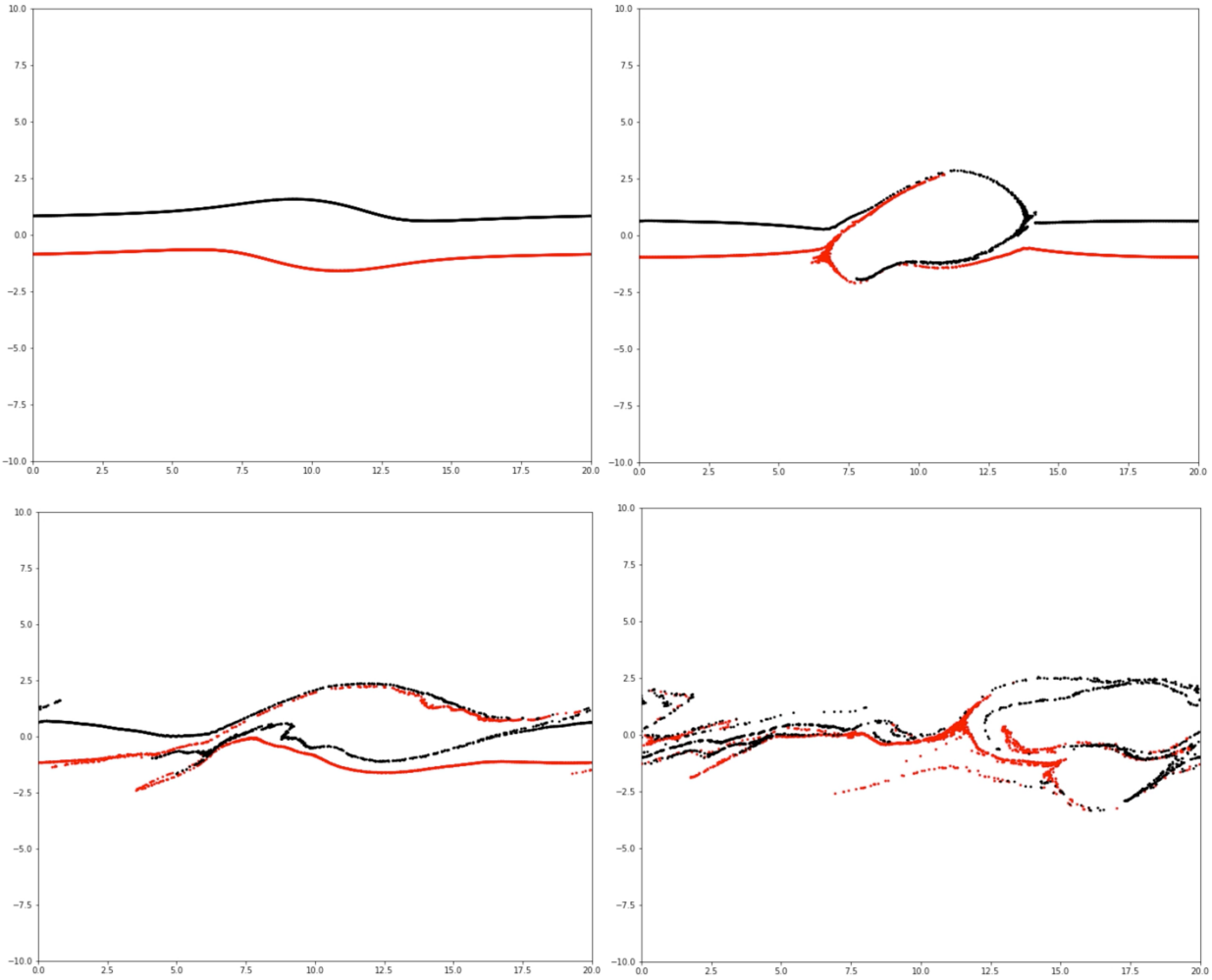
Plasma Perturbation Amplitude:

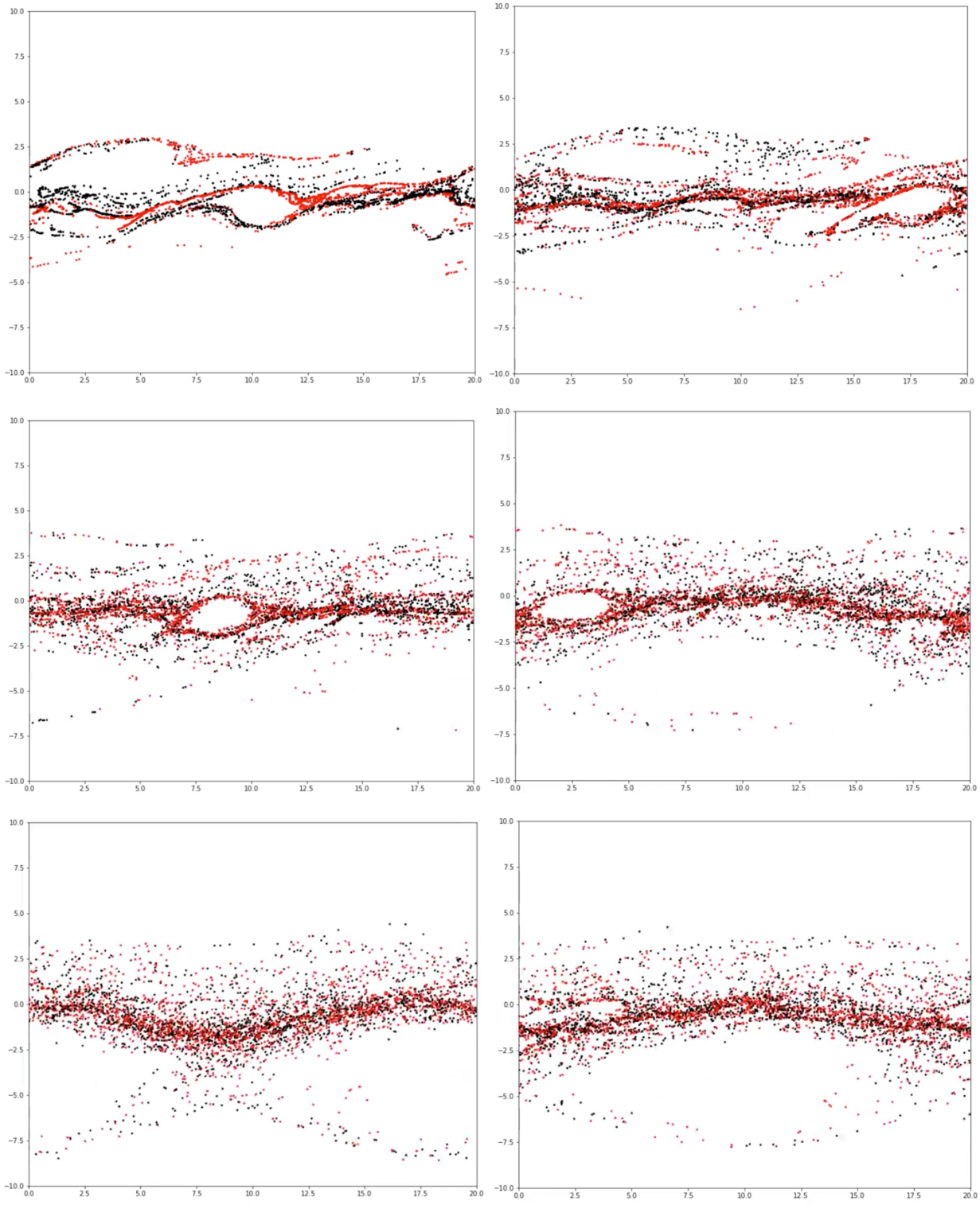
The next thing that was studied further was the effect of varying the plasma perturbation amplitude on the collective behavior of the two streams. It was found that decreasing the amplitude increased the stability of the collective two stream dynamics such that the instability decreased over time, at least in this single trial. The following images show the low amplitude behavior of a 0.001-unit amplitude, 2000 step, 2000 particle simulation to have increasing stability over time.

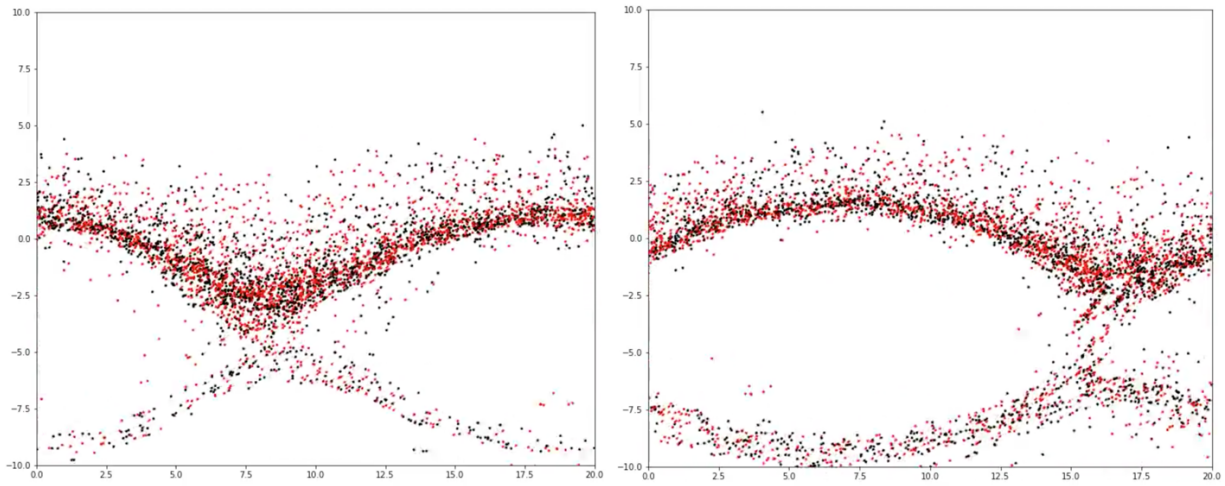









These results are very intriguing when compared to the results of a 0.1-unit amplitude, 2000 step, 4000 particle simulation seen below:







It can be seen clearly that the increased amplitude in the second set of images show an increasing instability that grows over time. Each of the two sets of animations were 1:17 minutes long with images taken at approximately even intervals from start to finish.

Notable Resources Used:

Algorithms code:

1) http://www.southampton.ac.uk/~feeg6002/lecturenotes/feeg6002_numerical_methods01.html

2) https://github.com/haykh/1D_p-i-c

Plasma physics via computer simulation; Adam Hilger Series on Plasma Physics. CK Birdsall and AB Langon. http://www.phy.pku.edu.cn/~lei/cp/ppvcs.pdf

PIC Particle simulation methods:
http://theory.ipp.ac.cn/~yj/research_notes/particle_simulation.pdf

Stanford two stream instability lecture:
http://sun.stanford.edu/~sasha/PHYS780/PLASMA_PHYSICS/phys780_2014_l9.pdf

Linear system of equations and central difference:
http://www.mathematik.uni-dortmund.de/~kuzmin/cfdintro/lecture4.pdf

Triangular Form and Forward Substitution:
https://en.wikipedia.org/wiki/Triangular_matrix#Forward_and_back_substitution