Computational Methods in Plasma Physics

Richard Fitzpatrick

Institute for Fusion Studies

University of Texas at Austin

Purpose of Talk

- Describe use of numerical methods to solve simple problem in plasma physics that, nevertheless, cannot be solved via standard analytic methods.

Plasma Physics Problem

- Collisionless, unmagnetized, 1D (i.e., $\partial/\partial y = 0$, $\partial/\partial z = 0$) plasma consisting of N electrons and N singly-charged ions.
- Ions much more massive than electrons, so on sufficiently short time-scales can treat ions as static, uniform, neutralizing background, and only consider motion of electrons.
- Initial state: Two counter-propagating warm electron beams.
- If beam speed significantly larger than thermal speed then system unstable to *two-stream instability* (see Wikipedia page).
- At small perturbation amplitudes, two-stream instability can be treated analytically. At large perturbation amplitudes, analytic methods break down, necessitating numerical approach.

Electron Equations of Motion

- Let r_i be x-coordinate of ith electron. Equations of motion of ith electron are

$$\frac{dr_i}{dt} = v_i,$$

$$\frac{dv_i}{dt} = -\frac{e}{m_e} E(r_i),$$

  where $e > 0$ is magnitude of electron charge, m_e is electron mass, and $E(x)$ is x-component of electric field-strength at position x.

- N.B. 1D nature of problem implies that mode is *longitudinal*: i.e., $\mathbf{k} \propto \mathbf{E}$. Longitudinal plasma modes do not self-generate magnetic fields. Hence, absence of magnetic term in equations of motion.

Poisson's Equation

- Electric field-strength expressed in terms of scalar potential,

$$E(x) = -\frac{d\phi}{dx}.$$

- Poisson-Maxwell equation yields

$$\frac{d^2\phi}{dx^2} = -\frac{e}{\epsilon_0}\,[n_0 - n(x)]\,,$$

where ϵ_0 is permittivity of free-space, $n(x)$ is electron number density [i.e., number of electrons in interval x to $x + dx$ is $n(x)\,dx$], and n_0 is uniform ion number density.

- Spatial average of $n(x)$ equal to n_0, since equal numbers of ions and electrons.

Initial State

- Initial electron distribution function consists of two counter-propagating Maxwellian beams of mean speed v_b and thermal spread v_t: i.e.,

$$f(x,v) = \frac{n_0}{2}\left[\frac{1}{\sqrt{2\pi}\,v_t}\,e^{-(v-v_b)^2/(2v_t^2)} + \frac{1}{\sqrt{2\pi}\,v_t}\,e^{-(v+v_b)^2/(2v_t^2)}\right].$$

- $f(x,v)\,dx\,dv$ is probable number of electrons between x and $x+dx$ with velocities in range v to $v+dv$.
- Beam temperature, T, related to thermal spread via $v_t = \sqrt{T/m_e}$, where T measured in energy units.

Normalization

- First step in solving physics problem numerically is to *normalize* governing equations: i.e., render all variables dimensionless, and as many as possible $\mathcal{O}(1)$.
- Normalization reduces likelihood of arithmetic underflow and overflow.
- Normalization facilities identification of terms in governing equations that can be safely neglected (such terms having magnitudes much less than unity).

Normalization Scheme

- All times expressed in units of $\omega_{\rm p}^{-1}$, where

$$\omega_{\rm p} = \sqrt{\frac{n_0\, e^2}{\epsilon_0\, m_e}}$$

  is *electron plasma frequency*: i.e., typical frequency of electrostatic electron oscillations.

- All lengths expressed in units of *Debye length*,

$$\lambda_{\rm D} = \frac{v_{\rm t}}{\omega_{\rm p}} :$$

  i.e., typical length-scale above which electrons exhibit collective (i.e., plasma-like) effects, instead of acting like individual particles.

Normalized Equations

- Normalized electron equations of motion:

$$\frac{dr_i}{dt} = v_i,$$

$$\frac{dv_i}{dt} = -E(r_i),$$

$$E(x) = -\frac{d\phi}{dx},$$

$$\frac{d^2\phi}{dx^2} = \frac{n(x)}{n_0} - 1.$$

- Normalized initial distribution function:

$$f(x,v) = \frac{n_0}{2}\left[\frac{1}{\sqrt{2\pi}}\, e^{-(v-v_b)^2/2} + \frac{1}{\sqrt{2\pi}}\, e^{-(v+v_b)^2/2}\right].$$

Spatial Domain of Solution

- Equations solved in spatial domain

$$0 \leq x \leq L,$$

  where $L \gg 1$: i.e., plasma many Debye lengths in extent.

- For sake of simplicity, adopt *periodic* boundary conditions: i.e., identify left and right boundaries of solution domain.

- Follows that $n(0) = n(L)$, $E(0) = E(L)$, $\phi(0) = \phi(L)$ at all times.

- Electrons that cross right boundary of solution domain must reappear at left boundary with same velocity, and vice versa.

Spatial Discretization

- Define set of J equally-spaced spatial grid points located at coordinates

$$x_j = j\,\delta x,$$

for $j = 0$ to $J - 1$, where $\delta x = L/J$.

- Let

$$n_j = n(x_j),$$

$$\phi_j = \phi(x_j),$$

et cetera.

Evaluation of Electron Number Density - I

- Electron number density at grid points calculated from electron coordinates via *particle-in-cell* (PIC) approach.
- Suppose ith electron lies between jth and $(j+1)$th grid points: i.e., $x_j < r_i < x_{j+1}$. Let

$$n_j \rightarrow n_j + \left(\frac{x_{j+1} - r_i}{x_{j+1} - x_j}\right) \Big/ \delta x,$$

$$n_{j+1} \rightarrow n_{j+1} + \left(\frac{r_i - x_j}{x_{j+1} - x_j}\right) \Big/ \delta x.$$

Thus, $n_j\,\delta x$ increases by 1 if electron at jth grid point, $n_{j+1}\,\delta x$ increases by 1 if electron at $(j+1)$th grid point, $n_j\,\delta x$ and $n_{j+1}\,\delta x$ both increase by $1/2$ if electron halfway between grid points, et cetera.

Evaluation of Electron Number Density - II

- Performing analogous assignment for each electron, in turn, allows the n_j to be determined from the r_i (assuming that all the n_j initialized to zero at start of process).
- Clear that

$$\int_0^L n(x)\,dx \simeq \sum_{j=0,J-1} n_j\,\delta x = N,$$

  as must be the case if the system contains N electrons.
- Relative error in calculation of the n_j from the r_i is $\mathcal{O}(1/J^2)$.

Fourier Solution of Poisson's Equation - I

- Poisson's equation (n.b., $n_0 = N/L$ in normalized units):

$$\frac{d^2\phi}{dx^2} = \rho(x),$$
$$\rho(x) = n(x)/n_0 - 1.$$

- Let

$$\phi_j = \sum_{j'=0,J-1} \bar{\phi}_{j'} \, e^{i\,j\,j'\,2\pi/J},$$
$$\rho_j = \sum_{j'=0,J-1} \bar{\rho}_{j'} \, e^{i\,j\,j'\,2\pi/J},$$

for $j = 0$ to $J-1$, which automatically satisfies periodic boundary conditions $\phi_J = \phi_0$ and $\rho_J = \rho_0$.

Fourier Solution of Poisson's Equation - II

- $\bar{\rho}_0 = \int_0^L \rho(x)\,dx/L = 0$, because $\int_0^L n(x)\,dx/L = n_0$.
- Remaining ρ_j obtained via standard Fourier inversion theorem:

$$\bar{\rho}_j = J^{-1} \sum_{j'=0,J-1} \rho_{j'}\, e^{-i\,j\,j'\,2\pi/J}.$$

- Fourier inversion of Poisson's equation yields: $\bar{\phi}_0 = 0$, and

$$\bar{\phi}_j = -\bar{\rho}_j / \left[j^2\,(2\pi/L)^2 \right]$$

  for $j = 1$ to $J/2$.

- Also,

$$\bar{\phi}_j = \bar{\phi}^*_{J-j}$$

  for $j = J/2 + 1$ to $J - 1$, which ensures that ϕ_j are real.

Calculation of E_j

- Electric field at grid points:

$$E_j = \frac{\phi_{j-1} - \phi_{j+1}}{2\,\delta x},$$

  for $j = 0$ to $J - 1$.

- N.B. $j = 0$ and $j = J - 1$ are special cases that are resolved using periodic boundary conditions.

- Relative error in above first-order discretization scheme is $\mathcal{O}(1/J^2)$.

Calculation of $E(r_i)$

- Suppose that r_i lies between jth and $(j+1)$th grid points: i.e., $x_j < r_i < x_{j+1}$.
- Linear interpolation:

$$E(r_i) = \left(\frac{x_{j+1} - r_i}{x_{j+1} - x_j} \right) E_j + \left(\frac{r_i - x_j}{x_{j+1} - x_j} \right) E_{j+1}.$$

- Relative error in interpolation scheme is $\mathcal{O}(1/J^2)$.

Fast Fourier Transform

- Discrete Fourier transform (DFT) is computationally expensive because of large number of trigonometric function evaluations.
- DFT typically require $\mathcal{O}(\mathrm{J}^2)$ arithmetic operations.
- *Fast Fourier transform* (FFT) algorithm (see Wikipedia page) rapidly computes DFT via factorization.
- FFT only requires $\mathcal{O}(\mathrm{J}\ \ln \mathrm{J})$ arithmetic operations.
- Since $\mathrm{J} = 1000$ in program, use of FFT algorithm implies reduction in arithmetic operations per time-step by factor of 100.
- Program implements FFT algorithm via calls to `fftw` library.[a]

[a] `http://www.fftw.org`

Euler Method

- Suppose that function $y(t)$ satisfies first-order ordinary differential equation (ODE)

$$\frac{dy}{dt} = f[y(t)].$$

- Equation can be solved numerically via repeated use of *Euler method* (see Wikipedia page):

$$y(t+h) = y(t) + h\, f[y(t)] + \mathcal{O}(h^2).$$

- Truncation error per time-step is $\mathcal{O}(h^2)$, necessitating relatively small time-steps to keep overall truncation error to acceptable levels.

- Generalization to system of N coupled first-order ODEs straightforward.

Fourth-Order Runge-Kutta Method

- *Fourth-order Runge-Kutta* (RK4) method (see Wikipedia page):

$$k_1 = h\, f[y(t)],$$

$$k_2 = h\, f[y(t) + k_1/2],$$

$$k_3 = h\, f[y(t) + k_2/2],$$

$$k_4 = h\, f[y(t) + k_3],$$

$$y(t + h) = y(t) + k_1/6 + k_2/3 + k_3/3 + k_4/6 + \mathcal{O}(h^5).$$

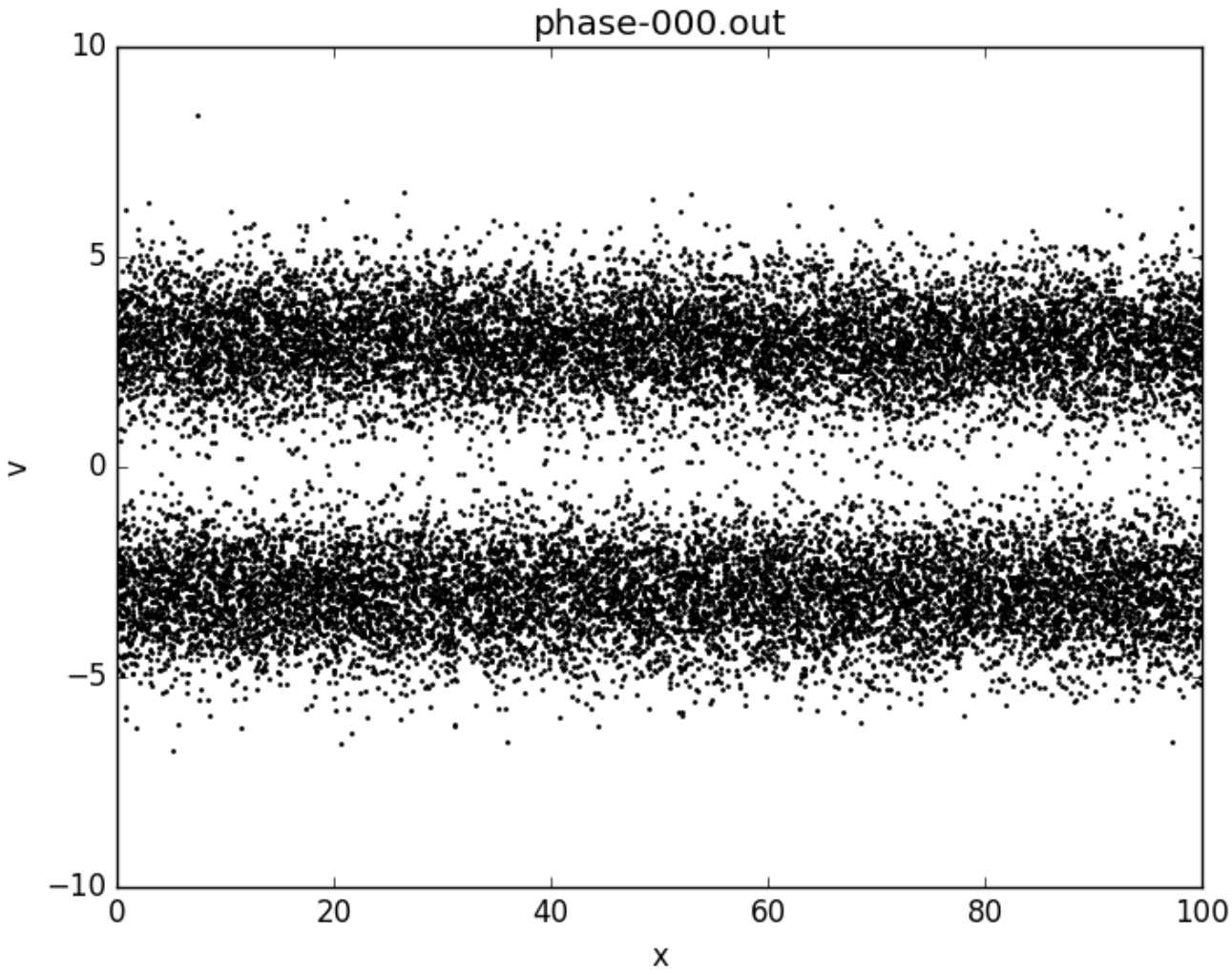
- Algorithm requires approximately 3 times number of arithmetic operations per time-step as Euler method. However, truncation error per time-step much smaller [$\mathcal{O}(h^5)$ rather than $\mathcal{O}(h^2)$].
- Can take much larger time-steps using RK4 method. More than makes up for factor of 3 increase in operations per time-step.

Program `Pic`

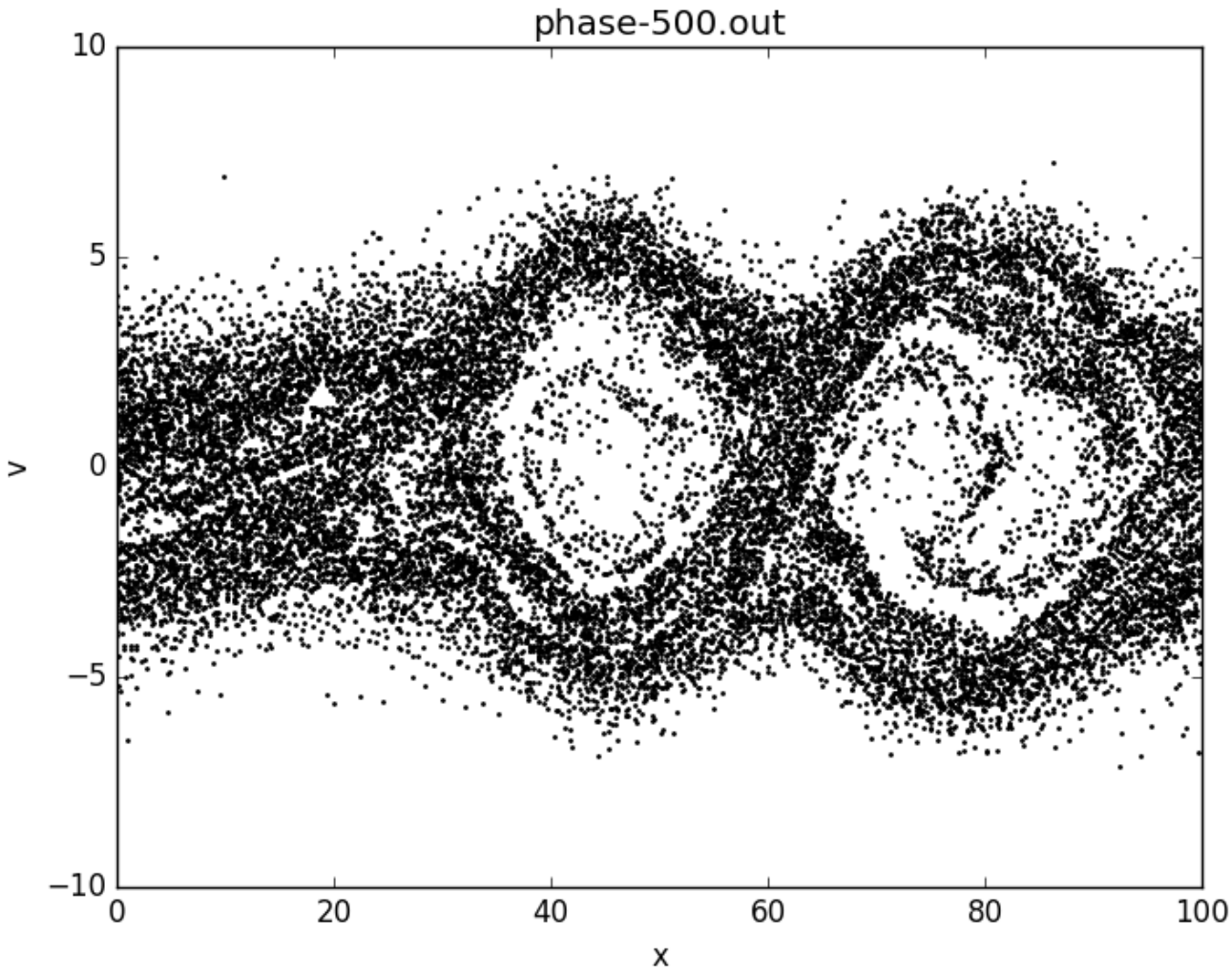
- Program `Pic`[a] is PIC code (which implements previously described calculation) written in C++ language.
- Program uses RK4 algorithm to integrate electron equations of motion. At each time-step, Poisson's equation solved via calls to `fftw` library.
- Default program parameters: $\mathrm{N} = 20,000$, $\mathrm{L} = 100$, $v_b = 3$, $\mathrm{J} = 1000$, $\mathrm{h} = 0.1$.

[a] `http://farside.ph.utexas.edu/teaching/plasma/plasma.html`

Initial State: $t = 0$



Final State: $t = 50$



Visualization

- Phase-space motion of electrons visualized as follows:
  - Program writes electron phase-space coordinates to file at each time-step (with separate file for each time-step).
  - `Python`[a] script reads coordinate files and employs `Matplotlib`[b] package to produce separate `png` plot of electron phase-space positions at each time-step.
  - Finally, `Python` script calls `ImageMagick`[c] image processing application to combine separate `png` plots into animated `gif`. (Animated `gif`s can be viewed by all modern web-browsers.)

[a] `http://www.python.org`
[b] `http://matplotlib.org`
[c] `http://www.imagemagick.org`

Energy Conservation

- Only force in problem (self-generated electric field) is *conservative*. So, total system energy should be *constant of motion*.
- (Normalized) total system energy:

$$\mathcal{E} = K + U,$$

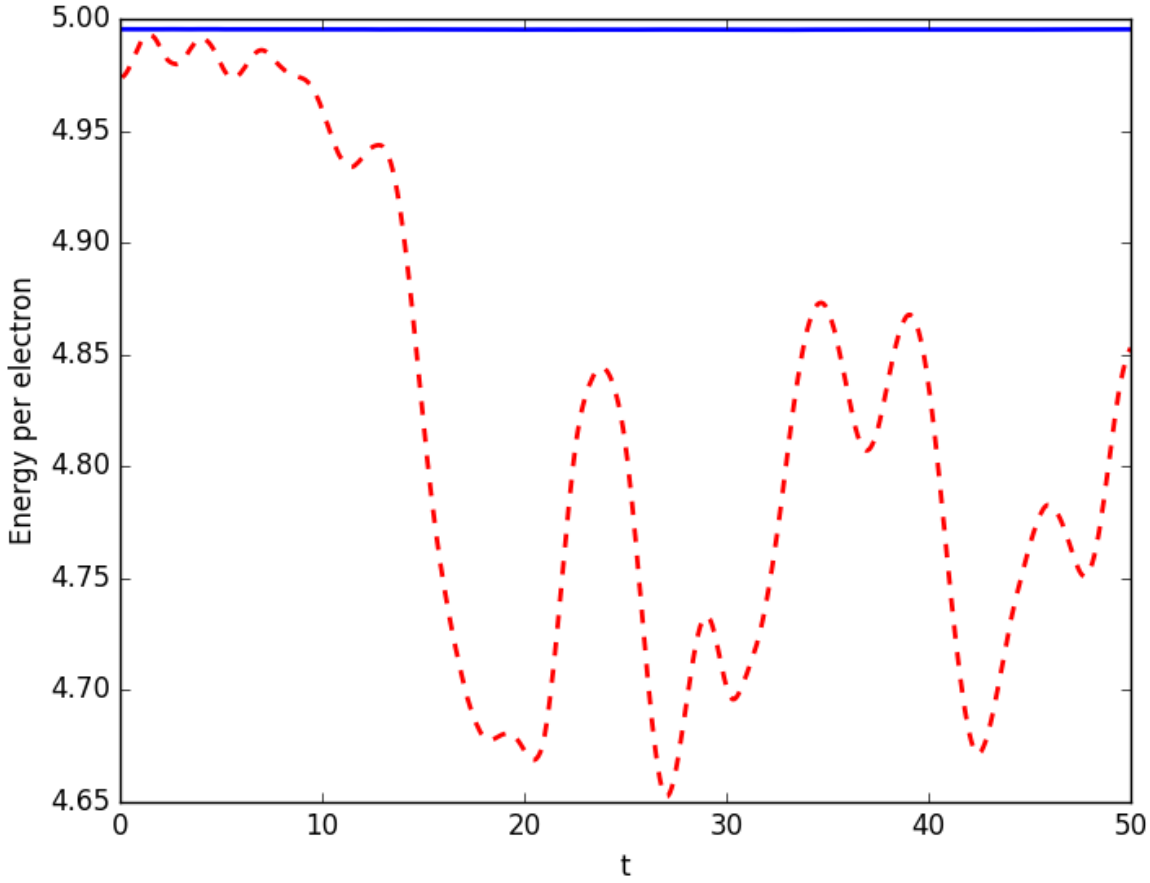
where

$$K = \sum_{i=1,N} v_i^2/2$$

is kinetic energy of electrons, and energy stored in electric field is

$$U = \frac{n_0}{2}\int_0^L E^2(x)\,dx = -\frac{1}{2}\int_0^L n(x)\,\phi(x)\,dx = -\sum_{i=1,N}\phi(r_i)/2.$$

Test of Energy Conservation

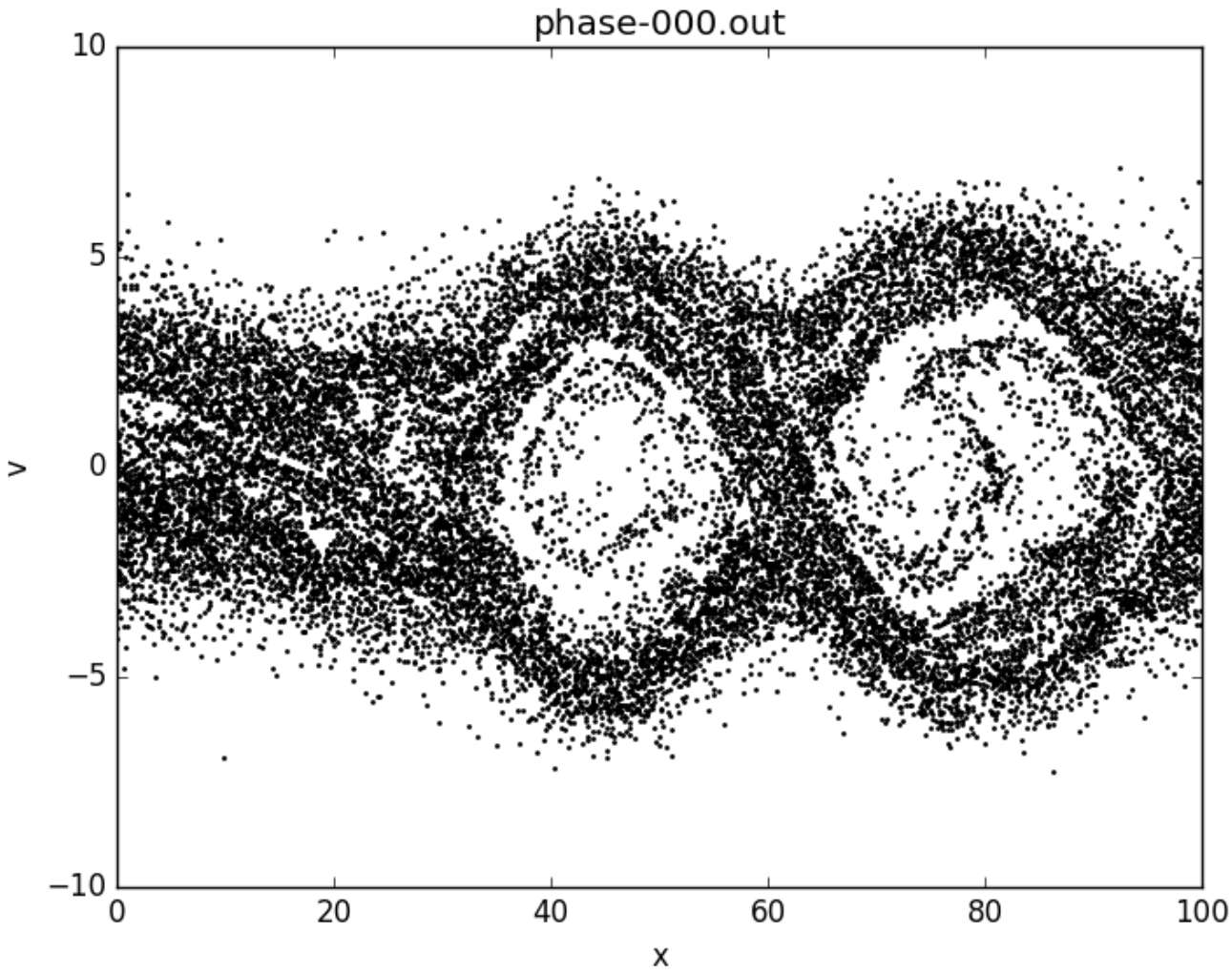


Blue curve is $\mathcal{E}$/N. Red curve is K/N.

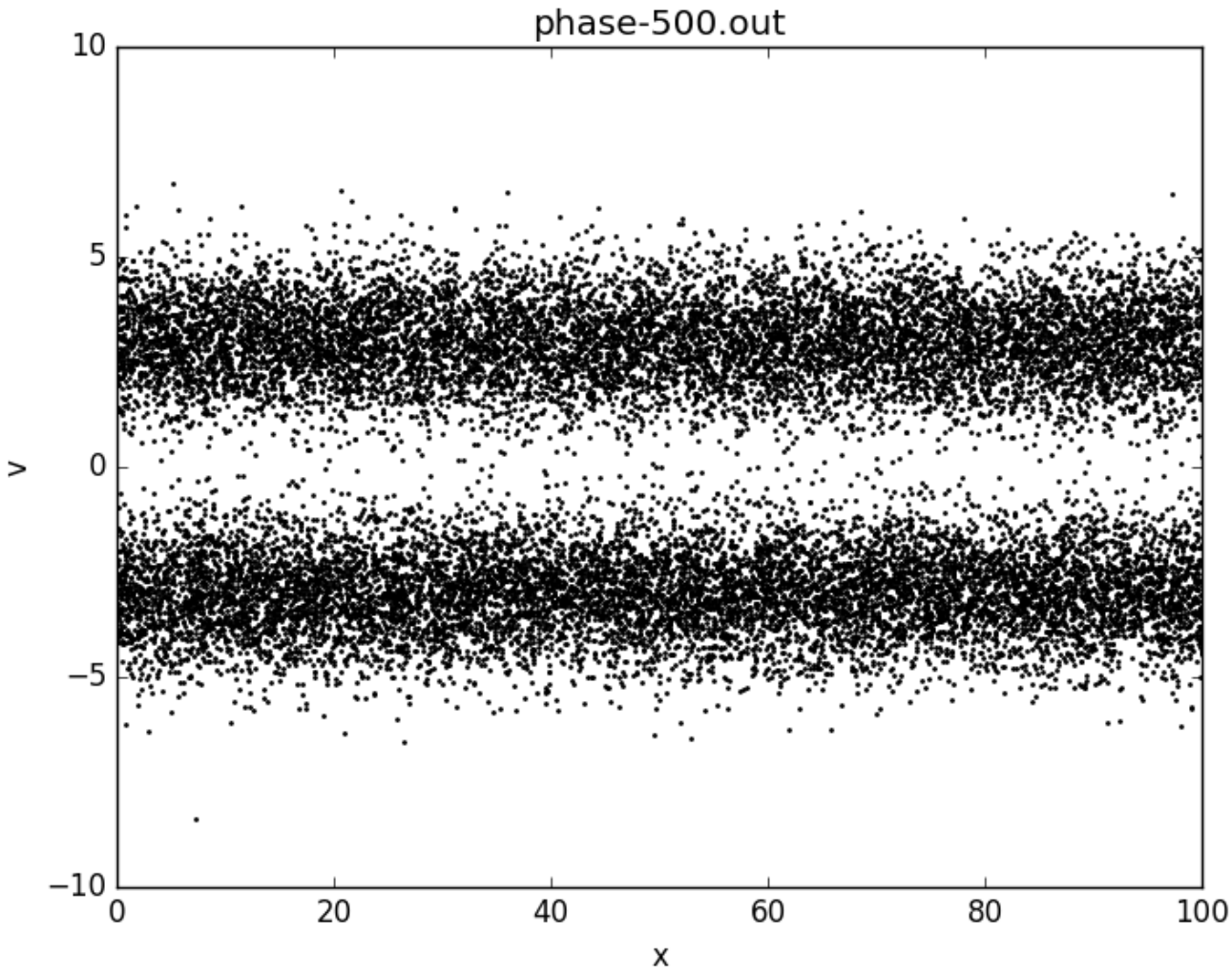
Entropy Conservation

- Motion of electrons in self-generated electric field should be *reversible* in time: i.e., if we take final state, reverse velocities of all electrons, and evolve system forward in time by same amount, then should recover initial state (with reversed velocities).
- Equivalent to saying total *entropy* of system is constant of motion.

Initial State: $t = 0$



Final State: $t = 50$



Using PIC Code to Simulate Real Plasma

- Real plasmas typically consist of $N_A \sim 10^{24}$ electrons. This is far greater number than could ever be accommodated in PIC code.
- Particle motion in PIC code only depends on *charge-to-mass ratio*. Could identify particles in PIC code as *super-particles* consisting of correlated clumps of real particles.
- Alternatively, could identify particles in PIC code as *markers* used to reconstruct *electron distribution function*, $f(x, v, t)$, at each time-step, in much same manner as electron number density, $n(x)$, reconstructed from electron coordinates in code. Hence, PIC code effectively evolving *Vlasov equation*:

$$\frac{\partial f}{\partial t} + v \frac{\partial f}{\partial x} - \frac{e E(x)}{m_e} \frac{\partial f}{\partial v} = 0.$$

Statistical Noise in PIC Codes

- PIC codes suffer from *statistical noise*. If phase-space binned into regions of 'volume' $\mathrm{d}x\,\mathrm{d}v$ then distribution function in given bin has irreducible statistical variation of relative magnitude $1/\sqrt{\mathrm{N_{bin}}}$, where $\mathrm{N_{bin}}$ is typical number of electrons in each bin.
- In 1D (1 space dimension, 1 velocity dimension), with 100×100 bins in phase-space, keeping statistical noise below 1% requires $100 \times 100 \times 100^2 = 10^8$ electrons.
- In 2D, (2 space dimensions, 2 velocity dimensions), with $100^2 \times 100^2$ bins in phase-space, keeping statistical noise below 1% requires $100^2 \times 100^2 \times 100^2 = 10^{12}$ electrons.
- In 3D, (3 space dimensions, 3 velocity dimensions), with $100^3 \times 100^3$ bins in phase-space, keeping statistical noise below 1% requires $100^3 \times 100^3 \times 100^2 = 10^{16}$ electrons!

Continuum Codes

- Alternative to PIC approach is to solve Vlasov equation,

$$\frac{\partial f}{\partial t} + v\,\frac{\partial f}{\partial x} - \frac{e\,E(x)}{m_e}\,\frac{\partial f}{\partial v} = 0,$$

  directly, using analogous methods to those typically used to solve fluid codes.

- Main advantage of so-called *continuum* codes is absence of statistical noise. Disadvantage is that numerical algorithms used in continuum codes more complicated that those used in PIC codes. Also, continuum codes harder to parallelize that PIC codes.

Scientific Computation Resources

- Scientific computation course: `http://farside.ph.utexas.edu`.
- Gnu Scientific Library (GSL).[a] Free (!) C library of numerical routines for scientific computation. Includes special functions, linear algebra, eigensystems, adapative integration of odes, etc.
- `Matplotlib`.[b] Free `python` 2D plotting library.
- `Maxima`.[c] Free computer algebra system.
- `Petsc`.[d] Free suite of data structures and routines for scalable (parallel) solution of partial differential equations.

[a] `http://www.gnu.org/software/gsl`
[b] `http://matplotlib.org`
[c] `http://maxima.sourceforge.net`
[d] `http://www.mcs.anl.gov/petsc`