Example: 1D PIC code for Two-Stream Plasma Instability

Setup

Periodic box $0 \le x \le L$

N Electrons with position (x) and velocity (v)

Initial electron distribution function:

2 counter-propagating Maxwellian beams of mean speed v_b

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

Method

Solve electron EOM as $2N$ coupled first order ODEs using RK4

$$\frac{dx_i}{dt} = v_i \qquad \frac{dv_i}{dt} = -E(x_i)$$

where

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

Method

Need electric field $E(x)$ at every time step. Solve Poisson's equation (using spectral method)

Calculate the electron number density $n(x)$ source term using PIC approach

DriverPIC.m

== 1D PIC code for the Two-Stream Plasma Instability Problem ==

Parameters

```
L = 100;     % domain of solution 0 <= x <= L
N = 20000;   % number of electrons
J = 1000;    % number of grid points
vb = 3;      % beam velocity
dt = 0.1;    % time-step (in inverse plasma frequencies)
tmax = 80;   % simulation run from t = 0 to t = tmax
```

DriverPIC.m

Initialize solution

```
t = 0;
rng(42);                              % seed the rand # generator
r = L*rand(N,1);                      % electron positions
v = double_maxwellian(N,vb);          % electron velocities
```

DriverPIC.m

Evolve solution

```
while (t<=tmax)
    % load r,v into a single vector
    solution_coeffs = [r; v];
    % take a 4th order Runge-Kutta timestep
    k1 = AssembleRHS(solution_coeffs,L,J);
    k2 = AssembleRHS(solution_coeffs + 0.5*dt*k1,L,J);
    k3 = AssembleRHS(solution_coeffs + 0.5*dt*k2,L,J);
    k4 = AssembleRHS(solution_coeffs + dt*k3,L,J);
    solution_coeffs = solution_coeffs + dt/6*(k1+2*k2+2*k3+k4);
    % unload solution coefficients
    r = solution_coeffs(1:N);
    v = solution_coeffs(N+1:2*N);
    % make sure all coordinates are in the range 0 to L
    r = r + L*(r<0) - L*(r>L);
    t = t + dt;
end
```

AssembleRHS.m

```
function RHS = AssembleRHS( solution_coeffs, L, J )
  r = solution_coeffs(1:N);    v = solution_coeffs(N+1:2*N);
  r = r + L*(r<0) - L*(r>L);
  % Calculate electron number density
  ne = GetDensity( r, L, J );
  % Solve Poisson's equation
  n0 = N/L;
  rho = ne/n0 - 1;
  phi = Poisson1D( rho, L );
  % Calculate electric field
  E = GetElectric( phi, L );
  % equations of motion
  dx = L/J;
  js = floor(r0/dx)+1;
  ys = r0/dx - (js-1);
  js_plus_1 = mod(js,J)+1;
  Efield = E(js).*(1-ys) + E(js_plus_1);
  rdot = v;
  vdot = -Efield;
  RHS = [rdot; vdot];
end
```

GetDensity.m

```
function n = GetDensity( r, L, J )
% Evaluate number density n in grid of J cells, length
L, from the electron positions r

  dx = L/J;

  js = floor(r/dx)+1;
  ys = r/dx - (js-1);
  js_plus_1 = mod(js,J)+1;
  n = accumarray(js,(1-ys)/dx,[J,1]) + ...
      accumarray(js_plus_1,ys/dx,[J,1]);

end
```

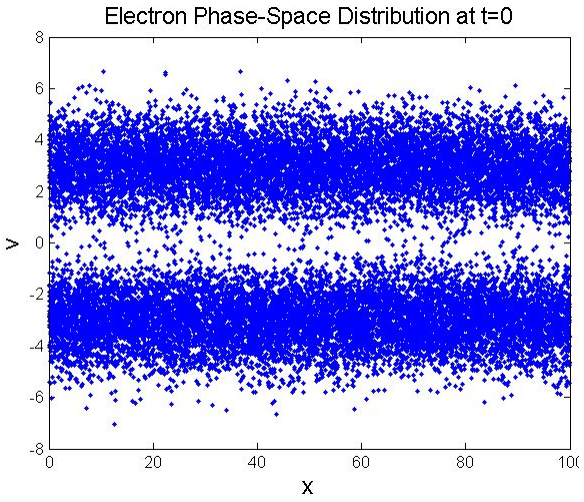
Poisson1D.m

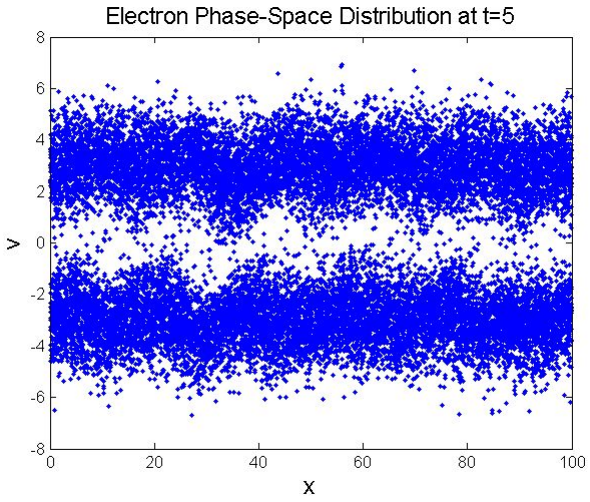
```
function u = Poisson1D( v, L )
  % Solve 1-d Poisson equation:
  %      d^u / dx^2 = v   for  0 <= x <= L
  % using spectral method
  J = length(v);
  % Fourier transform source term
  v_tilde = fft(v);
  % vector of wave numbers
  k = (2*pi/L)*[0:(J/2-1) (-J/2):(-1)]';
  k(1) = 1;
  % Calculate Fourier transform of u
  u_tilde = -v_tilde./k.^2;
  % Inverse Fourier transform to obtain u
  u = real(ifft(u_tilde));
  % Specify arbitrary constant by forcing corner u = 0;
  u = u - u(1);
end
```

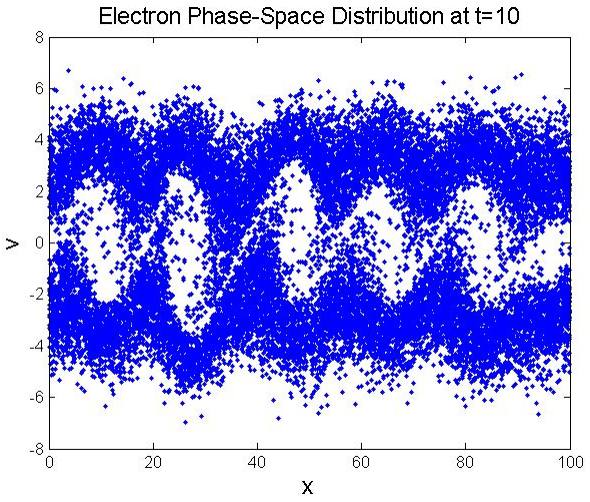
GetElectric.m

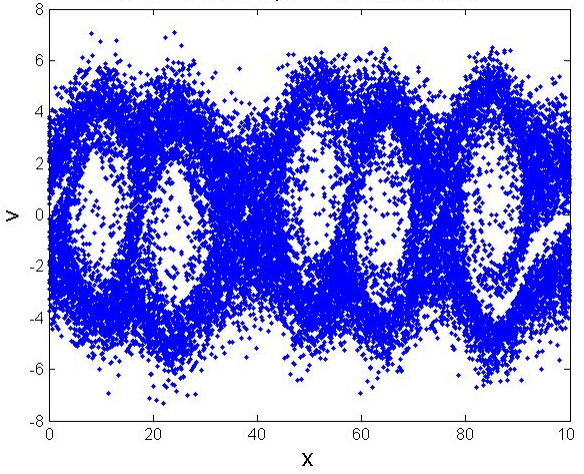
```
function E = GetElectric( phi, L )
  % Calculate electric field from potential
  J = length(phi);
  dx = L/J;
  % E(j) = (phi(j-1) - phi(j+1)) / (2*dx)
  E = (circshift(phi,1)-circshift(phi,-1))/(2*dx);
end
```

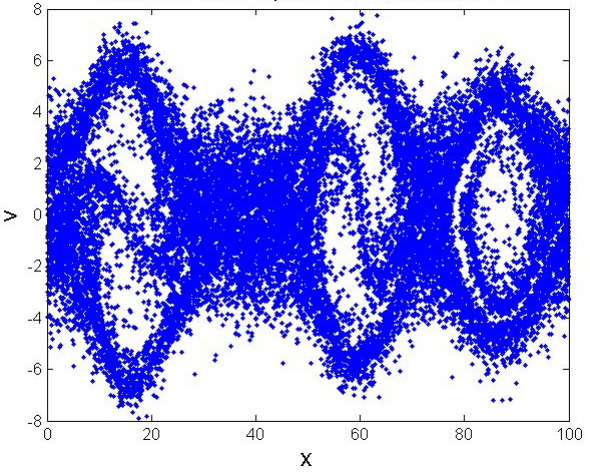
Results

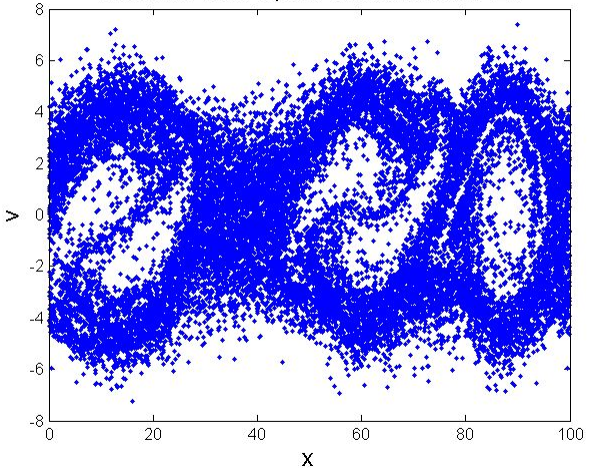

Electron Phase-Space Distribution at t=0
v
x


Electron Phase-Space Distribution at t=5
v
x


Electron Phase-Space Distribution at t=10
v
x


Electron Phase-Space Distribution at t=15
v
x


Electron Phase-Space Distribution at t=20
v
x


Electron Phase-Space Distribution at t=25
v
x