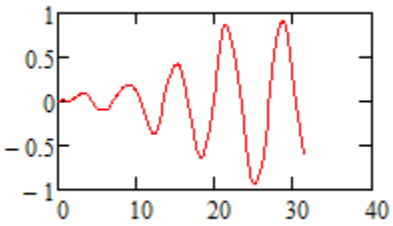
Particle-In-Cell Code for Beam Plasma Instability

This exercise uses the previously-developed PIC code to investigate the beam-plasma instability. The electron distribution function is composed of a Maxwellian background plasma and an electron beam with a narrow beam width, also described by a Maxwellian distribution. We will observe that a wave grows from noise spontaneously because of instability.



Wave amplitude
as a function of time

The equations to be solved

For an electron, the equations of motion are:

$$\frac{ds}{dt} = v \qquad \frac{dv}{dt} = \frac{-qE}{m}$$

The particle position is s, m is the electron mass, and the other variables have their usual meaning. The grid points will be labelled x.

The electric field is found from the particle density: $\frac{dE}{dx} = -\frac{n_e q}{\varepsilon_0}$

Dimensionless units

The characteristic time and distance scales are: $\hat{t} = \sqrt{\frac{\varepsilon_0 m}{n_0 q^2}}$ $\hat{x} = \sqrt{\frac{\varepsilon_0 T_e}{n_0 q^2}}$

We divide t by the characteristic time above and we divide s by the characteristic distance to obtain the dimensionless forms of our equations:

$$\frac{d\bar{s}}{d\bar{t}} = \bar{v} \qquad \frac{d\bar{v}}{d\bar{t}} = -\bar{E} \qquad \text{and} \qquad \frac{d\bar{E}}{d\bar{x}} = \frac{n_i q}{n_0 q} - \frac{n_e q}{n_0 q} = 1 - \bar{n}_e$$

Some additional characteristic scales are: $\hat{\phi} = \frac{T_e}{q}$ $\hat{v} = \frac{\hat{x}}{\hat{t}} = \sqrt{\frac{T_e}{m}}$ $\hat{E} = \frac{\hat{\phi}}{\hat{x}} = \sqrt{\frac{n_0 T_e}{\varepsilon_0}}$

Equations in finite-difference form

The finite-difference scheme is discussed in the first PIC code exercise.

The size of the simulation domain

We specify L, the width of the simulation domain in dimensionless units. $L := 16$

Theory often shows that the most unstable wave has the longest wavelength. For the PIC simulation, the longest wavelength is the one that just fits in the simulation domain.

Grid points and spatial resolution

In dimensionless units, we expect the spatial variation to be on a distance scale of order unity. Hence a grid spacing of 0.25 is sufficent to resolve the interesting physics. We will use x for the grid point locations and s (or S) for the electron locations. We will use the subscript i to indicate the values at successive times. Our program will output values of $M_{k,i}$ which is a matrix of values of the locations for the k^{th} particle at the i^{th} time step. The vector S of positions will be saved in successive columns of M at successive time steps.

$\Delta x := 0.25$ We specify the spatial resolution.

$jmax := floor\left(\frac{L}{\Delta x}\right)$ $jmax = 64$ This is the number of x grid points we need for the desired spatial resolution and desired L.

We will define a grid with jmax+1 points separated by distance Δx:

$j := 0 .. jmax$ $x_j := \Delta x \cdot j$ $x_{jmax} = 16$

Time step

A beam velocity of 1 in the dimensionless units (for example) and a grid spacing $\Delta x = 0.25$ results in a typical particle crossing a cell in a time of 0.25 in dimensionless time units. We will use a time step Δt that is half the time for the particle to cross a cell. The code will use a distribution in velocities, and the fastest particle could cross several cells in a time step.

The characteristic velocity; $V := 1$ The time step: $\Delta t := 0.5 \cdot \frac{\Delta x}{V}$ $\Delta t = 0.125$

Particle number per cell

Experience shows that 100 electrons per cell with about 4 cells per unit length provides sufficiently low noise that a wave of small amplitude (0.1 for example) can be seen above the noise.

$Ncell := 100$ The number of electrons in a cell.
Ncell determines the level of noise in the charge density calculation.

Initial electron positions

$kmax := jmax \cdot Ncell$ kmax is the number of particles to fill jmax cells with Ncell particles. The particles will be numbered 0 through kmax-1.

$kmax = 6400$

$k := 0 .. kmax - 1$ The index of the particle positions s_k.

The program loop at right distributes the particles randomly in each cell. A random distance between 0 and Δx is added to x_j, the location of the left boundary of the cell. A uniform spacing within a cell could be used, but this would be misleading about the level of random noise in the program.

$s :=$
$k \leftarrow 0$
for $j \in 0 .. jmax - 1$
  for $kk \in 0 .. Ncell - 1$
    $s_k \leftarrow x_j + \Delta x \cdot rnd(1)$
    $k \leftarrow k + 1$
s

The average density is: $n0 := \frac{kmax}{L}$ $n0 = 400$

Charge density based on electron positions

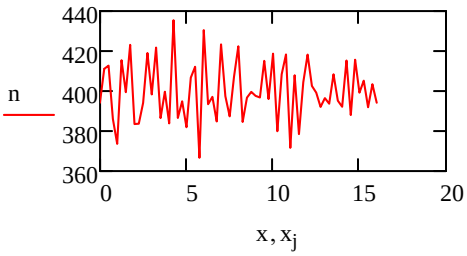
The program loop below determines electron density at grid points by counting electrons between grid points and using a weighting factor to assign a fraction of the density to the nearest two points. In the program loop, the vector N is the weighted number of particles assigned to the grid point and $n = N/\Delta x$ is the particle density per unit length evaluated at the grid point.

$inv\Delta x := \frac{1}{\Delta x}$ This variable is the inverse of Δx. The loop will execute more quickly if the divisions by Δx are replaced with multiplications by $inv\Delta x$.

$NumberDensity(s) :=$
$n_{jmax} \leftarrow 0$
$N_{jmax} \leftarrow 0$
for $k \in 0 .. rows(s) - 1$
  $jlower \leftarrow floor(s_k \cdot inv\Delta x)$
  $N_{jlower+1} \leftarrow (s_k - x_{jlower}) \cdot inv\Delta x + N_{jlower+1}$
  $N_{jlower} \leftarrow (x_{jlower+1} - s_k) \cdot inv\Delta x + N_{jlower}$
$N_0 \leftarrow N_0 + N_{jmax}$
$N_{jmax} \leftarrow N_0$
$n \leftarrow N \cdot inv\Delta x$

The next line calls the NumberDensity function above. $n := NumberDensity(s)$

Plot of the electron density



The plot shows the noise due to the random locations of the particles within cells. The noise is approximately the square root of Ncell.

$\sqrt{Ncell} = 10$ The noise level of n.

Initial electric field at grid points based on electron positions

This program loop finds the electric field by numerical integration of the particle density. The density in a cell is found by averaging the density at the grid points at each end of the cell. The constant of integration is assumed to be zero.

$$E := \left| \begin{array}{l} E_{jmax} \leftarrow 0 \\ \text{for } j \in 1 .. jmax \\ \quad E_j \leftarrow E_{j-1} + \left[1.0 - 0.5 \cdot \left(\frac{n_j + n_{j-1}}{n0} \right) \right] \cdot \Delta x \\ E \end{array} \right.$$

This program loop $\varphi(E)$ integrates the electric field to find the potential at the grid points.

$$\varphi(E) := \left| \begin{array}{l} \varphi_{jmax} \leftarrow 0 \\ \text{for } j \in 1 .. jmax \\ \quad \varphi_j \leftarrow \varphi_{j-1} - 0.5 \cdot \left(E_j + E_{j-1} \right) \cdot \Delta x \\ \varphi \end{array} \right.$$

This program loop combines the loops above. It defines a function Efield(n) that returns a vector containing the E that both satisfies Poisson's equation and the boundary conditions for φ. After a trial set of values for E is found, the trial value for φ at the right boundary is used to adjust E so that the new φ at the boundary is zero.

$$Efield(n) := \left| \begin{array}{l} E_0 \leftarrow 0 \\ \text{for } j \in 1 .. jmax \\ \quad E_j \leftarrow E_{j-1} + \left[1.0 - 0.5 \cdot \left(\frac{n_j + n_{j-1}}{n0} \right) \right] \cdot \Delta x \\ \varphi \leftarrow -0.5 \cdot \Delta x \sum_{j=1}^{jmax} \left(E_{j-1} + E_j \right) \\ E \leftarrow E + \frac{\varphi}{L} \end{array} \right.$$

Initial electric field at particle locations

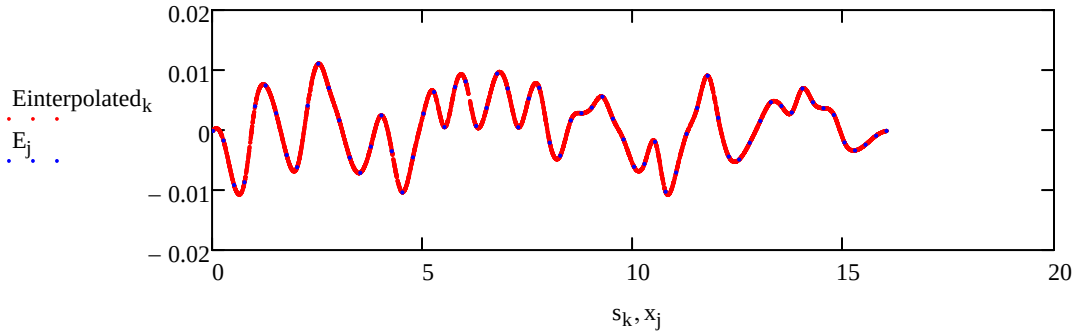
In order to find the electric field at the particle locations, we will need to interpolate using the values at the nearest grid points. We will use the function **interp** which fits a second order polynomial locally. The coefficients for the polynomial are obtained by calling the function **pspline**.

This function call finds the vector array of coefficients: $\text{coeffs} := \text{pspline}(x, E)$

This interpolation routine finds the electric field at the particle locations. $\text{Einterpolated}_k := \text{interp}(\text{coeffs}, x, E, s_k)$

The electric field arises from noise in the particle locations.

Plot of E(x) at grid points and at particle locations



Specifying a beam-plasma electron distribution function f(v)

We will need a program loop to assign electron velocities chosen from a distribution f(v) that is a Maxwellian background plasma with a thermal velocity of 1.0 plus a beam that is a drifting Maxwellian. The beam will have density (beam strength) BS and have velocity vbeam. The width of the distribution describing the beam will be BW (beam width).

$BW := 0.01$ Beam width

$BS := 0.05$ Beam strength Beam parameters

$vbeam := 3$ Beam velocity

This is the Maxwellian distribution function in our dimensionless system of units:

$$f(v) := \frac{1}{\sqrt{2\pi}} \cdot \left[(1 - BS) \cdot e^{-\left(\frac{v^2}{2}\right)} + \frac{BS}{\sqrt{BW}} \cdot e^{-\frac{(v - vbeam)^2}{2 \cdot BW}} \right]$$

$$\int_{-\infty}^{\infty} f(v)\, dv = 1$$ We verify that our distribution is normalized to unity.

$vmax := 4$ Define a range of v values to use for plots that extends from -vmax to +vmax.

$m := 0 .. 100$ $vplot_m := -vmax + \frac{2 \cdot vmax \cdot m}{100}$ The range is spanned by 100 points.

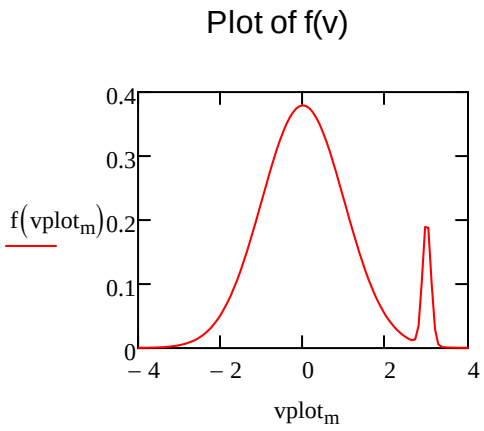
We will not select particles with v > vmax or v < -vmax because f(v) is small at these velocities.

$test := max(f(vplot))$

$test := 1.1 \cdot test$

$test = 0.417$

The variable test is set to a value a little larger than the maximum value of f(v) and is used below.



Program loop to select particles from a distribution function

This program loop chooses a velocity randomly on the interval -vmax to vmax and then chooses a random test value from 0 to test. If the value of f(v) is less than the test value, the v value is kept, otherwise it is rejected and a new v is created. Because the probability of v being kept is proportional to amplitude of f(v), the selected values of v have a probability proportional to f(v). This procedure is called the rejection method.

```
Vselect(b) := v ← 0
              y ← test
              while y > f(v)
                  v ← 2·vmax·(rnd(1) − 0.5)
                  y ← test·rnd(1)
              v
```

The random test value Vselect(b) is a function of a variable b that is not specified. Why? If Vselect is made a function of a variable, it is evaluated each time it is called. If Vselect is not made a function, then it is given a value only once in the program. Hence making it a function is a "work around" that causes the program loop to be executed each time Vselect(b) is called.

Test of the velocity selector Vselect(b)

We will call the function Vselect(b) many times and **histogram** the result.

$Ntest := 5000$ Number of Vtest values.

```
Vtest := hmax ← 5000
         G_hmax ← 0
         for h ∈ 0 .. hmax
             G_h ← Vselect(2)
         G
```

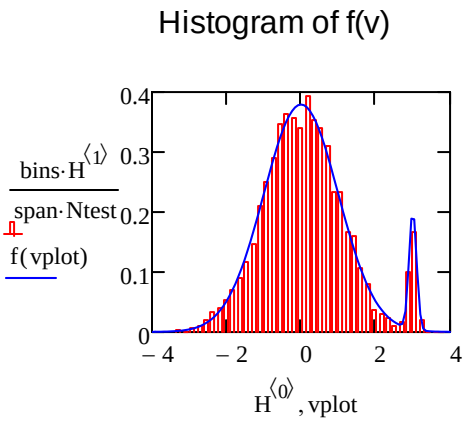
Below we create a histogram of velocities:

$bins := 40$

$span := max(Vtest) - min(Vtest)$

$H := histogram(bins, Vtest)$

Our program Vtest is working correctly because the scaled histogram of the velocities agrees with f(v). The scale factor for the histogram height is determined from the bin width (span/bins) and the total number of test particles Ntest.



Number of iterations

We will follow the evolution of the instability for 5 plasma periods.

$$Noscillations := 5$$

$$iters := floor\left(Noscillations \cdot \frac{2 \cdot \pi}{\Delta t}\right)$$

$$iters = 251$$

Shorten caclulation time

$\Delta t \Delta t := \Delta t^2$ We will use this variable in place of Δt^2 in order to reduce the number of multiplications. This variable appears in the innermost **for** loop which is executed the largest number of times.

$iters \cdot kmax = 1.606 \times 10^6$ Number of times the innermost loop is calculated.

We also reduce execution time by using the vector variable $E\Delta t\Delta t$ which is defined as $E*\Delta t*\Delta t$.

The program loop

Recall that: kmax = 6400 jmax = 64 iters = 251

We will use the **time** function to measure the time to do the program loop: StartTime := time(2)

```
M := | E_jmax ← 0
     | n_jmax ← 0
     | M_(kmax−1, iters) ← 0
     | ΔS_(kmax−1) ← 0
     | EΔtΔt_(kmax−1) ← 0
     | S ← s
     | for k ∈ 0 .. kmax − 1
     |     V ← Vselect(3)
     |     ΔS_k ← V·Δt
     | for i ∈ 0 .. iters
     |     n ← NumberDensity(S)
     |     E ← Efield(n)
     |     vs ← pspline(x, E)
     |     EΔtΔt ← ΔtΔt·interp(vs, x, E, S)
     |     ΔS ← ΔS − EΔtΔt
     |     S ← S + ΔS
     |     for k ∈ 0 .. kmax − 1
     |         S_k ← S_k + L if S_k < 0
     |         S_k ← S_k − L if S_k > L
     |     M^⟨i⟩ ← stack(S, ΔS/Δt)
     | M
```

These lines initialize the vectors E, n, S, ΔS and the matrix of answers M.

Recall that s is the initial vector of electron positions.

Mathcad "knows" that EΔtΔt is a vector with an element for each S value.

If the electron moves outside the domain 0 to L, it is moved into the domain (recycled) using the notion of periodic conditions. For example, if the particle moves to L+0.01, the new location is 0.01.

EndTime := time(2)

The time to run the loop is: EndTime − StartTime = 4.369 seconds

The answer matrix M contains the postions S stacked on top of the velocites ΔS/Δt.

We will recover the velocities from the bottom half of the matrix using the submatrix command:

$$Velocities := submatrix(M, kmax, kmax + kmax - 1, 0, iters) \qquad rows(Velocities) = 6400$$

We will remove the velocities from the bottom half of the matrix M so that M has only particle positions:

$$M := submatrix(M, 0, kmax - 1, 0, iters) \qquad rows(M) = 6400$$

This line defines the time T for each time step that is useful for plots.

$$ii := 0 .. iters \qquad T_{ii} := ii \cdot \Delta t$$

Program loop for potential as a function of time

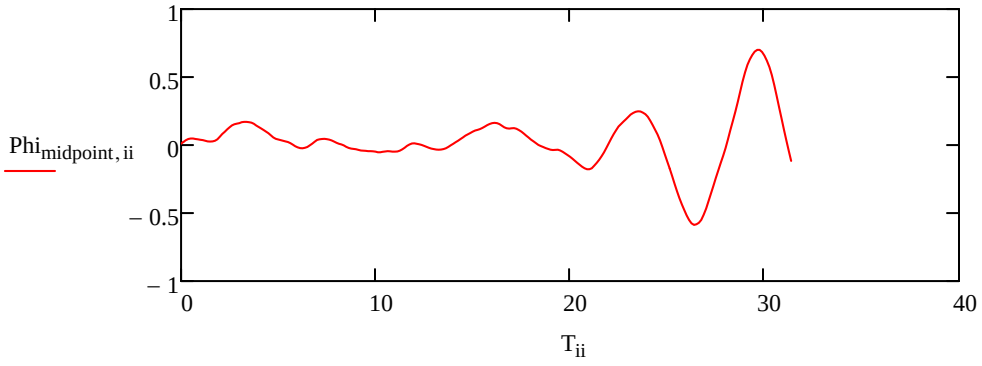
This program loop takes the matrix M and calculates the potential at the grid points at each time step. The columns of the new matrix Phi contain the values of φ at grid points.

The plot below will be φ(t) evaluated at the midplane. Recall that the potential is forced to be zero at the two boundaries, hence the midplane is the likely location of the greatest change in potential.

```
Phi :=  Phi[jmax, cols(M)-1] ← 0
        for i ∈ 0 .. cols(M) - 1
            s ← M^⟨i⟩
            n ← NumberDensity(s)
            E ← Efield(n)
            Phi^⟨i⟩ ← φ(E)
        Phi
```

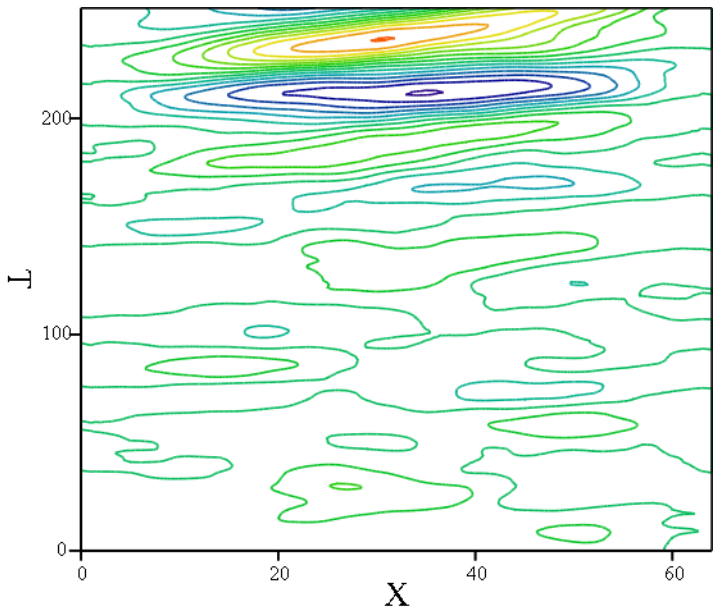
$$midpoint := round(jmax \div 2)$$ j value for the midpoint

Plot of φ(t) at the midpoint



Observe that the wave grows from noise.

Contour plot of the potential φ(x,t)



Phi

The vertical axis is the index i of the time step. The horizontal axis is the index j for the distance x.

Energy conservation as a check on accuracy of our PIC code

We will examine the "goodness" of our code by looking at energy conservation. We will find the kinetic energy of the particles and the energy in the wave and see if their sum is constant with time.

Kinetic Energy

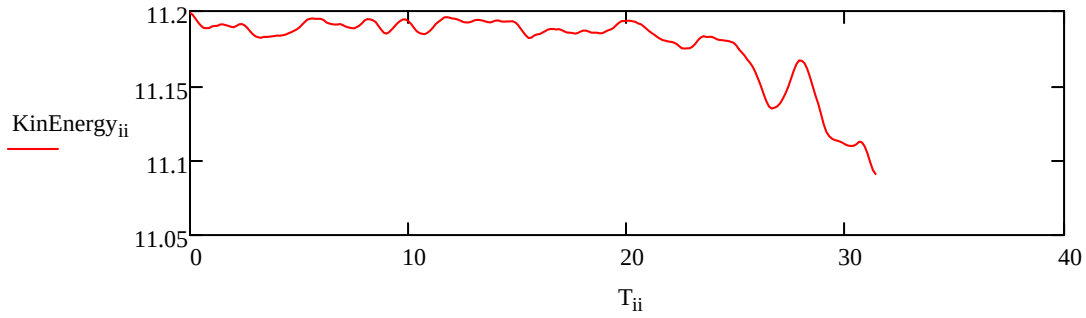
Create a vector that is the sum over particles of 0.5 v^2 at each time step.

$Nparticles := rows(Velocities)$ $Nparticles = 6400$ Number of particles.

$$KinEnergy_{ii} := \frac{0.5 \cdot L}{Nparticles} \cdot \sum_{k=0}^{Nparticles-1} \left(Velocities_{k,ii}\right)^2$$

The sum of the squares of the velocities divided by the number of PIC particles gives the mean squared velocity. The dimensionless density is unity hence the "true" number of particles is L. The rms velocity is multiplied by L to obtain the total kinetic energy of particles.

Plot of kinetic energy of particles as a function of time



The kinetic energy of the particles decreases in time because the beam is slowed by the wave.

Wave energy

This program loop Ematrix takes the matrix of particle positions M and calculates the electric field at the grid points at each time step.

$$\text{Ematrix} := \left| \begin{array}{l} \text{Ematrix}_{\text{jmax},\,\text{cols}(M)-1} \leftarrow 0 \\ \text{for } i \in 0 \,..\, \text{cols}(M) - 1 \\ \quad \left| \begin{array}{l} s \leftarrow M^{\langle i \rangle} \\ n \leftarrow \text{NumberDensity}(s) \\ E \leftarrow \text{Efield}(n) \\ \text{Ematrix}^{\langle i \rangle} \leftarrow E \end{array} \right. \\ \text{Ematrix} \end{array} \right.$$

$\text{rows}(\text{Ematrix}) = 65 \qquad \text{jmax} = 64$

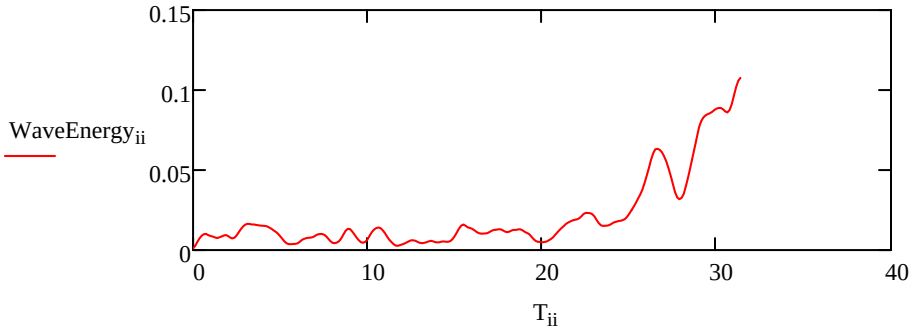
Note that the number of rows of Ematrix is the number of grid points, jmax+1.

We will integrate 0.5 E^2 with distance to find the electric field energy:

$$\text{WaveEnergy}_{ii} := 0.5 \cdot \Delta x \cdot \sum_{k=1}^{\text{jmax}} \left(\frac{\text{Ematrix}_{k,\,ii} + \text{Ematrix}_{k-1,\,ii}}{2} \right)^2$$

E is evaluated at the center of the cell by averaging the values at the adjacent grid points.

Plot of the electric field energy in the wave

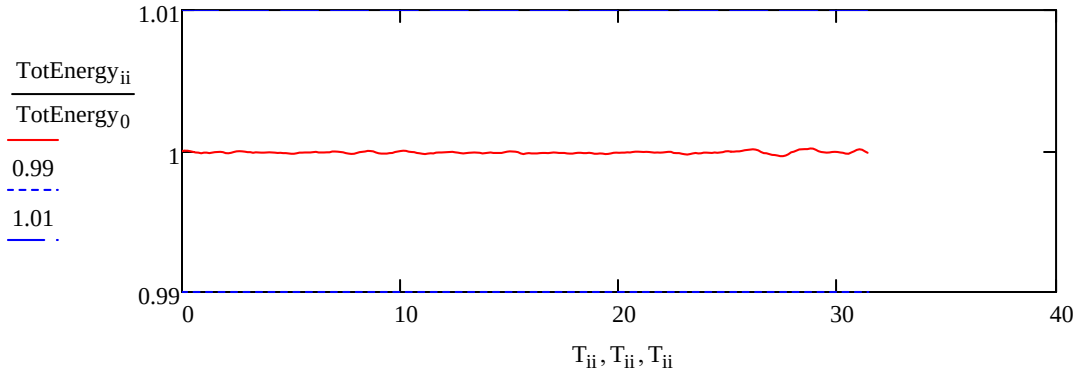


Total Energy is nearly conserved

$TotEnergy := WaveEnergy + KinEnergy$ This sum is constant if energy is conserved.

$\frac{max(TotEnergy)}{TotEnergy_0} = 1.0002$ $\frac{min(TotEnergy)}{TotEnergy_0} = 0.9996$ The maximum, minimum, and starting energies are compared.

Normalized total energy as a function of time



The total energy has been divided by the initial energy to obtain a normalized value near unity. The graph will "autoscale" to make the variation look large, hence we have added points at 99% and 101% of the starting value so that the variation is shown on an appropriate scale.

Note that the error in the total energy is less than 1%, indicating that our PIC code conserves energy (almost).

Modification of the distribution function by wave-particle interaction

To see the effect of the wave on f(v), we can plot f(v) before and after the wave grows.
The last column of the matrix Velocities contains the velocites at the last time step.

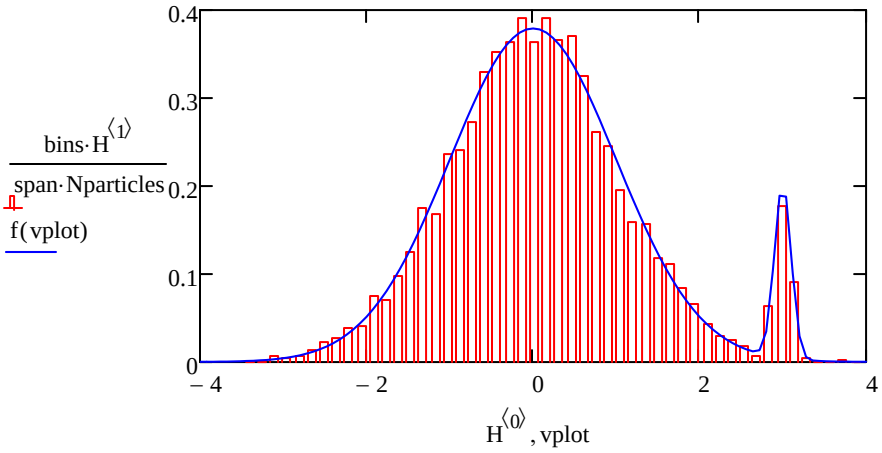
$cols(Velocities) - 1 = 251$ This is the same as the number of time steps. $iters = 251$

These commands prepare a histogram of the velocites at the zeroth time step:

$span := max\left(Velocities^{\langle 0 \rangle}\right) - min\left(Velocities^{\langle 0 \rangle}\right)$ $span = 7.421$ $bins := 50$

$H := histogram\left(bins, Velocities^{\langle 0 \rangle}\right)$

Histogram of f(v) initially

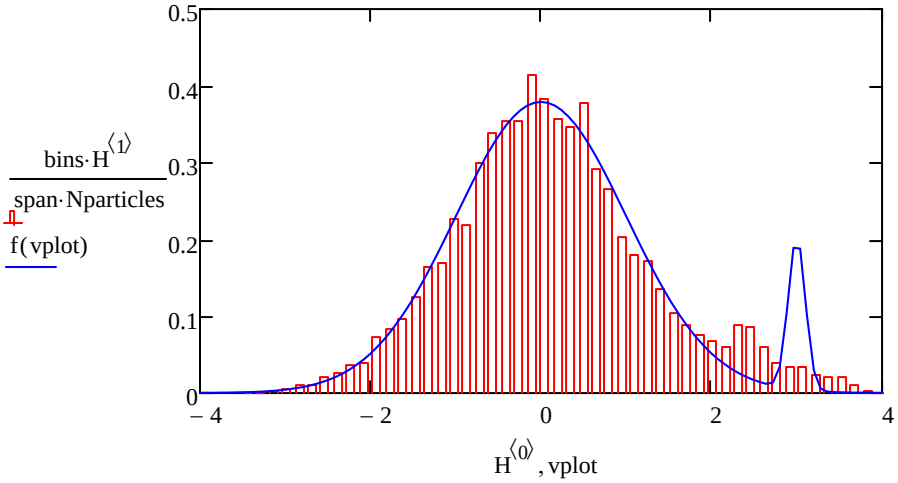


These commands make a histogram of the velocities at the last time step:

$H := \text{histogram}\left(\text{bins}, \text{Velocities}^{\langle \text{iters} \rangle}\right)$

$\text{span} := \max\left(\text{Velocities}^{\langle \text{iters} \rangle}\right) - \min\left(\text{Velocities}^{\langle \text{iters} \rangle}\right)$ $\quad \text{span} = 7.572$

Histogram of f(v) after growth of the unstable wave



Comparison of the histograms shows that the beam has become more like a plateau on the dsitribution function as a result of its interaction with the electrostatic wave.

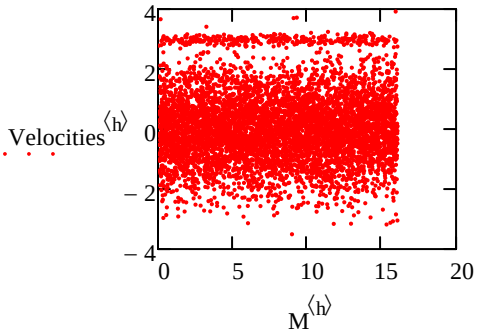
Phase space plots

The matrix M has the positions and the matrix Velocities has the velocities hence we easily can make a phase space plot that has the particle positions on the horizontal axis and the particle velocities on the vertical axis. The column number h will choose the time step.

$h := 0$ Column with data for the initial time

Phase space plot at t = 0

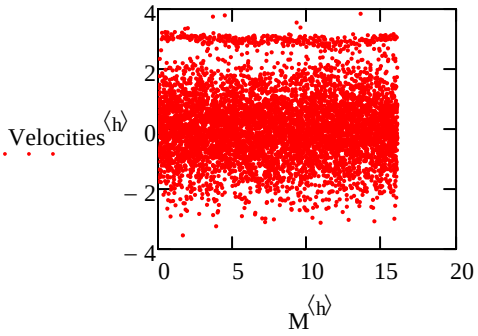
At the initial time, the two streams are clearly seen.



$h := \text{round}\left(\frac{\text{iters}}{2}\right)$ Half time

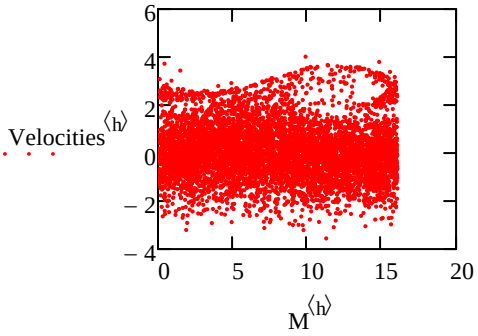
Phase space plot at "halftime"

Notice that a hole has formed in phase space.

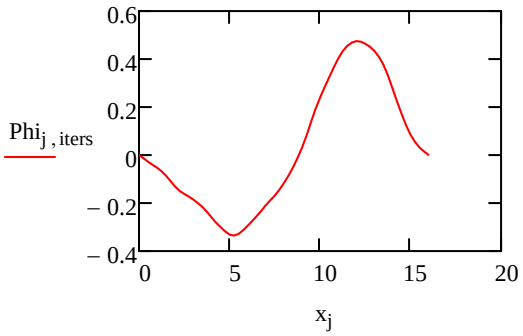


$h := \text{iters}$ Final time

Phase space plot at the final time

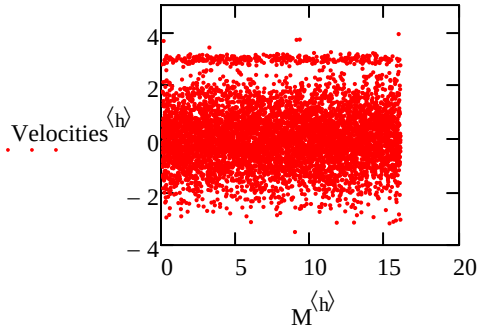


Compare the phase space plot at the final time above to the potential at the final time, plotted at right. The high electron velocities in the phase space plot are at the same location as the maximum in the potential, as would be expected from energy conservation. The electrons speed up as they fall into the potential well.



Phase space movie

h := FRAME iters = 251



If the plots have too many points, there will be insufficient computer memory.

To see the movie, click on **tools** then **animation** then **record** and set the frames from 0 to the value of iters. Highlight the region with the plot, and click on **Animate**. This procedure works for Mathcad 14.

Try it: Change the size of the simulation domain from L = 16 to some other value and observe the interaction of the beam with the wave.

Try it: Change the beam strength from 0.05 to 0.01 and observe the way in which the growth rate and amplitude of the wave are changed. It is helpful to cut the plot of E(t) and paste it as an image in order to make a comparison.

Try it: The instability grows from noise. Does it grow the same way if the code is run a second time? Should it?

Reference :

"Plasma Physics via Computer Simulation" by C.K. Birdsall and A.B Langdon, Institute of Physics, Bristol and Philadelphia, 1991.