Part II

Simulation Techniques

Chapter 2

KEMPO1

Technical Guide to One-dimensional Electromagnetic Particle Code

Yoshiharu Omura and Hiroshi Matsumoto

2.1 Introduction

We have developed a one-dimensional electromagnetic particle code which is a simplified version of the original two-dimensional KEMPO: Kyoto university's ElectroMagnetic Particle cOde [Matsumoto and Omura, 1985]. The simplified one-dimensional version is designed for a tutorial purpose so that the beginners in the field of space plasma physics can study various types of micro-instabilities and nonlinear processes involved in the instabilities. Hereafter, we will call the simplified one-dimensional code as KEMPO1.

The KEMPO1 is designed to assist physicists and engineers to develop their own particle codes by studying the techniques implemented in it. The KEMPO1 consists of a set of subroutines, each of which is a relatively short program providing a definite function. There is a main program calling the subroutines to formulate the algorithm of the simulation code. By configuring calls for the subroutines properly, one can construct different codes other than the electromagnetic code such as a test particle code or an electrostatic code. One can also modify the subroutines for diagnostics or develop one's own diagnostic subroutines.

For efficient diagnostics, a graphic subroutine package is provided. This package consists of subroutines calling a set of basic graphic subroutines such as PLOT for drawing lines and SYMBOL for drawing characters. These are the so-called CALCOMP graphic subroutines, which are available in many computer systems and can be implemented by writing simple interface subroutines for any graphic systems.

We describe function and usage of each of major subroutines of the KEMPO1 in each of the following sections. Among them, Section 2.2 for the MAIN program gives an overall description of the structure of the KEMPO1. Section 2.3 for the subroutine INPUT serves as User's Guide of the KEMPO1 giving explanation of the input parameters to be specified for a run of the KEMPO1. Sections 2.4 – 2.13 provide explanations of the essential subroutines forming the KEMPO1. Section 2.14 describes the diagnostic subroutines in the KEMPO1 with examples of the graphic outputs. Section 2.15 gives a series of exercises with the KEMPO1.

In the following subsections, we describe the basic equations and the associated technical subjects of the KEMPO1.

2.1.1 Basic equations

We solve Maxwell's equations for the electric field $\boldsymbol{E} \equiv (E_x, E_y, E_z)$ and magnetic field $\boldsymbol{B} \equiv (B_x, E_y, E_z)$

$$\nabla \times \boldsymbol{B} = \mu_o \boldsymbol{J} + \frac{1}{c^2} \frac{\partial \boldsymbol{E}}{\partial t} \tag{2.1}$$

$$\nabla \times \boldsymbol{E} = -\frac{\partial \boldsymbol{B}}{\partial t} \tag{2.2}$$

where $\boldsymbol{J} \equiv (J_x, J_y, J_z)$, c and μ_o are the current density, light speed and magnetic permeability, respectively. We assume a one-dimensional system taken along the x-axis. The electric field E_x should satisfy the initial condition given by Poisson's equation

$$\frac{\partial E_x}{\partial x} = \frac{\rho}{\epsilon_o} \tag{2.3}$$

where ρ and ϵ_o are charge density and electric permittivity, respectively. It is noted that Poisson's equation is solved only for the initial condition. It is satisfied automatically, if (2.1) is solved correctly in time based on the current density $\boldsymbol{J}$ satisfying the continuity equation of the charge density ρ. The magnetic field B_x should satisfy the initial condition given by

$$\frac{\partial B_x}{\partial x} = 0 \tag{2.4}$$

This condition means that B_x is constant in space and time, because there is no term for B_x in Maxwell's equations (2.1) and (2.2).

The current density $\boldsymbol{J}$ and charge density ρ are computed from the motion of a large number of particles. The equations of motion for particles with a charge q and a mass m is

$$\frac{\mathrm{d}\boldsymbol{v}}{\mathrm{d}t} = \frac{q}{m}(\boldsymbol{E} + \boldsymbol{v} \times \boldsymbol{B}) \tag{2.5}$$

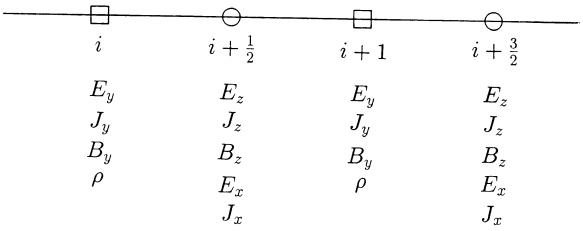

Figure 2.1: Grid assignment.

$$\frac{\mathrm{d}x}{\mathrm{d}t} = v_x \tag{2.6}$$

The basic equations described above are written in the form of the MKS unit system. In a simulation, however, values of the permittivity ϵ_o and permeability μ_o can be defined arbitrarily, as far as they satisfy the relation

$$\epsilon_o \mu_o = \frac{1}{c^2} \tag{2.7}$$

In the KEMPO1, for simplicity, we adopt the following definition

$$\epsilon_o = 1, \qquad \mu_o = \frac{1}{c^2}. \tag{2.8}$$

In the following sections, we write equations in terms of the above definition.

The $\boldsymbol{E}$, $\boldsymbol{B}$ and $\boldsymbol{J}$ are defined at the spatial grid points, while particles can take arbitrary positions. The $\boldsymbol{E}$ and $\boldsymbol{B}$ in the equations of motion (2.5) are interpolated from those at the adjacent grid points.

2.1.2 Grid assignment

We define full-integer grids at $i\Delta x$ $(i = 1, 2, 3, ..., N_x)$ and half-integer grids at $(i+1/2)\Delta x$. The E_y, B_y, J_y and ρ are defined at the full-integer grids, and E_x, E_z, B_z, J_x, J_z at the half-inter grids as shown in Figure 2.1. This assignment of the electric and magnetic fields $\boldsymbol{E}$ and $\boldsymbol{B}$ realizes centered difference forms for the spatial derivatives in Maxwell's equations. The components J_x, J_y, J_z of the current density must be assigned to the same grids of E_x, E_y, E_z, respectively, because $\boldsymbol{J}$ contributes directly to the time integration of $\boldsymbol{E}$ as in (2.1).

2.1.3 Time step chart

The quantities of the field and particles are advanced in time based on the sequence shown in Figure 2.2. We define a full-integer time $n\Delta t$ and a half-

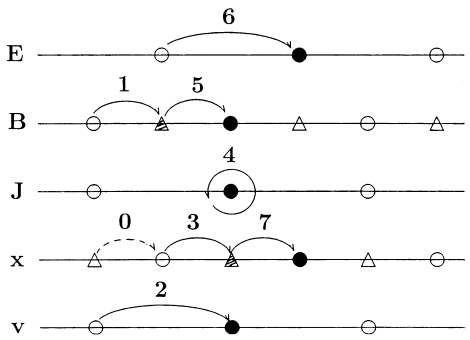

Figure 2.2: Time step chart.

integer time $(n + 1/2)\Delta t$ with a time step Δt. Basically, the electric field $\boldsymbol{E}$ at the full-integer time and the magnetic field $\boldsymbol{B}$ at the half-integer time are integrated in time by the leap-frog method. However, the magnetic field $\boldsymbol{B}$ is advanced twice by a half time step $\Delta t/2$ to obtain intermediate values for the particle pushing fields at the full-integer time. The particle positions $\boldsymbol{x}$ at the full-integer time and velocities $\boldsymbol{v}$ at the half-integer time are also advanced by the leap-frog method. The positions are advanced twice with a half time step $\Delta t/2$ to obtain intermediate values for computation of the current density $\boldsymbol{J}$ at the half-integer time. The current density $\boldsymbol{J}$ is computed from the positions and velocities of particles based on the method described in Section 2.12 for the subroutine CURRNT.

2.1.4 Courant condition

In solving Maxwell's equations by the centered difference scheme in space and by the leap-frog method in time, the grid spacing Δx and the time step Δt should satisfy the following inequality, which is called the Courant condition,

$$\Delta x > c\Delta t \tag{2.9}$$

where c is the light speed.

The condition is easily derived from the numerical dispersion relation of the light mode. Let us see the numerical effect in solving the a differential equation by a centered difference equation. We assume a quantity $A(x,t)$ has a wave structure with a wavenumber k and a frequency ω as

$$A(x,t) = A_o \exp(ikx - i\omega t) \tag{2.10}$$

We compute the derivative by a centered-difference equation as

$$\begin{aligned}\frac{\Delta A}{\Delta x} &= \frac{A(x_o+\Delta x/2,t)-A(x_o-\Delta x/2,t)}{\Delta x} \\ &= \frac{\exp(ik\Delta x/2)-\exp(-ik\Delta x/2)}{\Delta x}A(x_o,t) \\ &= i\frac{\sin(k\Delta x/2)}{\Delta x/2}A(x_o,t) \end{aligned} \tag{2.11}$$

Comparing the $\Delta A/\Delta x$ withe the spatial derivative $\partial A/\partial x$, we find that the wavenumber k is replaced by K represented by

$$K = \frac{\sin(\mathrm{k}\Delta\mathrm{x}/2)}{\Delta\mathrm{x}/2}, \tag{2.12}$$

in converting the differential equations to the difference equations. In the same manner, we find that the frequency ω is replaced by Ω defined as

$$\Omega = \frac{\sin(\omega\Delta\mathrm{t}/2)}{\Delta\mathrm{t}/2} \tag{2.13}$$

The dispersion relation of the light mode is obtained by neglecting the current density $\boldsymbol{J}$ and assuming the electromagnetic wave with a frequency ω and the wavenumber k as

$$\omega^2 = c^2k^2 \tag{2.14}$$

Replacing k and ω with K and Ω, we have the numerical dispersion relation for the light wave

$$\Omega^2 = c^2K^2 \tag{2.15}$$

For the maximum wavenumber $k_{max} = \pi/\Delta x$, we have

$$\sin^2(\omega\Delta\mathrm{t}/2) = (\frac{\mathrm{c}\Delta\mathrm{t}}{\Delta\mathrm{x}})^2 \tag{2.16}$$

If $c\Delta t/\Delta x > 1$, then the ω becomes complex, giving rise to a numerical instability. If $c\Delta t/\Delta x = 1$, then the system is marginally stable. Therefore, we have the Courant condition (2.9).

2.1.5 Debye length

To avoid a nonphysical instability caused by the grids [Birdsall and Langdon, 1985], we should choose the grid spacing Δx not much larger than the Debye length λ_e given by

$$\lambda_e = \frac{v_{th,e}}{\omega_{pe}} \tag{2.17}$$

where $v_{th,e}$ and ω_{pe} are the thermal velocity and the plasma frequency of electrons in a plasma, respectively.

An empirical condition for the linear weighting method used in the KEMPO1 is the following, although it is not a clear and critical condition such as the Courant condition.

$$\Delta x \leq 3\lambda_e \tag{2.18}$$

2.2 MAIN

This program controls the sequence of computations performed in the KEMPO1. In the following, we show the source code of the MAIN program, and make a brief description for each line of the program. The several subroutine calls prior to the DO-loop (100) statement is needed for setting up the initial conditions. The DO-loop provides a set of computations to advance the wave fields and particle motions by one time step. Diagnostics are made before the DO-loop for the initial condition and at each time step in the DO-loop.

The main program consists of subroutine calls which can be arranged flexibly so that the code is used as an electrostatic code. In the following we first describe the main program for the full-electromagnetic particle code, and then we describe a modification for an electrostatic code.

2.2.1 Electromagnetic simulation

The source code of the main program is the following.

```
      common /timecm/ itime,ntime,iecrct,iwrite,jobnum
      common /diagcm/ iediag,ifdiag,ikdiag,
     #                ipdiag,isdiag,ivdiag,
     #                ieplot,ifplot,ikplot,
     #                ipplot,isplot,ivplot
c
      call plots
      call factor(0.9)
      call input
      call chkprm
      call pltprm
      call renorm
      if(jobnum.le.1) then
        itime = 0
        call inital
        call positn
        call charge
```

```
        call ecrrct
      else
        call reader
      endif
      call fldplt
      call phsplt
      call vdsplt
      ist = itime
c
      do 100 j = ist+1, ist+ntime
        itime = j
        call bfield
        call velcty
        call positn
c       call currnt
        call curntv
        call positn
        call bfield
        call efield
        if( mod(j,iecrct).eq.0) then
            call charge
            call ecrrct
        endif
        if( mod(j,ifdiag).eq.0 ) then
            if( mod(ifdiag,iecrct).ne.0 ) call charge
            call fldplt
        endif
        if( mod(j,ikdiag).eq.0 ) call kspplt
        if( mod(j,ipdiag).eq.0 ) call phsplt
        if( mod(j,ivdiag).eq.0 ) call vdsplt
        if( mod(j,isdiag).eq.0 ) call spectr
        if( mod(j,iediag).eq.0 ) call energy
        if( mod(j,iwrite).eq.0 ) call writer
 100  continue
      call writer
      call plot(0.,0.,999)
      stop
      end
```

Among these subroutine calls, some of them are essential for the computation, and others are for graphic diagnostics which can be omitted. We explain the essential subroutines first and then diagnostics subroutines.

The initialization of the simulation run is made by the subroutines INPUT, RENORM, INITAL, POSITN, CHARGE and ECRRCT. In the main DO-loop, particle motion is solved by VELCTY and POSITN. Maxwell's equations are solved by CURRNT (vectorized version: CURNTV), BFIELD and EFIELD. Poisson's equation is solved by CHARGE and ECRRCT. As an electromagnetic code, Poisson's equation does not have to be solved at each time step, because the continuity equation for the charge density and currnt density is strictly satisfied in computing the current density in CURRNT (CURNTV). To avoid accumulation of round-off errors in the electrostatic field, we compute the charge density and correct the electrostatic field using Poisson's equation at an interval specified by the parameter *IECRCT*.

Diagnostics of the fields and particles are made by subroutines FLDPLT, KSPPLT, PHSPLT, VDSPLT, SPECTR and ENERGY. Among these subroutines, FLDPLT, KSPPLT, PHSPLT and VDSPLT plot snap-shots of the field, wavenumber spectra, and particle diagnostics at intervals specified by the parameters *IFDIAG*, *IKDIAG*, *IPDIAG*, respectively. The subroutines SPECTR and ENERGY perform diagnostics of the wave spectra and energy densities at intervals specified by *ISDIAG* and *IEDIAG*, respectively. These subroutines store the data to the arrays defined in the subroutines. After the data are accumulated for a certain number of time steps, the subroutines plot time histories of the wave modes and energy densities. In addition to the history plots of wave modes, SPECTR also plots the $\omega - k$ diagrams showing the wave dispersion relations.

The input parameters are checked for their appropriateness by the subroutine CHKPRM. If the input parameters violate the Courant condition 2.9, the program is terminated by executing a *STOP* FORTRAN statement. The program is also terminated if the specified numbers of grid points and number of particles exceed the declared array sizes, which are specified by the parameters in the file "paramt.h" included in each subroutine.

```
parameter(ix=1026, is=3, in = 32768)
```

where *IX* should satisfy the condition $NX + 2 \leq IX$, and *IS* and *IN* are the maximum numbers of particle species and superparticles, respectively. After the checking, the input parameters are plotted as a graphic output by the subroutine PLTPRM.

To perform a very long run, we need to split the run into several jobs. To connect jobs, we need to save all the on-memory data such as parameters and variables to a data file. To start a subsequent job, we need to load the saved data to the on-line memory. These functions are provided by the subroutines WRITER and READER. For an unexpected trouble of the computer system, the WRITER is called at a certain interval specified by the parameter IWRITE. Therefore, even if a job is terminated abnormally before completing the time steps specified by *NTIME*, one can resume the job from the latest time step

when the WRITER was called. A parameter *JOBNUM* is defined as a job number in the sequence of subsequent jobs. For the initial job, we assign the $JOBNUM = 0$ or 1. If $JOBNUM = 0$, the data file for a subsequent job is not created.

The call for the subroutine PLOTS at the beginning of the main program is for initialization of the graphics. The call for FACTOR(0.9) is just an adjustment of the graphic outputs. With this argument 0.9, the size of output figures is reduced by 90 %. One can modify the size by changing the argument. The call for PLOT(0.,0.,999) at the end of the MAIN program is for termination of the graphics.

Most of the CPU time is spent in the subroutines VELCTY and CURRNT. On a vector processor, the subroutine VELCTY is vectorized by the vector compiler without difficulty. However, the subroutine CURRNT cannot be vectorized without incorporating a special algorithm. For the vector processor, we have prepaired the subroutine CURNTV, in which a large two-dimensional arrays are used as a working memory area for vectorization. The size of the working area is

$$IX \times LVEC \times 3\text{words} \tag{2.19}$$

where IX is the size of arrays for the current density defined at grid points. The parameter $LVEC$ is a number of repetition in the vectorized DO-loop of the current density computation.

2.2.2 Electrostatic simulation

The KEMPO1 is converted to an electrostatic particle simulation code by replacing the DO-loop (100) by the following source code. The advantage of the electrostatic simulation is that the time step Δt is not restricted by the Courant condition.

```
      do 100 j = ist+1, ist+ntime
        itime = j
        call velcty
        call positn
        call positn
        call charge
        call ecrrct
        if( mod(j,ikdiag).eq.0 ) call kspplt
        if( mod(j,ipdiag).eq.0 ) call phsplt
        if( mod(j,ivdiag).eq.0 ) call vdsplt
        if( mod(j,isdiag).eq.0 ) call spectr
        if( mod(j,iediag).eq.0 ) call energy
        if( mod(j,iwrite).eq.0 ) call writer
100   continue
```

2.3 INPUT

The subroutine INPUT specifies the input parameters for the simulation run. One can change the input parameters by editing the source file of the subroutine INPUT.

In the following we describe each of the parameters specified in the subroutine INPUT.

- *DX* : Grid spacing.
- *DT* : Time step.
- *CV* : Light speed.
- *WC* : Cyclotron frequency of Species 1, Ω_1. From the cyclotron frequency, we compute the magnitude of the the static magnetic field B_o, using

$$B_o = \frac{\Omega_1}{(q/m)_1}, \tag{2.20}$$

  where $(q/m)_1$ is the charge-to-mass ratio of Species 1.
- *ANGLE* : Angle between the static magnetic field $\boldsymbol{B}_o$ and the wavenumber vector $\boldsymbol{k}$. The static magnetic field $\boldsymbol{B}_o$ is taken in the x-y plane.
- *VMIN, VMAX* : The minimum and maximum values of the velocity scales used in the diagnostics made by the subroutines PHSPLT and VDSPLT. If *VMIN* = *VMAX*, then the range is automatically computed from the particle velocities.
- *NX* : Number of grid points. The *NX* must be a power of 2, because the FFT is used to solve Poisson's equation in the subroutine ECRRCT.
- *NTIME* : Number of time steps in a simulation run. It is recommended to set the number which is a power of 2, because we use the FFT to obtain frequency spectra as diagnostics.
- *IEDIAG* : Number of time steps in an interval at which the energy diagnostics is made.
- *ISDIAG* : Number of time steps in an interval at which the wavenumber spectra is computed for the further analyses of the time history plots of wavenumber modes and the wave frequency spectra($\omega - k$ diagram).
- *IFDIAG* : Number of time steps in an interval at which the spatial profiles of the electric field (E_x, E_y, E_z) and the magnetic field (B_y, B_z) are plotted.
- *IKDIAG* : Number of time steps in an interval at which the wavenumber spectra are computed and plotted.
- *IPDIAG* : Number of time steps in an interval at which the phase diagrams of particles are plotted.

- *IVDIAG* : Number of time steps in an interval at which the velocity distribution function is plotted. The distribution functions for three velocity components of v_x, v_y and v_z are plotted.
- *IEPLOT* : Control parameter for time history plots of the energy diagnostics made by the subroutine ENERGY. After the *IEPLOT* times of samplings are made in time, the subroutine ENERGY plots a set of history plots of energy densities such as thermal energy, drift energy, electric field energy, magnetic field energy and total energy. The maximum value of *IEPLOT* is the parameter *IW* specified in the subroutine ENERGY.
- *ISPLOT* : Control parameter for ω - k diagrams plotted by the subroutine SPECTR. The subroutine SPECTR computes the wavenumber spectra at an interval specified by *ISDIAG*. After *ISPLOT* times of such computation, the accumulated wavenumber spectra are Fourier transformed in time. The *ISPLOT* should be a power of 2. The maximum value of *ISPLOT* is the parameter *JMAX* specified in the subroutine SPECTR.
- *IKPLOT* : Control parameter for the wavenumber diagnostics made by the subroutine KSPPLT. The *IKPLOT* is the maximum mode number to be plotted in snap-shot plots of the wavenumber spectra.
- *IFPLOT* : Control parameter for the diagnostics made in the subroutine FLDPLT. Two different diagnostics for the wave fields are available. One is a combination of 6 panels showing the spatial variations of five field components E_x, E_y, E_z, B_y, B_z and the charge density ρ (Case 1). The other is a combination of 3 panels showing the spatial variations of E_x, ρ and the electrostatic potential ϕ (Case 2). This diagnostics is especially useful for an analysis of electrostatic waves.

  If *IFPLOT* = 1 or 3, the diagnostics of Case 1 is made.
  If *IFPLOT* = 2 or 3, the diagnostics of Case 2 is made.

- *IPPLOT* : Control parameter for the diagnostics made in the subroutine PHSPLT. Phase diagrams are plotted for different combinations of the components such as $x - v_x$, $v_z - v_{\perp xy}$ and $v_\parallel - v_{\perp xy}$. The $v_{\perp xy}$ is one of the two velocity components forming the velocity $\boldsymbol{v}_\perp$ perpendicular to the static magnetic field $\boldsymbol{B}_o$ assumed in the $x - y$ plane. The other perpendicular component is v_z.

  One can specify any combination of these diagrams by giving a number (1 - 7) to *IPPLOT*, which is the sum of the three numbers $n1$, $n2$ and $n3$, corresponding to switches for the phase spaces $x - v_x$, $v_z - v_{\perp xy}$ and $v_\parallel - v_\perp$, respectively. The numbers $n1$, $n2$ and $n3$ take the values 1, 2 and 4, if the phase diagrams are plotted. Otherwise, they take 0's.

  If *IPPLOT* = 1, 3, 5 or 7, the $x - v_x$ diagram is plotted.
  If *IPPLOT* = 2, 3, 6 or 7, the $v_z - v_{\perp xy}$ diagram is plotted.
  If *IPPLOT* = 4, 5, 6 or 7, the $x_\parallel - v_\perp$ diagram is plotted.

- *IECRCT* : Control parameter to solve Poisson's equation for correction of the longitudinal electric field E_x. At every *IECRCT* time step, the charge density is computed, and Poisson's equation is solved.
- *IWRITE* : Control parameter to save all the parameters and variables to a data file "kempo1.cont", which is used to restart the run for further time steps. In case of an abnormal end of the run because of an unexpected system trouble, one can restart the run from the last time step when the on-memory data was saved to the file. For a relatively long run, it is recommended to save the data at a certain interval specified by *IWRITE*.
- *JOBNUM* : The job number for a continuous run staring from the final time step of the previous run. If $JOBNUM \geq 2$, all the on-memory data are loaded from the file "kempo1.cont". If *JOBNUM = 1*, the simulation run is initialized with the input parameters, and all the on-memory data are saved in the data file "kempo1.cont" at the end of the run. If *JOBNUM = 0*, the run is initialized with the input parameters, but the data file "kempo1.cont" is not created.

Input parameters for particles are specified for each different species of particles. These parameters are stored in the arrays defined in the *COMMON* area:

```
      common /ptprmc/ wp(is), qm(is), q(is), vpe(is), vpa(is),
     #                vd(is), pch(is), np(is)
```

where *IS* is the maximum number of species declared in the parameter file "paramt.h" to be included in the source file on compilation.

- *NS* : Number of particle species ($NS \leq IS$).
- *QM(i)* : Charge-to-mass ratio of Species i: q_i/m_i. For electrons, *QM(i)* must be negative.
- *WP(i)* : Plasma frequency of Species i, defined by

$$\omega_{pi} = \sqrt{\frac{n_i q_i^2}{m_i}} \tag{2.21}$$

  where n_i, q_i and m_i are number density, charge and mass of Species i, respectively. The *WP(i)* and *QM(i)* determine the charge density ρ_i of Species i, which is given by

$$\rho_i = \frac{\omega_{pi}^2}{q_i/m_i} \tag{2.22}$$

  In a system where both electrons and ions are assumed as mobile particles, the following charge neutrality condition must be satisfied.

$$\sum_i \rho_i = \sum_i \frac{\omega_{pi}^2}{q_i/m_i} = 0 \tag{2.23}$$

On the other hand, in a system where only electrons are computed as superparticles, the charge density of the background immobile ions are automatically computed to establish the charge neutrality.

- *VPE(i)* : Thermal velocity perpendicular to the static magnetic field
- *VPA(i)* : Thermal velocity parallel to the static magnetic field
- *VD(i)* : Drift velocity. In combination with the pitch angle *PCH(i)*, the drift velocities of parallel and perpendicular components are specified as follows;

$$V_{d\perp} = V_d \sin\phi \tag{2.24}$$

$$V_{d\parallel} = V_d \cos\phi \tag{2.25}$$

where V_d and ϕ are drift velocity and pitch angle, respectively. The parallel drift velocity is an average velocity of parallel velocities. The perpendicular drift velocity is defined as a radius of a ring distribution function in the velocity phase space formed by two components of the perpendicular velocity. First, the particles are distributed uniformly in phase of $\boldsymbol{v}_\perp$ with a constant $|v_\perp|$. Then, they are scattered to realize a thermal spread of the perpendicular thermal velocity *VPE*.

- *PCH(i)* : pitch angle to define parallel and perpendicular drift velocity $V_{d\parallel}$ and $V_{d\perp}$.
- *NP(i)* : Number of superparticles for Species i in the simulation system. The number of superparticles has nothing to do with the physical number density n_i. It determines thermal fluctuation levels of the electrostatic and electromagnetic fields. In a plasma at an equilibrium, it is well known that the energy density W of the electrostatic thermal fluctuation is proportional to the temperature of the plasma T_e, that is, [e.g., Nicholson, 1983; Ueda et al., 1993],

$$W = \frac{T_e}{2} \int \frac{dk}{2\pi} \frac{1}{1 + k^2\lambda_e^2} \tag{2.26}$$

where T_e is an electron temperature given by $m_e v_{th}^2$. The temperature is a statistical quantity defined for a single particle. Since a single superparticle in a simulation system represents a number of real particles, the mass of a superparticle is inversely proportional to the number of superparticles in a system. Therefore, the energy density of the electrostatic fluctuations are inversely proportional to *NP(i)*.

To make the thermal fluctuation level low, we need to assign a sufficient number of superparticles to the major species which contributes mostly to the kinetic energy in the system.

2.4 RENORM

This subroutine renormalizes the input parameters so that they are converted to those normalized to the unit system used in the simulation. This subroutine must be called before calling the INITAL subroutine which initializes physical quantities computed in the simulation.

In order to attain computational efficiency, it is necessary to reduce the number of operations involved in the difference equations of the fields and particles. Since the operations of multiplication and division are frequently performed with the grid spacing Δx and the half time step $\Delta t/2$, we renormalize distance and time by Δx and $\Delta t/2$, respectively. Let us define two different unit systems. One is a unit system for real physical quantities used as the input parameters. The other is a renormalized unit system used in the simulation code. Let us call the former "R-unit system", and the latter "S-unit system". We distinguish quantities in the two different unit systems by adding a subscript R or S. The grid spacing $(\Delta x)_R$ and time step $(\Delta t)_R$ for the R-unit system can take arbitrary values. However, in the S-unit system, they are fixed as $(\Delta x)_S = 1$ and $(\Delta t)_S = 2$.

All input parameters other than the grid spacing DX and time step DT are renormalized as follows, before they are used for initialization by the subroutine INITAL.

distance	$x_S = (1/\Delta x)x_R$
time	$t_S = (2/\Delta t)t_R$
velocity	$v_S = (\Delta t/2)(1/\Delta x)v_R$
electric field	$E_S = (\Delta t/2)^2(1/\Delta x)E_R$
magnetic field	$B_S = (\Delta t/2)B_R$
charge density	$\rho_S = (\Delta t/2)^2\rho_R$
current density	$J_S = (\Delta t/2)^3(1/\Delta x)J_R$
energy density	$\sigma_S = (\Delta t/2)^4(1/\Delta x)^2\sigma_R$
number density	$n_S = \Delta x n_R$
charge	$q_S = (\Delta t/2)^2(1/\Delta x)q_R$
mass	$m_S = (\Delta t/2)^2(1/\Delta x)m_R$

The renormalization coefficients are defined in the subroutine, and they are kept in the $COMMON$ area named $/RESCLC/$. It is noted that the coefficients for the electric filed, charge and mass are identical.

2.5 INITAL

This subroutine sets up the initial condition of the simulation. One can modify this subroutine to set up an arbitrary initial condition. The quantities to be initialized are itemized in the following.

1. Constants defined in the $COMMON$ $/CONSTC/$, $/ROTATC/$ and $/ECRCTC/$

2. Initial values of particles: x, v_x, v_y and v_z defined in the *COMMON /PRTCLC/*
3. Initial values of fields: E_x, E_y, E_z, B_y and B_z defined in the *COMMON /FIELDC/*

These quantities are initialized by simple methods as implemented in the following source program. One may use a more sophisticated method of particle initialization to suppress fluctuations of the initial field (see, for example, Birdsall and Langdon [1985]).

The initial values of the particle positions, velocities and the magnetic fields are defined at $t = -\Delta t/2$, while those of the electric fields are defined at $t = 0$. One should take care of the time difference of the initial magnetic fields and electric fields in setting up a plasma wave as an initial condition.

```
      common /inputc/ dx, dt, cv, wc, angle
      common /ptprmc/ wp(is), qm(is), q(is), vpe(is), vpa(is),
     #                vd(is), pch(is), np(is)
      common /constc/ tcs, bx0, rho0, slx, nx, nxp1, nxp2, npt, ns
      common /rotatc/ sinth, costh
      common /ecrctc/ rkfact(ix)
      common /prtclc/ x(in), vx(in), vy(in),vz(in)
      common /fieldc/ ex(ix), ey(ix), ez(ix), by(ix), bz(ix),
     #                ajx(ix), ajy(ix), ajz(ix), rho(ix)
c
      dimension xs(is),xl(is)
c
      twopi = 6.283185308
      theta = twopi/360.0*angle
      sinth = sin(theta)
      costh = cos(theta)
      bx0  = wc/qm(1)*costh
      by0  = wc/qm(1)*sinth
      tcs = 2.0*cv*cv
      slx = nx
      nxp1 = nx + 1
      nxp2 = nx + 2
c
      npt=0
      rho0 = 0.0
      xs(1)=0.0
      xl(1)=slx
      xs(2)=0.0
      xl(2)=slx
      xs(3)=0.0
```

```
      xl(3)=slx*0.25
      do 10 k = 1,ns
        npt=npt+np(k)
        q(k) = (slx/float(np(k))) * (wp(k)**2) / qm(k)
        rho0 = rho0 + q(k)*np(k)
   10 continue
      rho0 = -rho0/slx
c
      rkmin = twopi/slx
      nxh = nx/2
      fft = 1.0/float(nxh)
      do 300 i=1,nxh-1
        rk  = sin(rkmin*i*0.5)*2.0
        rkfact(2*i+1) = (1.0/rk**2) * fft
        rkfact(2*i+2) = rkfact(2*i+1)
  300 continue
      rkfact(1) = 0.0
      rk  = sin(rkmin*nxh*0.5)*2.0
      rkfact(2) = (1.0/rk**2) * fft
c ------------- Particle Initialization -----------------
      l = 0
      m = 0
      n2=0
      do 200 k=1,ns
        n1=n2
        n2=n1+np(k)
        phi = twopi/360.0*pch(k)
        vdpa = vd(k)*cos(phi)
        vdpe = vd(k)*sin(phi)
        do 100 i=n1+1,n2
          x(i)  = xs(k) + xl(k)*(i-n1-1)/float(np(k))
          vxi   = vpa(k)*strndm(l) + vdpa
          phase = twopi*unrndm(m)
          vyi   = vpe(k)*strndm(l) + vdpe*cos(phase)
          vz(i) = vpe(k)*strndm(l) + vdpe*sin(phase)
          vx(i) = costh*vxi - sinth*vyi
          vy(i) = sinth*vxi + costh*vyi
  100   continue
  200 continue
c ------------- Field Initialization -----------------
      do 20 i = 1,nxp2
        ex(i) = 0.0
        ey(i) = 0.0
```

```
      ez(i) = 0.0
      by(i) = by0
      bz(i) = 0.0
   20 continue
```

2.6 POSITN

This subroutine advances particle positions x using velocities v_x. In one time step Δt, particle positions are advanced twice each by a half time step $\Delta t/2$ as in the following.

$$x^{t+\Delta t/2} = x^t + v_x^{t+\Delta t/2}\frac{\Delta t}{2} \tag{2.27}$$

$$x^{t+\Delta t} = x^{t+\Delta t/2} + v_x^{t+\Delta t/2}\frac{\Delta t}{2} \tag{2.28}$$

These operations can be performed by calling the subroutine POSITN with the following source code twice in a cycle of one time step.

```
      do 100 i = 1, npt
          x(i) = x(i)+vx(i)
          if(x(i).lt.0.0) x(i) = x(i)+slx
          if(x(i).ge.slx) x(i) = x(i)-slx
   100 continue
```

where SLX is the size of the spatial simulation region.

2.7 VELCTY

This subroutine advances particle velocities by integrating the equation of motion

$$\frac{d\boldsymbol{v}}{dt} = \frac{q_s}{m_s}(\boldsymbol{E} + \boldsymbol{v} \times \boldsymbol{B}) \tag{2.29}$$

The difference equation of (2.29) is

$$\frac{\boldsymbol{v}^{t+\Delta t/2} - \boldsymbol{v}^{t-\Delta t/2}}{\Delta t} = \frac{q_s}{m_s}(\boldsymbol{E}^t + \frac{\boldsymbol{v}^{t+\Delta t/2} + \boldsymbol{v}^{t-\Delta/2}}{2} \times \boldsymbol{B}^t) \tag{2.30}$$

Defining new variables $\boldsymbol{v}^-$ and $\boldsymbol{v}^+$ as

$$\boldsymbol{v}^- = \boldsymbol{v}^{t-\Delta t/2} + \frac{q_s}{m_s}\boldsymbol{E}^t\frac{\Delta t}{2} \tag{2.31}$$

$$\boldsymbol{v}^+ = \boldsymbol{v}^{t+\Delta t/2} - \frac{q_s}{m_s}\boldsymbol{E}^t\frac{\Delta t}{2} \quad , \tag{2.32}$$

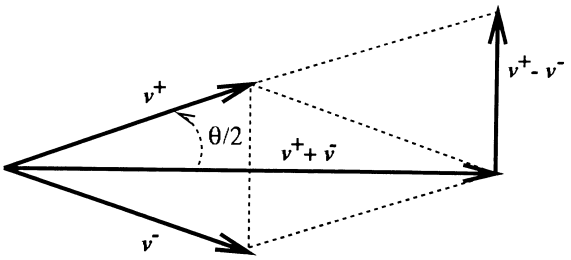

Figure 2.3: Vector relation in Buneman-Boris method.

we rewrite Eq. (2.30) as

$$\frac{\boldsymbol{v}^+ - \boldsymbol{v}^-}{\Delta t} = \frac{1}{2}\frac{q_s}{m_s}(\boldsymbol{v}^+ + \boldsymbol{v}^-) \times \boldsymbol{B}^t \tag{2.33}$$

Taking the inner product of (2.33) with $(\boldsymbol{v}^+ + \boldsymbol{v}^-)$, we have

$$(\boldsymbol{v}^+)^2 = (\boldsymbol{v}^-)^2 \tag{2.34}$$

As shown in Figure 2.3, this means that Eq. (2.33) expresses a rotation by an angle θ

$$\theta = -2\tan^{-1}(\frac{\Delta \mathrm{t}}{2}\frac{\mathrm{q_s}}{\mathrm{m_s}}\mathrm{B^t}) \tag{2.35}$$

The actual computation is realized by the following four steps, which are called Buneman-Boris method [Hockney and Eastwood, 1981; Birdsall and Langdon, 1985].

$$\boldsymbol{v}^- = \boldsymbol{v}^{t-\Delta t/2} + (q/m)_s \boldsymbol{E}^t \frac{\Delta t}{2} \tag{2.36}$$

$$\boldsymbol{v}^o = \boldsymbol{v}^- + \boldsymbol{v}^- \times (q/m)_s \boldsymbol{B}^t \frac{\Delta t}{2} \tag{2.37}$$

$$\boldsymbol{v}^+ = \boldsymbol{v}^- + \frac{2}{1 + ((q/m)_s B^t \Delta t/2)^2} \boldsymbol{v}^o \times \boldsymbol{B}^t \frac{\Delta t}{2} \tag{2.38}$$

$$\boldsymbol{v}^{t+\Delta t/2} = \boldsymbol{v}^+ + (q/m)_s \boldsymbol{E}^t \frac{\Delta t}{2} \tag{2.39}$$

where $\boldsymbol{E}^t$ and $\boldsymbol{B}^t$ are electric and magnetic fields linearly interpolated from the values at the adjacent grid points, respectively. The quantity $(q/m)_s$ is a charge-to-mass ratio for a particle species 's'. All $\boldsymbol{v}$, $\boldsymbol{E}$ and $\boldsymbol{B}$ are vector quantities.

Since we have adopted the staggered dual grid system for different components of electric and magnetic fields, the interpolation must be done differently

for different components of the fileds. Before the interpolation, however, relocation of the field quantities is necessary to avoid the electrostatic and magnetostatic self-forces.

The electrostatic field E_x is computed from the charge density ρ so that they satisfy the equation

$$\frac{\partial E_x}{\partial x} = \rho \tag{2.40}$$

whose difference form in the code is

$$E_{x,i+1/2} - E_{x,i-1/2} = \rho_i \tag{2.41}$$

The electric field E_x is defined at half-integer grids, while the charge density ρ is defined at the full-integer grids. Let us assume we have a single charged particle in a simulation system. We distribute the charge of the particle to the adjacent full-integer grid points by the linear weighting to compute the charge density ρ. The electric field satisfying the above equation is computed at the half-integer grid points. If the electric filed acting on the particle is interpolated linearly from E_x at the adjacent half-integer grid points, the particle feels a self-force ($E_x \neq 0$) which varies depending on the distance between the particle and the grid points. To avoid the self-force, we need to relocate the electric field defined at the half-integer grid points to the full-integer grid points [Matsumoto and Omura, 1985]. The relocation is made by the following equation.

$$E_{x,i} = \frac{1}{2}(E_{x,i-1/2} + E_{x,i+1/2}) \tag{2.42}$$

The electric field acting on the particle is then interpolated from the relocated $E_{x,i}$.

The principle of the self-force cancellation is that the same grid points must be used for distribution of the particle charge and for interpolation of the electrostatic field at the particle.

The same principle is also applied to the current density $\boldsymbol{J}$ and the magnetic field $\boldsymbol{B}$ related by the magnetostatic equation

$$\nabla \times \boldsymbol{B} = \frac{1}{c^2}\boldsymbol{J} \tag{2.43}$$

whose difference form in the code is written as,

$$B_{z,i+1/2} - B_{z,i-1/2} = -\frac{J_{y,i}}{c^2} \tag{2.44}$$

$$B_{y,i+1} - B_{y,i} = \frac{J_{z,i+1/2}}{c^2} \tag{2.45}$$

Since B_z and J_y in (2.44) and B_y and J_z in (2.45) are defined at different grid points, we need to relocate them. As we will see in the subroutine CURRNT/CURNTV, however, the current density J_y is first computed at the half-grid points, and then it is relocated to the full-grid points. Therefore, relocation

of B_z is not necessary. Since the J_z is computed at the half-grid points, the B_y defined at the full-grid points must be relocated to the half-grid points by the following equation.

$$B_{y,i+1/2} = \frac{1}{2}(B_{y,i} + B_{y,i+1}) \tag{2.46}$$

In the following source code of the VELCTY subroutine, we first relocate the fields E_x and B_y, and then advance velocities of particles interpolating the fields at particle positions from the adjacent grid points. Interpolation of E_x and E_y is performed from the full-integer grid points, while interpolation of E_z, B_y and B_z is performed from the half-integer grid points.

```
      do 100 i = 2, nxp1
        work1(i) = 0.5 * ( ex(i-1) + ex(i) )
  100 continue
      work1(nxp2) = work1(2)
c
      do 110 i = 2, nxp1
        work2(i) = 0.5 * ( by(i+1) + by(i) )
  110 continue
      work2(1) = work2(nxp1)
c
      n2=0
      do 210 k=1,ns
         n1  = n2
         n2  = n1 + np(k)
         bx1 = bx0*qm(k)
         const = 1.0 + bx1*bx1
      do 200 m = n1+1, n2
c
            i   =  x(m) + 2.0
            sf2 = (x(m) + 2.0 - i)*qm(k)
            sf1 = qm(k) - sf2
            ih  =  x(m) + 1.5
            sh2 = (x(m) + 1.5 - ih)*qm(k)
            sh1 = qm(k) - sh2
            i1  = i + 1
            ih1 = ih+ 1
          ex1 = sf1*work1(i)  + sf2*work1(i1)
          ey1 = sf1*ey(i)     + sf2*ey(i1)
          ez1 = sh1*ez(ih)    + sh2*ez(ih1)
          by1 = sh1*work2(ih) + sh2*work2(ih1)
          bz1 = sh1*bz(ih)    + sh2*bz(ih1)
c
```

```
                boris = 2./(const + by1*by1 + bz1*bz1)
c
                vx(m) = vx(m) + ex1
                vy(m) = vy(m) + ey1
                vz(m) = vz(m) + ez1
c
                vxt    = vx(m) + vy(m)*bz1 - vz(m)*by1
                vyt    = vy(m) + vz(m)*bx1 - vx(m)*bz1
                vzt    = vz(m) + vx(m)*by1 - vy(m)*bx1
c
                vx(m) = vx(m) + boris*(vyt*bz1 - vzt*by1)
                vy(m) = vy(m) + boris*(vzt*bx1 - vxt*bz1)
                vz(m) = vz(m) + boris*(vxt*by1 - vyt*bx1)
c
                vx(m) = vx(m) + ex1
                vy(m) = vy(m) + ey1
                vz(m) = vz(m) + ez1
c
  200 continue
  210 continue
```

2.8 EFIELD

This subroutine advances the electric field in time by integrating one of Maxwell's equations

$$\frac{\partial \boldsymbol{E}}{\partial t} = c^2 \nabla \times \boldsymbol{B} - \boldsymbol{J} \tag{2.47}$$

We can rewrite the above equation for the one-dimensional system as

$$\frac{\partial E_x}{\partial t} = -J_x \tag{2.48}$$

$$\frac{\partial E_y}{\partial t} = -c^2 \frac{\partial B_z}{\partial x} - J_y \tag{2.49}$$

$$\frac{\partial E_z}{\partial t} = c^2 \frac{\partial B_y}{\partial x} - J_z \tag{2.50}$$

These equations are integrated for one time step of $\Delta t_S = 2$. The FORTRAN source program is written as follows;

```
      do 100 i=2,nxp1
        ex(i) = ex(i) - 2.*ajx(i)
        ey(i) = ey(i) - tcs*( bz(i) - bz(i-1) ) - 2.*ajy(i)
        ez(i) = ez(i) + tcs*( by(i+1) - by(i) ) - 2.*ajz(i)
  100 continue
```

```
ey(nxp2) = ey(2)
ez(1)    = ez(nxp1)
ex(1)    = ex(nxp1)
```

The last three lines are for the periodic boundary condition.

2.9 BFIELD

This subroutine advances the magnetic field in time by integrating one of Maxwell's equations

$$\frac{\partial \boldsymbol{B}}{\partial t} = -\nabla \times \boldsymbol{E} \tag{2.51}$$

We can rewrite the above equation for the one-dimensional system as

$$\frac{\partial B_y}{\partial t} = \frac{\partial E_z}{\partial x} \tag{2.52}$$

$$\frac{\partial B_z}{\partial t} = -\frac{\partial E_y}{\partial x} \tag{2.53}$$

These equations are integrated for a half time step of $\Delta t_S/2 = 1$. Therefore, this subroutine must be called twice within a loop of one time step before and after advancing particle velocities with electric fields and magnetic field.

The FORTRAN source program are writted as follows;

```
      do 200 i=2,nxp1
        by(i) = by(i) + ez(i) - ez(i-1)
        bz(i) = bz(i) - ey(i+1) + ey(i)
  200 continue
      by(nxp2) = by(2)
      bz(1)    = bz(nxp1)
```

The last two lines are for the periodic boundary condition.

2.10 CHARGE

This subroutine computes the charge density ρ from superparticles which have a square-shaped charge q with a width of a grid spacing Δx as shown in Figure 2.4. A superparticle at a position x_p has a charge density distribution $q/\Delta x$ in the range

$$x_p - \Delta x/2 \leq x_p < x_p + \Delta x/2 \tag{2.54}$$

On the other hand, each grid point at X_i has a territory which covers a range

$$X_i - \Delta x/2 \leq X_i < X_i + \Delta x/2 \tag{2.55}$$

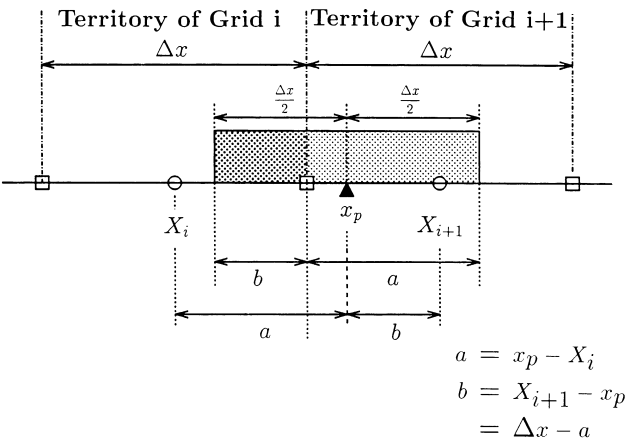

Figure 2.4: Area weighting method in computing charge density.

A superparticle which falls into the range $X_i \leq x_p < X_{i+1}$ has the charge distribution in the territories of the adjacent grid points at X_i and X_{i+1}. The charge distribution in each territory of the grid points is assigned to the grid points to form charge densities ρ defined at the grid points. In other words, the charge q of the superparticle is divided to the adjacent grid points proportional to the areas shared by the grid points. Numerically, $q(x_p - X_i)/\Delta x$ is assigned to $\rho(X_{i+1})$, and $q(X_{i+1} - x_p)/\Delta x$ is assigned to $\rho(X_i)$.

```
c
      do 100 i=1,nxp2
        rho(i) = rho0
  100 continue
c
          n2 = 0
      do 210 k=1,ns
         n1 = n2
         n2 = n1 + np(k)
      do 200 m = n1+1, n2
             i  =  x(m)+ 2.0
             s2 = (x(m)+ 2.0 - i)*q(k)
             s1 = q(k) - s2
             rho(i)   = rho(i)   + s1
             rho(i+1) = rho(i+1) + s2
  200     continue
  210     continue
        rho(2)    = rho(2) + rho(nxp2) - rho0
        rho(1)    = rho(nxp1)
```

```
rho(nxp2) = rho(2)
```

It is noted that the most time-consuming DO loop (200) is not vectorized, because of the recurrent accumulation of charges to the arrays *RHO*. Since this subroutine does not have to be called at every time step, we have not prepared the vectorized version. However, it is straightforward to vectorize this subroutine with the same algorithm used in the subroutine CURNTV, and left as an exercise.

2.11 ECRRCT

This subroutine corrects the electric field E_x so that it satisfies Poisson's equation. This subroutine is called at the beginning of the simulation to set up an initial electric field E_x based on a charge density distribution computed by the subroutine CHARGE. Although the charge conservation algorithm used in the computation of the current density J_x assures that the electric field E_x should satisfy Poisson's equation, the accumulation of round-off error requires correction of the electric field E_x at a certain interval.

For the one-dimensional model, the electric field E_x is purely electrostatic. Therefore, we may directly solve Poisson's equation

$$\frac{\partial E_x}{\partial x} = \rho \tag{2.56}$$

In two- or three-dimensional models, however, we use the following procedure used to correct the electric field in two-dimensional or three-dimensional model.

Assuming that we have the electric field $\boldsymbol{E}$ which contain an error, we compute a correcting electric field $\boldsymbol{E}_c$ which satisfies the following equation,

$$\nabla \cdot (\boldsymbol{E} + \boldsymbol{E}_c) = \rho \tag{2.57}$$

Defining

$$\rho_c = \rho - \nabla \cdot \boldsymbol{E} \tag{2.58}$$

we have

$$\nabla \cdot \boldsymbol{E}_c = \rho_c \tag{2.59}$$

Assuming a potential ϕ_c satisfying

$$\boldsymbol{E}_c = -\nabla \phi_c \tag{2.60}$$

we have Poisson's equation

$$\nabla^2 \phi_c = -\rho_c \tag{2.61}$$

whose difference form is

$$\frac{\partial^2 \phi_c}{\partial x^2} = \frac{\phi_c(x_{i+1}) - 2\phi_c(x_i) + \phi_c(x_{i-1})}{(\Delta x)^2} \tag{2.62}$$

There are a number of methods to solve Poisson's equation. For a periodic system, we apply the Fast Fourier Transform (FFT) to $\rho_c(x_i)$ to obtain $\rho_c(k_m)$, and compute $\phi_c(k_m)$ from Poisson's equation in the wavenumber domain,

$$K_m^2 \phi_c(k_m) = \rho_c(k_m) \tag{2.63}$$

where $k_m = 2\pi m/L : m = 1, 2, ..., N_x/2$, and

$$K_m = \frac{\sin(\mathrm{k_m}\Delta \mathrm{x}/2)}{\Delta \mathrm{x}/2} \tag{2.64}$$

We apply the Inverse-FFT to $\phi_c(k_m)$ to obtain $\phi_c(x_i)$. We compute $E_{xc}(x_{i+1/2})$ by

$$E_{xc}(x_{i+1/2}) = \frac{\phi_c(x_i) - \phi_c(x_{i+1})}{\Delta x} \tag{2.65}$$

Finally, the E_{xc} is added to the E_x.

The above operations are implemented in the following source code. The FFT and Inverse-FFT are performed by the subroutine REALFT which calculate the Fourier and Inverse-Fourier transforms of a set of NX real-valued data points. The coefficients *RKFACT(I)* are computed in the subroutine INITAL.

```
      do 100 i=2,nxp1
        work1(i-1) = rho(i) - ex(i) + ex(i-1)
100   continue
      call realft(work1,nx,1)
      do 200 i=1,nx
        work1(i) = work1(i)*rkfact(i)
200   continue
      call realft(work1,nx,-1)
      work1(nxp1) = work1(1)
      do 300 i=2,nxp1
        ex(i) = ex(i) + work1(i-1) - work1(i)
300   continue
        ex(1) = ex(nxp1)
```

2.12 CURRNT

This subroutine computes the current density $\boldsymbol{J}$ from velocities and positions of particles. For the difference forms of Maxwell's equations, J_x and J_z are defined at the half-integer grid points, while J_y are defined at the full-integer grid points. However, all components of $\boldsymbol{J}$ are computed at the half-grid points. After computing the current density, J_y is relocated from the half-integer grid points to the full-integer grid points. This makes the computation of the current

density simple, and also has an effect to eliminate the magnetostatic self-force from B_z field.

For J_y and J_z, we use the linear weighting to distribute the current $q_s \boldsymbol{v}$ of a particle as we do in computing the charge density. As for J_x we use the charge conservation method, proposed by Villasenor and Buneman [1991], which assures that the following equation of continuity for charge is satisfied locally.

$$\frac{\partial \rho}{\partial t} + \nabla \cdot \boldsymbol{J} = 0 \tag{2.66}$$

whose difference form is written as

$$\rho_i^{t+\Delta t} - \rho_i^t = -(J_{i+1/2}^{t+\Delta t/2} - J_{i-1/2}^{t+\Delta t/2}) \frac{\Delta t}{\Delta x} \tag{2.67}$$

where J is the x-component J_x. Since we use a square-shaped superparticle with a width Δx in computing the charge density ρ at the full-grid points, we can compute the current density J at the half-grid points by taking into account the amount of charge crossing the half-grid points from time t to $t + \Delta t$.

In the subroutine EFIELD, we use the current density $J^{t+\Delta t/2}$ to advance the electric field by the difference equation

$$- J_{i+1/2}^{t+\Delta t2} \Delta t = E_{i+1/2}^{t+\Delta t} - E_{i+1/2}^t \tag{2.68}$$

where J and E are J_x and E_x, respectively. Substituting $J_{i-1/2}^{t+\Delta t/2}$ and $J_{i+1/2}^{t+\Delta t/2}$ in (2.67) by (2.68), we have

$$\rho_i^{t+\Delta t} - \rho_i^t = (E_{i+1/2}^{t+\Delta t} - E_{i+1/2}^t - E_{i-1/2}^{t+\Delta t} + E_{i-1/2}^t) \frac{1}{\Delta x} \tag{2.69}$$

which is rewritten as

$$\rho_i^{t+\Delta t} - \frac{E_{i+1/2}^{t+\Delta t} - E_{i-1/2}^{t+\Delta t}}{\Delta x} = \rho_i^t - \frac{E_{i+1/2}^t - E_{i-1/2}^t}{\Delta x}. \tag{2.70}$$

Therefore, if the electric field E^t at time t satisfies the electrostatic equation, i.e.,

$$\frac{E_{i+1/2} - E_{i-1/2}}{\Delta x} = \rho_i, \tag{2.71}$$

then, the electric field field $E^{t+\Delta t}$ also satisfied the above equation.

To compute the current $J_x^{t+\Delta t/2}$ satisfying (2.67), we have to think of two cases, assuming that a particle does not move more than one grid spacing Δx by one time step Δt, i.e.,

$$x_p(t + \Delta t) - x_p(t) < \Delta x, \tag{2.72}$$

where $x_p(t)$ is a particle position at time t. One is Case 1 shown in Figure 2.5 where both $x_p(t)$ and $x_p(t + \Delta t)$ are in the same cell between X_i and X_{i+1}. The

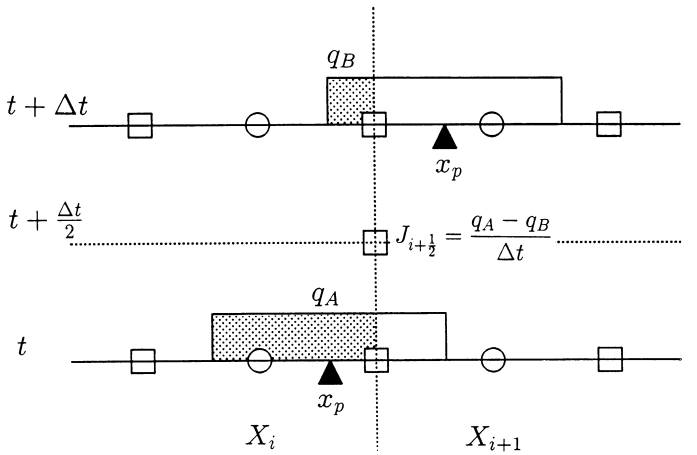

Figure 2.5: Charge conservation method in computing current density: Case 1.

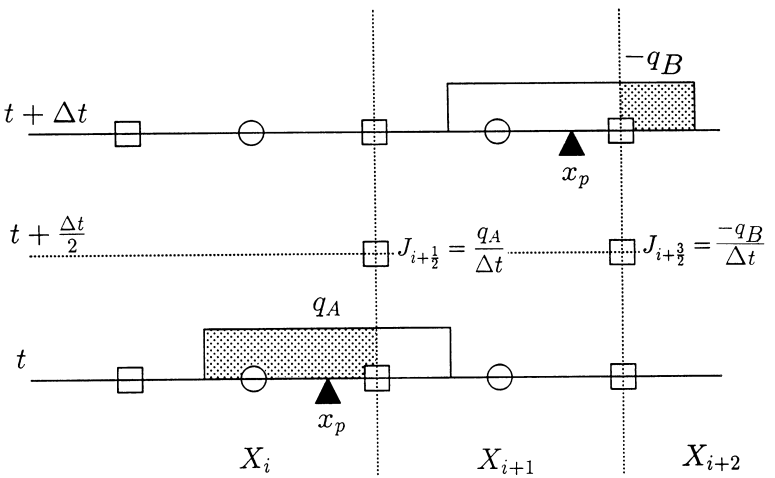

Figure 2.6: Charge conservation method in computing current density: Case 2.

other is Case 2 shown in Figure 2.6 where $x_p(t)$ and $x_p(t+\Delta t)$ are in different cells. In Case 1, the current $I_{i+1/2}$ at $X_{i+1/2}$ is given by computing the amount of the charge passing through the point of $X_{i+1/2}$ within the time step Δt.

$$I_{i+1/2} = \frac{q_A - q_B}{\Delta t}, \tag{2.73}$$

where q_A and q_B are given by

$$q_A = q\frac{X_i - x_p(t)}{\Delta x}, \qquad q_B = q\frac{X_i - x_p(t+\Delta t)}{\Delta x}. \tag{2.74}$$

In Case 2, the particle motion contributes to the currents at $X_{i+1/2}$ and $X_{i+3/2}$, which are give by

$$I_{i+1/2} = \frac{q_A}{\Delta t}, \qquad I_{i+3/2} = -\frac{q_B}{\Delta t}, \tag{2.75}$$

respectively. In Figures 2.5 and 2.6, we assumed that the particle velocity is positive. For the cases with a negative velocity, we need to multiply -1 to the left hand side of equations (2.73) and (2.75). These computation for different cases are realized by the simple coding without an "*IF*" statement as shown in the following listing of the subroutine CURRNT.

In a periodic system, a spatially uniform component of the current density $\boldsymbol{J}_u$ give rise to a spatially uniform component of the electric field $\boldsymbol{E}_u$, which oscillates in time with the plasma frequency in an unmagnetized plasma. Assuming an unmagnetized plasma with an electron density n_e, we have the following two equations from the equation of motion and Maxwell's equations,

$$\frac{\partial \boldsymbol{J}_u}{\partial t} = \frac{n_e e^2}{m_e}\boldsymbol{E}_u \tag{2.76}$$

$$\frac{\partial \boldsymbol{E}_u}{\partial t} = -\boldsymbol{J}_u \tag{2.77}$$

where $-e$ and m_e are the charge and mass of an electron. As a solution we have

$$\boldsymbol{J}_u = \boldsymbol{J}_o \exp(\omega_{pe} t), \qquad \boldsymbol{E}_u = \frac{i}{\omega_{pe}}\boldsymbol{J}_o \exp(\omega_{pe} t) \tag{2.78}$$

where $\omega_{pe} = \sqrt{n_e e^2/m_e}$. This uniform oscillation of $\boldsymbol{E}_u$ and $\boldsymbol{J}_u$ is not of our interest in doing simulations. It perturbs the simulation system, and makes it difficult to see the real physical process. Therefore, we need to eliminate the uniform component from the current density. We first compute the average of the current densities and subtract it from the current density at each half-grid point.

The above operations are implemented in the following source code.

```
c
      do 100 i=1,nxp2
        ajx(i) = 0.0
        ajy(i) = 0.0
        ajz(i) = 0.0
  100 continue
c
      n2 = 0
      do 210 k=1,ns
        n1 = n2
        n2 = n1 + np(k)
        qh = q(k)*0.5
        do 200 m = n1+1, n2
          ih =  x(m) + 1.5
          s2 = (x(m) + 1.5 - ih)*q(k)
          s1 = q(k) - s2
          ih1 = ih + 1
          ajy(ih)   = ajy(ih)  + vy(m)*s1
          ajy(ih1)  = ajy(ih1) + vy(m)*s2
          ajz(ih)   = ajz(ih)  + vz(m)*s1
          ajz(ih1)  = ajz(ih1) + vz(m)*s2
c--------- charge conservation method ---------
          qhs = qh * sign(1.0, vx(m))
          avx=abs(vx(m))
          x1 = x(m) + 2.0 - avx
          x2 = x(m) + 2.0 + avx
          i1 = x1
          i2 = x2
          ajx(i1) = ajx(i1) + (i2 - x1)*qhs
          ajx(i2) = ajx(i2) + (x2 - i2)*qhs
c-----------------------------------------------
  200   continue
  210 continue
c
      ajx(nxp1) = ajx(1) + ajx(nxp1)
      ajx(2)    = ajx(2) + ajx(nxp2)
      ajy(nxp1) = ajy(1) + ajy(nxp1)
      ajy(2)    = ajy(2) + ajy(nxp2)
      ajy(1)    = ajy(nxp1)
      ajz(nxp1) = ajz(1) + ajz(nxp1)
      ajz(2)    = ajz(2) + ajz(nxp2)
c
      do 300 i = nxp1, 2,-1
```

```
        ajy(i) = (ajy(i) + ajy(i-1))*0.5
  300 continue
c-------- cancel the uniform component ---------------
      juncan = 1
      if(juncan.eq.1) then
      ajxu = 0.0
      ajyu = 0.0
      ajzu = 0.0
      do 400 i = 2,nxp1
        ajxu = ajxu + ajx(i)
        ajyu = ajyu + ajy(i)
        ajzu = ajzu + ajz(i)
  400 continue
      ajxu = ajxu/float(nx)
      ajyu = ajyu/float(nx)
      ajzu = ajzu/float(nx)
      do 500 i = 2,nxp1
        ajx(i) = ajx(i) - ajxu
        ajy(i) = ajy(i) - ajyu
        ajz(i) = ajz(i) - ajzu
  500 continue
      endif
c
```

It is noted that the most time-consuming DO loop (200) can not be vectorized, because of the recurrent accumulation of currents to the arrays AJX, AJY and AJZ. For a computer with a vector processor, we have prepared a different subroutine named CURNTV, which should be called in place of the subroutine CURRNT in the MAIN program.

It is also noted that J_y is computed at the half-grid points, and then relocated to the full-grid points by the following procedure.

$$J_{y,i} = \frac{J_{y,i-1/2} + J_{y,i+1/2}}{2} \tag{2.79}$$

This procedure has the following three advantages compared with a direct computation of J_y at the full-grid points.

- The computation is simplified, because we can use the same area weightings for J_y and J_z.
- A relocation of B_z in interpolating the particle-pushing field in the subroutine VELCTY is not necessary, because the same area weightings are used as those in the computation of J_y.
- The electromagnetic fluctuations at the short wavelength consisting of several grid points are suppressed because of the filtering effect of the above

operation of Eq. (2.79). The fluctuations of J_y are suppressed by the factor $\cos(k\delta x/2)$, which varies from 1 to 0 with for $0 \leq k \leq \pi/\delta x$.

2.13 CURNTV

This subroutine is the vectorized version of the CURRNT subroutine, which computes the current density. The vectorization is achieved by preparing large two-dimensional arrays with a size of $(LVEC \times IX)$. The parameter $LVEC$ is a number of repetition in the most inner DO-loop which is vectorized. A larger value of $LVEC$ increases the vector efficiency in the DO-loop 200, while it increases the amount of computation in the DO-loop 300. There is an optimum value of $LVEC$ for a different type of vector computers. We normally set $LVEC$= $32 \sim 128$ depending of the availability of the main memory.

Some vector compilers require an explicit declaration that the recurrence does not occur in the most inner DO-loop 200 such as shown in the following source code.

```
      parameter(lvec=64)
      dimension wrk1(lvec,ix),wrk2(lvec,ix),wrk3(lvec,ix)
c
      do 100 i=1,nxp2
        ajx(i) = 0.0
        ajy(i) = 0.0
        ajz(i) = 0.0
  100 continue
      do 150 i=1,nxp2
      do 150 l=1,lvec
         wrk1(l,i) = 0.0
         wrk2(l,i) = 0.0
         wrk3(l,i) = 0.0
  150 continue
c
      n2 = 0
      do 210 k=1,ns
        n1 = n2
        n2 = n1 + np(k)
        qh = q(k)*0.5
        do 200 ik = n1+1,n2,lvec
c$dir no_recurrence
        do 200 m  = ik,min(ik+lvec-1,n2)
          l = m - ik + 1
          ih = x(m) + 1.5
          s2 = (x(m) + 1.5 - ih)*q(k)
```

```
          s1 = q(k) - s2
          ih1 = ih + 1
          wrk2(l,ih )  = wrk2(l,ih ) + vy(m)*s1
          wrk2(l,ih1)  = wrk2(l,ih1) + vy(m)*s2
          wrk3(l,ih )  = wrk3(l,ih ) + vz(m)*s1
          wrk3(l,ih1)  = wrk3(l,ih1) + vz(m)*s2
c--------- charge conservation method ---------
          qhs = qh * sign(1.0, vx(m))
          avx=abs(vx(m))
          x1 = x(m) + 2.0 - avx
          x2 = x(m) + 2.0 + avx
          i1 = x1
          i2 = x2
          wrk1(l,i1) = wrk1(l,i1) + (i2 - x1)*qhs
          wrk1(l,i2) = wrk1(l,i2) + (x2 - i2)*qhs
c-----------------------------------------------
  200   continue
  210 continue
c
      do 300 i=1,nxp2
      do 300 l=1,lvec
        ajx(i) = ajx(i) + wrk1(l,i)
        ajy(i) = ajy(i) + wrk2(l,i)
        ajz(i) = ajz(i) + wrk3(l,i)
  300 continue
c
      ajx(nxp1) = ajx(1) + ajx(nxp1)
      ajx(2)    = ajx(2) + ajx(nxp2)
      ajy(nxp1) = ajy(1) + ajy(nxp1)
      ajy(2)    = ajy(2) + ajy(nxp2)
      ajy(1)    = ajy(nxp1)
      ajz(nxp1) = ajz(1) + ajz(nxp1)
      ajz(2)    = ajz(2) + ajz(nxp2)
c
      do 350 i = nxp1, 2,-1
        ajy(i) = (ajy(i) + ajy(i-1))*0.5
  350 continue
c-------- cancel the uniform component ---------------
      juncan = 1
      if(juncan.eq.1) then
      ajxu = 0.0
      ajyu = 0.0
      ajzu = 0.0
```

```
      do 400 i = 2,nxp1
        ajxu = ajxu + ajx(i)
        ajyu = ajyu + ajy(i)
        ajzu = ajzu + ajz(i)
  400 continue
      ajxu = ajxu/float(nx)
      ajyu = ajyu/float(nx)
      ajzu = ajzu/float(nx)
      do 500 i = 2,nxp1
        ajx(i) = ajx(i) - ajxu
        ajy(i) = ajy(i) - ajyu
        ajz(i) = ajz(i) - ajzu
  500 continue
      endif
c
```

2.14 Diagnostics of KEMPO1

A run of KEMPO1 generates a series of on-line graphic diagnostics when it is executed on a computer. The first page of the KEMPO1 graphic outputs shows the input parameters, which are plotted by the subroutine PLTPRM. After the list of the input parameters, various diagnostics are plotted to show the initial condition and subsequent time evolution of the simulation system. These diagnostics are generated by the subroutines FLDPLT, KSPPLT, PHSPLT, VDSPLT, SPECTR and ENERGY, which are called from the MAIN program. There are two different kinds of diagnostics. One is a snap shot of the physical quantities showing their variation in space or configuration in phase spaces at a specific time, as generated by the subroutines KSPPLT, PHSPLT and VDSPLT. The other is a history plot of physical quantities showing their variation in time or frequency spectra, as generated by the subroutines SPECTR and ENERGY. In these subroutines, physical quantities are plotted in the R-unit system being converted from the S-unit system based on the coefficients given by the *COMMON* statements */RESCLC/*.

As described in Section 2.3, the timings of the snap shot diagnostics are specified by the input parameters IFDIAG, IKDIAG, IPDIAG and IVDIAG. The functions of these diagnostic subroutines are switched by the control parameters IFPLOT, IKPLOT, IPPLOT and IVPLOT. The data for the history plots generated by SPECTR and ENERGY are stored in the internal memory at time step intervals specified by ISDIAG and IEDIAG, and the history plots are generated at the timings specified by the ISPLOT and IEPLOT. The techniques used in the diagnostics subroutines and some examples of graphic outputs are given in the following subsections with the names of the diagnostic subroutines.

2.14.1 FLDPLT

This subroutine plots spatial profiles of the field quantities defined at grid points. Two different types of plots are implemented. One is for an electrostatic simulation, and it plots the electrostatic potential, electric field E_x and charge density ρ as shown in Figure 2.7(a). The dashed lines in the top and bottom panels indicate the zero levels of the potential and charge density, respectively. The other is for an electromagnetic simulation, and it plots the five elements of the electromagnetic field E_x, E_y, E_z, B_y, B_z and the charge density ρ as shown in Figure 2.7(b). These plots can be selected by the input parameter $IFPLOT$, as described in Section 2.3.

2.14.2 KSPPLT

This subroutine plots wavenumber spectra of the fields E_x, E_y, E_z, B_y and B_z at a time step interval specified by the parameter $IKDIAG$. The minimum and maximum wavenumbers to be plotted are

$$k_{min} = \frac{2\pi}{SLX}, \qquad k_{max} = k_{min} \times IKPLOT \tag{2.80}$$

where $IKPLOT \leq NX/2$. It is necessary to specify an appropriate $IKPLOT$ so that we can see wave spectra of our interest. An example of the output is given in Figure 2.8.

2.14.3 PHSPLT

This subroutine plots phase diagrams of particles in three different sets of coordinates at an interval specified by $IPDIAG$. The first one is in the $x - v_x$ phase space which is suitable for analyzing the dynamics of particles interacting with electrostatic waves. The second one is in the $v_z - v_{\perp xy}$ which corresponds to the velocity space of the perpendicular velocity $\boldsymbol{v}_{perp}$. This diagnostics is useful for viewing diffusion of a ring distribution of particles or perpendicular heating of particles. The third one is in the $v_\parallel - v_\perp$ phase space which is useful for checking pitch angle scattering of particles by electromagnetic waves like whistler mode waves or ion cyclotron waves. The velocity range for these phase space plots can be specified by the input parameters $VMIN$ and $VMAX$, while an automatic scaling is activated by setting $VMIN = VMAX$. In case the number of particles is very large, the subroutine can limit the number of plotted particles by skipping the particles with an interval. One can change the maximum number of plotted particles by the parameter $IPMAX$ defined in the beginning of the subroutine. An example of these plots is given in Figure 2.9.

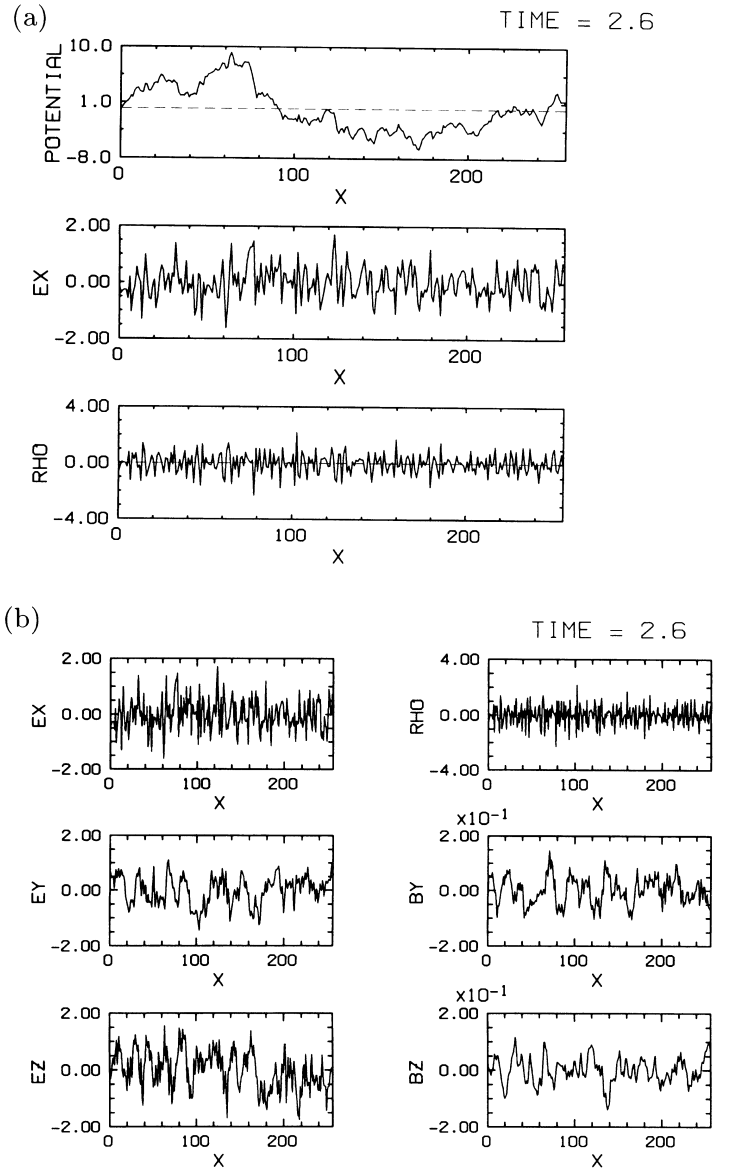

Figure 2.7: An example of outputs by FLDPLT subroutine. (a) $IFPLOT = 1$, (b) $IFPLOT = 2$.

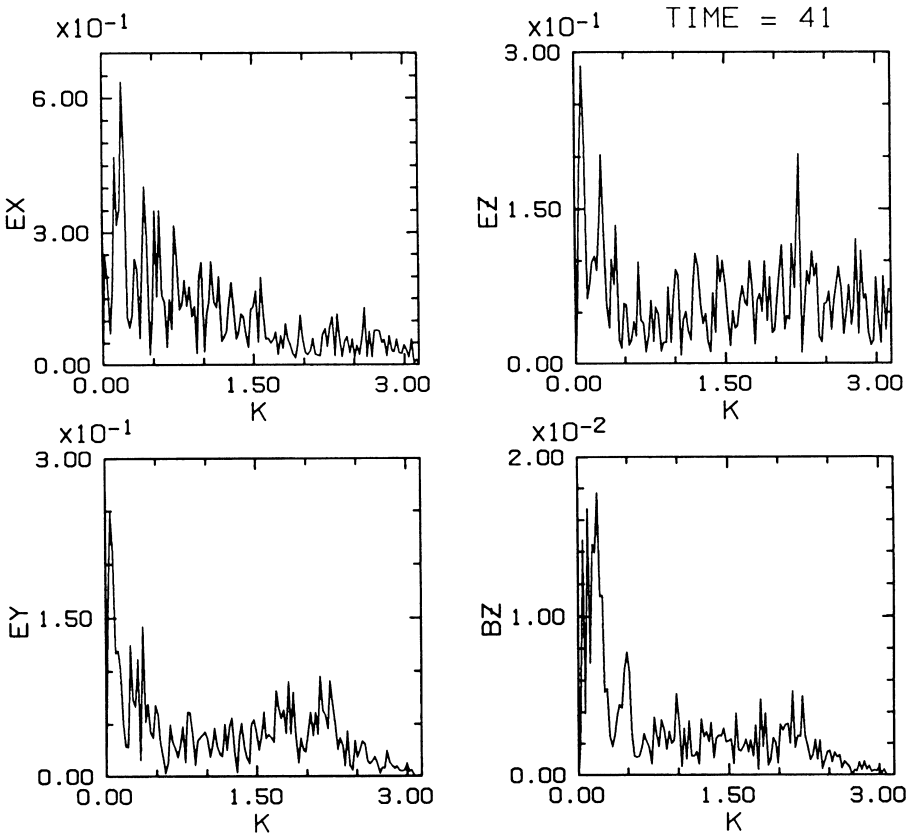

Figure 2.8: An example of outputs by KSPPLT subroutine.

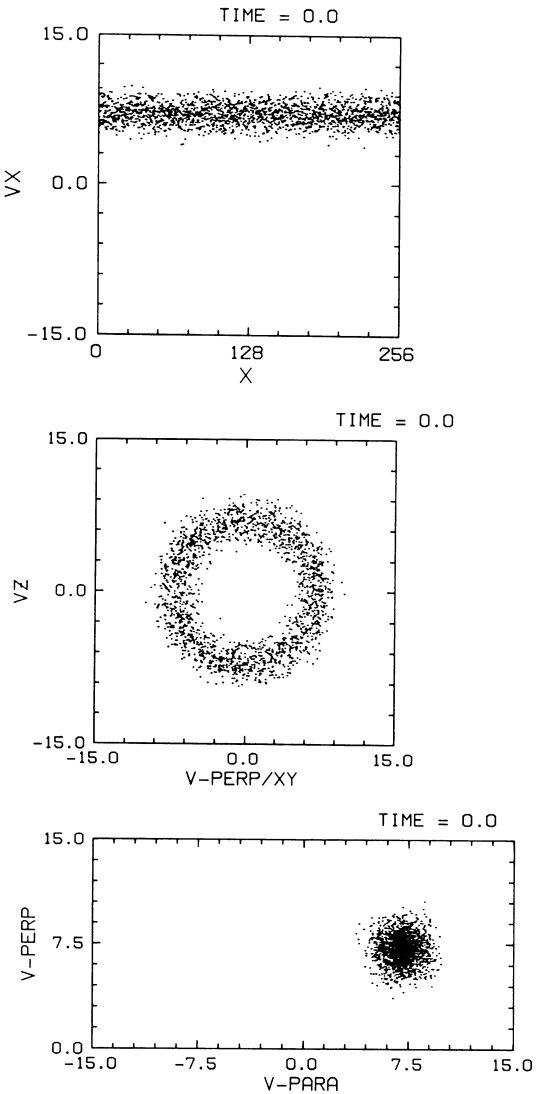

Figure 2.9: An example of outputs by PHSPLT subroutine.

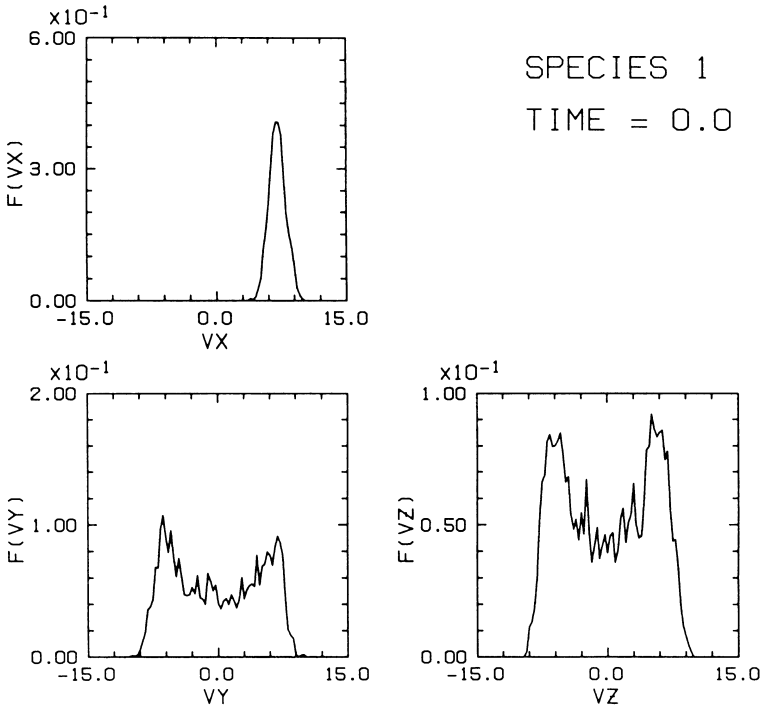

Figure 2.10: An example of the outputs by VDSPLT subroutine.

2.14.4 VDSPLT

This subroutine plots velocity distribution functions of particles for v_x, v_y and v_z at an time step interval specified by $IVDIAG$. The input parameter $IVPLOT$ is currently a dummy parameter for possible variations of the diagnostics. The range of the velocities for the distribution functions is specified by the input parameter $VMIN$ and $VMAX$. When $VMIN = VMAX$ is specified, the range is automatically taken from the minimum and maximum values of the velocities. The scale of the distribution functions is given so that the integration of $f(v_i)$ over v_i becomes unity. An example of the outputs is given in Figure 2.10.

2.14.5 SPECTR

This subroutine generates two different diagnostics based on the time history of each component of electromagnetic field E_x, E_y, E_z, B_y and B_z. One is a mode history plot with a fixed wavenumber $k_i = k_{min} \times i$ as shown in Figure 2.11. The other is a $\omega - k$ diagram showing spectra of each component of the fields

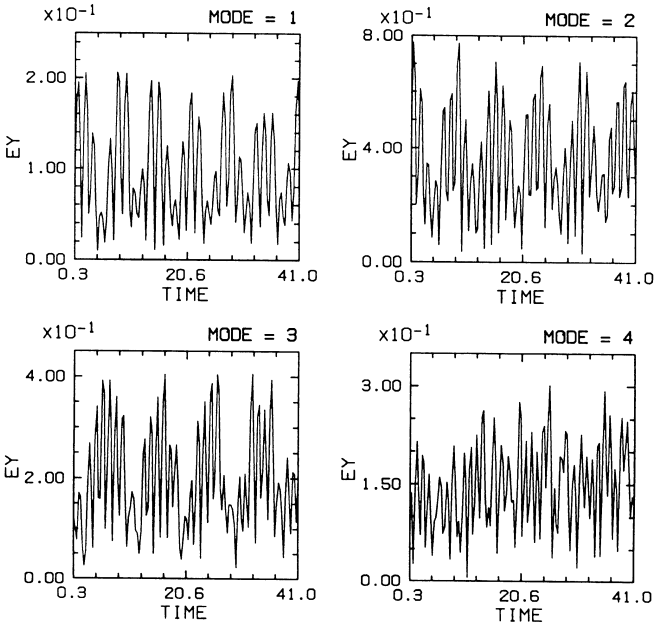

Figure 2.11: An example of the mode history plots by SPECTR subroutine.

as shown in Figure 2.12. The range of mode numbers to be plotted as the mode history plot is specified by the parameters $MINMOD$ and $MAXMOD$ in the subroutine. The $\omega - k$ diagram plots spectra for the wavenumbers from 0 to $(k_{min} \times IKP)$ and for the frequencies from 0 to $(\omega_{min} \times IWP)$, where IKP and IWP are parameters defined in the subroutine. The minimum frequency ω_{min} is given by

$$\omega_{min} = \frac{2\pi}{DT \times ISDIAG \times ISPLOT} \tag{2.81}$$

The parameters IKP and IWP cannot exceed $IMAX/2$ and $ISPLOT/2$, respectively. The $\omega - k$ diagram is produced by calling the subroutines WKFFT and WKPLOT. The technique to generate the spectrum is described in Matsumoto and Omura [1985]. We can separate the forward and backward traveling waves by this technique, and it can be specified by modifying the parameter IFB in the subroutine. If $IFB = 0$, the sum of the forward and backward wave spectra is plotted. If $IFB = 1(-1)$, then only forward (backward) wave spectra are plotted. For purely electrostatic simulation, we only need to make diagnostics for E_x. This can be done by setting the parameter $ICOMP = 1$ instead of $ICOMP = 5$ in the subroutine.

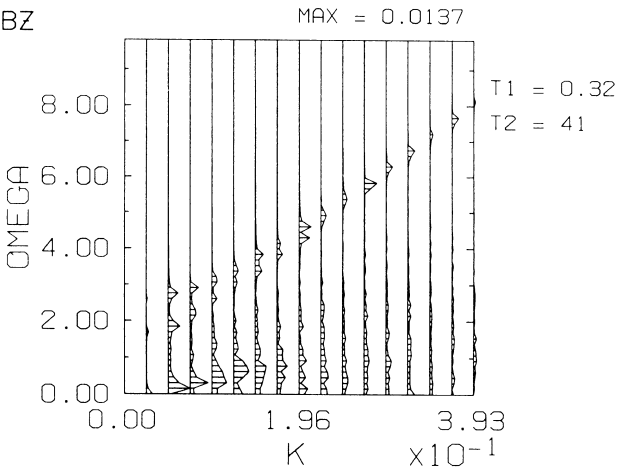

Figure 2.12: An example of $\omega - k$ diagrams by SPECTR subroutine.

2.14.6 ENERGY

Every time this subroutine is called at an interval specified by the input parameter *ISDIAG*, the energy diagnostics is made for the field and particles. The results are stored in the internal array. After accumulating the data for *ISPLOT* time sequences, the subroutine plots time histories of the energies of fields and particles. In the first page of the energy diagnostics, the kinetic, electric, magnetic, and total energy are plotted. In the second page, variations of these energies from the initial values are plotted. In each of the subsequent pages, the drift and thermal energy, and their summation and temperature anisotropy are plotted for each species. The temperature anisotropy A_i of particle species i is defined as a temperature ratio

$$A_i = \frac{T_{y,i} + T_{z,i}}{2T_{x,i}} \tag{2.82}$$

rather than a temperature ratio between perpendicular and parallel temperatures with respect to the static magnetic field. Examples of these energy history plots are given in Figure 2.13.

2.15 Exercises of KEMPO1

2.15.1 Single particle motion

Put a single electron in the system with a drift velocity in parallel and/or perpendicular direction. Confirm that there is no electrostatic self-force acting on a particle, and the kinetic energy of the particle is conserved.

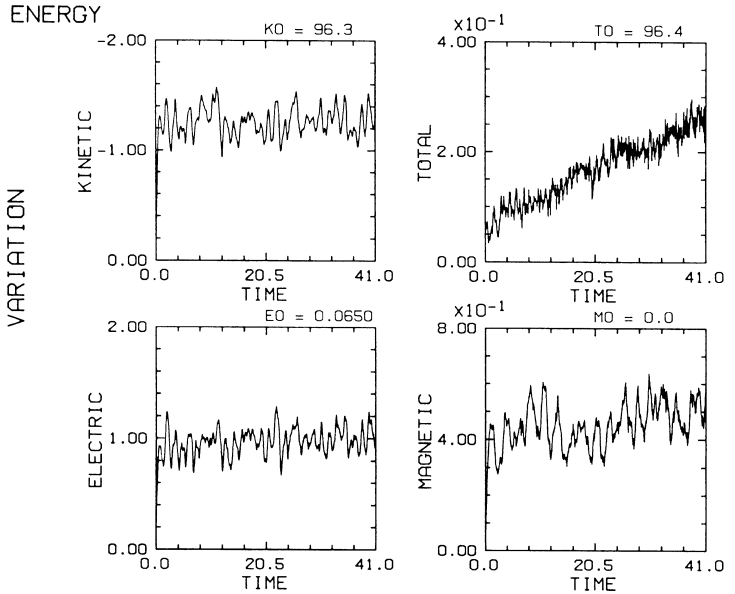

Figure 2.13: An example of outputs by ENERGY subroutine.

Modify the source code of the VELCTY subroutine so that the electrostatic field E_x is interpolated from the half-grid points rather than the relocated E_x at the full-grid points. Find out what will happen.

2.15.2 Thermal fluctuations

Set up a magnetized plasma consisting of thermal electrons. Call the SPECTR subroutine at a proper interval, and draw the $\omega - k$ diagram for each component of the fields. Compare the spectra with the normal modes of the plasma. Vary the plasma parameters such as the plasma frequency, thermal velocity or angle of the static magnetic field.

2.15.3 Two stream instability

Set up two streams of electrons with different drift velocities and with equal electron densities. Find formation of an electrostatic field and nonlinear dynamics of beam electrons. Change the density ratio between the two electron beams to find a weak beam instability.

2.15.4 Buneman instability

Set up an unmagnetized plasma consisting of cold electrons and drifting cold ions forming a current. Find the nonlinear evolution of the famous Buneman instability [e.g, Melrose, 1986].

2.15.5 Non-cancellation of the uniform current

Modify the CURRNT/CURNTV subroutine not to cancel the uniform component of the current J_x. Run a job with the same parameters used for Buneman instability.

2.15.6 Whistler mode beam instability

Eliminate the electrostatic interaction in the KEMPO1 by setting $J_x = 0$. Put a weak electron beam in a plasma and find that whistler mode waves are generated through the cyclotron resonance interaction.

2.15.7 Weibel instability

Set up an unmagnetized electron plasma with a distribution with the large perpendicular thermal velocity and a small parallel thermal velocity. The "perpendicular" and "parallel" is referenced with respect to the wavenumber vector $\boldsymbol{k}$ in the x direction. Specify proper diagnostics of particle and field to find particle scattering in the $v_{\parallel} - v_{\perp}$ phase space by the unstable magnetic field. Also modify the subroutine PHSPLT to plot the phase diagram in $x - v_y$ phase space.

2.15.8 Violation of Courant condition

Comment out the call of the CHKPRM subroutine in the MAIN program. Specify the time step which does not satisfy the Courant condition, and run the program.

2.15.9 Numerical heating of a plasma

Set up an unmagnetized plasma with a very small thermal velocity, where the Debye length λ_D is much smaller than the grid spacing Δx. Examine the energy conservation.

2.15.10 Electrostatic simulation

Modify the main program to solve only the electrostatic field E_x using the ECRRCT subroutine. Run the program with the parameters of two-stream

instability with a time step not satisfying the Courant condition.

2.15.11 Vectorization of CHARGE

Using the vectorization algorithm of the CURNTV subroutine, we can also vectorize the CHARGE subroutine. Make the vectorized version of the CHARGE subroutine (ex. CHARGV). Run the code with the electrostatic MAIN program, and compare the CPU time with the run with the non-vectorized CHARGE subroutine.

2.16 LIBKEMPO1

To run the KEMPO1, we need various subroutines other than those described above. We prepared the subroutine library named LIBKEMPO1, which consists of 25 subroutines categorized as follows;

- Fast Fourier Transform
  REALFT, FOUR1
- Spectrum Analyses
  WKFFT, RKFFT, SKFFT
- Random Number Generator
  UNRNDM, STRNDM
- Graphics
  QLOOK, QLOOK2, PRMPLT, WKPLOT, XAXIS1, XAXIS2, YAXIS1, YAXIS2, LXAXIS, LYAXIS, ENUMBR, GNUMBR DPLOT
- Miscellaneous
  MAXMIN, RNDOFF, ETRANS, ASCALE, XYROT

The graphic application programs uses the following basic graphic subroutines as external references.

PLOT, SYMBOL, NUMBER, NEWPEN

These are the basic subroutines in the CALCOMP graphic library.

2.17 Installation of KEMPO1

To install the KEMPO1, you have to prepare two libraries on your computer system. One is the LIBKEMPO1 whose source code is distributed with the KEMPO1. The other is the graphic subroutine library which consists of the basic graphic subroutines of the so-called CALCOMP graphics. They are the followings.

- PLOTS : initialize the graphics. We assume the virtual graphic window of 34.5×26.5 mapped onto a graphic output area of a paper or a terminal screen of any size. A position (X, Y) specified by the graphic subroutines should be inside the virtual widow, i.e., $0 \leq X \times F \leq 34.5$, $0 \leq Y \times F \leq 26.5$, where F is the factor specified by the FACTOR subroutine.
- PLOT$(X, Y, IPEN)$: draw lines by moving the pen to a point (X, Y).
- SYMBOL$(X, Y, HIGHT, TEXT, ANGLE, NT)$: draw a set of characters $TEXT$.
- NUMBER$(X, Y, HIGHT, R, ANGLE, NR)$: draw a number R.
- NEWPEN(IC) : change the color or width of the pen. As color graphics, we assume that $IC = 1, 2, 3, 4, 5, 6, 7$ correspond to white, blue, cyan, green, yellow, red, magenta, respectively.
- FACTOR(F) : change the factor of the figures to be drawn after this subroutine.

To control the paging of the graphic outputs, we need the following subroutine

- CHART : control the paging of the graphic outpus. If the graphics is operated on a graphic terminal, this subroutine holds an execution of an application program to wait for an input from the terminal. After the input (this can be just a carriage return), it clears the graphic terminal and resumes the execution of the program.

On most of supercomputer systems, the CALCOMP graphics is available. If not, it is relatively easy to realize the CALCOMP basic graphic subroutines using other graphic libraries on various computer systems. We can make the NUMBER subroutine by calling the SYMBOL subroutine. We can implement the SYMBOL subroutine by calling the PLOT subroutine, if the vector font data are available. If not, we may use the text output subroutine in the graphic library. Then, what is essential is to write the PLOT subroutine to draw lines, which is easily implemented on any graphics systems. An example of the graphic interface subroutines for the X Window System is given in Chapter 8.

Acknowledgments

We thank H. Usui, A. Sawada, H. Kojima, M. Okada, H. Ueda and T. Murata for their supports for the computer system where the KEMPO1 has been developed and for their comments on the the KEMPO1 and the present manuscript. We thank Y. Zhang and L. Borda de Agua for their careful reading and comments on the manuscript. We thank T. Shimizu for reporting a bug in the earlier version of the KEMPO1.

References

Birdsall, C. K., and A. b. Langdon, *Plasma Physics via Computer Simulation*, McGraw-Hill, 1985.

Hockney, R. W., and J. W. Eastwood, *Computer Simulation using particles*, McGraw-Hill, 1981.

Matsumoto, H., and Y. Omura, Particle simulations of electromagnetic waves and their applications to space plasmas, *Computer Simulations of Space Plasmas*, ed. by H. Matsumoto and T. Sato, Terra Pub. and Reidel Co., 1985.

Melrose, D. B., *Instabilities in space and laboratory plasmas*, Cambridge Univ. Press, 1986.

Nicholson, D. R., *Introduction to Plasma Theory*, John Wiley & Sons, 1983.

Ueda, H., Y. Omura, H. Matsumoto, and T. Okuzawa, An analysis on numerical heating of electrostatic particle code, submitted to *Computer Physics Communication*, 1993.

Villasenor, J., and O. Buneman, Rigorous charge conservation for local electromagnetic field solvers, *Computer Physics Communication*, *69*, 306-316, 1992.