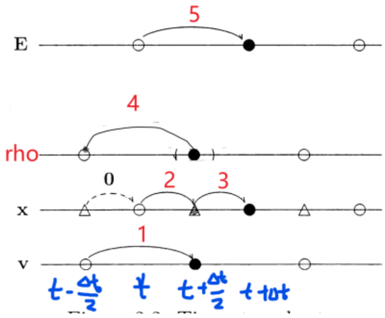
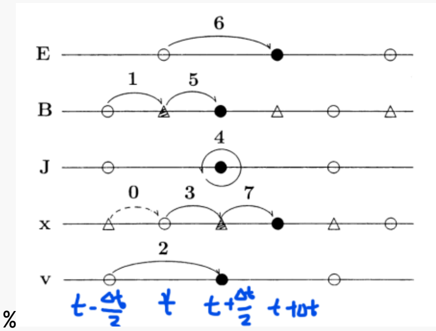
%1 kempo1main.m

```
function kempo1main
    %*************伪随机数************%
    %用于随机生成粒子的初始坐标和相位角
    rng('default');
    rng(1);
    global flag_exit
    flag_exit=0

    %*************读取参数************%
    prm = Parameters

    %*************归一化************%
    [prm,ren] = renorm(prm);

    %*************初始化************%
    %初始化画图设置，粒子坐标和速度等
    [hdiag,output] = diagnostics_init(prm);
    particle = Particle(prm);
    field = Field(prm);

    %初次更新粒子位置和求电荷量、电场分布
    position(particle,prm); %首次更新半个时间步的粒子位置
    if prm.iex              %静电开关 : iex=1 为电磁，iex=2 为静电
       charge(particle, field, prm);  %求网格点电荷密度
       poisson(field, prm);           %通过泊松方程求电场分布
    end

    %***********main loop***********%
    jtime=0;
    jdiag=1;  %诊断计数

    %-- Diagnostics at initial time --
    hdiag=diagnostics(hdiag,particle,field,output,prm,jtime,jdiag,ren);
    if prm.nplot == 0  %nplot: number of output
       return
    end
```

```
    % Time advance loop
   for jtime = 1:prm.ntime   %? 时间间隔 dt 和总时间步数的选择
       if prm.iex==2  %iex=2 静电; iex=1 电磁
```




```
           rvelocity(particle, field, prm);
           position(particle, prm);
           position(particle, prm);  %? 两次位置变化? ?  每次更新半个时间步位置
           charge(particle, field, prm);
           poisson(field, prm);
       else

           %
```




```
           bfield(field,prm);
           rvelocity(particle, field, prm);  %Boris 方法
           position(particle, prm);
           current(particle, field, jtime, prm);
           bfield(field, prm);
           efield(field, prm);
           position(particle, prm);
       end

      %-- 时变诊断 diagnostics --
      if mod(jtime,prm.ifdiag)==0
         jdiag = jdiag+1;
         hdiag = diagnostics(hdiag, particle, field,output, prm, jtime, jdiag,ren);
      end
      if flag_exit
         break;
      end
```

```
    end
    %-- diagnostics --
    if ~flag_exit
        diagnostics_last(hdiag, prm, jtime,output,ren);
    end
end
```

%2 Parameter.m

```
%********读取输入参数*********%
classdef Parameters < handle

    properties
        %网格距和时间步长，需满足库朗条件Δx > cΔt 且小于德拜长度 Δx ≤ 3λ_e
        dx double {mustBePositive}
         dt double {mustBePositive}
        %网格点数
        nx double {mustBePositive}
        %时间步数
        ntime double {mustBeInteger,mustBePositive}
        % number of outputs
        nplot double {mustBeInteger, mustBePositive}
        %光速，程序输入的光速为 20 ？？
        cv double {mustBePositive}
        %粒子的回旋频率
        wc double {mustBeReal}
        %外部电流的振幅 jz
        ajamp double {mustBeNonnegative}
        %电场振幅
        eamp double {mustBePositive}
        %maximum range for plotting the electric field
        emax double {mustBeReal}
        %磁场的振幅
        bamp double {mustBePositive}
        %maximum range for plotting the magnetic field
        bmax double {mustBeReal}
        % control parameter for electrostatic option
        iex double {mustBeMember(iex,[0,1,2])}
        % maximum range for plotting velocity
        vmax double {mustBeReal}
        %number of bins for deriving the particle distribution function
        nv double {mustBeInteger}
        % 外部电流的频率 jz
        wj double {mustBeNonnegative}
```

```
        % 粒子种类的数量
        ns double {mustBeInteger, mustBePositive}
        % number of particles for species
        np double {mustBeInteger, mustBePositive}
        % plasma frequency of species
        wp double {mustBePositive}
        % charge-to-mass ratio of species
        qm double {mustBeReal}
        % parallel thermal velocity of species
        vpa double {mustBePositive}
        % perpendicular thermal velocity of species
        vpe double {mustBePositive}
        % drift velocity of species
        vd double {mustBeNonnegative}
        %漂移速度和水平方向的夹角，满足以下关系（Ø=pch*pi/180）
        %$v_{d\perp} = v_d \sin\phi$
        %$v_{d||} = v_d \cos\phi$
        pch double {mustBeGreaterThanOrEqual(pch,0), mustBeLessThanOrEqual(pch,180)}
        %颜色开关
        icolor
        %画图开关
        iparam
        %诊断方式
        diagtype
        angle  double {mustBeGreaterThanOrEqual(angle,0), mustBeLessThanOrEqual(angle,90)}
    end

% Actually I am not sure if this should be calculated only once. It
% might be better.
% For example, in the initialization part, slx has been used several
% times inside the nested loop. q is used in function charge.
properties (Dependent)
    slx
    npt
    nxp1
    nxp2
    X1
    X2
    X3
    cs
    tcs
    q
```

```
        mass
        rho0
        bx0
        by0
        ifdiag
    end

    methods
        function obj = Parameters(fname)
            % read input parameters and set values

            if nargin==0
                fname = 'input_tmp.dat';   % default input filename
            end

            try
                fid = fopen(fname);
                C = textscan(fid,'%s%s','delimiter','=;','commentstyle','matlab');
                [StrName,StrValue] = C{:};
                fclose(fid);
            catch
                errordlg(sprintf('Can''t open input file: %s',fname),'Error')
             end

             for l=1:length(StrName)
                 value = eval(char(StrValue(l)));
                 prmname = strtrim(StrName{l});
                 switch prmname
                     case 'dx'
                         obj.dx = value;
                     case 'dt'
                         obj.dt = value;
                     case 'nx'
                         obj.nx = value;
                     case 'ntime'
                         obj.ntime = value;
                     case 'nplot'
                         obj.nplot = value;
                     case 'cv'
                         obj.cv = value;
                     case 'wc'
                         obj.wc = value;
                     case 'ajamp'
```

```
                    obj.ajamp = value;
                case 'eamp'
                    obj.eamp = value;
                case 'emax'
                    obj.emax = value;
                case 'bamp'
                    obj.bamp = value;
                case 'bmax'
                    obj.bmax = value;
                case 'iex'
%                     obj.iex = value;
                      obj.iex = 2;
                case 'vmax'
                    obj.vmax = value;
                case 'nv'
                    obj.nv = value;
                case 'wj'
                    obj.wj = value;
                case 'ns'
                    obj.ns = value;
                case 'np'
                    obj.np = value;
                case 'wp'
                    obj.wp = value;
                case 'qm'
                    obj.qm = value;
                case 'vpa'
                    obj.vpa = value;
                case 'vpe'
                    obj.vpe = value;
                case 'vd'
                    obj.vd = value;
                case 'pch'
                    obj.pch = value;
                case 'icolor'
                    obj.icolor = value;
                case 'iparam'
                    obj.iparam = value;
                case 'diagtype'
%                     obj.diagtype = value;
                    obj.diagtype =  [1, 4, 5,  10, 11.000000,15, 25.000000, 3
0.000000];
                case 'angle'
                    obj.angle = value;
```

```
                  otherwise
                      error('Plese check input parameter %s.',prmname)
              end
          end
      end

      % Get the dependent vars

      %网格点数
      function value = get.slx(obj)
          value = obj.nx;
      end

      %总的网格点数
      function value = get.npt(obj)
          %总的粒子数量
          value = sum(obj.np(1:obj.ns));
      end

      %网格点数+1
      function value = get.nxp1(obj)

          value = obj.nx+1;
      end

      %网格点数+2
      function value = get.nxp2(obj)
         value = obj.nx+2;
      end

      %1:128
      function value = get.X1(obj)
         value = 1:obj.nx;
      end

      %2:129
      function value = get.X2(obj)
         value = 2:(obj.nx+1);
      end

      %3:130
      function value = get.X3(obj)
         value = 3:(obj.nx+2);
      end
```

```

        %光速平方
        function value = get.cs(obj)
            value = obj.cv^2;
        end

        %光速平方的 2 倍
        function value = get.tcs(obj)
            value = 2*obj.cs;
        end

        %电荷量
        function value = get.q(obj)
            value = obj.nx ./ obj.np(1:obj.ns) .* (obj.wp(1:obj.ns).^2) ./ ...
                obj.qm(1:obj.ns);
        end

        % This is a case where you have inter-dependency.
        function value = get.mass(obj)
            value = obj.q ./ obj.qm(1:obj.ns);
        end

        %电荷密度
        function value = get.rho0(obj)
            value = -sum(obj.q(1:obj.ns) .* obj.np(1:obj.ns)) / obj.nx *...
                ones(obj.nxp2,1);
        end

        function value = get.bx0(obj)
            theta = pi/180*obj.angle;
            value = obj.wc/obj.qm(1)*cos(theta);
        end

        function value = get.by0(obj)
            theta = pi/180*obj.angle;
            value = obj.wc/obj.qm(1)*sin(theta);
        end

        %每次画的步长数
        function value = get.ifdiag(obj)
            value = ceil(obj.ntime/obj.nplot);
        end
    end
```

```
end
```

%3 renorm.m

```
%**********归一化系数*********%
%ren.*=实际/模拟（归一化指的是网格距归一，其他参数并不归一而是等比例变化）
function [prm,ren]=renorm(prm)
    ren.x=prm.dx                %网格距系数，将网格距归一化为 1
    ren.t=prm.dt/2              %时间步长系数，将时间步统一化为 2
    ren.v=ren.x/ren.t           %速度系数
    ren.e=ren.x/(ren.t^2)       %电场系数
    ren.b=1.0/ren.t             %磁场系数
    ren.j=ren.x/(ren.t^3)       %电流密度系数
    ren.r=1.0/(ren.t^2)         %电荷密度系数
    ren.s=(ren.x^2)/(ren.t^4)%能量密度系数

    prm.cv=prm.cv/ren.v         %等比列变化后的光速
    prm.wc=prm.wc*ren.t         %等比列变化后的回旋频率

    prm.wp=prm.wp   .*ren.t   %等比列变化后的等离子体频率
    prm.vpa=prm.vpa ./ren.v   %等比列变化后的平行速度
    prm.vpe=prm.vpe ./ren.v   %等比列变化后的垂直速度
    prm.vd=prm.vd   ./ren.v   %等比列变化后的漂移速度

    prm.vmax=prm.vmax ./ren.v%等比列变化后的速度上限

    prm.wj=prm.wj*ren.t         %等比列变化后的外部电流的频率 jz
    prm.ajamp=prm.ajamp/ren.j%等比列变化后的外部电流的振幅 ajamp

end
```

%4 Particle.m

```
classdef Particle < handle

     properties
          x double     %粒子坐标
          vx double
          vy double
          vz double
     end

     methods
          function obj=Particle(prm)
```

```
        %粒子初始化
        obj.x =zeros(prm.npt,1);
        obj.vx=zeros(prm.npt,1);
        obj.vy=zeros(prm.npt,1);
        obj.vz=zeros(prm.npt,1);

        n2=0;
        for k=1:prm.ns
            n1=n2;
            n2=n2+prm.np(k);

            phi = pi/180.0*prm.pch(k); %漂移速度与水平方向的夹角
            vdpa = prm.vd(k)*cos(phi); %漂移速度的平行分量
            vdpe = prm.vd(k)*sin(phi); %漂移速度的垂直分量

            xx = 0;
            nphase = 1;
            phase = 0;

            %对每一个粒子进行操作
            for i=(n1+1):n2
                if mod(i,nphase) == 0
                   phase = 2*pi*rand;          %随机粒子的相位
                   xx = xx+ prm.nx/prm.np(k);    %等间距分配每一个粒子坐标
                else
                   phase = phase + 2*pi/nphase;  %随机粒子相位角
                end

                obj.x(i) = xx;                 %初始化粒子坐标
                if obj.x(i) < 0.0
                   obj.x(i) = obj.x(i) + prm.slx;%slx 网格点数=nx
                end
                if obj.x(i) >= prm.slx
                   obj.x(i) = obj.x(i) - prm.slx;%保证粒子坐标在网格内
                end

                uxi = prm.vpa(k)*randn + vdpa;           %平行热速度+平行漂移速度
                uyi = prm.vpe(k)*randn + vdpe*cos(phase);%垂直热速度的 y 分量+平行漂移速度的 y 分量
                uz  = prm.vpe(k)*randn + vdpe*sin(phase);%垂直热速度的 z 分量+平行漂移速度的 z 分量

                % rotation to the direction of the magnetic field
```

```
                    % angle 为静磁场 B0 与波矢 k 的夹角，其中 B0 在 x-y 平面内
                    costh = cos(pi/180*prm.angle); %costh=1
                    sinth = sin(pi/180*prm.angle); %sinth=0
                    ux = costh*uxi - sinth*uyi;    %当 angle=0 时，
                    uy = sinth*uxi + costh*uyi;    %静磁场对速度没偏转

                    %? 抑制初始场的波动(Birdsall and Langdon[1985])
                    g = prm.cv /sqrt(prm.cs + ux*ux + uy*uy + uz*uz);
                    obj.vx(i) = ux*g;
                    obj.vy(i) = uy*g;
                    obj.vz(i) = uz*g;
                end
            end
        end
    end
end
```

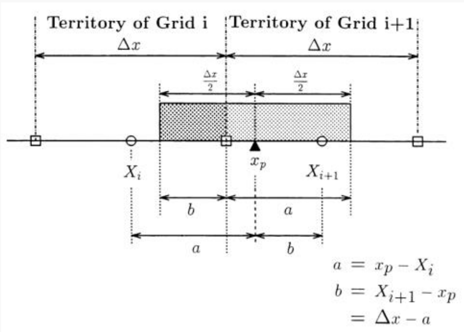
%5 position.m

```
function particle = position(particle,prm)
% Update the position in one step

slx = prm.slx
p = particle

%更新半个时间步的位置，Δt时间内需调用 2 次(注：一个时间步为 2)
% x^{t+\Delta t/2} = x^t + v_x^{t+\Delta t/2}\frac{\Delta t}{2}
% x^{t+\Delta t} = x^{t+\Delta t/2} + v_x^{t+\Delta t/2}\frac{\Delta t}{2}
p.x = p.x + p.vx

% 周期性边界条件：保证粒子在网格内  Periodic BC
p.x = p.x + slx.*(p.x<0.0) - slx.*(p.x>=slx)
end
```

%6 charge.m

```
function charge(particle,field,prm)
% Calculate the charge on the grid from particles

p = particle; % reference to the particle obj

% Field.rho = zeros(prm.nxp2,1);
```

```
field.rho = prm.rho0; % $w_{pi} = \sqrt{\frac{n_i q_i^2}{m_i}}$  ->  $\rho_i = \frac{w_{pi}^2}{q_i/m_i}$

n2 = 0;
for k=1:prm.ns
    n1 = n2;
    n2 = n1 + prm.np(k);
    for m = (n1+1):n2
        i  = floor(p.x(m) + 2.0);           %每个粒子所在的网格点坐标
        i1 = i + 1;                         %每个粒子坐标的下一个网格点坐标
        s2 = (p.x(m)+ 2.0 - i)*prm.q(k);    %分到右侧网格点的电荷密度
        s1 = prm.q(k) - s2;                 %分配到左侧网格点的电荷量
```




```
        field.rho(i ) = field.rho(i ) + s1; %累加分配到左侧网格的电荷量
        field.rho(i1) = field.rho(i1) + s2; %累加分配到右侧网格的电荷量
        %? 这里分配的电荷密度是否归一化 % $w_{pi} = \sqrt{\frac{n_i q_i^2}{m_i}}$  ->  $q_i^2 = \frac{w_{pi}^2 m_i}{n_i}$

 end
end

%电荷密度边界条件
field.rho(2) = field.rho(2) + field.rho(prm.nxp2) - prm.rho0(2);%?
field.rho(1) = field.rho(prm.nxp1);
field.rho(prm.nxp2) = field.rho(2);


end
```

%7 poisson.m

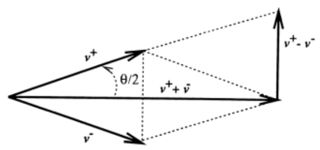
```
function poisson(field, prm)
    % Calculate Ex from Poisson equation

    f = field; % reference to the Field obj

```

```
    %泊松方程 $\frac{\partial E_x}{\partial x}=\rho$
    %差分形式 $E_{x,i+1/2}-E_{x,i-1/2}=\rho_i$
    %这里定义真空介电常数等于 1
    %$\varepsilon_0\mu_0=\frac{1}{c^2}$, $\varepsilon_0=1$, $\mu_0=\frac{1}{c^2}$
    f.ex(prm.X2) = f.ex(prm.X2-1) + f.rho(prm.X2);

   ex0 = sum(f.ex(prm.X2))/prm.nx      %所有网格点的平均电场强度
   f.ex(prm.X2) = f.ex(prm.X2) - ex0;  %？？？
   f.ex(1) = f.ex(prm.nxp1);           %对称边界条件 nxp1=nx+1
   f.ex(prm.nxp2) = f.ex(2);           %对称边界条件 nxp2=nx+2
end
```

%8 rvelocity.m

```
function rvelocity(particle, field, prm)
% Update velocity in one step

% References to class obj
p = particle;
f = field;

nxp1 = prm.nxp1; nxp2 = prm.nxp2; ns = prm.ns; np = prm.np;
qm = prm.qm;
bx0 = prm.bx0;  %bx0=prm.wc/prm.qm(1)*cos(theta);
cs = prm.cs;
X1 = prm.X1; X2 = prm.X2; X3 = prm.X3; %X2=2:(prm.nx+1);X3=3:(prm.nx+2)

%********通过 Boris 方法更新粒子速度*******%
%牛顿第二定律 $\frac{d\boldsymbol{v}}{dt}=\frac{q_s}{m_s}(\boldsymbol{E}+\boldsymbol{v}\times\boldsymbol{B})$
%差分形式 $\frac{\boldsymbol{v}^{t+\Delta t/2}-\boldsymbol{v}^{t-\Delta t/2}}{\Delta t}=\frac{q_s}{m_s}(\boldsymbol{E}^t+\frac{\boldsymbol{v}^{t+\Delta t/2}+\boldsymbol{v}^{t-\Delta/2}}{2}\times\boldsymbol{B}^t)$
%定义 $\boldsymbol{v}^-=\boldsymbol{v}^{t-\Delta t/2}+\frac{q_s}{m_s}\boldsymbol{E}^t\frac{\Delta t}{2}$
%定义 $\boldsymbol{v}^+=\boldsymbol{v}^{t+\Delta t/2}-\frac{q_s}{m_s}\boldsymbol{E}^t\frac{\Delta t}{2}$
%将$v^+$和$v^-$代入差分形式得 $\frac{\boldsymbol{v}^+-\boldsymbol{v}^-}{\Delta t}=\frac{1}{2}\frac{q_s}{m_s}(\boldsymbol{v}^++\boldsymbol{v}^-)\times\boldsymbol{B}^t$
%两边同乘以$(v^+-v^-)$得 $(\boldsymbol{v}^+)^2=(\boldsymbol{v}^-)^2$
```

%Boris 方法矢量关系图



%由矢量可以求得夹角θ，即

$$\theta = -2\tan^{-1}\left(\frac{\Delta t}{2}\frac{q_s}{m_s}B^t\right)$$

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

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

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

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

%

```
work1 = zeros(nxp2,1);
work2 = zeros(nxp2,1);
work1(X2) = 0.5*(f.ex(X1) + f.ex(X2));
work1(nxp2)= work1(2);
work2(X2) = 0.5*(f.by(X3) + f.by(X2));
work2(1) = work2(nxp1);

%-----
n2 = 0;
for k=1:ns
   n1 = n2;
   n2 = n2 + np(k);

   for m=(n1+1):n2
      i = floor(p.x(m) + 2.0);
      sf2 = (p.x(m) + 2.0 - i)*qm(k);
      sf1 = qm(k) - sf2;

      ih = floor(p.x(m) + 1.5);
      sh2 = (p.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*f.ey(i)  + sf2*f.ey( i1);
      ez1 = sh1*f.ez(ih) + sh2*f.ez(ih1);
```

```

        bx1 = bx0*prm.qm(k);
        by1 = sh1*work2(ih) + sh2*work2(ih1);
        bz1 = sh1*f.bz(ih)  + sh2*f.bz(ih1);

        g = prm.cv / sqrt(cs - p.vx(m)^2 - p.vy(m)^2 - p.vz(m)^2);

        ux = p.vx(m)*g + ex1;%?
        uy = p.vy(m)*g + ey1;
        uz = p.vz(m)*g + ez1;

        g = prm.cv/sqrt(cs + ux*ux + uy*uy + uz*uz);

        bx1 = bx1*g;
        by1 = by1*g;
        bz1 = bz1*g;

        boris = 2.0/(1+ bx1*bx1 + by1*by1 + bz1*bz1);

        uxt = ux + uy*bz1 - uz*by1;
        uyt = uy + uz*bx1 - ux*bz1;
        uzt = uz + ux*by1 - uy*bx1;

        ux = ux + boris*(uyt*bz1 - uzt*by1) + ex1;
        uy = uy + boris*(uzt*bx1 - uxt*bz1) + ey1;
        uz = uz + boris*(uxt*by1 - uyt*bx1) + ez1;

        g = prm.cv /sqrt(cs + ux*ux + uy*uy + uz*uz);

        % this is causing speed issue. Why?
        p.vx(m) = ux*g;
        p.vy(m) = uy*g;
        p.vz(m) = uz*g;
    end
end

end
```

%9 bfield.m

```
function [field] = bfield(field,prm)
    % Update magnetic field in one step
    X2 = prm.X2;  %2:129
    f = field; % reference to Field obj
```

```

    % $\frac{\partial \boldsymbol{B}}{\partial t} = -\nabla \times \boldsymbol{E}$ , 标量形式如下：$\nabla \times \boldsymbol{E} = \left(\frac{\partial E_z}{\partial y} - \frac{\partial E_y}{\partial z}\right)\boldsymbol{i} + \left(\frac{\partial E_x}{\partial z} - \frac{\partial E_z}{\partial x}\right)\boldsymbol{j} + \left(\frac{\partial E_y}{\partial x} - \frac{\partial E_x}{\partial y}\right)\boldsymbol{k}$
    % $\frac{\partial B_y}{\partial t} = \frac{\partial E_z}{\partial x}$
    % $\frac{\partial B_z}{\partial t} = -\frac{\partial E_y}{\partial x}$
    f.by(X2) = f.by(X2) + f.ez(X2) - f.ez(X2-1);
    f.bz(X2) = f.bz(X2) - f.ey(X2+1) + f.ey(X2  );  %?

    f.by(prm.nxp2)= f.by(2);%对称边界条件
    f.bz(prm.nxp2)= f.bz(2);
    f.by(1)    = f.by(prm.nxp1);
    f.bz(1)    = f.bz(prm.nxp1);
end
```

%10 current.m

```
function current(particle, field, jtime, prm)
% Calculate the current in one step

nx = prm.nx; nxp1 = prm.nxp1; nxp2 = prm.nxp2;
X2 = prm.X2;
np = prm.np; ns = prm.ns;
q = prm.q;

% references to class obj
f = field;
p = particle;

f.ajx = zeros(nxp2,1);
f.ajy = zeros(nxp2,1);
f.ajz = zeros(nxp2,1);

%----
n2 = 0;
for k=1:ns
   n1 = n2;
   n2 = n2 + np(k);
   qh = q(k)*0.5;

   for m=(n1+1):n2
      ih = floor( p.x(m) + 1.5 );
      s2 = (p.x(m) + 1.5 - ih)*q(k);
```

```
        s1 = q(k) - s2;
        ih1= ih+1;
        %jy 和 jz 按线性比重分配到相邻网格点
        f.ajy(ih)  = f.ajy(ih)  + p.vy(m)*s1;
        f.ajy(ih1) = f.ajy(ih1) + p.vy(m)*s2;
        f.ajz(ih)  = f.ajz(ih)  + p.vz(m)*s1;
        f.ajz(ih1) = f.ajz(ih1) + p.vz(m)*s2;

        %--Jx 采用电荷守恒方法–
        % \frac{\partial \rho}{\partial t} + \nabla \cdot \boldsymbol{J} = 0
        % \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}
        if prm.iex
            qhs = qh * sign(p.vx(m));
            avx = abs(p.vx(m));

            x1 = p.x(m) + 2.0 - avx;
            x2 = p.x(m) + 2.0 + avx;
            i1 = floor(x1);
            i2 = floor(x2);

            % This is causing me speed issue
            f.ajx(i1) = f.ajx(i1) + (i2 - x1)*qhs;
            f.ajx(i2) = f.ajx(i2) + (x2 - i2)*qhs;
        end
    end
end

%-- boundary --
f.ajx(nxp1) = f.ajx(1) + f.ajx(nxp1);
f.ajx(2)    = f.ajx(2) + f.ajx(nxp2);

f.ajy(nxp1) = f.ajy(1) + f.ajy(nxp1);
f.ajy(2)    = f.ajy(2) + f.ajy(nxp2);
f.ajy(1)    = f.ajy(nxp1);

f.ajz(nxp1) = f.ajz(1) + f.ajz(nxp1);
f.ajz(2)    = f.ajz(2) + f.ajz(nxp2);

%--
f.ajy(X2) = 0.5*(f.ajy(X2) + f.ajy(X2-1));

%-- external current source ----
```

```
if prm.ajamp
    f.ajz(nx/2+1) = f.ajz(nx/2+1) + prm.ajamp*sin(prm.wj*jtime);
else
    %--cancellation of uniform Jx,Jy,Jz components---
    ajxu = sum(f.ajx(X2))/nx;
    ajyu = sum(f.ajy(X2))/nx;
    ajzu = sum(f.ajz(X2))/nx;
    f.ajx(X2) = f.ajx(X2) - ajxu;
    f.ajy(X2) = f.ajy(X2) - ajyu;
    f.ajz(X2) = f.ajz(X2) - ajzu;
end

end
```

%11 efield.m

```
function [field] = efield(field,prm)
% Update the electric field in one step


nxp1 = prm.nxp1; nxp2 = prm.nxp2;
tcs = prm.tcs;                %2*obj.cs;
X1 = prm.X1; X2 = prm.X2; X3 = prm.X3;

f = field; % reference to class obj

```

% $\frac{\partial \boldsymbol{E}}{\partial t} = c^2 \nabla \times \boldsymbol{B} - \boldsymbol{J}$ %$\varepsilon_0 \mu_0 = \frac{1}{c^2}$, $\boldsymbol{\varepsilon_0} = \mathbf{1}$, $\mu_0 = \frac{1}{C^2}$

% $\frac{\partial E_x}{\partial t} = -J_x$,标量形式如下

% $\frac{\partial E_x}{\partial t} = -J_x$

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

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

```

if prm.iex == 0  %iex==0:没有电场
    f.ex(:) = 0;
else
    f.ex(X2) = f.ex(X2) - 2*f.ajx(X2);
    f.ex(1) = f.ex(nxp1);
    f.ex(nxp2) = f.ex(2);
end

```

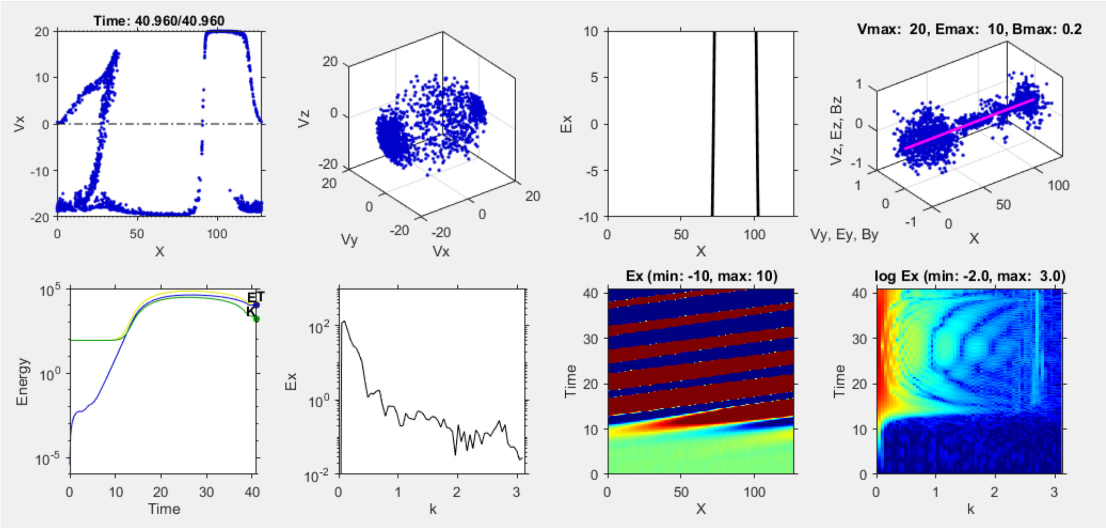
```
f.ey(X2) = f.ey(X2) - tcs*(f.bz(X2) - f.bz(X1)) - 2*f.ajy(X2);
f.ez(X2) = f.ez(X2) + tcs*(f.by(X3) - f.by(X2)) - 2*f.ajz(X2);

f.ey(1) = f.ey(nxp1);
f.ez(1) = f.ez(nxp1);
f.ey(nxp2) = f.ey(2);
f.ez(nxp2) = f.ez(2);

end
```

%12 Inputs and Outputs

%12.1 Inputs

```
dx = 1.000000;
dt = 0.040000;
nx = 128.000000;
ntime = 1024.000000;
cv = 20.000000;
wc = 0.000000;
angle = 0.000000;

ns = 1.000000;
np = [2048.000000, ];
wp = [1.000000, ];
qm = [-1.000000, ];
vpa = [0.500000, ];
vpe = [10.000000, ];
vd = [0.000000, ];
pch = [0.000000, ];

iex = 2;

ajamp = 0.000000;
wj = 0.000000;

nplot = 256.000000;
nv = 100.000000;
icolor = 1.000000;
iparam = 1.000000;
vmax = 20.000000;
emax = 10.000000;
bmax = 0.200000;
diagtype = [11.000000, 23.000000, 18.000000, 4.000000, ];
```

%12.2 Outputs(iex=2)



%12.3 Outputs(iex=1)