IBSimu Particle Diagnostic

Working Draft

Duccio Marco Gasparri

2020-11-30

1 variables

- m [kg] particle mass (provided in u)
- q [J] charge of beam particle (provided in multiples of e)
- J [A/m$\hat{2}$] beam current density
- E [J] mean energy (provided in eV)
- Tp [J] parallel temperature (provided in eV)
- Tt [J] transverse temperature (provided in eV)
- $(x_1, r_1), (x_2, r_2)$ [m] beam emission line vectors
- N number of particles
- IQ [A] (A/m?) particle current
- v [m/s] from E and m

2 Setup

2.1 Geometry

Relevant IBSimu files:

- geometry.hpp
- geometry.cpp
- mesh.hpp

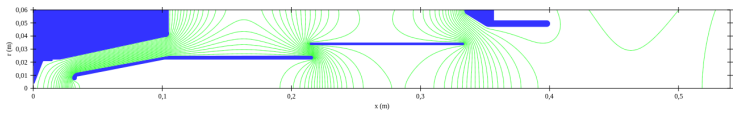

Figure 1: Geometry DXF starting at -10e-3 m. Setting origin-x = 0 mm

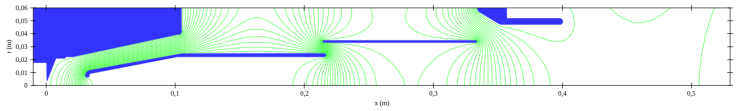

Figure 2: Geometry DXF starting at -10e-3 m. Setting origin-x = -10 mm

- mesh.cpp

Relevant client files:

- configuration-setup.hpp
- configuration-setup.cpp

Parameters:

- mesh-cell-size-h [eg. 0.5e-4]
- origin-x
- origin-y
- origin-z
- geometry-start-x [UNUSED]
- geometry-start-y [UNUSED]
- geometry-start-z [UNUSED]
- geometry-size-x
- geometry-size-y
- geometry-size-z

The class Geometry creates the mesh of the simulation starting from the CAD files and/or function definitions.

The parameters origin-x, origin-y and origin-z are used to move the reference frame relative to the bbox in the DXF file [see mesh.hpp line 70-73, mesh.cpp line 60].

IBSimu does not render the space before the origin-x, so also moving the geomtery with the mouse would not show the part before the origin-x.

NB: Non si capisce perché non calcoli il campo elettrico prima dello zero della geometria. Avevo la geometria traslata a -10mm, con zero la punta dell'elettrodo di estrazione. Tra [-10mm;0] l'epot era = a Vplasma

The physical length/height/depth of the simulation is size-x/y/z * mesh-cell-size. Specifically:

$$size - x = (int)floor(geometry - size - x/mesh - cell - size - h) + 1$$

$$x_{max} = origin - x + mesh - cell - size - h * size - x$$

3 ParticleDataBaseCylImp::add_2d_beam_with_energy

Function ParticleDataBaseCylImp::add_2d_beam_with_energy (file: *particle-databaseimp.cpp*, line: 968) is used to add a beam of N particles with average energy E to a cylindrical geometry.

The charge q is provided by the user and is set constant for all the particles.

The beam emission line norm s [m] is defined:

$$s = \sqrt{(x_2 - x_1)^2 + (r_2 - r_1)^2} \tag{1}$$

The current IQ [A] is set for each particle as follows:

$$IQ = \frac{2\pi sJ}{N}(r_1 + \frac{(r_2 - r_1)}{N}(n + 0.5)) \tag{2}$$

where $n \in [0, 1, ..., N - 1]$.

The particles are distributed evenly spaced along the emission line defined by the vectors $(x_1, r_1), (x_2, r_2)$. The particle velocities v_x, v_r [m/s] are:

$$v_x = \frac{(x_2 - x_1)}{s}\sqrt{\frac{Tt}{m}}rnd_0 + \frac{(r_2 - r_1)}{s}\sqrt{\frac{2E}{m} + (\sqrt{\frac{Tp}{m}}rnd_1)^2} \tag{3}$$

$$v_r = \frac{(r_2 - r_1)}{s}\sqrt{\frac{Tt}{m}}rnd_0 + \frac{-(x_2 - x_1)}{s}\sqrt{\frac{2E}{m} + (\sqrt{\frac{Tp}{m}}rnd_1)^2} \tag{4}$$

and

$$w = \frac{d\theta}{dt} = \frac{\sqrt{\frac{Tt}{m}} rnd_2}{r_1 + \frac{(r_2 - r_1)}{N}(n + 0.5)} \tag{5}$$

with rnd_0, rnd_1 and rnd_2 normally distributed random variables.

4 Iteration

The method ParticleDataBasePPImp::iterate_trajectories (inherits ParticleDataBaseImp, file: particledatabaseimp.hpp::641) outputs the message "Using non-relativistic iterator", clears the scharge, creates a iterators vector of size equal to the set thread, populates it with new instances of ParticleIterator¡ PP¿ and waits for the iteration to be finished. Then calls scharge_finalize_linear (file:scharge.cpp:149) Finally it publishes the particle hisories ("Particle histories").

scharge_finalize_liear prints "Finalizing space charge density map (LINEAR method)"

The core of the iteration isinside the ParticleIterator¡PP¿

The method ParticleDataBasePPImp::iterate_trajectories (inherits ParticleDataBaseImp, file: particledatabaseimp.hpp::641) outputs the message "Using non-relativistic iterator", clears the scharge, creates a iterators vector of size equal to the set thread, populates it with new instances of ParticleIterator¡PP¿ and waits for the iteration to be finished. Then calls scharge_finalize_linear (file:scharge.cpp:149) Finally it publishes the particle hisories ("Particle histories").

scharge_finalize_liear prints "Finalizing space charge density map (LINEAR method)"

The core of the iteration isinside the ParticleIterator¡PP¿

Scheduler class

Scheduler is templated with Solver, Problem and Error classes.

From ParticleDatabasePPImp (particledatabaseimp.hpp:291):

std::vector¡Particle¡PP¿ *¿ _particles; Scheduler¡ParticleIterator¡PP¿ (as Solver), Particle¡PP¿ (as Problem), Error (as Error)¿ _scheduler;

Scheduler class requires the Solver class ParticleIterator¡PP¿ to provide an operator

void ParticleIterator¡PP¿::operator()(Problem *p, int32_t pi)

or, as implemented, Problem is Particle¡PP¿

void ParticleIterator¡PP¿::operator()(Particle¡PP¿ *particle, uint32_t pi)

to solve problem p with index location pi. (particleiterator.hpp:1258)

The operator calls the PP::get_derivatives(0.0, &x[1], dxdt, (void *)&_pidata); function to fill dxdt (particleiterator.hpp:1298).

double dxdt[PP::size()-1];

For the ParticlePCyl::get_derivatives(...) function, documentation reports:

"Returns time derivatives dxdt of coordinates at time t and coordinates x = (x,vx,r,vr,w) for one particle.

The calculation of particle trajectory is done by integrating the Lorentz equation in a form of a set of ordinary differential equations. In the case of cylindrical coordinates the set is:

$$\frac{dx}{dt} = v_x$$

$$\frac{dr}{dt} = v_r$$

$$\frac{dv_x}{dt} = a_x = \frac{q}{m}(E_x + v_r B_\theta - v_\theta B_r)$$

$$\frac{dv_r}{dt} = a_r + r\left(\frac{d\theta}{d}\right)^2 = \frac{q}{m}(E_y + v_\theta Bx - v_x B_\theta) + r\left(\frac{d\theta}{d}\right)^2$$

$$\frac{d^2\theta}{dt^2} = \frac{1}{r}\left(a_\theta - \frac{dr}{dt}\frac{d\theta}{dt}\right) = \frac{1}{r}\left(\frac{q}{m}(v_x B_r - v_r B_x) - 2\frac{dr}{dt}\frac{d\theta}{dt}\right)$$

where

$$v_\theta = r\frac{d\theta}{dt}$$

End quote. Source: http://ibsimu.sourceforge.net/manual_1_0_6/classParticlePCyl.html#adb44dff844a1af2615727a1920c18c8b

, then

double dt = ParticleIterator¡PP¿calculate_dt(x, dxdt);

to get initial dt.

Then the GSL odeiv functions are called to advance the particle, and the result is checked with the ParticleIterator¡PP¿::handle_trajectory() function (particleiterator.hpp:1361)

The bool ParticleIterator¡PP¿::handle_trajectory() function searches mesh intersections between points x1 and x2 and builds ColData. Checks for collisions with solids and boundaries and sets space charge at each mesh line crossing.

Error class has to have a default constructor. Scheduler catches the errors of this type from the working threads and saves caught errors in a container. If an error is caught, all the working threads are interrupted and problem solving is finished. The scheduler does indicate the error state by returning false from finish(). Error state can also be queried with is_error(). Errors can be fetched from the internal containers with get_errors(). The threads can add problems. The problem vector is a shared resource so it must be protected with lock_mutex() and unlock_mutex().

Parallel processing is started with run() function and the end of processing can be waited with finish().

5 Particle Diagnostic

The relevant functions are in files gtkparticlediagdialog.cpp (the GTK dialog file) and particlediagplot.cpp (does the actual plotting). It has the following methods:

- ParticleDiagPlot::build_data(): the function extracts the data from the ParticleDatabase
- ParticleDiagPlot::build_plot(): calls build_data(). the function extracts the data from the ParticleDatabase, set the decorations and add the graph to the _frame

Trajectories are obtained from ParticleDataBase::trajectories_at_plane() that calls ParticleDataBaseImp::trajectories_at_plane(). It requires the axis (X,

Y, Z, R) and the distance from origin. It returns a vector of particle point objects ParticleP¡PP¿ (either ParticleP2D, ParticlePCyl, or ParticleP3D), each particle carrying its information about position and momentum

q is the direction normal to the diagnostic plane

DIAG_X: x position [m] DIAG_R: r position [m]

DIAG_RP: $\frac{v_r}{v_q}$

DIAG_AP: $\frac{v_\theta}{v_q}$

$$v_\theta = r\frac{d\theta}{dt}$$

Emittance:

- X axis, r-r': DIAG_R (r position), DIAG_RP ($\frac{v_r}{v_q}$)
- X axis, r-a': DIAG_R (r position), DIAG_AP ($\frac{v_\theta}{v_q}$)
- X axis, z-z': using DIAG_R, DIAG_RP, DIAG_AP and EmittanceConv class instead

The diagnostic data are obtained from the Emittance class. For Cylindrical coordinates, the class EmittanceConv converts to (x,x') equivalent emittance from the following data:

- r (radius)
- rp (radial angle)
- ap (skew angle)
- I (current)

It builds (x,x') data in a grid array of size n by m. Here the skew angle is $r\omega/v_z$, where v_z is the velocity to the direction of beam propagation.

This description from class reference list is not clear:

The conversion is done by rotating each trajectory diagnostic point around the axis in rotn steps (defaults to 100). The output grid size can be forced by setting (xmin,xpmin,xmax,xpmax) variables, otherwise the grid is autotomatically sized to fit all data.

The emittance statistics is built using original data and not the gridded data for maximized precision.

6 ToDo

Particle Types

template¡class PP¿ class Particle { std::vector¡PP¿ _trajectory; ¡- trajectories PP _x; ¡- current position } Typedef Particle¡ParticleP2D¿ Particle2D; Typedef Particle¡ParticlePCyl¿ ParticleCyl; Typedef Particle¡ParticleP3D¿ Particle3D;

Particle Types in Beam

ParticleDataBaseCylImp::add_*_beam -¿ ParticlePCyl-¿ParticleCyl ParticleDataBase2DImp::add_*_beam-¿ParticleP2D-¿Particle2D ParticleDataBase3DImp::add_*_beam-¿ParticleP3D-¿Particle3D

ParticleDataBasePPImp¡ PP ¿::trajectories_at_plane

Int ParticleP2D::trajectory_intersections_at_plane (NON CONST std::vector¡ ParticleP2D ¿ & intsc) Return the number of trajectory intersections with plane Intersection points are appended to vector intsc. Int TrajectoryRep1D::solve() Returns solutions found [Linear [0,1], Quadratic[0,1,2], Cubic [0,1,2,3]]

IBSimu DXF bug in mydxffile.cpp #define CODE_STRING(x) code (x) == 101 is not included as it is a later standard in DXF file.