The particle diagnostic in SMILEI

F. Pérez

March 19, 2015

The *particle diagnostic* collects data from the macro-particles and processes them during runtime. It does not provide information on individual particles: instead, it produces averaged quantities like the particle density, currents, etc.

The data may be collected from one or several particle species.

The data is discretized inside a "grid" chosen by the user. This grid may be of any dimension.
Examples:

- 1-dimensional grid along the position x (gives density variation along x)
- 2-dimensional grid along positions x and y (gives density map)
- 1-dimensional grid along the velocity v_x (gives the velocity distribution)
- 2-dimensional grid along position x and momentum p_x (gives the phase-space)
- 1-dimensional grid along the kinetic energy E_{kin} (gives the energy distribution)
- 3-dimensional grid along x, y and E_{kin} (gives the density map for several energies)
- 1-dimensional grid along the charge $Z^\star$ (gives the charge distribution)

Each dimension of the grid is called "axis".

The user may choose to average the data over several time-steps.

1 How to add a particle diagnostic in the input file

In the input file, you can add a particle diagnostic with the following syntax.

```
diagnostic particles
        output = output
        every = every
        time_average = time_average
        species = species1 species2 ...
        axis = type min max nsteps [logscale] [edge_inclusive]
        axis = type min max nsteps [logscale] [edge_inclusive]
        ...
end
```

- *output* must be set to `density`, `charge_density`, `current_density_x`, `current_density_y`, `current_density_z`. It determines the data that is summed in each cell of the grid. In the case of `density`, the *weights* are summed. In the case of `charge_density`, the *weights*×*charge* are summed. In the case of `current_density_x`, the *weights*×*charge*×v_x are summed (etc.)

- *every* must be a positive integer. It is the number of time-steps between each output.

- *time_average* must be a positive integer. It is the number of time-steps during which the data is averaged before output.

- *species1 species2 ...* must be one or several species. Species are recognized by their parameter `species_type` that is included in each group `species ... end`

- `axis` is an argument that describes one axis of the grid.
  Syntax: `axis = type min max nsteps [logscale] [edge_inclusive]`
  type can be `x`, `y`, `z`, `px`, `py`, `pz`, `p`, `gamma`, `ekin`, `vx`, `vy`, `vz`, `v` or `charge`.
  The axis is discretized for *type* from *min* to *max* in *nsteps* bins.
  The optional keyword `logscale` sets the axis scale to logarithmic instead of linear.
  The optional keyword `edge_inclusive` includes the particles outside the range [*min*, *max*] into the extrema bins.

There may be as many `axis` arguments as wanted in one `diagnostic particles` block.

There may be as many `diagnostic particles` blocks as wanted in the input file.

In the input file, you may add the following block as a reminder.

```
# DIAGNOSTICS ON PARTICLES - project the particles on a N-D arbitrary grid
# ----------------------------------------------------------------------
# output = density, charge_density or current_density_[xyz]
#              => parameter that describes what quantity is obtained
# every        => integer > 0 : number of time-steps between each output
# time_average => integer > 0 : number of time-steps to average
# species      => list of one or several species whose data will be used
# axis   = _type_ _min_ _max_ _nsteps_ [logscale] [edge_inclusive]
#              => _type_ can be one of the following:
#                  x, y, z, px, py, pz, p, gamma, ekin, vx, vy, vz, v or charge
#              => the data is discretized for _type_ between _min_ and
#                  _max_, in _nsteps_ bins
#              => the optional [logscale] sets the scale to logarithmic
#              => the optional [edge_inclusive] forces the particles
#                  outside (_min_,_max_) to be counted in the extrema bins
#   example : axis = x 0 1 30
#   example : axis = px -1 1 100
# >>>> MANY AXES CAN BE ADDED IN A SINGLE DIAGNOSTIC <<<<
```

2 Examples

Variation of the density of species `electron1` from $x = 0$ to 1, every 5 time-steps, without time-averaging.

```
diagnostic particles
        output = density
        every = 5
        time_average = 1
        species = electron1
        axis = x    0    1    30
end
```

Density map from $x = 0$ to 1, $y = 0$ to 1.

```
diagnostic particles
        output = density
        every = 5
        time_average = 1
        species = electron1
        axis = x    0    1    30
        axis = y    0    1    30
end
```

Velocity distribution from $v_x = -0.1$ to 0.1.

```
diagnostic particles
        output = density
        every = 5
        time_average = 1
        species = electron1
        axis = vx    -0.1    0.1    100
end
```

Phase space from $x = 0$ to 1 and from $px = -1$ to 1.

```
diagnostic particles
        output = density
        every = 5
        time_average = 1
        species = electron1
        axis = x    0    1    30
        axis = px    -1    1    100
end
```

Energy distribution from 0.01 to 1 MeV in logarithmic scale. Note that the input units are $m_e c^2 \sim 0.5$ MeV.

```
diagnostic particles
        output = density
        every = 5
        time_average = 1
        species = electron1
        axis = ekin    0.02    2   100  logscale
end
```

x-y density maps for three bands of energy: $[0, 1]$, $[1, 2]$, $[2, \infty]$. Note the use of `edge_inclusive` to reach energies up to ∞.

```
diagnostic particles
        output = density
        every = 5
        time_average = 1
        species = electron1
        axis = x    0    1    30
        axis = y    0    1    30
        axis = ekin    0   6   3  edge_inclusive
end
```

Charge distribution from $Z^\star = 0$ to 10.

```
diagnostic particles
        output = density
        every = 5
        time_average = 1
        species = electron1
        axis = charge    -0.5   10.5   11
end
```

3 How to view and post-process a particle diagnostic

Each diagnostic produces one file like *ParticleDiagnostic0.h5*, *ParticleDiagnostic1.h5*, etc. They are numbered the same way as the order of appearance in the input file.

A python script *ParticleDiagnostic.py* is provided to view or extract data from these files. To run this script, you will need *python2.7* with the following packages: numpy, matplotlib, pylab, h5py.

First, run python and include the script:

```
python -i scripts/ParticleDiagnostic.py
```

(Alternately, you can include this file into your own script using, for example, `execfile("scripts/ParticleDiagnostic.py")`)

To prepare the data:

```
ParticleDiagnostic(results_path, diagNumber=None, timesteps=None, slice=None,
                   units="code", data_log=False)
```

- `results_path` = `_string_`
  Path to the directory where the outputs are stored.
  (Also, this has to contain one and only one input file *.in)

- `diagNumber` = `_int_` (optional)
  Number of the diagnostic. The first diagnostic has number 0.
  If not given, then a list of available diagnostics is printed.

- `timesteps` = `_int_` (optional)
  `timesteps` = `[_int_, _int_]` (optional)
  If omitted, all timesteps are used.
  If one number given, the nearest timestep available is used.
  If two numbers given, all the timesteps in between are used.

- `slice` = { *axis* : `"all", ...` } (optional)
  `slice` = { *axis* : `_double_, ...` } (optional)
  `slice` = { *axis* : `[_double_, _double_], ...` } (optional)
  This parameter is used to reduce the number of dimensions of the array.
  axis must be `"x"`, `"y"`, `"z"`, `"px"`, `"py"`, `"pz"`, `"p"`, `"gamma"`, `"ekin"`, `"vx"`, `"vy"`, `"vz"`, `"v"` or `"charge"`.
  Any axis of the same name will be removed with the following technique:
  - If the value is `"all"`, then a sum is performed over all the axis.
  - If the value is `_double_`, then only the bin closest to the value is kept.
  - If the value is `[_double_,_double_]`, then a sum is performed between the two values.
  Example: `{"x":[4,5]}` will sum all the data for x in the range [4,5].

- `units` = `"nice"` (optional)
  If `"nice"` is chosen, then units are converted into usual units.
  Distances in microns, density in cm^{-3}, energy in MeV.

- `data_log` = `True` or (`False`) (optional)
  If True, then $\log 10$ is applied to the output array.

To obtain the data as an array:

```
ParticleDiagnostic( ... ).getData()
```

This method returns only the data array.

```
ParticleDiagnostic( ... ).get()
```

This method returns the results as a python dictionary. The dictionary is of the form `{"data":data_array, axis1:axis1_array, axis2:axis2_array, ...}` where:

- *data_array* is the final array containing the data.
- *axis1* and *axis2* are the names of the axes (for example `"x"` or `"px"`).
- *axis1_array* and *axis2_array* are the locations of the axes bins.

Example:

```
>>> result = ParticleDiagnostic("path/to/my/results", diagNumber=3,
    slice={"ekin":[1,10]}, timesteps=1000).get()
```

This will take the diagnostic #3 for the timestep nearest to 1000, and sum for all energies between 1 and 10. The results are stored in the variable `result`.
The data can be accessed with `result["data"]`.
If one of the axes is *x*, you can access the locations of the bins with `result["x"]`.

To plot the data:

```
ParticleDiagnostic( ... , figure=1, data_min=None, data_max=None,
                          xmin=None, xmax=None, ymin=None, ymax=None ).plot()
```

- `figure` = `_int_` (optional)
  The figure number that is passed to matplotlib.
  If absent, figure 1 is used.
- `data_min` = `_double_` (optional)
  `data_max` = `_double_` (optional)
  If present, output is rescaled before plotting.
- `xmin` = `_double_` (optional)
  `xmax` = `_double_` (optional)
  `ymin` = `_double_` (optional)
  `ymax` = `_double_` (optional)
  If present, axes are rescaled before plotting.

Example:

```
>>> ParticleDiagnostic("path/to/my/results", diagNumber=1, slice={"y":"all"},
    units="nice").plot(figure=1,data_min=0, data_max=3e14)
```

This will take the diagnostic #1, and sum the grid over all the y axis. Then it will plot the resulting array in figure 1 from 0 to 3e14.

To simultaneously plot multiple diagnostics in the same figure:

```
multiPlot(diag1, diag2, ... , figure=1, shape=None)
```

- `diag1` = diagnostic prepared by `ParticleDiagnostic(...)`
  `diag2` = diagnostic prepared by `ParticleDiagnostic(...)`
  ...
- `figure` = `_int_` (optional)
  The figure number that is passed to matplotlib.
  If absent, figure 1 is used.
- `shape` = `[_int_ , _int_]` (optional)
  The arrangement of plots inside the figure. For instance, `[2, 1]` makes two plots stacked vertically, and `[1, 2]` makes two plots stacked horizontally.
  If absent, stacks plots vertically.

Example:

```
>>> A = ParticleDiagnostic("path/to/my/results", diagNumber=0)
>>> B = ParticleDiagnostic("path/to/my/results", diagNumber=1)
>>> multiPlot( A, B, figure=1 )
```

This will plot the diagnostics #0 and #1 on the same figure, and make an animation for all available timesteps.

4 Tutorial

If you don't know how to run SMILEI, please refer to the appropriate documentation first. The commands can vary depending on your system and installation. A typical example of the command to run SMILEI is: `mpiexec -np 1 smilei mycase.in`

In the `benchmarks` directory, we provide a test case `tst1d_6_particle_diagnostic.in`. This case is very simple: it consists of a one-dimensional uniform neutral plasma composed by ions and electrons. The electrons all have a drift velocity of $0.05c$.

Run this case using SMILEI and collect the results in a directory of your choice. In this tutorial, we suppose that the results are in the directory `tst1d_6_particle_diagnostic`. Make sure this directory contains the input file `tst1d_6_particle_diagnostic.in` and the output files `ParticleDiagnostic0.h5`, `ParticleDiagnostic1.h5`, etc.

An example of the commands you may use from a *terminal* (or *console*, or *command line window*) is

```
$ mkdir tst1d_6_particle_diagnostic
$ cp   benchmarks/tst1d_6_particle_diagnostic.in    tst1d_6_particle_diagnostic
$ cd tst1d_6_particle_diagnostic
$ mpiexec -np 1 smilei tst1d_6_particle_diagnostic.in
$ cd ..
```

From the same terminal, launch *python* using the command

```
$ python -i scripts/ParticleDiagnostic.py
```

Your are now in the *python* prompt.
Obtain a list of available particle diagnostics using

```
>>> ParticleDiagnostic('tst1d_6_particle_diagnostic')
Printing available particle diagnostics:
----------------------------------------
Diag#0 - density of species # 1
    Every 4 timesteps, averaging over 2 timesteps
    x from 0.0 to 6.28319 in 100 steps
    vx from -0.1 to 0.1 in 100 steps
Diag#1 - density of species # 0
    Every 4 timesteps, no time-averaging
    x from 0.0 to 6.28319 in 100 steps
    vx from -0.001 to 0.001 in 100 steps
Diag#2 - density of species # 1
    Every 10 timesteps, averaging over 5 timesteps
    ekin from 0.0001 to 0.1 in 100 steps  [ LOG SCALE ]
```

We want to have a look at the diagnostic #0, which is the density of species #1 (here, electrons) with two axes: the position x and the velocity v_x. In other words, it is the phase-space of electrons.
To plot the phase-space in the initial conditions, use

```
>>> ParticleDiagnostic('tst1d_6_particle_diagnostic', 0, timesteps=0 ).plot()
```

A window appears showing the plot in Figure 1. We can see that the electrons have indeed a drift velocity of $0.05c$.
To obtain the equivalent plot for the ions, use the diagnostic #1 with the command

```
>>> ParticleDiagnostic('tst1d_6_particle_diagnostic', 1, timesteps=0 ).plot()
```

This results in the plot in Figure 2. The ions have a zero average velocity.

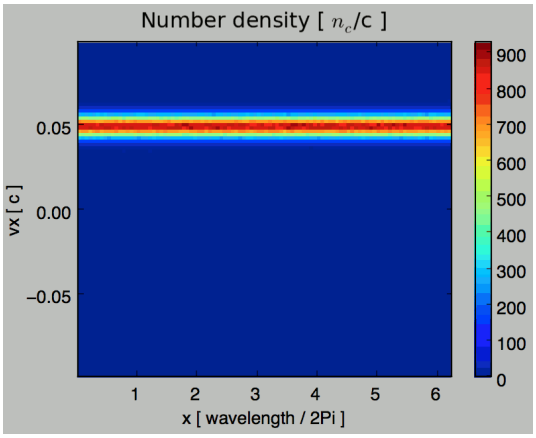

Figure 1: Phase-space of electrons at $t = 0$.

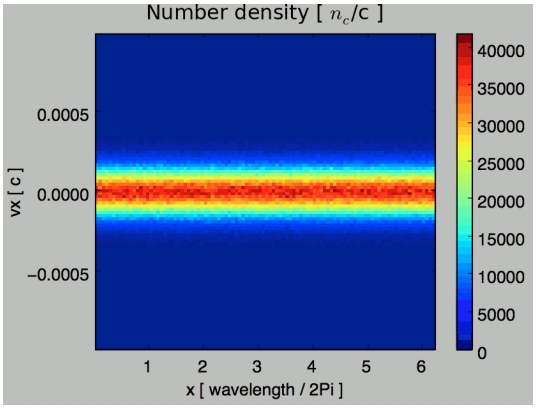

Figure 2: Phase-space of ions at $t = 0$.

Use the following command to have an animation of the electron phase-space with time:

```
>>> ParticleDiagnostic('tst1d_6_particle_diagnostic', 0 ).plot()
```

You will see the electron velocity oscillate from $0.05c$ to $-0.05c$. This is due to the fact that we are simulating a plasma wave with infinite wavelength.

Use the following commands to have the same animation with both electrons and ions on the same figure:

```
>>> A = ParticleDiagnostic('tst1d_6_particle_diagnostic', 0 )
>>> B = ParticleDiagnostic('tst1d_6_particle_diagnostic', 1 )
>>> multiPlot(A, B)
```