

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^* (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 *weight* \times *charge* are summed. In the case of `current_density_x`, the *weights* \times *charge* \times *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 $Z^* = 0$ 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 ParticleDiagnostic.py
```

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

In python, you can run `ParticleDiagnostic(...)` to extract the data and optionally plot it.

Syntax of the function *ParticleDiagnostic*:

```
ParticleDiagnostic(results_path, diagNumber=None, timesteps=None, slice=None,  
                  units="code", data_log=False, data_min=None, data_max=None,  
                  xmin=None, xmax=None, ymin=None, ymax=None,  
                  figure=None)
```

- `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 before plotting.
- `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.
- `figure = None` (default)
`figure = _int_` (optional)
Chooses the figure number that is passed to matplotlib.
If absent or None, returns the first data without plotting.

Output of the function *ParticleDiagnostic*

If `figure` is a number, then the function does not return anything. Instead, it displays the results in a graph.

If `figure` is not provided, or `figure=None`, then the function 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.

Examples of usage from the python prompt:

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

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.

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

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"]`.