p2sat Documentation

Release 1.4.2

lesnat

Oct 18, 2018

CONTENTS

p2sat is an open-source, object-oriented python package created to simplify particle phase-space analysis.

Core features of the package are :

- Automatic calculation of kinetic energy, divergence angle and gamma factor of the particles from phase space informations
- Histogram making (1D, 2D, 3D) and data fits (1D)
- Plotting (1D to 3D histograms, scatter and contour plots) with automatic normalizations and legend
- Particle filtering with a given property (for example select all the particles at a given position)
- Statistical tools (standard deviation, covariance, . . .)
- Import data from simulation files (Smilei, Geant4, text files, . . .)
- Low memory load

This allows to process complex operations in a very concise and clear way, as shown in the examples.

See documentation for more informations.

Notes :

- This package was made for my personal use and then contains only few methods to import data from code results, but you can easily add your own (please, share !) and use it to perform your data analysis. See sub-object `_Extract` for more informations.
- This tool can be usefull to physicists working with Particle-In-Cell or Monte Carlo codes

Contents :

CHAPTER

ONE

INTRODUCTION

1.1 Problematic

When you deal with phase space analysis, the philosophy is often the same :

- Import data from a file or generate it with defined laws
- Make calculations with this raw data (kinetic energy, divergence, . . .)
- Filter some of it (divergence with a condition on energy, . . .)
- Make histograms
- Make fits
- Plot results
- Format the axes

It is often long and leads to many programmation errors, and when there is a need to compare several set of datas, it become more and more complicated.

p2sat objective is to unify a set of phase space data in a single object, and give all the needed methods to manipulate and analyse these datas.

This way the analysis and comparison of several sources need much less efforts and physicists can do what they like to do : physics.

1.2 Package Structure

All the package structure relies on one single object : *PhaseSpace*.

This object is a "box" containing all the informations about a given particle phase space, and all the methods to interact with it (make histograms, statistics or plots).

Once you create such object, the data it contains are isolated from the rest of your script (see examples below to show how to acess data), so there is no risk of confuse between different data sources. This object-oriented approach is so completely coherent when you need to compare data from several simulation files for examples : the methods you use are the same, only the instances names are changing.

1.3 Particle phase space

The phase space of a set of particles gives the most complete description of a dynamical system (assuming particles are independant), and all the needed informations can be reconstructed from it.

This phase space contains informations such as particles positions, momentum and time. There is also given a statistical weight to each configuration. The phase space is then a 7D space (3 for momentum, 3 for space, 1 for time). If it is discretized, this represent a tremendous amount of possible configurations, so give a statistical weight to **EACH** of them is not the good approach. The p2sat approach is to save only the informations on configurations that have a non-zero statistical weight, and list them, one configuration per line. To have access to a given configuration, there is only the need of knowing its index.

Informations about units can be found in the *_Data* object (access via *PhaseSpace.data*)

1.4 Quick example

Assuming *p2sat* is imported, you can instanciate a *PhaseSpace* object for, let say, electrons, and import a simulation file containing the phase space informations.

```
>>> eps = p2sat.PhaseSpace(particle="electron")
>>> eps.extract.txt("example.csv",sep=",")
```

All the data in you simulation file can now be found at *eps.data*

```
>>> # List of all the statistical weights
>>> print(eps.data.raw.w)
[1456.0, 1233.0 , 756.0, ... ]
>>> # List of all the x position
>>> print(eps.data.raw.x)
[10.0, 50.0, 30.0, ... ]
>>> # List of all the kinetic energies
>>> print(eps.data.raw.ekin)
[1.58, 4.61, 3.28, ... ]
```

This means that at the first index, there is an electron at position x=10.0 um with 1.58 MeV of kinetic energy and with statistical weight of 1456.0.

You can now do statistics with this data, for example get the standard deviation of theta angle for all the electrons

```
>>> theta_std = eps.stat.standard_deviation('theta')
```

and get an histogram (Number/degree, bin width = 1 degree) of this quantity

```
>>> theta,Ntheta = eps.hist.h1('theta', bwidth=0.1)
```

It is also possible to make simple or complicated plot in a elegant way

```
>>> eps.plot.figure(0)
>>> eps.plot.h1('theta', log=True, bwidth=0.1)
>>> eps.plot.figure(1)
>>> eps.plot.h2('theta','ekin',
...             log=True, polar=True
...             bwidth1=1.0,bwidth2=0.1,
...             select={'x':10.0,'t':[0.0,100.0]})
```

Other examples and a more complete documentation can be found at : https://github.com/lesnat/p2sat

CHAPTER

TWO

INSTALLATION

The most simple way to install p2sat is to use pip (https://pypi.org/project/p2sat/)

```
pip install p2sat
```

Otherwise, you can also download the source code from github (https://github.com/lesnat/p2sat), extract it and and type the following commands

```
cd p2sat
python setup.py install
```

If it is not working, you can add the following lines at the beginning of your script

```
p2sat_path="/path/to/p2sat/"
import sys
if p2sat_path not in sys.path: sys.path.append(p2sat_path)

import p2sat
```

p2sat is written for python 2.7 but might be compatible with 3.

Its only dependancies are python packages numpy and matplotlib.

CHAPTER

THREE

USE

3.1 PhaseSpace

class p2sat.**PhaseSpace**(*particle*)

Main class for particle phase-space analysis.

Parameters **particle** (*str*) – Name of the particle. Availables are gamma,e-,e+,mu-,mu+.

Variables

- **particle** (*dict*) – contains informations about particle particle, such as name, mass and label in TeX format
- **data** (*sub-object*) – contains raw data and methods to manipulate it, such as discretization or transformation.
- **hist** (*sub-object*) – make histograms from data
- **plot** (*sub-object*) – plot histograms
- **extract** (*sub-object*) – load phase space from a file
- **export** (*sub-object*) – export phase space into a file
- **stat** (*sub-object*) – make statistics on particle phase space

Examples

Assuming you already imported p2sat, you can create a PhaseSpace object for, let say, electrons, as follows

```
>>> eps = p2sat.PhaseSpace(particle="e-")
```

You can then import data from a file, using the *txt* method of sub-object *extract*

```
>>> eps.extract.txt("example.csv")
```

and look at the imported data

```
>>> print(eps.data.raw.w)
```

or print general informations about your data set

```
>>> print(eps)
```

You can also make histograms, plots or statistics . . .

Notes

See sub-objects documentation for more informations

copy(*verbose=False*)
Return a copy of the current PhaseSpace object.

Parameters **verbose** (*bool*) – verbosity of the function. If True, a message is displayed when the attributes are loaded in memory

3.2 PhaseSpace.data

class p2sat._Data.**_Data**(*PhaseSpace*)
Class containing raw data and methods to manipulate it.

Variables **raw** (*sub-object*) – class containing raw data and physical quantities calculations

Notes

Units :

- lengths are defined in 10^{-6} meters (um)
- momentums are defined in 10^{6} electron-volt/speed of light (MeV/c)
- times are defined in 10^{-15} seconds (fs)
- energies are defined in 10^{6} electron-volt (MeV)
- angles are defined in degrees (deg)

As all the calculations are done with the previously defined units, the input data might be firstly converted to those units.

All the attributes can not be overwriden as they are defined as properties. Please call the *update* method to update particle phase-space data.

discretize(*with_time=True*, *split=4*, *MP=True*, *verbose=True*, ***kargs*)
Discretize the particles phase space in a 6 or 7 D histogram.

Parameters

- **with_time** (*bool, optional*) – discretize with time (7D). Default is True
- **verbose** (*bool, optional*) – verbosity. Default is True
- **kargs** – optional keyword arguments to pass to the hist.hn function

Notes

This method can be used to significantly reduce disk space usage when saving data into output file.

See also:

`hist.hn()`

full_select(*faxes*, *frange*, *fpp=1e-07*, *update=False*, *verbose=True*)
Select all the phase space with given condition

Parameters

- **faxes** (*list of str or list of numpy.ndarray*) – filtering axis
- **frange** (*list of int, float, list/tuple of 2 float*) – filtering value/range (value if int, range if float or list/tuple). If a frange element is None, the minimum/maximum value is taken
- **fpp** (*float, optional*) – relative floating point precision. Default is 1e-7
- **update** (*bool, optional*) – update or not the current *PhaseSpace* instance. Default is false
- **verbose** (*bool, optional*) – verbosity

See also:

`data.select()`

generate (*Nconf*, *Npart*, *ekin*, *theta*, *phi*, *x=None*, *y=None*, *z=None*, *r=None*, *t=None*, *verbose=True*)

Generate a particle phase space from given laws.

Parameters

- **Nconf** (*int*) – total number of configurations
- **Npart** (*float*) – total number of particles
- **ekin** (*dict*) – parameters to generate kinetic energy
- **theta** (*dict*) – parameters to generate theta angle distribution
- **phi** (*dict*) – parameters to generate phi angle distribution
- **x,y,z** (*dict, optional*) – parameters to generate position distribution. Default is 0 for x,y,z
- **r** (*dict,optional*) – parameters to generate transverse position distribution. Default is 0
- **t** (*dict, optional*) – parameters to generate time distribution. Default is 0

Notes

The dictionnaries must each time at least contain the key 'law' with a value depending on which law are available for each physical quantity

For dict *ekin*, available laws are :

- 'mono', for a mono-energetic source. Energy must be given as a value of keyword 'ekin0'
- 'exp', for exponential energy. Charasteristic energy must be given as a value of keyword 'ekin0'

For dict *theta* and *phi*, available laws are :

- 'mono', for a directional source. Angle must be given as a value of keyword 'theta0' or '/phi0'
- 'iso', for an isotropic source. Optional keywords 'min' and 'max' can be given to specify a minimum/maximum angle (centered on 0 deg)
- 'gauss', for a gaussian spreading. Center of the distribution must be given with keyword 'mu', and standard deviantion with keyword 'sigma'

For dict *x*,'y','z','t', available laws are :

- 'mono', for a unique position/time. This parameter must be given as a value of keyword *x0*/*y0*/*z0*/*t0*

- 'range', for a uniform law in a given range. Keywords 'min' and 'max' MUST be given to specify a minimum and maximum
- 'exp', for exponential distribution. Charasteristic length/time must be given as a value of keyword *x0/y0/z0/t0*
- 'gauss', for a gaussian distribution. Center of the distribution must be given with keyword 'mu', and standard deviantion with keyword 'sigma'
- 'grid', for uniformly placed on a given grid. Keywords 'bins' OR 'min' + 'max' + 'Nbins' MUST be given.

For dict *r*, available laws are :

- 'range', for a uniform law in a given range. Keywords 'min' and 'max' MUST be given to specify a minimum and maximum
- 'gauss', for a gaussian distribution. Center of the distribution must be given with keyword 'mu', and standard deviantion with keyword 'sigma'

Details of the calculations :

Considering $E_T = E_k + m_0$ being the total energy, with E_k the kinetic energy and m_0 the rest mass energy.

We also have $E_T^2 = p^2 + m_0^2$ and $p^2 = p_x^2 + p_y^2 + p_z^2$ with p in MeV/c.

Assuming $\cos\theta = \frac{p_x}{p}$ and $\tan\phi = \frac{p_z}{p_y}$ we finaly get

- $p = E_k^2 - 2E_k m_0$
- $p_x = p\cos\theta$
- $p_y = \frac{\phi}{|\phi|}\sqrt{\frac{p^2 - p_x^2}{1+\tan^2\phi}}$
- $p_z = p_y \tan\phi$

Examples

With a *PhaseSpace* object instanciated as *eps*, you can generate a mono-energetic source in isotropic direction for 1e12 particles represented by 1e6 configurations as follows

```
>>> eps.data.generate(Nconf=1e6,Npart=1e12,
...                   ekin={"law":"mono","ekin0":20.0},
...                   theta={"law":"iso"},
...                   phi={"law":"iso"})
...
```

get_axis (*axis*, *select=None*)
Return given axis

Parameters

- **axis** (*str*) – axis name
- **select** (*dict*) – filtering dictionnary

Examples

```
>>> eps.data.get_axis("w")
>>> eps.data.get_axis("w",select={'x':150,'ekin':[0.511,None]})
```

get_ps()
Return current phase space raw data.

propagate(*x=None*, *t=None*, *update=True*, *verbose=True*)
Propagate the phase space to a given position or time.

Parameters

- **x** (*float, optional*) – propagate the phase-space to position x. Default is None (no propagation)
- **t** (*float, optional*) – propagate the phase-space to time t. Default is None (no propagation)
- **verbose** (*bool, optional*) – verbosity

Notes

x and t can not be defined simultaneously.

select(*axis*, *faxes*, *frange*, *fpp=1e-07*)
Filter an axis with a value/range on another axis.

Parameters

- **axis** (*str or numpy.ndarray*) – axis to filter
- **faxes** (*list of str or list of numpy.ndarray*) – filtering axis
- **frange** (*list of int, float, list/tuple of 2 float*) – filtering value/range (value if int, range if float or list/tuple). If a frange element is None, the minimum/maximum value is taken
- **fpp** (*float, optional*) – relative floating point precision. Default is 1e-7

Returns **axis** – filtered axis

Return type numpy.ndarray

Examples

Given the *PhaseSpace* instance *eps*, it is possible to filter by an int value (Select all the w satisfying $x = 3$)

```
>>> w,x = eps.data.select('w',['x'],[3])
```

or filter by a range (Select all the θ with $E_{kin} \in [0.511, +\infty]$ MeV)

```
>>> theta,ekin = eps.data.select('theta',['ekin'],[[0.511,None]])
```

If frange is a list/tuple or a float, the filtering is done with a fpp precision

transformate(*T=None*, *R=None*, *rotate_first=False*, *verbose=True*)
Transformate the particle phase space with given translation and rotation.

Parameters

- **T** (*tuple of 3 float, optional*) – translate (x,y,z) position of T um. Default is (0,0,0)
- **R** (*tuple of 3 float, optional*) – rotate (x,y,z) and (px,py,pz) of R degree. Default is (0,0,0)
- **rotate_first** (*bool, optional*) – process rotation before translation. Default is false
- **verbose** (*bool, optional*) – verbosity

update(*w*, *x*, *y*, *z*, *px*, *py*, *pz*, *t*, *verbose=True*)

Update class attributes with new values.

Parameters

- **w,x,y,z,px,py,pz,t** (*list or numpy.ndarray*) – particle phase space. More information can be found in data object documentation
- **verbose** (*bool*) – verbosity of the function. If True, a message is displayed when the attributes are loaded in memory

3.3 PhaseSpace.data.raw

class p2sat._Data.**_Raw**(*PhaseSpace*)

Class containing raw data and physical quantities calculations.

Variables

- ***w*** (*numpy.ndarray*) – statistical weight
- ***x***, ***y***, ***z*** (*numpy.ndarray*) – x,y,z position in um
- ***px***, ***py***, ***pz*** (*numpy.ndarray*) – momentum in x,y,z direction in MeV/c
- ***t*** (*numpy.ndarray*) – time in fs
- **others** (*numpy.ndarray*) – see also documentation to look at all the quantities calculated from phase-space data

Notes

To add a new quantity, please add a new function to this file with the name of the quantity, only the *self* parameter and with the decorator *@property*. Please also add label and unit definition of this new quantity in *__init__*.

beta

Particle normalized velocity

Notes

beta is calculated as follow :

$$\beta = \sqrt{1 - \frac{1}{\gamma^2}}$$

References

https://en.wikipedia.org/wiki/Lorentz_factor

ekin
Particle kinetic energy (in MeV)

Notes

ekin is calculated as follow :

$$E_{kin} = E_{tot} - m_0$$

with m_0 being the rest mass energy

References

https://en.wikipedia.org/wiki/Energy-momentum_relation

ekin_density
Particle energy density (in MeV)

Notes

ekin_density is calculated as follow :

$$E_{kin} \times w$$

etot
Particle total energy (in MeV)

Notes

etot is calculated as follow :

$$E_{tot} = \sqrt{p^2 + m_0^2}$$

with m_0 being the rest mass energy

References

https://en.wikipedia.org/wiki/Energy-momentum_relation

gamma
Particle Lorentz factor

Notes

gamma is calculated as follow :

$$\gamma = E_{tot}/m_0$$

with m_0 being the rest mass energy

References

https://en.wikipedia.org/wiki/Lorentz_factor

m

Particle relativistic mass (in MeV)

Notes

m is calculated as follow :

$$m = \gamma m_0$$

with m_0 being the rest mass energy

References

https://en.wikipedia.org/wiki/Lorentz_factor

p

Particle absolute momentum (in MeV/c)

Notes

p is calculated as follow :

$$p = \sqrt{p_x^2 + p_y^2 + p_z^2}$$

phi

Particle azimutal angle (between py and pz) (in deg)

Notes

phi is calculated as follow :

$$\phi = \arctan p_z/p_y$$

with using the arctan2 function to have a result between -180 and 180 deg

References

https://en.wikipedia.org/wiki/List_of_common_coordinate_transformations#To_spherical_coordinates with switching (x->py, y->pz, z->px). A figure can be found at https://en.wikipedia.org/wiki/Spherical_coordinate_system

https://en.wikipedia.org/wiki/Atan2

px

Particle momentum in propagation direction x (in MeV/c)

py

Particle momentum in transverse direction y (in MeV/c)

pz
Particle momentum in transverse direction z (in MeV/c)

r
Particle distance to the propagation direction x (in um)

Notes

r is calculated as follow :

$$r = \sqrt{y^2 + z^2}$$

t
Particle time (in fs)

theta
Particle polar angle (between px and the plane (py,pz)) (in deg)

Notes

theta is calculated as follow :

$$\theta = \arccos p_x/p$$

References

https://en.wikipedia.org/wiki/List_of_common_coordinate_transformations#To_spherical_coordinates with switching (x->py, y->pz, z->px). A figure can be found at https://en.wikipedia.org/wiki/Spherical_coordinate_system

ux
Particle direction on x axis

Notes

ux is calculated as follow :

$$u_x = \frac{p_x}{p}$$

uy
Particle direction on y axis

Notes

uy is calculated as follow :

$$u_y = \frac{p_y}{p}$$

uz
Particle direction on z axis

Notes

uz is calculated as follow :

$$u_z = \frac{p_z}{p}$$

v

Particle absolute velocity (in um/fs)

Notes

v is calculated as follow :

$$v = \beta \times c$$

with c being the speed of light

References

https://en.wikipedia.org/wiki/Lorentz_factor

w

Particle statistical weight

x

Particle position in propagation direction x (in um)

y

Particle position in transverse direction y (in um)

z

Particle position in transverse direction z (in um)

3.4 PhaseSpace.hist

class p2sat._Hist.**_Hist**(*PhaseSpace*)

Create histograms from raw data.

f1(*axis*, *func_name*, *weight='w'*, *return_fit=False*, *verbose=True*, ***kargs*)

Fit a 1D histogram with given law.

Parameters

- **axis** (*str or np.array*) – axis to fit
- **func_name** (*str*) – name of the fit law. Available are *exp* for exponential law and *gauss* for gaussian law
- **return_fit** (*bool, optional*) – returns the spectrum instead of fited parameters
- **verbose** (*bool, optional*) – verbosity
- **kargs** (*dict, optional*) – dictionnary to pass to the hist.h1 method

Returns

- **x** (*np.array*) – fit abscissa
- **param1,param2** (*float,optional*) – fit parameters. Returned if *return_fit=False* (default)
- **w** (*np.array, optional*) – fit weight. Returned if *return_fit=True*

Notes

The *exp* law is defined as $\frac{N}{T}\exp(-x/T)$ and returns fit parameters N,T.

The *gauss* law is defined as $\frac{N}{\sigma\sqrt{2\pi}}\exp(-\frac{(x-\mu)^2}{2\sigma^2})$ and returns fit parameters N,sigma,mu.

h1(*axis*, *weight='w'*, *bwidth=None*, *brange=None*, *normed=True*, *select=None*)
Create and return the 1 dimensional histogram of given axis.

Parameters

- **axis** (*str or np.array*) – axis to hist
- **bwidth** (*float, optional*) – bin width. If None, a calculation is done to have 10 bins in the axis
- **brange** (*list of 2 float, optional*) – bin maximum and minimum. If a brange element is None, the axis minimum/maximum is taken
- **normed** (*bool, optional*) – weight normalization. If a normed element is True, the bin width is taken as weight normalization
- **select** (*dict, optional*) – filtering dictionnary

Returns

- **b** (*np.array*) – bins
- **h** (*np.array*) – histogram

Notes

the h1 method is just a different way to call the generic method hn

See also:

`hist.hn()`, `hist.h2()`, `hist.h3()`

h2(*axis1*, *axis2*, *weight='w'*, *bwidth1=None*, *bwidth2=None*, *brange1=None*, *brange2=None*, *normed=True*, *select=None*)
Create and return the 2 dimensional histogram of given axis.

Parameters

- **axis1,axis2** (*str or np.array*) – axis to hist
- **bwidth1,bwidth2** (*float, optional*) – bin width. If None, a calculation is done to have 10 bins in the axis
- **brange1,brange2** (*list of 2 float, optional*) – bin maximum and minimum. If a brange element is None, the axis minimum/maximum is taken
- **normed** (*bool, optional*) – weight normalization. If a normed element is True, the bin width is taken as weight normalization
- **select** (*dict, optional*) – filtering dictionnary

Returns

- **b1,b2** (*np.array*) – bins
- **h** (*np.array*) – histogram

Notes

the h2 method is just a different way to call the generic method hn

See also:

`hist.hn()`, `hist.h1()`, `hist.h3()`

h3(*axis1*, *axis2*, *axis3*, *weight='w'*, *bwidth1=None*, *bwidth2=None*, *bwidth3=None*, *brange1=None*, *brange2=None*, *brange3=None*, *normed=True*, *select=None*)
Create and return the 3 dimensional histogram of given axis.

Parameters

- **axis1,axis2,axis3** (*str or np.array*) – axis to hist
- **bwidth1,bwidth2,bwidth3** (*float, optional*) – bin width. If None, a calculation is done to have 10 bins in the axis
- **brange1,brange2,brange3** (*list of 2 float, optional*) – bin maximum and minimum. If a brange element is None, the axis minimum/maximum is taken
- **normed** (*bool, optional*) – weight normalization. If a normed element is True, the bin width is taken as weight normalization
- **select** (*dict, optional*) – filtering dictionnary

Returns

- **b1,b2,b3** (*np.array*) – bins
- **h** (*np.array*) – histogram

Notes

the h3 method is just a different way to call the generic method hn

See also:

`hist.hn()`, `hist.h1()`, `hist.h2()`

hn(*axis*, *weight='w'*, *bwidth=None*, *brange=None*, *normed=True*, *select=None*)
Create and return the n-dimensional histo of axis list.

Parameters

- **axis** (*list of str/np.array*) – list of axis to hist
- **bwidth** (*list of float, optional*) – list of bin width. If a bwidth element is None, a calculation is done to have 100 bins in the correspondant axis
- **brange** (*list of list of 2 float, optional*) – list of bin minimum and maximum. If a brange element is None, the minimum/maximum of the axis is taken
- **normed** (*bool or list of bool, optional*) – weight normalization. If a normed element is True, the bin width is taken as weight normalization
- **select** (*dict, optional*) – filtering dictionary

Returns

- **b** (*np.array*) – bins
- **h** (*np.array*) – number of particles per bin unit

Notes

The weight normalization allows to be independant of choosen bin width. If normalized, the weight unit is *Number/(unit1 x unit2 x . . .)* with *unit1, unit2, . . .* the units of axes *1, 2, ...*

If the given maximum bin range does not match with an int number of bins, the bin width is over sized.

The select dictionary takes the name of filtering axes as dict keys, and value/range of filtering axis as dict values.

Examples

Using *eps* as a *PhaseSpace* instance

```
>>> w,x = eps.hist.hn(['x'],
...                   bwidth=[50],brange=[[0,1000]],
...                   normed=[True],
...                   select={'ekin':(0.511,None)})
...
```

returns the number of particles with $ekin \in [0.511, +\infty] MeV$ in function of x normed=[True] to divide weight by bin width, so weight unit is Number/um

```
>>> w,r,ekin=eps.hist.hn(['r','ekin'],
...                      bwidth=[10.0,0.1],
...                      brange=[[0,1000],[0.1,50.0]],
...                      select={'x':150})
...
```

returns the number of particle per um per MeV at x=150 um

See also:

`data.select()`

3.5 PhaseSpace.plot

class p2sat._Plot.**_Plot**(*PhaseSpace*)

Plot raw data.

Variables

- **autoclear** (*bool*) – True to auto clear figure when calling a plot method. Default is False
- **cmap** (*str*) – color map to use in 2d plot. Default is viridris
- **rcParams** (*dict*) – shortcut to matplotlib.rcParams dictionnary (change plot appearence)

a1(*axis*, *t=None*, *pause=0.5*, *where='post'*, *log=False*, *polar=False*, *reverse=False*, ***kargs*)

Plot the animation of 1d histogram of given axis.

Parameters

- **axis** (*str*) – Name of the axis to plot
- **t** (*dict*) – parameters to construct plot times
- **pause** (*float*) – waiting time between each plot (in seconds)
- **where** (*str, optional*) – ...
- **log** (*bool, optional*) – True to set log scale on y axis
- **polar** (*bool, optional*) – True to use a polar plot. axis must be an angle
- **reverse** (*bool, optional*) – True to plot axis against number instead of number against axis
- **kargs** (*dict, optional*) – Dictionnary to pass to the hist.h1 method

Notes

t dictionnary must contains following keys :

- "min", to define the first plot time
- "max", to define the last plot time
- "Nbins", to define the number of timesteps to plot

Examples

```
>>> eps.plot.autoclear = False
>>> eps.plot.a1('r',t={'min':0,'max':1000,'Nbins':10},bwidth=1.)
```

See also:

`plot.h1()`

a2 (*axis1*, *axis2*, *t=None*, *pause=0.5*, *log=False*, *polar=False*, ***kargs*)
Plot the animation of 2d histogram of given axes.

Parameters

- **axis1,axis2** (*str*) – Name of the axes to plot
- **t** (*dict*) – parameters to construct plot times
- **pause** (*float*) – waiting time between each plot (in seconds)
- **log** (*bool, optional*) – True to set log scale on y axis
- **polar** (*bool, optional*) – True to use a polar plot. axis1 must be an angle
- **kargs** (*dict, optional*) – Dictionnary to pass to the hist.h2 method

Notes

t dictionnary must contains following keys :

- "min", to define the first plot time
- "max", to define the last plot time

- "Nbins", to define the number of timesteps to plot

Examples

```
>>> eps.plot.a2('x','y',t={'min':0,'max':1000,'Nbins':10},log=True)
```

See also:

`plot.h2()`

c2 (*axis1*, *axis2*, *log=False*, *polar=False*, *gfilter=0.0*, ***kargs*)
Plot the 2d contour of given axes.

Parameters

- **axis1,axis2** (*str*) – Name of the axes to plot
- **log** (*bool, optional*) – True to set log scale on y axis
- **polar** (*bool, optional*) – True to use a polar plot. axis1 must be an angle
- **gfilter** (*float, optional*) – Filtering scipy.ndimage.filters.gaussian_filter
- **kargs** (*dict, optional*) – Dictionnary to pass to the hist.h2 method

See also:

`hist.h2()`

clear (*number=None*)
Clear a plot.

Parameters **number** (*int, optional*) – Figure number to clear. If None, clear the current figure

f1 (*axis*, *func_name*, *log=False*, *polar=False*, *reverse=False*, ***kargs*)
Plot the 1d fit of given axis.

Parameters

- **axis** (*str*) – Name of the axis to plot
- **func_name** (*str*) – name of the fit function
- **log** (*bool, optional*) – True to set log scale on y axis
- **polar** (*bool, optional*) – True to use a polar plot. axis must be an angle
- **reverse** (*bool, optional*) – True to plot axis against number instead of number against axis
- **kargs** (*dict, optional*) – Dictionnary to pass to the hist.h1 method

See also:

`hist.f1()`

figure (*number=None*, *clear=True*)
Creates a new figure with given number.

Parameters

- **number** (*int, optional*) – Figure number to create
- **clear** (*bool, optional*) – Call or not the *clear* method for given number. Default is True

See also:

`plot.clear()`

get_labels (*axes*, *weight*, *normed*)
Returns the labels of given axes.

Parameters

- **axes** (*list of str*) – Names of the axes
- **normed** (*list of bool or None*) – Weight normalization. If None, the last labels element is "Number", otherwise it is "Number/unit1/unit2/..."

Returns **labels** – Labels of given axes and label of weight

Return type list of str

h1 (*axis*, *where='post'*, *log=False*, *polar=False*, *reverse=False*, ***kargs*)
Plot the 1d histogram of given axis.

Parameters

- **axis** (*str*) – Name of the axis to plot
- **where** (*str, optional*) – ...
- **log** (*bool, optional*) – True to set log scale on y axis
- **polar** (*bool, optional*) – True to use a polar plot. axis must be an angle
- **reverse** (*bool, optional*) – True to plot axis against number instead of number against axis
- **kargs** (*dict, optional*) – Dictionnary to pass to the hist.h1 method

See also:

`hist.h1()`

h2 (*axis1*, *axis2*, *log=False*, *polar=False*, ***kargs*)
Plot the 2d histogram of given axes.

Parameters

- **axis1,axis2** (*str*) – Name of the axes to plot
- **log** (*bool, optional*) – True to set log scale on y axis
- **polar** (*bool, optional*) – True to use a polar plot. axis1 must be an angle
- **kargs** (*dict, optional*) – Dictionnary to pass to the hist.h2 method

See also:

`hist.h2()`

h2h1 (*axis1*, *axis2*, *log=False*, ***kargs*)
TODO

h3 (*axis1*, *axis2*, *axis3*, *s=5*, *wmin=0*, *log=False*, ***kargs*)
Plot the 3d histogram of given axes.

Parameters

- **axis1,axis2,axis3** (*str*) – Name of the axes to plot
- **s** (*float, optional*) – square sizes. Default is 5

- **wmin** (*float, optional*) – minimum weight to plot. Default is 0
- **log** (*bool, optional*) – log color scale. Default is False
- **kargs** (*dict, optional*) – Dictionnary to pass to the hist.h3 method

See also:

`hist.h3()`

s2 (*axis1*, *axis2*, *weight='w'*, *snorm=1.0*, *log=False*, *polar=False*, *select=None*)
Plot the 2d scattering plot of given axes.

Parameters

- **axis1,axis2** (*str*) – name of the axes to plot
- **weight** (*str, optional*) – weight to plot. Default is w
- **snorm** (*float, optional*) – dots size normalization. Default is 1
- **log** (*bool, optional*) – True to set log scale on y axis
- **polar** (*bool, optional*) – True to use a polar plot. axis1 must be an angle
- **select** (*dict, optional*) – select dictionnary as in the hist.h2 method

s2h1 (*axis1*, *axis2*, *log=False*)
TODO

s3 (*axis1*, *axis2*, *axis3*, *weight='w'*, *snorm=1.0*, *log=False*, *select=None*)
Plot the 3d histogram of given axes.

Parameters

- **axis1,axis2,axis3** (*str*) – Name of the axes to plot
- **weight** (*str, optional*) – weight to plot. Default is w
- **snorm** (*float, optional*) – dots size normalization. Default is 1
- **log** (*bool, optional*) – log color scale. Default is False
- **select** (*dict, optional*) – filtering dictionnary

set_title (*title*, *number=None*)
Set the title of the figure.

3.6 PhaseSpace.extract

class p2sat._Extract.**_Extract** (*PhaseSpace*)
Import data from a file.

Notes

If you want to add a method to import data from another code, you must proceed as follow :

- Add a method to this object, with the name of your code. It must contains the keyword *self* as a first argument (because of object-oriented paradigm), and all the other parameters you need
- Get the data from your file and put it in lists or numpy arrays, one line describing one particle
- Call the *update* method of *data* sub-object (access via *self._ps.data.update*)

- Please write a documentation and share !

You can copy-paste the *txt* method to have a basic example of file import.

Geant4_csv (*file_name*, *nthreads=1*, *verbose=True*)
Extract simulation results from a Geant4 NTuple csv output file

DEPRECATED

Parameters

- **file_name** (*str*) – name of the output file. If it ends with '*_t0.*', the number '0' will be replaced by the number of the current thread
- **nthreads** (*int*) – total number of threads to consider
- **verbose** (*bool, optional*) – verbosity

Smilei_Screen_1d (*Screen*, *xnorm*, *wnorm*, *tnorm*, *X=0*)
TODO

TrILEns_output (*path*, *verbose=True*)
Extract simulation results from a TrILEns output.txt file

Parameters

- **path** (*str*) – simulation path
- **verbose** (*bool, optional*) – verbosity

Examples

```
>>> eps = p2sat.PhaseSpace(particle="e-")
>>> eps.extract.TrILEns_output("../TrILEns/")
```

TrILEns_prop_ph (*path*, *verbose=True*)
TODO

gp3m2_csv (*base_name*, *verbose=True*)
Extract simulation results from a gp3m2 NTuple csv output file

Parameters

- **base_name** (*str*) – base file name
- **verbose** (*bool, optional*) – verbosity

Examples

For the gp3m2 output file name *Al_target_nt_electron_t0.csv*, the base_name is *Al_target*. Assuming a *p2sat.PhaseSpace* object is instanciated for particle *e-* as eps, you can import simulation results for all the threads as follows

```
>>> eps.extract.gp3m2_csv("Al_target")
```

txt (*file_name*, *sep=', '*, *verbose=True*)
Load particle phase space from a text file.

Parameters

- **file_name** (*str*) – name of the input file

- **sep** (*str*) – character used to separate values. Default is ‘,’
- **verbose** (*bool, optional*) – verbosity of the function. If True, a message is displayed when the data is imported

See also:

`export.txt()`

3.7 PhaseSpace.export

class p2sat._Export.**_Export**(*PhaseSpace*)

Export phase space into several file format.

TrILEns_input(*path*, *S1*, *S2*, *pasdt=1.0*, *maillage_spatial=None*, *with_time=True*, *verbose=True*)

Write default input file and export particle phase space in a TrILEns input file.

Parameters

- **path** (*str*) – path to the output folder
- **verbose** (*bool, optional*) – verbosity of the function. If True, a message is displayed when the data is exported

TrILEns_prop_ph(*path*, *with_time=True*, *verbose=True*)

Export particle phase space in a TrILEns input file.

Parameters

- **path** (*str*) – path to the output folder
- **verbose** (*bool, optional*) – verbosity of the function. If True, a message is displayed when the data is exported

gp3m2_input(*file_name*, *title=''*, *verbose=True*)

Export particle phase space in a gp3m2 input file

Parameters

- **file_name** (*str*) – name of the file
- **title** (*str, optional*) – short description of the file content
- **verbose** (*bool, optional*) – verbosity

txt(*file_name*, *header=True*, *title=''*, *sep=', '*, *verbose=True*)

Export particle phase space in a text file.

Parameters

- **file_name** (*str*) – name of the output file
- **header** (*bool, optional*) – True to put informations at the beginning of the file. Default is True
- **title** (*str, optional*) – short description of the file content
- **sep** (*str, optional*) – character to use to separate values. Default is ‘,’ (csv file)
- **verbose** (*bool, optional*) – verbosity of the function. If True, a message is displayed when the data is exported

Notes

The format in the output file is

```
# title
# legend
w x y z px py pz t
w x y z px py pz t
. . . . .  .  .  .
. . . . .  .  .  .
. . . . .  .  .  .
```

with 7 digits precision in scientific notation

Some text can be written if the first character of the line is a '#'.

3.8 PhaseSpace.stat

class p2sat._Stat.**_Stat**(*PhaseSpace*)

Get global statistics from phase space data.

correlation_coefficient(*axis1*, *axis2*, *select=None*)

Returns correlation coefficient of given axes.

Parameters

- **axis1** (*str or np.array*) – axis to consider
- **axis2** (*str or np.array*) – axis to consider
- **select** (*dict, optional*) – filtering dictionary

Returns **cc** – correlation coefficient of given axes

Return type float

Notes

correlation_coefficient is defined as covariance(axis1,axis2)/(standard_deviation(axis1)*standard_deviation(axis2))

covariance(*axis1*, *axis2*, *select=None*)

Returns covariance of given axes.

Parameters

- **axis1** (*str or np.array*) – axis to consider
- **axis2** (*str or np.array*) – axis to consider
- **select** (*dict, optional*) – filtering dictionary

Returns **cov** – covariance of given axes

Return type float

Notes

covariance is defined as expected_value((axis1-expected_value(axis1)) * (axis2-expected_value(axis2)))

expected_value(*axis*, *select=None*)
Returns expected value of given axis.

Parameters

- **axis** (*str or np.array*) – axis to consider
- **select** (*dict, optional*) – filtering dictionary

Returns **E** – expected value of given axis

Return type float

Notes

expected_value is defined as sum(p*axis) with p=w/sum(w)

standard_deviation(*axis*, *select=None*)
Returns standard deviation of given axis.

Parameters

- **axis** (*str or np.array*) – axis to consider
- **select** (*dict, optional*) – filtering dictionary

Returns **std** – standard deviation of given axis

Return type float

Notes

standard_deviation is defined as the square root of variance

total_energy(*unit='J'*, *select=None*)
Return total energy contained in the phase space

Parameters

- **unit** (*str, optional*) – unit of energy. Available are 'J' and 'MeV'. Default is 'J'
- **select** (*dict, optional*) – filtering dictionary

See also:

`data.select()`

variance(*axis*, *select=None*)
Returns variance of given axis.

Parameters

- **axis** (*str or np.array*) – axis to consider
- **select** (*dict, optional*) – filtering dictionary

Returns **var** – variance of given axis

Return type float

Notes

variance is defined as expected_value((axis - expected_value(axis))**2)

CHAPTER

FOUR

EXAMPLES

4.1 Make histo

```
#coding:utf8

"""
This is an example of how to use the `PhaseSpace.hist` object of p2sat.

It allows to make histogram from phase space data in a very simple way
"""

# Import p2sat
p2sat_path="../"
import sys
if p2sat_path not in sys.path:sys.path.insert(0,p2sat_path)
import p2sat

# Instanciate a PhaseSpace object for electron specie
eps = p2sat.PhaseSpace(particle="electron")

# Import data from a file
eps.extract.txt("example.csv",sep=",",verbose=False)

# Get spectrum (Number/MeV, bin width of 0.1 MeV)
ekin,spec = eps.hist.h1('ekin',bwidth=0.1)

# Fit the last spectrum for ekin > 0.511 MeV (Ne is total number of e-, Te its␣
↪temperature in MeV)
fekin,Ne,Te = eps.hist.f1('ekin', func_name="exp", bwidth=0.1, select={'ekin':[0.511,
↪None]})
print("Hot electron temperature (fit) : %.3E MeV"%Te)
```

4.2 Make plots

```
#coding:utf8

"""
This is an example of how to use the `PhaseSpace.plot` object of p2sat.

It allows to make plots from phase space data in a very simple way
"""
```

```
# Import p2sat
p2sat_path="../"
import sys
if p2sat_path not in sys.path:sys.path.insert(0,p2sat_path)
import p2sat

# Instanciate a PhaseSpace object for electron specie
eps = p2sat.PhaseSpace(particle="electron")

# Import data from a file
eps.extract.txt("example.csv",sep=",",verbose=False)

# Plot spectrum in log scale (Number/MeV, bin width of 0.1 MeV)
eps.plot.figure(0)
eps.plot.h1('ekin', log=True, bwidth=0.1)

# Fit the last spectrum for ekin > 0.511 MeV
eps.plot.f1('ekin', func_name="exp", log=True, bwidth=0.1, select={'ekin':[0.511,
↪None]})

# Plot Transverse particle dispersion for electrons with kinetic energy > 0.511 MeV␣
↪(bin width of 10 µm between -500 and 500 µm)
eps.plot.figure(1)
eps.plot.h2('y','z',log=True,
            bwidth1=5.0,bwidth2=5.0,
            brange1=[-300.,300.],brange2=[-300.,300.],
            select={'x':300,'ekin':[0.511,None]})

# Add a contour plot
eps.plot.c2('y','z',log=True, gfilter=3.5,
            bwidth1=5.0,bwidth2=5.0,
            brange1=[-300.,300.],brange2=[-300.,300.],
            select={'x':300,'ekin':[0.511,None]})

# Plot angle/energy polar distribution of the particles
eps.plot.figure(2)
eps.plot.h2('theta','ekin',
            log=True,polar=True,
            bwidth1=1.0,bwidth2=0.1)
```

4.3 Make statistics

```
#coding:utf8

"""
This is an example of how to use the `PhaseSpace.stat` object of p2sat.

It allows to make statistics on phase space data in a very simple way
"""

# Import p2sat
p2sat_path="../"
import sys
if p2sat_path not in sys.path:sys.path.insert(0,p2sat_path)
```

```
import p2sat

# Instanciate a PhaseSpace object for electron specie
eps = p2sat.PhaseSpace(particle="electron")

# Import data from a file
eps.extract.txt("example.csv",sep=",",verbose=False)

# Get the mean value of theta
theta_ev = eps.stat.expected_value('theta')

# Get the mean value of positive theta
theta_evp = eps.stat.expected_value('theta',select={'theta':[0.0,None]})

# Get standard deviation of theta
theta_std = eps.stat.standard_deviation('theta')

# Get correlation coefficient between theta and ekin
theta_ekin_cc = eps.stat.correlation_coefficient('theta','ekin')

# Print informations
print('theta expected value : %.4E deg'%theta_ev)
print('positive theta expected value : %.4E deg'%theta_evp)
print('theta standard deviation : %.4E deg'%theta_std)
print('theta ekin correlation coefficient : %.4E'%theta_ekin_cc)
```

4.4 Manipulate data

```
#coding:utf8

"""
This is an example of how to use the `PhaseSpace.data` object of p2sat.

It allows to generate and manipulate phase space
"""

# Import p2sat
p2sat_path="../"
import sys
if p2sat_path not in sys.path:sys.path.insert(0,p2sat_path)
import p2sat

# Boolean to export or not the generated phase space
export = False
check_input = True

# Instanciate a PhaseSpace object for electron specie
gps1 = p2sat.PhaseSpace(particle="gamma")

# Define energy and angle parameters
ekin_dict = {"law":"exp","ekin0":1.0}
theta_dict = {"law":"gauss","mu":0.,"sigma":5.}
phi_dict = {"law":"iso"}
x_dict = {"law":"gauss","mu":0.,"sigma":10.}
r_dict = {"law":"gauss","mu":0.,"sigma":50.}
```

```
t_dict = {"law":"exp","t0":150.}

# Generate particle phase space
gps1.data.generate(Nconf = 1e4, Npart = 1e12,
                   ekin=ekin_dict, theta=theta_dict, phi=phi_dict,
                   x=x_dict, r=r_dict, t=t_dict)

# Look at the consistency of phase space generation
if check_input:
    print(gps1)
    gps1.plot.figure(0)
    gps1.plot.h1('ekin',bwidth=.1,log=True)
    gps1.plot.f1('ekin',func_name="exp",log=True,bwidth=.1)

    gps1.plot.figure(1)
    gps1.plot.h2('theta','phi',log=True,
                 bwidth1=.5,bwidth2=1.)

    gps1.plot.figure(2)
    gps1.plot.h1('r',bwidth=1)
    gps1.plot.f1('r',func_name="gauss",bwidth=1)

# Copy current PhaseSpace in a new object
gps2 = gps1.copy()
# Rotate and translate phase spaces
gps1.data.transformate(T=(-200.,0.,0.),R=(0.,0.,45.))
gps2.data.transformate(T=(200.,0.,0.),R=(0.,0.,180.),rotate_first=True)

# Propagate to a given time
gps1.data.propagate(t=300.)
gps2.data.propagate(t=300.)

# Combine the 2 previous PhaseSpace
gps = gps1 + gps2

# Plot results
if check_input:
    gps.plot.figure(3)
    gps.plot.h2('x','px',bwidth1=1.,bwidth2=.1,log=True)
    gps.plot.figure(4)
    gps.plot.h2('x','y',bwidth1=1.,bwidth2=1.,log=True)
    gps.plot.figure(5)
    gps.plot.h3('x','y','z',wmin=1e6,s=10,log=True)

# Export phase space if needed
if export:
    # Discretize phase space, to limitate disk usage
    bwidth=[1.,2.,2.,1.,1.,1.]
    bwidth = None
    gps.data.discretize(with_time=False,split=2,MP=False,bwidth=bwidth)
    gps.export.txt("test_gps.csv",sep=",")
```

PYTHON MODULE INDEX

p

INDEX

T

U

V

W

X

Y

Z