p2sat Documentation

Release 1.1.2

lesnat

September 20, 2018

CONTENTS

1 Introduction **3**

2 Installation **5**

3 Use **7**

3.1 PhaseSpace 7
3.2 PhaseSpace.data 7
3.3 PhaseSpace.hist 11
3.4 PhaseSpace.plot 14
3.5 PhaseSpace.extract 16
3.6 PhaseSpace.export 18
3.7 PhaseSpace.stat 19

4 Examples **21**

4.1 Make histo 21
4.2 Make plots 21
4.3 Make statistics 22

Index **25**

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

CHAPTER TWO

INSTALLATION

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

Its only dependancies are python packages *numpy* and *matplotlib*.

Download and extract the source code, and use the following lines at the beginning of your script

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

CHAPTER

THREE

USE

PhaseSpace

PhaseSpace.data

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

Class containing raw data and methods to manipulate it.

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
- **`r`** (*`numpy.ndarray`*) – absolute distance to the x axis in um
- **`p`** (*`numpy.ndarray`*) – absolute momentum in MeV/c
- **`ekin`** (*`numpy.ndarray`*) – kinetic energy (for massive species) or total energy (for massless species) in MeV
- **`gamma`** (*`numpy.ndarray`*) – Lorentz factor
- **`beta`** (*`numpy.ndarray`*) – normalized velocity
- **`v`** (*`numpy.ndarray`*) – velocity in um/fs
- **`theta`** (*`numpy.ndarray`*) – polar angle (between px and the plane (py,pz)) in degree
- **`phi`** (*`numpy.ndarray`*) – azimutal angle (between py and pz) in degree

Notes

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

Definitions :

•$r = \sqrt{y^2 + z^2}$

•$p = \sqrt{p_x^2 + p_y^2 + p_z^2}$

•$\theta = \arccos p_x/p$

- $\phi = \arctan p_z/p_y$
- For massive species (with mass m)
  - $E_{kin} = (\sqrt{(p/mc)^2 + 1} - 1) \times mc^2$
  - $\gamma = E_{kin}/mc^2 + 1$
  - $\beta = \sqrt{1 - \frac{1}{\gamma^2}}$
  - $v = \beta \times c$
- For massless species
  - $E_{kin} = p$
  - $\gamma = +\infty$
  - $\beta = 1$
  - $v = c$

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

References

For ekin, gamma, beta, v : https://en.wikipedia.org/wiki/Lorentz_factor

For theta, phi : 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

`discretize`(*with_time=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()`

`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. An optional keyword 'max' can be given to specify a maximum angle
- 'gauss', for a gaussian spreading. Center of the distribution must be given with keyword 'mu', and standard deviantion with keyword 'sigma'

Details of the calculations :

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

We also have $E_T^2 = p^2 + E_m^2$ 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

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

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"})
...
```

lorentz(*beta_CM*, *verbose=True*)

Lorentz-transformate the particle phase-space with given speed of the center of mass.

TODO

References

https://en.wikipedia.org/wiki/Lorentz_transformation#Transformation_of_other_quantities

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

Parameters

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

Notes

x_pos and time can not be defined simultaneously.

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

Parameters

- **axis** (*str or numpy.ndarray*) – axis to filter
- **faxis** (*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 (*translation=(0.0, 0.0, 0.0)*, *rotation=(0.0, 0.0)*)
Transformate the particle phase space with given translation and rotation.

TODO

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

PhaseSpace.hist

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

Create histograms from raw data.

`f1`(*axis*, *func_name*, *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*, *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*, *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*, *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.

TODO

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*, *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 10 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()`

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
- **`rc`** (*`dict`*) – shortcut to matplotlib.rcParams dictionnary (change plot appearence)

`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*, *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*, *snorm=1.0*, *hmin=0.0*, ***kargs*)
Plot the 3d histogram of given axes.

TODO

Parameters

- **axis1,axis2,axis3** (*str*) – Name of the axes to plot
- **kargs** (*dict, optional*) – Dictionnary to pass to the hist.h2 method

See also:

`hist.h3()`

s2(*axis1*, *axis2*, *log=False*, *polar=False*, *select=None*)
Plot the 2d scattering plot 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
- **select** (*dict, optional*) – select dictionnary as in the hist.h2 method

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

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

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

Parameters

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

Examples

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

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 specie *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()`

PhaseSpace.export

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

Export phase space into several file format.

`TrILEns_input`(*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 '#'.

PhaseSpace.stat

class `p2sat._Stat.`**`_Stat`**(*PhaseSpace*)
Calculate statistics.

`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*, *p=None*, *select=None*)
Returns expected value of given axis.

Parameters

- **`axis`** (*`str or np.array`*) – axis to consider
- **`p`** (*`float, optional`*) – probability
- **`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

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

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.append(p2sat_path)
import p2sat

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

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.append(p2sat_path)
import p2sat

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

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.append(p2sat_path)
import p2sat

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

Symbols

_Data (class in p2sat._Data), 7
_Export (class in p2sat._Export), 18
_Extract (class in p2sat._Extract), 16
_Hist (class in p2sat._Hist), 11
_Plot (class in p2sat._Plot), 14
_Stat (class in p2sat._Stat), 19

C

c2() (p2sat._Plot._Plot method), 14
clear() (p2sat._Plot._Plot method), 14
correlation_coefficient() (p2sat._Stat._Stat method), 19
covariance() (p2sat._Stat._Stat method), 19

D

discretize() (p2sat._Data._Data method), 8

E

expected_value() (p2sat._Stat._Stat method), 19

F

f1() (p2sat._Hist._Hist method), 11
f1() (p2sat._Plot._Plot method), 14
figure() (p2sat._Plot._Plot method), 15

G

Geant4_csv() (p2sat._Extract._Extract method), 16
generate() (p2sat._Data._Data method), 8
get_labels() (p2sat._Plot._Plot method), 15
gp3m2_csv() (p2sat._Extract._Extract method), 17
gp3m2_input() (p2sat._Export._Export method), 18

H

h1() (p2sat._Hist._Hist method), 11
h1() (p2sat._Plot._Plot method), 15
h2() (p2sat._Hist._Hist method), 12
h2() (p2sat._Plot._Plot method), 15
h2h1() (p2sat._Plot._Plot method), 16
h3() (p2sat._Hist._Hist method), 12
h3() (p2sat._Plot._Plot method), 16
hn() (p2sat._Hist._Hist method), 13

L

lorentz() (p2sat._Data._Data method), 9

P

propagate() (p2sat._Data._Data method), 10

S

s2() (p2sat._Plot._Plot method), 16
s2h1() (p2sat._Plot._Plot method), 16
select() (p2sat._Data._Data method), 10
set_title() (p2sat._Plot._Plot method), 16
Smilei_Screen_1d() (p2sat._Extract._Extract method), 17
standard_deviation() (p2sat._Stat._Stat method), 20

T

transformate() (p2sat._Data._Data method), 10
TrILEns_input() (p2sat._Export._Export method), 18
TrILEns_output() (p2sat._Extract._Extract method), 17
txt() (p2sat._Export._Export method), 18
txt() (p2sat._Extract._Extract method), 17

U

update() (p2sat._Data._Data method), 10

V

variance() (p2sat._Stat._Stat method), 20