[2]p()*1/2-^ [1]p1-^ TJ 3
10000pt

0.00.5em

0.0.00.5em

0.0.0.00.5em

0.0.0.0.00.5em

0.0.0.0.0.00.5em

p2sat Documentation

Release 2.0.1

lesnat

Mar 31, 2020

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 :

InstallationInstallation 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.

UseUse

3.1 PhaseSpace

3.2 PhaseSpace.read

class p2sat.datasets._ReadPhaseSpace.**_ReadPhaseSpace**(*PhaseSpace*)

Read the dataset.

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 *metadata*.

property R

Particle distance to the origin.

Notes

R is calculated as follow :

$$R = \sqrt{x^2 + y^2 + z^2}$$

property 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

property betax

Particle normalized velocity along x axis.

Notes

betax is calculated as follow :

$$\beta_x = \sqrt{1 - \frac{1}{\gamma_x^2}}$$

References

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

property betay

Particle normalized velocity along y axis.

Notes

betay is calculated as follow :

$$\beta_y = \sqrt{1 - \frac{1}{\gamma_y^2}}$$

References

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

property betaz

Particle normalized velocity along z axis.

Notes

betaz is calculated as follow :

$$\beta_z = \sqrt{1 - \frac{1}{\gamma_z^2}}$$

References

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

property costhetax

Particle polar angle for reference axis x (angle between px and the plane (py,pz)).

Notes

costhetax is calculated as follow :

$$\cos(\theta_x) = p_x/p$$

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

property costhetay
Particle polar angle for reference axis y (angle between py and the plane (pz,px)).

Notes

costhetay is calculated as follow :

$$\cos(\theta_y) = p_y/p$$

References

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

property costhetaz
Particle polar angle for reference axis z (angle between px and the plane (py,pz)).

Notes

costhetaz is calculated as follow :

$$\cos(\theta_z) = p_z/p$$

References

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

property dataset
???

property ekin
Particle kinetic energy.

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

property etot
Particle total energy.

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

property 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

property gammax
Particle Lorentz factor along x axis.

Notes

gammax is calculated as follow :

$$\gamma_x = \sqrt{1 + (p_x/m_0)^2}$$

with m_0 being the rest mass energy

References

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

property gammay
Particle Lorentz factor along y axis.

Notes

gammay is calculated as follow :

$$\gamma_y = \sqrt{1 + (p_y/m_0)^2}$$

with m_0 being the rest mass energy

References

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

property gammaz
Particle Lorentz factor along z axis.

Notes

gammaz is calculated as follow :

$$\gamma_z = \sqrt{1 + (p_z/m_0)^2}$$

with m_0 being the rest mass energy

References

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

property id
Particle index.

property m
Particle relativistic mass.

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

property omegax
Minimum solid angle in which the particle is contained (in sr).

Notes

omegax is calculated as follow :

$$2\pi(1 - \cos\theta_x)$$

References

https://en.wikipedia.org/wiki/Solid_angle#Cone,_spherical_cap,_hemisphere

property omegay
Minimum solid angle in which the particle is contained (in sr).

Notes

omegay is calculated as follow :

$$2\pi(1 - \cos\theta_y)$$

References

https://en.wikipedia.org/wiki/Solid_angle#Cone,_spherical_cap,_hemisphere

property omegaz
Minimum solid angle in which the particle is contained (in sr).

Notes

omegaz is calculated as follow :

$$2\pi(1 - \cos\theta_z)$$

References

https://en.wikipedia.org/wiki/Solid_angle#Cone,_spherical_cap,_hemisphere

property p
Particle absolute momentum.

Notes

p is calculated as follow :

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

property particles
???

property phix
Particle azimutal angle for reference axis x (angle between py and pz).

Notes

phix is calculated as follow :

$$\phi_x = \arctan p_z/p_y$$

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

property phiy

Particle azimutal angle for reference axis y (angle between pz and px).

Notes

phiy is calculated as follow :

$$\phi_y = \arctan p_x/p_z$$

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->pz, y->px, z->py). A figure can be found at https://en.wikipedia.org/wiki/Spherical_coordinate_system

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

property phiz

Particle azimutal angle for reference axis z (angle between px and py).

Notes

phiz is calculated as follow :

$$\phi_z = \arctan p_z/p_y$$

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->px, y->py, z->pz). A figure can be found at https://en.wikipedia.org/wiki/Spherical_coordinate_system

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

property px

Particle momentum in direction x.

property py

Particle momentum in direction y.

property pz

Particle momentum in direction z.

quantity (*qty*, *select=None*)
Return the values of given quantity.

Parameters

- **qty** (*str*) – Quantity name.
- **select** (*dict*) – Filtering dictionnary.

Examples

```
>>> eps = ExamplePhaseSpace()
>>> eps.read.values("x")
array([-4.26043957, -8.225104  ,  0.25424565, ..., -3.19180518])
>>> eps.read.values("thetax",select={'x':[0.,1.],'ekin':[0.511,None]})
array([4.85380308, 3.79207276, 0.23348689, 0.59771946, 1.02382589,
   1.65421016, 6.84567286, 4.75129691, 3.82330291, 4.15075404,
   7.23677374, 2.69007983])
```

property rx
Particle distance to axis x.

Notes

rx is calculated as follow :

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

property ry
Particle distance to axis y.

Notes

ry is calculated as follow :

$$r_y = \sqrt{x^2 + z^2}$$

property rz
Particle distance to axis z.

Notes

rz is calculated as follow :

$$r_z = \sqrt{x^2 + y^2}$$

property t
Particle time (in fs).

property thetax
Particle polar angle for reference axis x (angle between px and the plane (py,pz)).

Notes

thetax is calculated as follow :

$$\theta_x = \arccos p_x/p$$

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

property thetay

Particle polar angle for reference axis y (angle between py and the plane (pz,px)).

Notes

thetay is calculated as follow :

$$\theta_y = \arccos p_y/p$$

References

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

property thetaz

Particle polar angle for reference axis z (angle between px and the plane (py,pz)).

Notes

thetaz 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->px, y->py, z->pz). A figure can be found at https://en.wikipedia.org/wiki/Spherical_coordinate_system

property ux

Particle direction on x axis.

Notes

ux is calculated as follow :

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

property uy
Particle direction on y axis.

Notes

uy is calculated as follow :

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

property uz
Particle direction on z axis.

Notes

uz is calculated as follow :

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

property v
Particle velocity.

Notes

v is calculated as follow :

$$v = \beta c$$

References

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

property vx
Particle velocity along x axis.

Notes

vx is calculated as follow :

$$v_x = \beta_x c$$

References

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

property vy
Particle velocity along y axis.

Notes

vy is calculated as follow :

$$v_y = \beta_y c$$

References

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

property vz
Particle velocity along z axis.

Notes

vz is calculated as follow :

$$v_z = \beta_z c$$

References

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

property w
Particle statistical weight.

property wekin
Particle kinetic energy density.

Notes

wekin is calculated as follow :

$$E_{kin} \times w$$

property x
Particle position in axis x.

property y
Particle position in axis y.

property z
Particle position in axis z.

3.3 PhaseSpace.data.edit

class p2sat.datasets._EditPhaseSpace.**_EditPhaseSpace**(*PhaseSpace*)
Edit the dataset.

filter(*select*, *verbose=True*)
Filter all the phase space with given condition

Parameters

- **select** (*dict*) – filtering dictionary
- **verbose** (*bool, optional*) – verbosity

Examples

```
>>> eps = ExamplePhaseSpace()
>>> eps.edit.filter(select={'x':[-5.,5.],'r':[0,10],'t':[150,None]})
Filtering e- phase space with axes ['x', 'r', 't'] ...
Done !
Updating raw values ...
Done !
>>> eps.read.ekin
array([1.14810454e-01, 1.78725015e+00, 6.30877382e-01, ..., 3.37300348e+00])
```

generate(*Nmp*, *Np*, *propagation_axis*, *ekin*, *phi=None*, *costheta=None*, *x=None*, *y=None*, *z=None*, *t=None*, *seed=None*, *verbose=True*)

Generate a particle phase space from given functions.

Parameters

- **Nmp** (*int*) – Number of macro-particles (number of lines in output file).
- **Np** (*float*) – Total number of particles to represent (sum of all the weights).
- **propagation_axis** (*str*) – Propagation axis. Must be 'x', 'y' or 'z'.
- **ekin** (*function*) – Function to call to generate the kinetic energy of macro-particles.
- **costheta** (*function, optional*) – Function to call to generate the cosinus of theta angle of macro-particles. Default is 1 (along propagation_axis).
- **phi** (*function, optional*) – Function to call to generate the phi angle (in radians) of macro-particles. Default is uniform between 0 and 2*pi (isotropic).
- **y, z** (*x,*) – Function to call to generate the positions of macro-particles. Default is 0.
- **t** (*function, optional*) – Function to call to generate the time of macro-particles. Default is 0.

Notes

All the functions must return values in user units.

Examples

```
>>> import numpy as np
>>> eps = ExamplePhaseSpace()
>>> eps.edit.generate(Nmp=1000, Np=1e12, propagation_axis="x", ekin=lambda:np.
↪random.exponential(scale=2.))
Generating phase space ...
Done !
Updating raw values ...
Done !
```

merge(*method='hist'*, *verbose=True*, ***kargs*)

Eliminate twin configurations by summing their weights.

merge ? compress ? merge_particles ?

Parameters

- **algo** (*{'bf','sort'}, optional*) – Algorithm to use. For 'sort' the phase space is assumed to be already sorted.
- **verbose** (*bool, optional*) – verbosity

Notes

The 'bf' algorithm stands for brute force. The 'sort' algorithm sort the values before merging.

References

https://en.wikipedia.org/wiki/Packing_problems https://en.wikipedia.org/wiki/Set_cover_problem

propagate_x(*x*, *verbose=True*)
Propagate the phase space to a given position x.

Parameters

- **x** (*float*) – Propagate the phase-space to position x (in user unit).
- **verbose** (*bool, optional*) – Verbosity.

rebin(*Dx*, *Dy*, *Dz*, *Dpx*, *Dpy*, *Dpz*, *Dt*, *nbins=100*, *MP=False*, *brange=None*, *deduplicate=False*, *verbose=True*)
Rebin all the phase space

Parameters

- **nbins** (*int or list of 7 int*) – number of bins to use
- **deduplicate** (*bool, optional*) – call or not the deduplicate_ps function
- **verbose** (*bool, optional*) – verbosity

See also:

`rebin_axis()`, `deduplicate_ps()`

Examples

```
>>> eps = ExamplePhaseSpace()
>>> eps.data.rebin_ps(nbins=10,deduplicate=True)
>>> eps.data.raw.w
[...]
```

rotate_x(*angle*, *verbose=True*)
Rotate the particle phase space along axis x.

Parameters

- **angle** (*float, optional*) – rotate (x,y,z) and (px,py,pz) of given angle along x axis.
- **verbose** (*bool, optional*) – verbosity

References

https://en.wikipedia.org/wiki/Rotation_matrix#In_three_dimensions

rotate_y(*angle*, *verbose=True*)
Rotate the particle phase space along axis y.

Parameters

- **angle** (*float, optional*) – rotate (x,y,z) and (px,py,pz) of given angle along y axis.
- **verbose** (*bool, optional*) – verbosity

References

https://en.wikipedia.org/wiki/Rotation_matrix#In_three_dimensions

rotate_z(*angle*, *verbose=True*)
Rotate the particle phase space along axis z.

Parameters

- **angle** (*float, optional*) – rotate (x,y,z) and (px,py,pz) of given angle along z axis.
- **verbose** (*bool, optional*) – verbosity

References

https://en.wikipedia.org/wiki/Rotation_matrix#In_three_dimensions

sort(*verbose=True*)
Sort the particles to have increasing number on x, then on y, then on z, ...

translate(*Tx=0.0*, *Ty=0.0*, *Tz=0.0*, *Tt=0.0*, *in_code_units=False*, *verbose=True*)
Translate the particle phase space.

Parameters

- **Tt** (*Tx, Ty, Tz,*) – Translate (x,y,z,t) position of Tx,Ty,Tz,Tt. Default is (0,0,0,0).
- **in_code_units** (*bool, optional*) – Specify whether the Ti are in code or user units. Default is in user units.
- **verbose** (*bool, optional*) – Verbosity.

update(*w*, *x*, *y*, *z*, *px*, *py*, *pz*, *t*, *in_code_units*, *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

3.4 PhaseSpace.load

class p2sat.datasets._LoadPhaseSpace.**_LoadPhaseSpace**(*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.edit.update*)
- Please write a documentation and share !

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

Smilei_Screen_1d(*path*, *nb*, *r*, *x=0*, *verbose=True*)

Extract phase space from Smilei 1D Screen diagnostic.

Parameters

- **path** (*str*) – path to the simulation folder
- **nb** (*int*) – Screen number
- **r** (*float*) – typical radius to consider in transverse direction (in um)
- **x** (*float, optional*) – diagnostic position
- **verbose** (*bool, optional*) –

Notes

On a 1D Smilei simulation, a typical DiagScreen must be declared as follows

```
DiagScreen(
    shape               = 'plane',
    point               = [xtarget[1] - 5*um],
    vector              = [1.],
    direction           = 'forward',

    deposited_quantity  = 'weight',
    species             = ['e'],
    axes                = [
         ['px' , pmin     , pmax     , 301],
         ['py' , -pmax/5  , pmax/5   , 301]
    ],

    every               = every
)
```

Smilei_TrackParticles(*path*, *species*, *dscale=1.0*, *verbose=True*)

Extract phase space from a TrackParticles Smilei diagnostic.

Parameters

- **path** (*str*) – path to the simulation folder
- **species** (*str*) – name of the specie in the Smilei namelist
- **dscale** (*float*) – Typical diameter to consider in the transverse direction if needed (in 1D or 2D). Should be given in meters.
- **verbose** (*bool, optional*) – verbosity

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

Parameters

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

TrILEns_prop_ph(*path*, *verbose=True*)
Extract simulation results from a TrILEns prop_ph file

Parameters

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

gp3m2_csv(*base_name*, *path='./'*, *thread=None*, *multiprocessing=False*, *in_code_units=False*, *verbose=True*)
Extract simulation results from a gp3m2 NTuple csv output file

Parameters

- **base_name** (*str*) – base file name
- **path** (*str*) – path to the simulation folder
- **thread** (*int, optional*) – number of the thread to import. By default it get the data of all the threads
- **multiprocessing** (*bool, optional*) – use or not the multiprocessing to paralelize import. Incompatible with thread != None.
- **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 = ExamplePhaseSpace()
>>> # eps.extract.gp3m2_csv("Al_target")
```

txt(*file_name*, *in_code_units*, *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:

`save.txt()`

3.5 PhaseSpace.save

class p2sat.datasets._SavePhaseSpace.**_SavePhaseSpace**(*PhaseSpace*)

Export phase space into several file format.

TrILEns_input(*path*, *source_1*, *source_2*, *lvl_diag=7*, *random_distrib=False*, *stat_col=True*, *chi_to_time=True*, *normalized_nb=False*, *pasdt=1.0*, *max_lvl_kd_tree=18*, *max_lvl_octree=5*, *max_number_swarms=20*, *threshold_merging=10*, *lvl_spatial=5*, *lvl_momentum=5*, *maillage_spatial=None*, *nb_max_par_espece=2000000*, *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 '#'.

3.6 PhaseSpace.metadata

class p2sat.datasets._MetadataPhaseSpace.**_MetadataPhaseSpace**(*specie*, *unit_system*)

3.7 hist

Create histograms from raw data.

p2sat.hist.**hist1d**(*ds*, *qty*, *weight='w'*, *bwidth=None*, *brange=None*, *normed=True*, *select=None*)

Create and return the 1 dimensional histogram of given quantity.

Parameters

- **ds** (*{PhaseSpace, EventLocation}*) – Dataset to use.
- **qty** (*str or np.array*) – qty to hist
- **bwidth** (*float, optional*) – bin width. If None, a calculation is done to have 10 bins in the qty
- **brange** (*list of 2 float, optional*) – bin maximum and minimum. If a brange element is None, the qty 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 hist1d method is just a different way to call the generic method histNd

See also:

`hist.histNd()`, `hist.hist2d()`, `hist.hist3d()`

p2sat.hist.**hist2d**(*ds*, *qty1*, *qty2*, *weight='w'*, *bwidth1=None*, *bwidth2=None*, *brange1=None*, *brange2=None*, *normed=True*, *select=None*)

Create and return the 2 dimensional histogram of given quantity.

Parameters

- **ds** (*{PhaseSpace, EventLocation}*) – Dataset to use.
- **qty1,qty2** (*str or np.array*) – qty to hist
- **bwidth1,bwidth2** (*float, optional*) – bin width. If None, a calculation is done to have 10 bins in the qty
- **brange1,brange2** (*list of 2 float, optional*) – bin maximum and minimum. If a brange element is None, the qty 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 hist2d method is just a different way to call the generic method histNd

See also:

`hist.histNd(),hist.hist1d(),hist.hist3d()`

p2sat.hist.**hist3d**(*ds*, *qty1*, *qty2*, *qty3*, *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 quantity.

Parameters

- **ds** (*{PhaseSpace, EventLocation}*) – Dataset to use.
- **qty1,qty2,qty3** (*str or np.array*) – qty to hist
- **bwidth1,bwidth2,bwidth3** (*float, optional*) – bin width. If None, a calculation is done to have 10 bins in the qty
- **brange1,brange2,brange3** (*list of 2 float, optional*) – bin maximum and minimum. If a brange element is None, the qty 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 hist3d method is just a different way to call the generic method histNd

See also:

`hist.histNd(),hist.hist1d(),hist.hist2d()`

p2sat.hist.**histNd**(*ds*, *qty*, *weight='w'*, *bwidth=None*, *brange=None*, *normed=True*, *select=None*)

Create and return the n-dimensional histo of qty list.

Parameters

- **ds** (*{PhaseSpace, EventLocation}*) – Dataset to use.
- **qty** (*list of str/np.array*) – List of qty 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 qty.
- **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 qty 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 qty as dict values.

Examples

```
>>> eps = ExamplePhaseSpace()
>>> w,x = eps.hist.histNd(['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.histNd(['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()`

3.8 plot

Plot raw data.

ivar ~p2sat.plot.autoclear

vartype ~p2sat.plot.autoclear bool

ivar True to auto clear figure when calling a plot method. Default is False

ivar ~p2sat.plot.cmap

vartype ~p2sat.plot.cmap str

ivar color map to use in 2d plot. Default is viridris

ivar ~p2sat.plot.rcParams

vartype ~p2sat.plot.rcParams dict

ivar shortcut to matplotlib.rcParams dictionnary (change plot appearence)

p2sat.plot.**arrow**()

p2sat.plot.**clear_figure**(*number=None*)

Clear a plot.

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

p2sat.plot.**figure**(*number=None*, *clear=False*)

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()

p2sat.plot.**get_labels**(*ds*, *qties*, *weight*, *normed*)

Returns the labels of given qties.

Parameters

- **ds** (*{PhaseSpace, ScalarField, EventLocation}*) – Dataset to use.
- **qties** (*list of str*) – Names of the qties
- **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 qty and label of weight

Return type list of str

p2sat.plot.**hist1d**(*ds*, *qty*, *where='post'*, *legend=''*, *log=False*, *polar=False*, *reverse=False*, *clear=False*, ***kargs*)

Plot the 1d histogram of given qty.

Parameters

- **ds** (*{PhaseSpace, ScalarField, EventLocation}*) – Dataset to use.

- **qty** (*str*) – Name of the qty to plot
- **where** (*str, optional*) – ...
- **log** (*bool, optional*) – True to set log scale on y qty
- **polar** (*bool, optional*) – True to use a polar plot. qty must be an angle
- **reverse** (*bool, optional*) – True to plot qty against number instead of number against qty
- **clear** (*bool, optional*) – Clear or not the figure before plotting
- **kargs** (*dict, optional*) – Dictionnary to pass to the hist.hist1d method

See also:

`hist.hist1d()`

p2sat.plot.**hist2d**(*ds*, *qty1*, *qty2*, *log=False*, *polar=False*, *clear=False*, ***kargs*)
Plot the 2d histogram of given qty.

Parameters

- **ds** (*{PhaseSpace, ScalarField, EventLocation}*) – Dataset to use.
- **qty1,qty2** (*str*) – Name of the qty to plot
- **log** (*bool, optional*) – True to set log scale on y qty
- **polar** (*bool, optional*) – True to use a polar plot. qty1 must be an angle
- **clear** (*bool, optional*) – Clear or not the figure before plotting
- **kargs** (*dict, optional*) – Dictionnary to pass to the hist.hist2d method

See also:

`hist.hist2d()`

p2sat.plot.**hline**(*y*, *text=''*, *xfactor=1.01*, *yfactor=1.01*, *linestyle='dashed'*, *color='k'*)

p2sat.plot.**set_title**(*title*, *number=None*)
Set the title of the figure.

p2sat.plot.**vline**(*x*, *text=''*, *xfactor=1.01*, *yfactor=1.01*, *linestyle='dashed'*, *color='k'*)

3.9 stat

Get global statistics from datasets.

p2sat.stat.**correlation_coefficient**(*ds*, *qty1*, *qty2*, *select=None*)
Returns correlation coefficient of given quantities.

Parameters

- **ds** (*{PhaseSpace, ScalarField, EventLocation}*) – Dataset to use.
- **qty1** (*str or np.array*) – Quantity to consider.
- **qty2** (*str or np.array*) – Quantity to consider
- **select** (*dict, optional*) – Filtering dictionary.

Notes

correlation_coefficient is defined as covariance(qty1,qty2)/(standard_deviation(qty1)*standard_deviation(qty2))

p2sat.stat.**covariance**(*ds*, *qty1*, *qty2*, *select=None*)
Returns covariance of given quantities.

Parameters

- **ds** (*{PhaseSpace, ScalarField, EventLocation}*) – Dataset to use.
- **qty1** (*str or np.array*) – Quantity to consider.
- **qty2** (*str or np.array*) – Quantity to consider
- **select** (*dict, optional*) – Filtering dictionary.

Notes

covariance is defined as expected_value((qty1-expected_value(qty1)) * (qty2-expected_value(qty2)))

p2sat.stat.**expected_value**(*ds*, *qty*, *select=None*)
Returns expected value of given quantity.

Parameters

- **ds** (*{PhaseSpace, ScalarField, EventLocation}*) – Dataset to use.
- **qty** (*str or np.array*) – Quantity to consider.
- **select** (*dict, optional*) – Filtering dictionary.

Notes

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

p2sat.stat.**peak_brilliance**(*ds*)

p2sat.stat.**standard_deviation**(*ds*, *qty*, *select=None*)
Returns standard deviation of given quantity.

Parameters

- **ds** (*{PhaseSpace, ScalarField, EventLocation}*) – Dataset to use.
- **qty** (*str or np.array*) – Quantity to consider.
- **select** (*dict, optional*) – Filtering dictionary.

Notes

standard_deviation is defined as the square root of variance

p2sat.stat.**total_charge**(*ds*, *unit='C'*, *select=None*)
Return total charge of the dataset.

Parameters

- **ds** (*{PhaseSpace}*) – Dataset to use.
- **unit** (*str, optional*) – unit of charge. Available are 'C' and 'nC', 'pC'. Default is 'nC'.

- **select** (*dict, optional*) – Filtering dictionary.

See also:

`data.select()`

p2sat.stat.**total_energy**(*ds*, *unit='J'*, *select=None*)

Return total energy contained in the dataset.

Parameters

- **ds** (*{PhaseSpace, ScalarField}*) – Dataset to use.
- **unit** (*str, optional*) – unit of energy. Available. are 'J' and 'MeV'. Default is 'J'
- **select** (*dict, optional*) – Filtering dictionary.

See also:

`data.select()`

p2sat.stat.**variance**(*ds*, *qty*, *select=None*)

Returns variance of given quantity.

Parameters

- **ds** (*{PhaseSpace, ScalarField, EventLocation}*) – Dataset to use.
- **qty** (*str or np.array*) – Quantity to consider.
- **select** (*dict, optional*) – Filtering dictionary.

Notes

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

4.1 Make histo

4.2 Make plots

4.3 Make statistics

4.4 Manipulate data

Python Module Index

p