p2sat Documentation

Release 1.0

lesnat

Aug 27, 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
- 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(specie="electron")
>>> eps.extract.txt("example.dat",sep=None)
```

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

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

See folder *examples/* or documentation to a more complete set of p2sat capabilities.

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

3.1 PhaseSpace

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

Base class for particle phase-space analysis.

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

Variables

- **`data`** (*`sub-object`*) – contains data data and methods to manipulate it, such as discretization or transformation.
- **`hist`** (*`sub-object`*) – make histograms from data 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

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

- **`w`** (*`numpy.ndarray`*) – particle statistical weight
- **`x,y,z`** (*`numpy.ndarray`*) – particle x,y,z position in um
- **`t`** (*`numpy.ndarray`*) – particle time in fs

- **r** (*numpy.ndarray*) – absolute distance to the x axis in um
- **px,py,pz** (*numpy.ndarray*) – particle momentum in x,y,z direction in MeV/c
- **p** (*numpy.ndarray*) – absolute particle momentum in MeV/c
- **ekin** (*numpy.ndarray*) – particle energy in MeV
- **theta** (*numpy.ndarray*) – angle between px and py in degree
- **phi** (*numpy.ndarray*) – angle between ??? in degree

Notes

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

Calculations :

- r is defined as $\sqrt{y^2 + z^2}$
- p is defined as $\sqrt{p_x^2 + p_y^2 + p_z^2}$
- theta is defined as $\arctan p_y/p_x$
- phi is defined (yet) as $\arctan p_z/p_x$
- ekin is defined as
  - $(\sqrt{(p/m_e c)^2 + 1} - 1) \times m_e c^2$ for massive species
  - p otherwise (here ekin is the total particle energy)
- gamma is defined as
  - $E_{kin}/m_e c^2 + 1$ for massive species
  - ... otherwise

Details of the calculations can be found at . . . TODO

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:

`hn()`

generate(***kargs*)
Generate a particle phase space from given laws

TODO

propagate()

Propagate the phase space to a given position or time.

TODO

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

It is possible to filter by an int value

```
>>> w = np.random.uniform(low=0.,high=10.,size=10)
>>> x = np.array([1,3,3,3,7,9,5,3,7,3])
>>> w = select(w,x,3) # Select all the w satisfying x==3
```

or filter by a range

```
>>> w = np.random.uniform(low=0.,high=10.,size=1000)
>>> ekin = np.random.exponential(scale=3.0,size=1000)
>>> w = select(w,ekin,[0.511,None]) # Select all the w with :math:`ekin \in␣
↪[0.511,+\infty] MeV`
```

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** (*list or numpy.ndarray*) – particle phase space. More information can be found in raw object documentation
- **verbose** (*bool*) – verbosity of the function. If True, a message is displayed when the attributes are loaded in memory
- **TODO** (*get np array to be immutable with x.writeable=False ?*) –

3.3 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*) – fit parameters

Notes

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

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

h1(*axis*, *bwidth=None*, *brange=None*, *wnorm=None*, *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
- **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:

hn(), *h2()*, *h3()*

h2 (*axis1*, *axis2*, *bwidth1=None*, *bwidth2=None*, *brange1=None*, *brange2=None*, *wnorm=None*, *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
- **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:

hn(), *h1()*, *h3()*

h3 (*axis1*, *axis2*, *axis3*, *bwidth1=None*, *bwidth2=None*, *bwidth3=None*, *brange1=None*, *brange2=None*, *brange3=None*, *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
- **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:

hn(), *h1()*, *h2()*

hn (*axis*, *blen=None*, *bwidth=None*, *brange=None*, *wnorm=None*, *select=None*)

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

Parameters

- **axis** (*list of str/np.array*) – list of axis to hist
- **blen** (*list of int, optional*) – list of number of bins. If a blen element is None, the default value is 10
- **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
- **wnorm** (*list of float, optional*) – weight normalization. If a wnorm element is None, the bin width is taken
- **select** (*dict, optional*) – filtering dictionary

Returns

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

Notes

TODO: If the given maximum bin range does not match with an int number of bins, the last bin is oversized ?? it reduce bwidth to fit brange[1]-brange[0] with a int nb of bins

If blen and bwidth are both defined, priority is given to blen.

Examples

```
>>> hn(['x'],bwidth=[50],brange=[[0,1000]],wnorm=[1.0],select={'ekin':(0.511,
↪None)})
```

returns the number of particles with $ekin \in [0.511, +\infty] MeV$ in function of x wnorm=[1.0] to not divide nb of particles by bin width (otherwise number per um)

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

returns a number of e- per um per MeV at x=150 um

3.4 PhaseSpace.plot

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

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:

`clear()`

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

Parameters

- **axes** (*list of str*) – Names of the axes
- **wnorm** (*float 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 : doc + kargs + delete labels on h2

`h3` (*axis1*, *axis2*, *axis3*, *snorm=1.0*, *hmin=0.0*, ***kargs*)
Plot the 3d histogram of given axes.

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*)

`set_title` (*title*)

3.5 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

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

Notes

The Geant4 NTuple format should be

```
w,x,y,z,px,py,pz,t
. . . . .  .  .  .
. . . . .  .  .  .
. . . . .  .  .  .
```

Examples

```
>>> eg = p2sat.PhaseSpaceGeant4()
>>> eg.extract("../Geant4/testem_nt_electron_t*.csv",nthreads=10)
```

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

TrILEns_output(*path*, *specie*, *verbose=True*)

Extract simulation results from a TrILEns output.txt file

Parameters

- **path** (`str`) – simulation path
- **specie** (`str`) – specie to find in the output. The specie name must be in plural form (i.e 'electrons' or 'positrons')

- **verbose** (*bool, optional*) – verbosity

Notes

…

Examples

```
>>> et = p2sat.PhaseSpaceTrILEns()
>>> et.extract("../TrILEns/",specie="positrons")
```

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()`

3.6 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

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*) – title of the file
- **sep** (*str, optional*) – character to use to separate values. Default is ','
- **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 '#'.

3.7 PhaseSpace.stat

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

Allows to do statistics with p2sat 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*, *p=None*, *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) with w being the statistical weight of the configuration

`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

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.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("input.tsv",sep=None,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
"""
```

(continues on next page)

(continued from previous page)

```
# 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("input.tsv",sep=None,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=10.0,bwidth2=10.0,
            brange1=[-500.,500.],brange2=[-500.,500.],
            select={'ekin':[0.511,None]})

# Add a contour plot
"""
eps.plot.c2('y','z',log=True,
            bwidth1=10.0,bwidth2=10.0,
            brange1=[-500.,500.],brange2=[-500.,500.],
            select={'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
```

(continues on next page)

(continued from previous page)

```python
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("input.tsv",sep=None,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)
```

PYTHON MODULE INDEX

INDEX