pymchelper Documentation

Release 1.10.0

Author

Nov 22, 2021

Contents

pymchelper is a toolkit for processing output files of particle transport codes, such as FLUKA or SHIELD-HIT12A. It provides a command line program, called **convertmc** which simplifies process of converting binary output files to graphs, tabulated plain data and MS Excel files and other formats. Toolkit can also serve as a library in the Python language, which can be used by programmers and data scientists to process output data in Python scripts.

CHAPTER 1

Getting Started

Brief overview of pymchelper and how to install it.

1.1 Introduction

Let us assume we are running particle transport simulation using Fluka MC code. Two estimators: particle fluence and deposited energy are defined in the input file. To get high statistics run is parallelised into 100 different jobs, each of them producing 2 binary files. That gives 200 binary output files, typically following `*_fort.*` pattern.

pymchelper comes with **convertmc** program which simplifies postprocessing of such binary data.

To convert all binary files into two text files (one for energy and one for particle fluence scorer) type

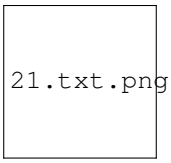
```
convertmc txt --many "*_fort.*"
```

pymchelper will automatically figure out how many scorers were defined and which files to merge. Two new files will be created: `21.txt` and `22.txt` which can further processed. By default they will contain 5 columns: X,Y,Z coordinates, data and error column:

```
0.0000000E+00  0.0000000E+00  0.5000000E-02  0.1881775795482099E-02  0.
↪2157818940293169E-04
0.0000000E+00  0.0000000E+00  0.1500000E-01  0.1893831696361303E-02  0.
↪2251415707395123E-04
0.0000000E+00  0.0000000E+00  0.2500000E-01  0.1887728041037917E-02  0.
↪1936414681456153E-04
0.0000000E+00  0.0000000E+00  0.3500000E-01  0.1897429465316236E-02  0.
↪1944586725074595E-04
```

In case estimated value was scored on 1-dimensional grid, it can be easily plotted. Same thing might be done if 2-D scoring grid was used - then a heatmap type plot can be produced. In order to get such plots instead of text files, replace first argument with `image`:

```
convertmc image --many "*_fort*"
```



Two new PNG files will be created which can be directly opened, for example

If you are SHIELD-HIT12A user, same effect can be achieved by processing `*.bdo` binary files.

These are just basic examples. To learn more about additional features, proceed to *User's Guide*. Among these features are:

- reading binary `*.bdo` files generated by SHIELD-HIT12A code
- reading binary `*_fort*` files generated by FLUKA
- calculating standard error when averaging many files
- writing PNG images (1 and 2D plots)
- writing tabulated text files
- writing gnuplot scripts

1.2 Quick Installation Guide

Be sure to have Python framework installed, then type:

```
pip install pymchelper
```

In case you don't have administrator rights, add `--user` flag to `pip` command. In this situation converter will be probably installed in `~/.local/bin` directory.

1.3 License

pymchelper is licensed under *MIT LICENCE*.

CHAPTER 2

User's Guide

2.1 Available converters

Detailed documentation about available converters:

2.1.1 Plain text

Plain text converter comes in two flavours: `txt` and `plotdata`. It accepts data with any dimension of scoring grid: 0-D (single number), 1-D, 2-D and 3-D.

txt converter

First one (`txt`) generates 4- or 5- column text files. After running: `convertmc txt proton0001_fort.21 dose.txt`, new file `dose.txt` will contain: X,Y,Z coordinates, data and error column:

```
0.0000000E+00  0.0000000E+00  0.5000000E-02  0.1881775795482099E-02  0.
↪2157818940293169E-04
0.0000000E+00  0.0000000E+00  0.1500000E-01  0.1893831696361303E-02  0.
↪2251415707395123E-04
0.0000000E+00  0.0000000E+00  0.2500000E-01  0.1887728041037917E-02  0.
↪1936414681456153E-04
0.0000000E+00  0.0000000E+00  0.3500000E-01  0.1897429465316236E-02  0.
↪1944586725074595E-04
(...)
```

When running `convertmc txt proton0001_fort.21 dose.txt --error none` fifth column will not be appended and file contents will be following:

```
0.0000000E+00  0.0000000E+00  0.5000000E-02  0.1881775795482099E-02
0.0000000E+00  0.0000000E+00  0.1500000E-01  0.1893831696361303E-02
0.0000000E+00  0.0000000E+00  0.2500000E-01  0.1887728041037917E-02
```

(continues on next page)

(continued from previous page)

```
0.0000000E+00  0.0000000E+00  0.3500000E-01  0.1897429465316236E-02
(...)
```

plotdata converter

In the examples shown above we could see that columns corresponding to X and Y axis contain fixed number (0.0). It comes from the fact that the 1-dimensional scoring grid spans along the Z axis. When plotting data from such file user needs to select third and fourth column.

convertmc can skip data of columns with fixed number and save to file only columns which contain plottable values. This can be achieved by running:

```
convertmc plotdata proton0001_fort.21 dose.txt
```

We will get a file containing following data - Z axis, data and error column:

```
0.5000000E-02  0.1881775795482099E-02  0.2157818940293169E-04
0.1500000E-01  0.1893831696361303E-02  0.2251415707395123E-04
0.2500000E-01  0.1887728041037917E-02  0.1936414681456153E-04
0.3500000E-01  0.1897429465316236E-02  0.1944586725074595E-04
(...)
```

Such file is easier to handle for plotting, as user can select only first and second column.

2.1.2 MS Excel file

MS Excel converters accepts data only with 1-D scoring. Data files with other scorers will not be converted.

An example usage

Conversion is done using standard command:

```
convertmc excel --many "*.bdo"
```

2.1.3 Plots and images

`image` converter is very useful tool to quickly inspect simulation results. It accepts data with any 1-D and 2-D scoring grid and is producing images in PNG format with plots. Conversion is done using standard command:

```
convertmc image --many "*.bdo"
```

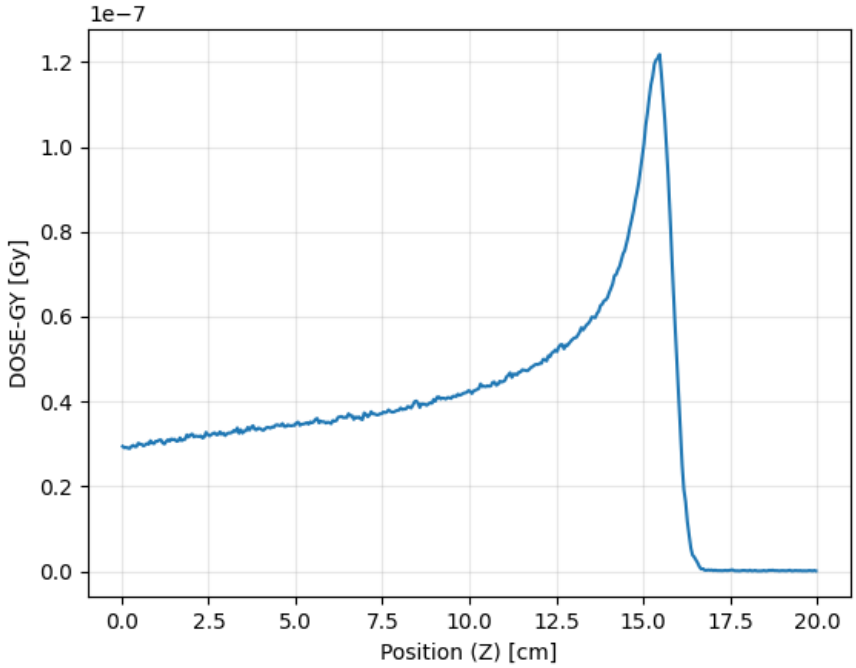
After converting data with 1-D scoring grid, following plot can be generated:

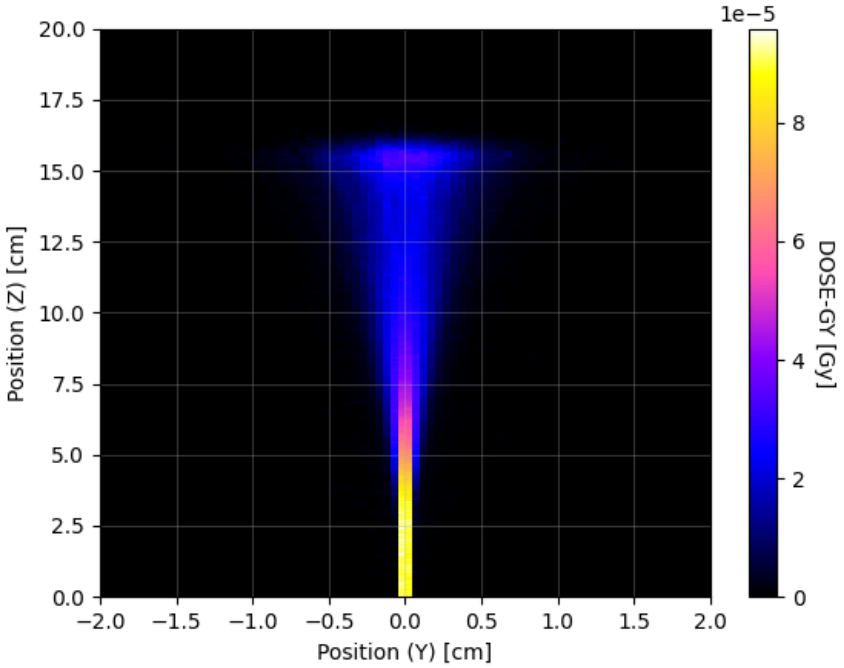
Data containing 2-D scoring grid are visualised as heatmap with color denoting scored value.

Options

Logarithmic scale

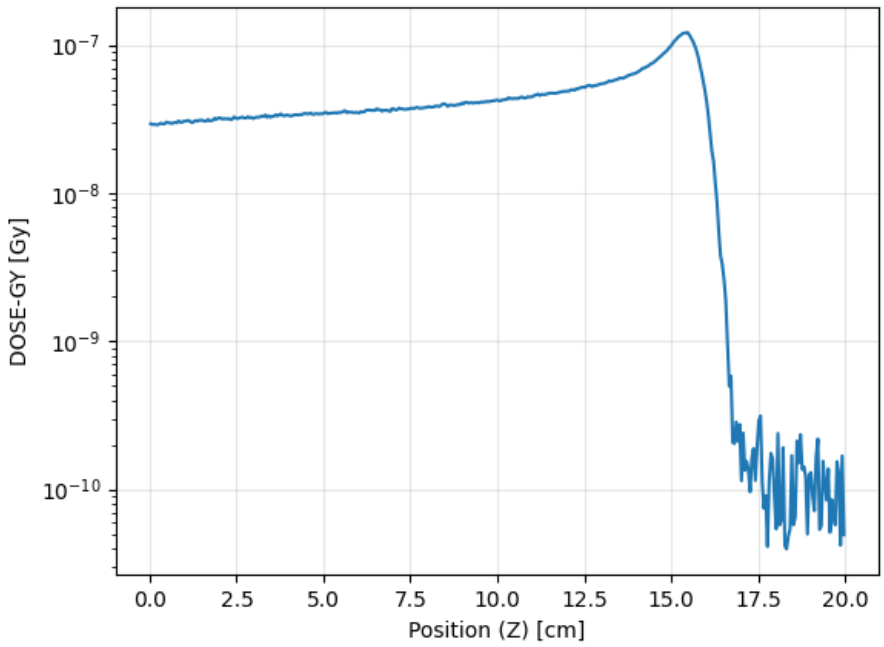
User can also set logscale on one or more axis in the plots using *–log* option.


1e−7
1.2
1.0
0.8
0.6
0.4
0.2
0.0
DOSE-GY [Gy]
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Position (Z) [cm]


1e−5
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
Position (Z) [cm]
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
Position (Y) [cm]
8
6
4
2
0
DOSE-GY [Gy]

An example plot with logarithmic scale on Y axis:

```
convertmc image --many "*.bdo" --log y
```



Scale can be also change on two axis at once:

```
convertmc image --many "*.bdo" --log x y
```

An example plot with 2-D heatmap and logarithmic scale on Z (color) axis:

```
convertmc image --many "*.bdo" --log z
```

Colormap

When generating 2D heatmaps it is also possible to specify colormap. List of available colormaps is available here: https://matplotlib.org/stable/tutorials/colors/colormaps.html. By default colormap called *gnuplot2* is used. An example plot obtained with other colormap (*Greys*) can be obtained with following command:

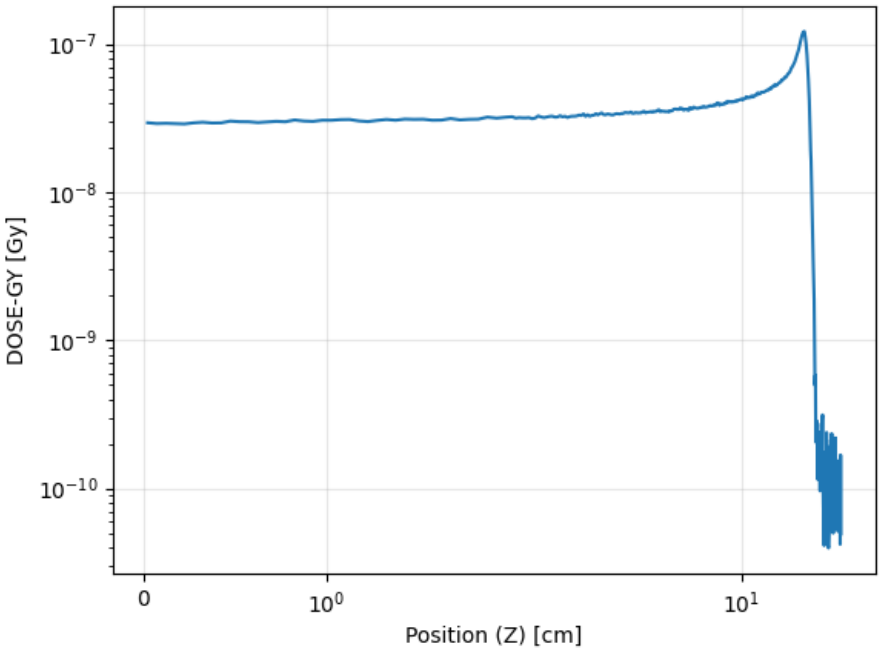
```
convertmc image --many "*.bdo" --colormap Greys
```

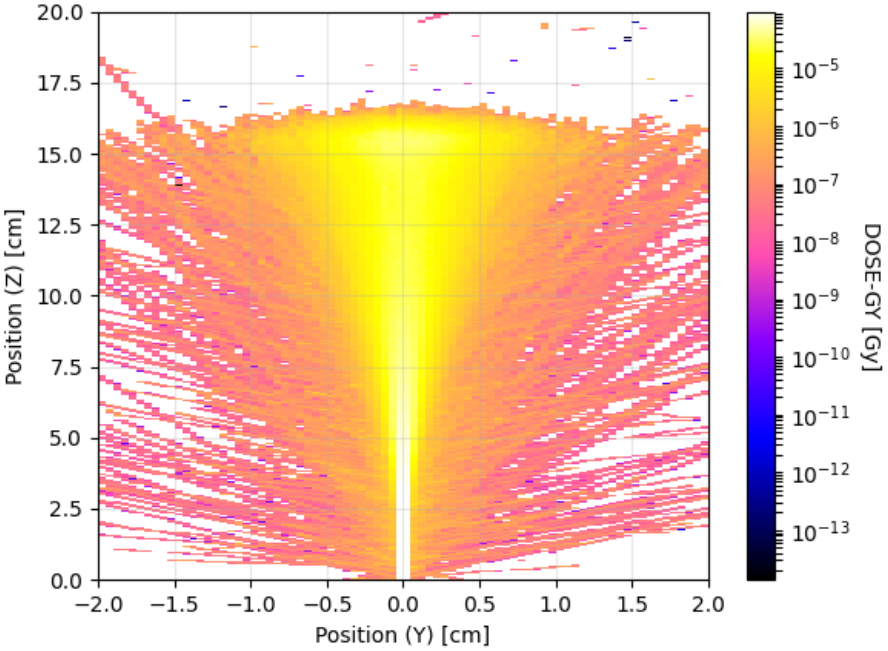
2.1.4 Gnuplot

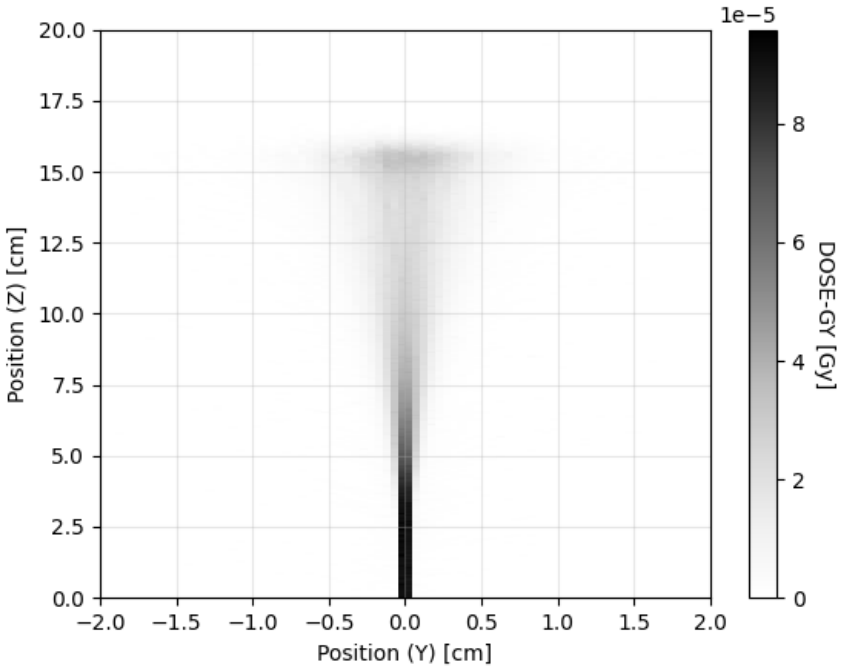
Gnuplot converter is producing scripts which can be used by gnuplot tool to produce plot. It accepts data with any 1-D and 2-D scoring grid. This converter might be useful for users who would like to adjust shape of the plots.

Data needed for plotting have to be generated using `plotdata` converter.

Let us start with generating some data first:


DOSE-GY [Gy]
10^{-7}
10^{-8}
10^{-9}
10^{-10}
0
10^{0}
10^{1}
Position (Z) [cm]


Position (Z) [cm]
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
Position (Y) [cm]
10^{-5}
10^{-6}
10^{-7}
10^{-8}
10^{-9}
10^{-10}
10^{-11}
10^{-12}
10^{-13}
DOSE-GY [Gy]




```
convertmc plotdata proton0001_fort.21
```

As expected we will get `proton0001_fort.dat` file. Now a gnuplot script can be generated:

```
convertmc gnuplot proton0001_fort.21
```

We will get a script `proton0001_fort.plot` with following content:

```
set term png
set output "proton0001_fort.png"
set title "dose_z_1"
set xlabel ""
set ylabel "dose_z_1 []"
max(x,y) = (x > y) ? x : y
plot  './proton0001_fort.dat' u 1:2 w l lt 1 lw 2 lc -1 title 'mean value'
```

After running `gnuplot proton0001_fort.plot` we will get a plot

proton0001_fort.png

2.1.5 Sparse matrix data

Sparse matrix format can greatly reduce size on disk in case most of the matrix is occupied with zeros. For every input detector a single file with `.npz` extension is written. It contains:

- shape of detector data: tuple of 3 numbers holding nx, ny and nz
- tuple of 3 numpy arrays with X, Y and Z coordinates, pointing to non-zero data cells
- plain, 1D numpy array with non-zero data values

Such format is similar to a so-called COO-rdinate format for 2D sparse matrices. Here it is used for any kind of data, with dimensionality between 0 and 3.

Output file is saved in uncompressed NPZ numpy data format, for details see https://docs.scipy.org/doc/numpy/reference/generated/numpy.savez.html

An example usage

Conversion is done using standard command:

```
convertmc sparse --many "*.bdo"
```

Assuming that we had "dose" detector output saved to `dose0001.bdo`, `dose0002.bdo` and `dose0003.bdo`, we should expect to get `dose.npz` as an output file.

Data reconstruction

Sparse data can be extracted from NPZ file with following Python code:

```
import numpy as np

# load contents of NPZ file:
npzfile = np.load(filename)
data = npzfile['data']    # plain 1-D numpy array
indices = npzfile['indices']  # 3-elements tuple with array of coordinates
shape = npzfile['shape'] # 3-element shape tuple

# reconstruct normal form of matrix from sparse data
result = np.zeros(shape)
for ind, dat in zip(indices, data):
    result[tuple(ind)] = dat
```

Threshold value

Additionally a threshold can be specified to discard noisy data. In this case, all data cells with absolute value less or equal than threshold will be treated as zeros. Please note that after reconstructing from sparse format we won't get the original data, as values lower than threshold will be overwritten with zeros. An example usage:

```
convertmc sparse --many "*.bdo" --threshold 1e-7
```

2.1.6 Inspect data

Inspector converter (not being in fact a true converter) is not producing any files, but prints on the screen metadata of the input binary file.

An example usage

As usual we can apply standard command:

```
convertmc inspect ex_dose.bdo
```

We should expect to see following output on the screen:

```
INFO:pymchelper.readers.shieldhit:Reading: ex_dose.bdo
dettyp                  : 'DOSE'
filedate                : 'Fri, 06 Oct 2017 23:05:49 +0200'
geotyp                  : 'CYL'
host                    : 'nz61-6'
mc_code_version         : '0.6.0-dev'
nstat                   : '10000'
nx                      : '1'
ny                      : '1'
nz                      : '1000'
pages                   : '1'
particle                : '-1'
particle_a              : '-1'
particle_z              : '-1'
title                   : 'Dose'
tripdose                : '0.0'
tripntot                : '-1'
units                   : '['cm', 'radians', 'cm', '(nil)', ' MeV/g/primary', 'Dose',
↪'Radius (R)', 'Angle (PHI)', 'Position (Z)', '', '']'
xmax                    : '10.0'
xmin                    : '0.0'
ymax                    : '6.28318530718'
ymin                    : '0.0'
zmax                    : '30.0'
zmin                    : '0.0'
zone_start              : '-1'
**************************************************************************
Data min: 3.04792e-05, max: 0.0715219
```

Advanced usage

Very simple ASCII-art plots can be also printed on the screen, assuming that two libraries: bashplotlib and hipsterplot will be installed first:

```
pip install bashplotlib hipsterplot
```

To activate plotting additional option has to be used:

```
convertmc inspect ex_dose.bdo --details
INFO:pymchelper.readers.shieldhit:Reading: ex_dose.bdo
dettyp                  : 'DOSE'
filedate                : 'Fri, 06 Oct 2017 23:05:49 +0200'
geotyp                  : 'CYL'
host                    : 'nz61-6'
mc_code_version         : '0.6.0-dev'
nstat                   : '10000'
nx                      : '1'
ny                      : '1'
```

(continues on next page)

(continued from previous page)

```
nz                      : '1000'
pages                   : '1'
particle                : '-1'
particle_a              : '-1'
particle_z              : '-1'
title                   : 'Dose'
tripdose                : '0.0'
tripntot                : '-1'
units                   : '['cm', 'radians', 'cm', '(nil)', ' MeV/g/primary', 'Dose',
↪'Radius (R)', 'Angle (PHI)', 'Position (Z)', '', '']'
xmax                    : '10.0'
xmin                    : '0.0'
ymax                    : '6.28318530718'
ymin                    : '0.0'
zmax                    : '30.0'
zmin                    : '0.0'
zone_start              : '-1'
************************************************************************
Data min: 3.04792e-05, max: 0.0715219
************************************************************************
Data scatter plot
    0.0691                                                          #:
    0.0644                                                          |.
    0.0596                                                         :.|
    0.0548                                                         # .
    0.0501                                                         # .
    0.0453                                                        #: .
    0.0405                                                       ##  :
    0.0358                                                      ##
    0.0310                                                  .###.    :
    0.0262                                          . #######        .
    0.0215                          #:.######################|        :
    0.0167 ########################## .                              .
    0.0119                                                           |
    0.0072                                                           :
    0.0024                                                           #########
************************************************************************
Data histogram

 106|  o
 100|  o                   o
  95|  o                 o o
  89|  o                 ooo
  84|  o                 ooo
  78|  o                 ooo
  73|  o                ooooo
  67|  o               oooooo
  62|  o               oooooo
  56|  o               oooooo
  50|  o               oooooo
  45|  o              ooooooo
  39|  o              ooooooo o
  34|  o              oooooooooo
  28|  o              ooooooooooo
  23|  o              ooooooooooo o
  17|  o              oooooooooooo oo
  12|  o              oooooooooooooooo
```

(continues on next page)

(continued from previous page)

```
   6|  o              ooooooooooooooooo o o     o
   1| ooooooo o o   oooooooooooooooooooooooo ooooooooooooooo ooo ooooooooooo o
     ------------------------------------------------------------------------


-------------------------------------
|              Summary              |
-------------------------------------
|         observations: 1000        |
|        min value: 0.000030        |
|          mean : 0.020207          |
|        max value: 0.071522        |
-------------------------------------
```

2.2 Common options

convertmc command line program needs several options to work. The first one, obligatory is converter name. User might choose among: `txt`, `excel`, `image`, `gnuplot`, `plotdata` and `sparse`.

All converters accepts following options:

2.2.1 Input files

It is obligatory to specify location of the entities to convert. It can be a single file or a wildcard expression (like `*.bdo`) specifying group of files.

To convert a single binary file `proton0001_fort.21` generated with Fluka to a text file you can use following command:

```
convertmc txt proton0001_fort.21
```

Converter will automatically figure out output filename `proton0001_fort.txt`

It is also possible to merge group of files into single one to obtain better statistics. Let us assume we have input files `proton0001_fort.21`, `proton0002_fort.21` and `proton0003_fort.21` corresponding to different runs but storing the output of the same scorer. To read data from these 3 files, average on-the-fly and save output to single file, type:

```
convertmc txt "proton0001_fort.*"
```

Again, a single file `proton0001_fort.txt` will be generated, containing averaged data from 3 binary input files.

Finally we could have many files, coming from many simulation runs and belonging to many different scorers. **convertmc** can read list of such files, automatically group them according to scorers. By adding `---many` flag we ask converter to scan group of files, discover sub-groups belonging to same converter and perform averaging in each of sub-groups. Finally some number of output files will be generated (one output file for one scorer). An example:

```
convertmc txt --many "proton0001_fort.*"
```

2.2.2 Output files

When working in single-file conversion mode or when merging group of files into a single file it is possible to provide a name of output file as a third argument:

```
convertmc txt proton0001_fort.21 dose.txt
```

When using `--many` option **convertmc** will generate many files. In this case user might provide an existing directory as a third parameter. All output files will be generated there:

```
convertmc txt --many "*_fort.*" /path/to/output/dir/
```

2.2.3 Scaling factor

Some of the scorers used in Monte-Carlo transport codes, such as dose or fluence dump the results per-primary-particle. For example dose is often reported in MeV/g/prim units. These units might be hard to interpret when doing dosimetric particle transport simulations. To report dose (or some other quantities) in more natural units knowledge of total particles in the experiment is necessary (not to be confused with number of simulated primaries). User can specify this number via option called `--nscale`. **convertmc** will then report dose (and some other quantities) in natural units and per user-provided particle number (not per single primary). Let us assume that `proton0001_fort.21` file was obtained by simulation of 10000 protons.

```
convertmc txt proton0001_fort.21 dose_per_prim.txt
```

will save a file `dose_per_prim.txt` with dose-per-primary and MeV/g/prim units. While:

```
convertmc txt proton0001_fort.21 dose.txt --nscale 1e8
```

will save a file `dose.txt` with rescaled dose of 10^8 protons in Gy units

2.2.4 Error calculation

In order to perform averaging converter needs to get at least two files per scorer. By default all **convertermc** during averaging will also calculate standard deviation, which can be presented to the user in various forms:

- additional column of numbers when saving data into text files (i.e. using `txt` converter)
- error band on the normal plot for 1-D scoring grid (when using `image` and `gnuplot` converters)
- additional heatmap error plot for 2-D scoring grid (when using `image` and `gnuplot` converters)

When working in single-file mode calculation of the startand deviation is not possible and user will not get such information. In this case output text files will have one column less and no error band is displayed on the plots.

User can control presented information by means of `--error` option. When this option is not specified **convertmc** will raport standard error if possible. Available options are:

- `none` - no uncertainty information is reported
- `stderr` - standard error information is reported
- `stddev` - standard deviation information is reported

2.2.5 Handling NaNs

In case of some simulations the results might include numbers decoded as NaN (not-a-number). NaN has this property that if a regular number is added to NaN then the results is again NaN (1.0 + NaN = NaN). This property might lead to a problems when averaging. Single NaN version will make a NaN when averaged with other 99 regular numbers. We have a special flag in **convertmc** which will exclude NaN from averaging: if `--nan` is used then average will be calculated only on regular numbers.

2.2.6 Obtaining help

To get a general help instructed printed on a screen type:

```
convertmc --help
```

or simply `convertmc -h`

Each of the converter comes with dedicated help page which can be printed using i.e.:

```
convertmc txt --help
```

2.2.7 Program version

To get program version type `convertmc --version` or `convertmc -V`

2.2.8 More verbose output

To get more verbose output use `-v` option (be careful, there is similar capital-V option for printing version number). Adding `-v` or `--verbose` will make the program printing more lines of comments explaining what is doing. By adding `-vv` even more logging output will be generated. Maximum logging level is obtained using `-vvv`.

2.2.9 Silent execution

If you do like the program printing any logging output on the screen, you can use `-q` or `--quiet` option. These options might be useful if your program is i.e. called repetitively in a script.

2.3 Using as a library

`pymchelper` is build around Detector class. I/O module provides `fromfile` method to get the data from binary file and is exposing to the user axis data, scoring information and data read from the file. See an example:

```
    parsed_args = parser.parse_args(args)

    # paths to binary files
    fluka_file_path = parsed_args.fluka
    sh12a_file_path = parsed_args.shield

    # creating empty Detector object and filling it with data read from Fluka file
    fluka_data = fromfile(fluka_file_path)

    # same as above but reading from SHIELD-HIT12A file
    sh12a_data = fromfile(sh12a_file_path)

    # printing some output on the screen
    print("Fluka bins in X: {:d}, Y: {:d}, Z: {:d}".format(fluka_data.x.n, fluka_data.
↪y.n, fluka_data.z.n))
    print("First bin of fluka data", fluka_data.pages[0].data[0, 0, 0, 0, 0])

    print("SHIELD-HIT12A bins in X: {:d}, Y: {:d}, Z: {:d}".format(sh12a_data.x.n,␣
↪sh12a_data.y.n, sh12a_data.z.n))
```

(continues on next page)

(continued from previous page)

```
18     print("First bin of SHIELD-HIT12A data", sh12a_data.pages[0].data[0, 0, 0, 0, 0])
19
```

CHAPTER 3

Detailed Installation Guide

Installation guide is divided in two phases: checking the prerequisites and main package installation.

3.1 Prerequisites

pymchelper works under Windows, Linux and Mac OSX operating systems.

First we need to check if Python interpreter is installed. Try if one of following commands (printing Python version) works:

```
python --version
python3 --version
```

At the time of writing Python language interpreter has two popular versions: 2.x (Python 2) and 3.x (Python 3) families. Command `python` invokes either Python 2 or 3, while `python3` can invoke only Python 3.

pymchelper supports most of the modern Python versions, mainly: 2.7, 3.4 - 3.10. Check if your interpreter version is supported.

If none of `python` and `python3` commands are present, then Python interpreter has to be installed.

We suggest to use the newest version available (from 3.x family).

Python installers can be found at the python web site (http://python.org/download/).

pymchelper also relies on these packages:

- NumPy – Better arrays and data processing.
- matplotlib – Needed for plotting, optional.

and if they are not installed beforehand, these will automatically be fetched by pip.

3.2 Installing using pip (all platforms)

The easiest way to install PyTRiP98 is using pip

Note: Starting from mid-2014 pip comes pre-installed with Python newer than 3.4 and 2.7.9 (for 2.x family)

3.2.1 Administrator installation (root access)

Administrator installation is very simple, but requires to save some files in system-wide directories (i.e. */usr*):

```
sudo pip install pymchelper
```

To upgrade the **pymchelper** to newer version, simply type:

```
sudo pip install --upgrade pymchelper
```

To completely remove **pymchelper** from your system, use following command:

```
sudo pip uninstall pymchelper
```

Now all **pymchelper** commands should be installed for all users:

```
convertmc --help
```

3.2.2 User installation (non-root access)

User installation will put the **pymchelper** under hidden directory *$HOME/.local*.

To install the package, type in the terminal:

```
pip install pymchelper --user
```

If *pip* command is missing on your system, replace *pip* with *pip3* in abovementioned instruction.

To upgrade the **pymchelper** to newer version, simply type:

```
pip install --upgrade pymchelper --user
```

To completely remove **pymchelper** from your system, use following command:

```
pip uninstall pymchelper
```

In most of modern systems all executables found in *$HOME/.local/bin* directory can be called like normal commands (i.e. *ls*, *cd*). It means that after installation you should be able to simply type in terminal:

```
convertmc --help
```

If this is not the case, please prefix the command with *$HOME/.local/bin* and call it in the following way:

```
$HOME/.local/bin/convertmc --help
```

CHAPTER 4

Developer documentation

4.1 Contributing Guide

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

4.1.1 Types of Contributions

Report Bugs

Report bugs at https://github.com/DataMedSci/pymchelper/issues

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

Fix Bugs or Implement Features

Look through the GitHub issues for bugs or features. Anything tagged with "bug" or "feature" is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything tagged with "feature" is open to whoever wants to implement it.

Write Documentation

pymchelper could always use more documentation, whether as part of the official *pymchelper* docs, in docstrings, or even on the web in blog posts, articles, and such.

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/DataMedSci/pymchelper/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

4.1.2 Get Started for GIT-aware developers

Ready to contribute? Here's how to set up *pymchelper* for local development. We assume you are familiar with GIT source control system. If not you will other instruction at the end of this page.

1. Fork the `pymchelper` repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/pymchelper.git
```

3. If you are not familiar with GIT, proceed to step 5, otherwise create a branch for local development:

```
$ cd pymchelper
$ git checkout -b feature/issue_number-name_of_your_bugfix_or_feature
```

4. Now you can make your changes locally.

As the software is prepared to be shipped as pip package, some modifications of PYTHONPATH variables are needed to run the code. Let us assume you are now in the same directory as `setup.py` file.

The standard way to execute Python scripts WILL NOT WORK. What users see as convertmc program, is basically pymchelper/run.py script:

```
$ python pymchelper/run.py --help
```

To have the code working, the PYTHONPATH has to be adjusted:

```
$ PYTHONPATH=. python pymchelper/run.py --help
 usage: run.py [-h] [-V] converter ...
 (...)
```

5. Make local changes to fix the bug or to implement a feature.
6. When you're done making changes, check that your changes comply with PEP8 code quality standards (flake8 tests) and run unit tests with pytest:

```
$ flake8 pymchelper tests
$ pytest tests
```

   To get flake8 and pytest, just pip install them.

7. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
```

8. Repeat points 4-6 until the work is done. Now its time to push the changes to remote repository:

```
$ git push origin feature/issue_number-name_of_your_bugfix_or_feature
```

9. Submit a pull request through the GitHub website to the master branch of `git@github.com:DataMedSci/pymchelper.git` repository.
10. Check the status of automatic tests ran by Travis system.

You can find them on the pull request webpage https://travis-ci.org/DataMedSci/pymchelper/pull_requests. In case some of the tests fails, fix the problem. Then commit and push your changes (steps 5-8).

4.1.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and extend the documentation where necessary.
3. The pull request should work for Python 2.7, 3.2, 3.3, 3.4 and 3.5. Check https://travis-ci.org/DataMedSci/pymchelper/pull_requests and make sure that the tests pass for all supported Python versions.

4.1.4 Get Started for non-GIT developers

1. Fetch the code from remote GIT repository to your local directory:

```
$ git clone git@github.com:DataMedSci/pymchelper.git
```

2. Follow steps 4-6 from the instruction for GIT-aware developers. To run code locally, prefix usual calls with `PYTHONPATH=.`:

```
$ PYTHONPATH=. python pymchelper/run.py --help
```

usage: run.py [-h] [-V] converter ... (...)

Make your changes and check that they comply with PEP8 code quality standards (flake8 tests) and run all unit tests with pytest:

```
$ flake8 pymchelper tests
$ pytest tests/
```

3. Compress your working directory and send it to us by email (see *authors*), describing your changes.

4.1.5 Tips

To run full tests type:

```
pytest tests/
```

To run only a single test type:

```
pytest tests/test_file_to_run.py
```

4.2 Source code

4.2.1 pymchelper package

Subpackages

pymchelper.readers package

Subpackages

pymchelper.readers.shieldhit package

Submodules

pymchelper.readers.shieldhit.binary_spec module

class pymchelper.readers.shieldhit.binary_spec.**SHBDOTagID**
Bases: enum.IntEnum

List of Tag ID numbers for BDO 2016 and 2019 formats. Must be synchronized with tags in sh_bdo.h and sh_detect.h in SH12A.

apro0 = 51969

apzlscl = 52234

beamdivk = 51982

beamdivx = 51980

beamdivy = 51981

beamphi = 51979

beamtheta = 51978

beamx = 51971

beamy = 51972

beamz = 51973

comment = 65484

ct_ang = 52736

ct_icnt = 52737

ct_len = 52738

data_block = 56763

debug = 65485

dele = 52224

```
demin = 52225
det_dif_start = 56590
det_dif_stop = 56591
det_difftype = 56584
det_dmat = 56582
det_dsize = 56586
det_dsizexyz = 56587
det_dtype = 56579
det_geotyp = 56576
det_nbin = 56577
det_nbine = 56583
det_part = 56578
det_parta = 56581
det_partz = 56580
det_thresh = 56593
det_voxvol = 56592
det_xyz_start = 56588
det_xyz_stop = 56589
det_zonestart = 56585
detector_type = 56624
detector_unit = 56764
error = 65486
estimator_number = 60929
estimator_rescale_per_particle = 60931
ext_nproj = 52239
ext_ptvdose = 52240
filedate = 2
filename_or_geotype = 60928
format = 5
geo_n_bins = 57348
geo_non_equidist_grid = 57352
geo_p_start = 57346
geo_q_stop = 57347
geo_rotation = 57349
geo_unit_ids = 57354
geo_units = 57353
```

```
geo_volume = 57350
geo_zones = 57351
geometry_name = 57345
geometry_type = 57344
host = 4
iemtrans = 52230
iextspec = 52231
intrfast = 52232
intrslow = 52233
inucre = 52229
ioffset = 52235
irifimc = 52236
irifitrans = 52237
irifizone = 52238
itypms = 52227
itypst = 52226
ixfirs = 52241
jpart0 = 51968
number_of_pages = 60930
oln = 52228
page_diff_flag = 56784
page_diff_size = 56788
page_diff_start = 56786
page_diff_stop = 56787
page_diff_type = 56785
page_diff_units = 56789
page_filter_e_min = 56818
page_filter_emax = 56819
page_filter_name = 56816
page_filter_rules_no = 56817
page_medium_scoring = 56630
page_medium_transport = 56629
page_normalized = 56626
```

- the data in page->data as x_j
- for j instances of this simulation
- which was done with I_j number of paritcles.

The resulting data will be termed X and has the units given by SHBDO_PAG_DATA_UNIT

0: X = x_1 for GEOMAP type scorers 1: X = sum_j x_j COUNT, ... 2: X = (sum_j x_j) / (sum_j I_j) NORMCOUNT, ... 3: X = (sum_j x_j * I_j) / (sum_j I_j) LET, ...

Type Given

page_number = 56625

page_offset = 56628

page_scale_factor = 56627

page_unit_ids = 56631

rt_nstat = 43520

rt_time = 43521

rt_timesim = 43522

shbuilddate = 1

shversion = 0

sigmat0 = 51977

sigmax = 51974

sigmay = 51975

tmax0 = 51976

tmax0amu = 51984

tmax0mev = 51983

tmax0nuc = 51985

user = 3

zpro0 = 51970

class pymchelper.readers.shieldhit.binary_spec.**SHBDOUnitID**

Bases: enum.IntEnum

An enumeration.

SH_SCORING_UNIT_AU = 1

SH_SCORING_UNIT_CM = 10

SH_SCORING_UNIT_CM2 = 11

SH_SCORING_UNIT_CM3 = 12

SH_SCORING_UNIT_COUNT = 60

SH_SCORING_UNIT_DEGREES = 50

SH_SCORING_UNIT_DOSERAD = 45

SH_SCORING_UNIT_DOSEREM = 46

SH_SCORING_UNIT_GPCM3 = 22

SH_SCORING_UNIT_GY = 41

SH_SCORING_UNIT_GYRBE = 42

SH_SCORING_UNIT_GYRE = 43

SH_SCORING_UNIT_INVALID = -1

SH_SCORING_UNIT_KEVPUM = 30

SH_SCORING_UNIT_KGPM3 = 23

SH_SCORING_UNIT_M = 16

SH_SCORING_UNIT_M2 = 17

SH_SCORING_UNIT_M3 = 18

SH_SCORING_UNIT_MATID = 90

SH_SCORING_UNIT_MEV = 70

SH_SCORING_UNIT_MEVCM2PG = 32

SH_SCORING_UNIT_MEVPAMU = 72

SH_SCORING_UNIT_MEVPC2 = 81

SH_SCORING_UNIT_MEVPCM = 31

SH_SCORING_UNIT_MEVPG = 40

SH_SCORING_UNIT_MEVPNUC = 71

SH_SCORING_UNIT_NONE = 0

SH_SCORING_UNIT_NUCN = 80

SH_SCORING_UNIT_NZONE = 91

SH_SCORING_UNIT_PCM = 13

SH_SCORING_UNIT_PCM2 = 14

SH_SCORING_UNIT_PCM3 = 15

SH_SCORING_UNIT_PCT = 2

SH_SCORING_UNIT_PM = 19

SH_SCORING_UNIT_PM2 = 20

SH_SCORING_UNIT_PM3 = 21

SH_SCORING_UNIT_PMIL = 3

SH_SCORING_UNIT_RADIANS = 51

SH_SCORING_UNIT_RELATIVE = 4

SH_SCORING_UNIT_SR = 52

SH_SCORING_UNIT_SV = 44

SH_SCORING_UNIT_U = 82

SH_SCORING_UNIT_UNKNOWN = -2

pymchelper.readers.shieldhit.general module

class pymchelper.readers.shieldhit.general.**SHFileFormatId**
Bases: enum.IntEnum

SHIELD-HIT12A file format ids, as described in sh_file_format.h file

ascii = 3

bdo2016 = 1

bdo2019 = 2

bin2010 = 0

csv = 4

class pymchelper.readers.shieldhit.general.**SHReaderASCII**(*filename*)
Bases: object

Reads plain text files with data saved by binary-to-ascii converter.

read(*detector*)

read_header(*detector*)

read_payload(*detector*)

class pymchelper.readers.shieldhit.general.**SHReaderFactory**(*filename*)
Bases: *pymchelper.readers.common.ReaderFactory*

get_reader()
Inspect binary file and return appropriate reader object :return:

pymchelper.readers.shieldhit.general.**extract_sh_ver**(*filename*)
BDO binary files, introduced in 2016 (BDO2016 and BDO2019 formats) contain information about SH VER :param filename: Binary file filename :return: SH12 version (as a string, i.e. 0.7) or None if version information was not found in the file

pymchelper.readers.shieldhit.general.**file_has_sh_magic_number**(*filename*)
BDO binary files, introduced in 2016 (BDO2016 and BDO2019 formats) starts with 6 magic bytes xSH12A :param filename: Binary file filename :return: True if binary file starts with SH magic number

pymchelper.readers.shieldhit.general.**read_token**(*filename*, *token_id*)
TODO :param filename: :param token_id: :return:

pymchelper.readers.shieldhit.reader_base module

class pymchelper.readers.shieldhit.reader_base.**SHReader**(*filename*)
Bases: *pymchelper.readers.common.Reader*

Reads binary output files generated by SHIELD-HIT12A code.

corename
TODO :return:

read_data(*estimator*, *nscale=1*)
TODO :param estimator: :param nscale: :return:

pymchelper.readers.shieldhit.reader_base.**mesh_unit_and_name**(*estimator*, *axis*)
Gets units and names depending on detector type.

pymchelper.readers.shieldhit.reader_base.**read_next_token**(*f*)
returns a tuple with 4 elements: 0: payload id 1: payload dtype string 2: payload number of elements 3: payload itself f is an open and readable file pointer. returns None if no token was found / EOF

pymchelper.readers.shieldhit.reader_bdo2016 module

class pymchelper.readers.shieldhit.reader_bdo2016.**SHReaderBDO2016**(*filename*)
Bases: *pymchelper.readers.shieldhit.reader_base.SHReader*

Binary format reader from version >= 0.6 This format doesn't have support for multiple pages per estimator

read_data(*estimator*)
TODO :param estimator: :param nscale: :return:

pymchelper.readers.shieldhit.reader_bdo2019 module

class pymchelper.readers.shieldhit.reader_bdo2019.**SHReaderBDO2019**(*filename*)
Bases: *pymchelper.readers.shieldhit.reader_base.SHReader*

Experimental binary format reader version >= 0.7

read_data(*estimator*)
TODO :param estimator: :param nscale: :return:

pymchelper.readers.shieldhit.reader_bin2010 module

class pymchelper.readers.shieldhit.reader_bin2010.**SHReaderBin2010**(*filename*)
Bases: *pymchelper.readers.shieldhit.reader_base.SHReader*

Binary format reader from 0.1 <= version <= 0.6

read_data(*estimator*)
TODO :param estimator: :param nscale: :return:

read_header(*estimator*)

read_payload(*estimator*)

Submodules

pymchelper.readers.common module

class pymchelper.readers.common.**Reader**(*filename*)
Bases: object

corename

read(*estimator*)

read_data(*estimator*)

class pymchelper.readers.common.**ReaderFactory**(*filename*)
Bases: object

get_reader()

pymchelper.readers.fluka module

class pymchelper.readers.fluka.**FlukaReader**(*filename*)
Bases: *pymchelper.readers.common.Reader*

corename
corename_fort.XX. :return: corename part of output file or None in case filename doesn't follow Fluka naming pattern

Type Fluka output filenames follow this pattern

parse_data(*estimator*)
TODO :param estimator: an Estimator object, will be modified here and filled with data

parse_resnuclei(*estimator*)
TODO add support for resnuclei RESNUCLEi Scores residual nuclei produced in inelastic interactions on a region basis :param estimator: an Estimator object, will be modified here and filled with data

parse_usrbdx(*estimator*)
USRBDX defines a detector for a boundary crossing fluence or current estimator :param estimator: an Estimator object, will be modified here and filled with data

parse_usrbin(*estimator*)
USRBIN scores distribution of one of several quantities in a regular spatial structure (binning detector) independent from the geometry. :param estimator: an Estimator object, will be modified here and filled with data

parse_usrtrack(*estimator*)

Parameters **estimator** – an Estimator object, will be modified here and filled with data

USRTRACK defines a detector for a track-length fluence estimator

read_data(*estimator*, *nscale=1*)
TODO :param estimator: an Estimator object, will be modified here and filled with data

class pymchelper.readers.fluka.**FlukaReaderFactory**(*filename*)
Bases: *pymchelper.readers.common.ReaderFactory*

Class responsible for discovery of filetype.

get_reader()
Try reading header of Fluka binary file and return a corresponding FlukaReader object

Returns FlukaReader class if file is digested by Usrxxx Flair reader. None is returned otherwise

pymchelper.shieldhit package

Subpackages

pymchelper.shieldhit.detector package

Submodules

pymchelper.shieldhit.detector.detector_type module

class pymchelper.shieldhit.detector.detector_type.**SHDetType**
Bases: enum.IntEnum

List of available detector types below is based on IDET(5,*) in detect.f in SHIELD-HIT12A. If new detectors are added, class SHEstimator in estimator.py should also be updated.

```
a = 30
alanine = 13
alanine_gy_bdo2016 = 213
amass = 31
amu = 32
angle = 25
angle_bdo2016 = 121
avg_beta = 9
avg_energy = 8
counter = 14
crossflu = 3
ddd = 12
dedx = 35
dlet = 6
dletg = 16
dose = 5
dose_av_q = 48
dose_av_z2beta2 = 46
dose_eqv = 40
dose_gy = 39
dose_gy_bdo2016 = 205
dzeff2beta2 = 54
energy = 1
energy_amu = 29
energy_nuc = 28
eqv_dose = 41
flu_char = 22
flu_neqv = 24
flu_neut = 23
fluence = 2
gen = 33
id = 34
invalid = 32767
kinetic_energy = 27
```

let_bdo2016 = 120

letflu = 4

mass_dedx = 36

material = 11

medium = 19

n_eqv_dose = 44

nkerma = 38

none = 0

pet = 15

q = 21

rho = 20

spc = 10

tlet = 7

tletg = 17

trace = 26

track_av_q = 49

track_av_z2beta2 = 47

track_length = 37

tzeff2beta2 = 53

user1 = 42

user2 = 43

z = 50

z2beta2 = 45

zeff = 51

zeff2beta2 = 52

zone = 18

pymchelper.shieldhit.detector.estimator module

class pymchelper.shieldhit.detector.estimator.**SHEstimator**

Bases: object

allowed_detectors = {<SHGeoType.unknown: 0>: (), <SHGeoType.zone: 1>: <generator ob

is_valid()

pymchelper.shieldhit.detector.estimator_type module

class pymchelper.shieldhit.detector.estimator_type.**SHGeoType**
Bases: enum.IntEnum

An enumeration.

cyl = 2

dcyl = 6

dcylz = 10

dmsh = 7

dmshz = 11

dplane = 8

dzone = 5

geomap = 15

msh = 3

plane = 4

trace = 13

unknown = 0

voxscore = 14

zone = 1

pymchelper.shieldhit.detector.fortran_card module

class pymchelper.shieldhit.detector.fortran_card.**CardLine**(*data=None*)
Bases: object

Represents single line of detect.dat file

static any_to_element(*s*, *align_right=True*)

comment = '*----0---><----1---><----2---><----3---><----4---><----5---><----6--->'

credits = '* generated by pymchelper (https://github.com/DataMedSci/pymchelper) *'

element_length = 10

no_of_elements = 7

class pymchelper.shieldhit.detector.fortran_card.**EstimatorReader**
Bases: object

class pymchelper.shieldhit.detector.fortran_card.**EstimatorWriter**
Bases: object

Helper class. Gives textual representation of Estimator objects.

static get_lines(*estimator*)
Converts estimator to CardLine objects :param estimator: valid estimator object :return: tuple of CardLine objects

static get_text(*estimator*, *add_comment=False*)
Converts Estimator to text :param estimator: valid Estimator object :param add_comment: if True, prepends textual representation with comment line :return: text representation (set of text lines) of Estimator

pymchelper.shieldhit.detector.geometry module

class pymchelper.shieldhit.detector.geometry.**Axis**(*name=''*, *start=None*, *stop=None*, *nbins=None*, *number=None*)
Bases: object

Represents named sequence of nbins numbers, with known start and end position. Can be used as container to describe scoring geometry along one of the axis.

set(*start=None*, *stop=None*, *nbins=None*, *number=None*)

class pymchelper.shieldhit.detector.geometry.**CarthesianMesh**
Bases: *pymchelper.shieldhit.detector.geometry.Geometry*

Carthesian mesh along X,Y and Z axis

static allowed_estimators()

Returns List of compatible estimators for this scoring geometry.

class pymchelper.shieldhit.detector.geometry.**CylindricalMesh**
Bases: *pymchelper.shieldhit.detector.geometry.Geometry*

Cylindrical mesh along R, PHI and Z axis

static allowed_estimators()

Returns List of compatible estimators for this scoring geometry.

class pymchelper.shieldhit.detector.geometry.**Geometry**
Bases: object

Base class for all types of scoring geometries. Holds information about three axes. Three axis are used to allow scoring up to 3 dimensions. Lower number of dimensions (i.e. scoring on square) are achieved by setting nbins=1 in the axis.

static allowed_estimators()

Returns List of compatible estimators for this scoring geometry.

set_axis(*axis_no*, *start=None*, *stop=None*, *nbins=None*)
Fill axis data :param axis_no: integer number, from 0 to 2, specifies axis number :param start: start position :param stop: stop position :param nbins: number of elements :return: None

class pymchelper.shieldhit.detector.geometry.**Plane**
Bases: object

Plane scoring geometry

static allowed_estimators()

set_normal(*x=None*, *y=None*, *z=None*)

set_point(*x=None*, *y=None*, *z=None*)

class pymchelper.shieldhit.detector.geometry.**Zone**
Bases: object

Zone scoring geometry - start and stop

static allowed_estimators()

Submodules

pymchelper.shieldhit.particle module

class pymchelper.shieldhit.particle.**SHHeavyIonType**
Bases: object

particle_type = 25

class pymchelper.shieldhit.particle.**SHParticleType**
Bases: enum.IntEnum

Particle list based on JPART from SHIELD-HIT12A, extended with ids 0 and -1.

all = -1

anti_neutrino_e = 18

anti_neutrino_mu = 20

anti_neutron = 6

anti_proton = 7

deuteron = 21

electron = 13

gamma = 12

heavy_ion = 25

helium_3 = 23

helium_4 = 24

kaon_minus = 8

kaon_plus = 9

kaon_tilde = 11

kaon_zero = 10

muon_minus = 15

muon_plus = 16

neutrino_e = 17

neutrino_mu = 19

neutron = 1

pi_minus = 3

pi_plus = 4

pi_zero = 5

positron = 14

proton = 2

triton = 22

unknown = 0

pymchelper.writers package

Submodules

pymchelper.writers.common module

class pymchelper.writers.common.**Converters**
Bases: enum.IntEnum

Available converters

excel = 6

fromname = <bound method Converters.fromname of <enum 'Converters'>>

fromnumber = <bound method Converters.fromnumber of <enum 'Converters'>>

gnuplot = 2

hdf = 9

image = 3

inspect = 8

plotdata = 1

sparse = 7

tripcube = 4

tripddd = 5

txt = 0

pymchelper.writers.excel module

class pymchelper.writers.excel.**ExcelWriter**(*filename*, *options*)
Bases: object

Supports writing XLS files (MS Excel 2003 format)

write(*estimator*)

pymchelper.writers.fortranformatter module

pymchelper.writers.fortranformatter.**format_d**(*w*, *d*, *val*)
TODO :param w: :param d: :param val: :return:

pymchelper.writers.fortranformatter.**format_e**(*w*, *d*, *val*)
TODO :param w: :param d: :param val: :return:

pymchelper.writers.hdf module

class pymchelper.writers.hdf.**HdfWriter**(*filename*, *options*)
Bases: object

Supports writing HDF file format. HDF is designed to store large amounts of data organized in convenient way. One HDF file can handle many single- or multi-dimensional tables.

write(*estimator*)

pymchelper.writers.inspector module

class pymchelper.writers.inspector.**Inspector**(*filename*, *options*)
Bases: object

write(*estimator*)
Print all keys and values from estimator structure

they include also a metadata read from binary output file

pymchelper.writers.plots module

class pymchelper.writers.plots.**GnuplotDataWriter**(*filename*, *options*)
Bases: object

TODO

write(*estimator*)
TODO

class pymchelper.writers.plots.**ImageWriter**(*filename*, *options*)
Bases: object

Writer responsible for creating PNG images using matplotlib library

default_colormap = 'gnuplot2'

get_page_figure(*page*)
Calculate matplotlib figure object for a single page in estimator

write(*estimator*)
Go through all pages in estimator and save corresponding figure to an output file

class pymchelper.writers.plots.**PlotAxis**
Bases: enum.IntEnum

An enumeration.

x = 1

y = 2

z = 3

class pymchelper.writers.plots.**PlotDataWriter**(*filename*, *options*)
Bases: object

gnuplot data writer

write(*estimator*)
TODO

write_single_page(*page*, *filename*)
TODO

pymchelper.writers.shieldhit module

class pymchelper.writers.shieldhit.**SHBinaryWriter**(*filename*, *options*)
Bases: object

write(*estimator*)

class pymchelper.writers.shieldhit.**TxtWriter**(*filename*, *options*)
Bases: object

write(*estimator*)

pymchelper.writers.sparse module

class pymchelper.writers.sparse.**SparseWriter**(*filename*, *options*)
Bases: object

Supports writing sparse matrix format

write(*estimator*)

pymchelper.writers.trip98cube module

class pymchelper.writers.trip98cube.**TRiP98CubeWriter**(*filename*, *options*)
Bases: object

write(*estimator*)

pymchelper.writers.trip98ddd module

class pymchelper.writers.trip98ddd.**DebuggingPlots**(*base_data*)
Bases: object

Debugging plots, mostly needed to inspect if Gaussian function fitting was successful

base_data(*zmax_cm*, *threshold*, *logy=False*)

static fit_summary(*fit_results*)

map2d(*zlog=False*)

class pymchelper.writers.trip98ddd.**FitResultCollection**(*n*)
Bases: object

Fit results collection (along Z axis)

class pymchelper.writers.trip98ddd.**FittingMethods**
Bases: object

Functions describing Gaussian functions modelling lateral dose distributions

classmethod gauss2_MeV_g(*x_cm*, *amp_MeV_cm_g*, *sigma1_cm*, *weight*, *sigma2_add_cm*)

classmethod gauss2_MeV_g_1st(*x_cm*, *amp_MeV_cm_g*, *sigma1_cm*, *weight*, *sigma2_add_cm*)

classmethod gauss2_MeV_g_2nd(*x_cm*, *amp_MeV_cm_g*, *sigma1_cm*, *weight*, *sigma2_add_cm*)

classmethod gauss2_r_MeV_cm_g(*x_cm*, *amp_MeV_cm_g*, *sigma1_cm*, *weight*, *sigma2_add_cm*)

classmethod gauss_MeV_g(*x_cm*, *amp_MeV_cm_g*, *sigma_cm*)

classmethod gauss_r_MeV_cm_g(*x_cm*, *amp_MeV_cm_g*, *sigma_cm*)

class pymchelper.writers.trip98ddd.**LateralDepthDoseProfile**(*r_cm_1d*, *z_cm_1d*, *dose_MeV_g_2d*, *dose_error_MeV_g_2d*, *r_step_cm=None*, *z_step_cm=None*)

Bases: object

Base data for fitting

static cumulative_dose(*dose_1d*)

static cumulative_dose_left(*cumsum*)

class pymchelper.writers.trip98ddd.**TRiP98DDDWriter**(*filename*, *options*)

Bases: object

Writer for TRiP98 DDD files. File format is described here: http://bio.gsi.de/DOCS/TRiP98/PRO/DOCS/trip98fmtddd.html

Only liquid water target is supported now.

write(*estimator*)

write_single_page(*estimator*, *page*, *filename*)

Submodules

pymchelper.run module

pymchelper.run.**add_default_options**(*parser*)

pymchelper.run.**main**(*args=None*)

4.3 Badges and links

docs	
tests	
package	

4.4 Credits

4.4.1 Development

- Leszek Grzanka - IFJ-PAN, Poland <leszek.grzanka@ifj.edu.pl>
- Niels Bassler - Stockholm University, Sweden

4.4.2 Contributors

None yet. Why not be the first?

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z