
fastjet

Aryan Roy

Aug 31, 2021

TUTORIAL

1	Clustering Specification	3
2	The JetAlgorithms	5
3	The Data	7
4	ClusterSequence Class	9
5	Extracting Information	11
6	Limitations	13
7	The Object Oriented Interface	15
7.1	Clustering the data	15
7.2	Clustering Specification	15
7.3	The JetAlgorithms	15
7.4	The Data	16
7.5	ClusterSequence Class	16
7.6	Extracting Information	16
8	The Classic Interface classes	17
9	fastjet.ClusterSequence	19
10	Documentation	25
11	Installation	27
12	The Interfaces	29
12.1	Indices and tables	29
	Index	31

The tutorial on this page describes how the user can use [Awkward Arrays](#) to perform clustering on the particle data.

CLUSTERING SPECIFICATION

The fastjet library has some classes specifically made to provide the different parameters for clustering. This includes the following classes :

- `JetDefinition`
- `AreaDefinition`
- `RangeDefinition`

For example, the `JetDefinition` class can be instantiated in the following way:

```
import fastjet
jetdef = fastjet.JetDefinition(fastjet.antikt_algorithm, 0.6)
```

The `JetDefinition` class takes varied number of arguments, the first argument is always the type of algorithm, the number of rest of the arguments depends on how many parameters the given algorithm requires.

THE JETALGORITHMS

The `JetDefinition` class takes `JetAlgorithms` as arguments. In the above example we have chosen the Anti-kt algorithm. The list of algorithms is as following:

- `ee_genkt_algorithm`
- `ee_kt_algorithm`
- `genkt_algorithm`
- `genkt_for_passive_algorithm`
- `kt_algorithm`
- `cambridge_for_passive_algorithm`
- `cambridge_algorithm`
- `cambridge_aachen_algorithm`
- `antikt_algorithm`

THE DATA

The input data for the Multi-event interface has to be an Awkward Array. One such example is as follows:

```
>>> import awkward as ak
>>> array = ak.Array(
... [
...     {"px": 1.2, "py": 3.2, "pz": 5.4, "E": 2.5, "ex": 0.78},
...     {"px": 32.2, "py": 64.21, "pz": 543.34, "E": 24.12, "ex": 0.35},
...     {"px": 32.45, "py": 63.21, "pz": 543.14, "E": 24.56, "ex": 0.0},
... ],
... )
```

The Awkward Array here is a Record Array of Lorentz Vectors.

Note: The inputs can be provided in more Awkward Array formats than described here.

CLUSTERSEQUENCE CLASS

After defining the JetDefinition class, the user can provide this instance to the ClusterSequence class as an argument, along with the input data to perform the clustering:

```
>>> cluster = fastjet.ClusterSequence(array, jetdef)
      <fastjet._pyjet.AwkwardClusterSequence object at 0x7f1413120a90>
```


EXTRACTING INFORMATION

Any output that has to be an Array will be an Awkward Array in the array oriented interface. For example:

```
>>> cluster.inclusive_jets()  
<Array [{px: 1.2, py: 3.2, ... E: 48.7}] type='2 * Momentum4D["px": float64, "py...']>
```


LIMITATIONS

The Awkward Array interface is only available for the `fastjet.ClusterSequence` class. The Awkward Array functionality is likely to be expanded to other classes in the future.

THE OBJECT ORIENTED INTERFACE

7.1 Clustering the data

The fastjet library provides many options for the user to perform clustering on HEP data. The library has been designed keeping in mind the different requirements of users. The basic clustering process is described below.

7.2 Clustering Specification

The fastjet library has some classes specifically made to provide the different parameters for clustering. This includes the following classes :

- [JetDefinition](#)
- [AreaDefinition](#)
- [RangeDefinition](#)

For example, the JetDefinition class can be instantiated in the following way:

```
import fastjet
jetdef = fastjet.JetDefinition(fastjet.antikt_algorithm, 0.6)
```

The JetDefinition class takes varied number of arguments, the first argument is always the type of algorithm, the number of rest of the arguments depends on how many parameters the given algorithm requires.

7.3 The JetAlgorithms

The JetDefinition class takes [JetAlgorithms](#) as arguments. In the above example we have chosen the Anti-kt algorithm. The list of algorithms is as following:

- [ee_genkt_algorithm](#)
- [ee_kt_algorithm](#)
- [genkt_algorithm](#)
- [genkt_for_passive_algorithm](#)
- [kt_algorithm](#)
- [cambridge_for_passive_algorithm](#)
- [cambridge_algorithm](#)

- `cambridge_aachen_algorithm`
- `antikt_algorithm`

7.4 The Data

The input for the classic interface is a list of `PseudoJets`. To use the classic interface here's what the data should look like (This is a single event interface, one function call can only process one event):

```
>>> array = [fastjet.PseudoJet(1.1,1.2,1.3,1.4),  
... fastjet.PseudoJet(2.1,2.2,2.3,2.4),  
... fastjet.PseudoJet(3.1,3.2,3.3,3.4)]
```

7.5 ClusterSequence Class

After defining the `JetDefinition` class, the user can provide this instance to the `ClusterSequence` class as an argument, along with the input data to perform the clustering:

```
fastjet.ClusterSequence(inputs, jetdef)
```

7.6 Extracting Information

Any output that has to be an `Array` will be a list of `PseudoJets` if it's particle data. For example:

```
>>> inc_jets = cluster.inclusive_jets()  
>>> for elem in inc_jets:  
...     print("px:", elem.px(), "py:", elem.py(), "pz:", elem.pz(), "E:", elem.E(),)  
px: 6.3000000000000001 py: 6.6000000000000005 pz: 6.8999999999999995 E: 7.199999999999999
```

THE CLASSIC INTERFACE CLASSES

The documentation for the C++ Fastjet covers all the classes in fastjet. The python classes of the classic interface behave exactly like they do in C++, therefore we are providing a link to the respective doxygen pages:

- [PseudoJet](#)
- [JetDefinition](#)
- [ClusterSequenceArea](#)
- [AreaDefinition](#)
- Every Other C++ Class (Almost all available through the classic interface)

FASTJET.CLUSTERSEQUENCE

class `fastjet.ClusterSequence(data, jetdef)`

The base class for all clustering.

Parameters

- **data** (*awkward.highlevel.Array*) – The data for clustering.
- **jetdef** (*fastjet._swig.JetDefinition*) – The JetDefinition for clustering specification.

Q()

Returns the sum of all energies in the event (relevant mainly for e+e-)

Parameters **None** –

Returns Returns an Awkward Array of the same type as the input.

Return type *awkward.highlevel.Array*

Q2()

Return $Q()^2$

Parameters **None** –

Returns Returns an Awkward Array of the same type as the input.

Return type *awkward.highlevel.Array*

childless_pseudojets()

Return the list of pseudojets in the ClusterSequence that do not have children (and are not among the inclusive jets).

Parameters **None** –

Returns Returns an Awkward Array of the same type as the input.

Return type *awkward.highlevel.Array*

constituent_index(min_pt=0)

Returns the index of the constituent of each Jet.

Parameters **min_pt** (*float*) – The minimum value of the pt for the inclusive jets.

Returns Returns an Awkward Array of the same type as the input.

Return type *awkward.highlevel.Array*

constituents(min_pt=0)

Returns the particles that make up each Jet.

Parameters **min_pt** (*float*) – The minimum value of the pt for the inclusive jets.

Returns Returns an Awkward Array of the same type as the input.

Return type `awkward.highlevel.Array`

exclusive_dmerge(*njets=10*)

Returns the dmin corresponding to the recombination that went from $n+1$ to n jets.

Parameters **n_jets** (*int*) – The number of jets it was clustered to.

Returns Returns an Awkward Array of the same type as the input.

Return type `awkward.highlevel.Array`

exclusive_dmerge_max(*njets=10*)

Returns the maximum of the dmin encountered during all recombinations up to the one that led to an n -jet final state.

Parameters **n_jets** (*int*) – The number of jets it was clustered to.

Returns Returns an Awkward Array of the same type as the input.

Return type `awkward.highlevel.Array`

exclusive_jets(*n_jets=-1, dcut=-1*)

Returns the exclusive jets after clustering in the same format as the input awkward array. Either takes *njets* or *dcut* as argument.

Parameters

- **n_jets** (*int*) – The number of jets it was clustered to.
- **dcut** (*float*) – The dcut for the result.

Returns Returns an Awkward Array of the same type as the input.

Return type `awkward.highlevel.Array`

exclusive_jets_ycut(*ycut=-1*)

Returns the exclusive jets after clustering in the same format as the input awkward array.

Parameters **ycut** (*float*) – The dcut for the result.

Returns Returns an Awkward Array of the same type as the input.

Return type `awkward.highlevel.Array`

exclusive_subdmerge(*data, nsub=0*)

Returns the dij that was present in the merging $nsub+1 \rightarrow nsub$ subjects inside this jet.

Parameters

- **data** (*awkward.highlevel.Array*) – An Awkward Array containing the Jets.
- **n_sub** (*int*) – The number of subjects.

Returns Returns an Awkward Array of the same type as the input.

Return type `awkward.highlevel.Array`

exclusive_subdmerge_max(*data, nsub=0*)

Returns the maximum dij that occurred in the whole event at the stage that the $nsub+1 \rightarrow nsub$ merge of subjects occurred inside this jet.

Parameters

- **data** (*awkward.highlevel.Array*) – An Awkward Array containing the Jets.
- **n_sub** (*int*) – The number of subjects.

Returns Returns an Awkward Array of the same type as the input.

Return type `awkward.highlevel.Array`

exclusive_subjets(*data*, *dcut*=-1, *nsub*=-1)

Returns an Awkward Array of all subjets of the current jet (in the sense of the exclusive algorithm) that would be obtained when running the algorithm with the given *dcut*.

Parameters

- **data** (*awkward.highlevel.Array*) – An Array containing the Jets.
- **dcut** (*float*) – The *dcut* for the result.
- **n_sub** (*int*) – The number of subjets.

Returns Returns an Awkward Array of the same type as the input.

Return type `awkward.highlevel.Array`

exclusive_subjets_up_to(*data*, *nsub*=0)

Returns the list of subjets obtained by unclustering the supplied jet down to *nsub* subjets (or all constituents if there are fewer than *nsub*).

Parameters

- **data** (*awkward.highlevel.Array*) – An Awkward Array containing the Jets.
- **n_sub** (*int*) – The number of subjets.

Returns Returns an Awkward Array of the same type as the input.

Return type `awkward.highlevel.Array`

exclusive_ymerge(*njets*=10)

Returns the *ymin* corresponding to the recombination that went from *n*+1 to *n* jets.

Parameters **n_jets** (*int*) – The number of jets it was clustered to.

Returns Returns an Awkward Array of the same type as the input.

Return type `awkward.highlevel.Array`

exclusive_ymerge_max(*njets*=10)

Same as `exclusive_dmerge_max`, but normalised to squared total energy.

Parameters **n_jets** (*int*) – The number of jets it was clustered to.

Returns Returns an Awkward Array of the same type as the input.

Return type `awkward.highlevel.Array`

get_child(*data*)

If the jet has parents in the clustering, it returns them.

Parameters **data** (*awkward.highlevel.Array*) – An Array containing the Jets.

Returns Returns an Awkward Array of the same type as the input.

Return type `awkward.highlevel.Array`

get_parents(*data*)

If the jet has parents in the clustering, it returns them.

Parameters **data** (*awkward.highlevel.Array*) – An Array containing the Jets.

Returns Returns an Awkward Array of the same type as the input.

Return type `awkward.highlevel.Array`

has_child(*data*)

If the jet has children in the clustering, it returns true.

Parameters **data** (*awkward.highlevel.Array*) – An Array containing the Jets.

Returns Returns an Awkward Array of the same type as the input.

Return type *awkward.highlevel.Array*

has_parents(*data*)

if the jet has parents in the clustering, it returns true.

Parameters **data** (*awkward.highlevel.Array*) – An Array containing the Jets.

Returns Returns an Awkward Array of the same type as the input.

Return type *awkward.highlevel.Array*

inclusive_jets(*min_pt=0*)

Returns the inclusive jets after clustering in the same format as the input awkward array

Parameters **min_pt** (*float*) – The minimum value of the pt for the inclusive jets.

Returns Returns an Awkward Array of the same type as the input containing inclusive jets.

Return type *awkward.highlevel.Array*

jet_def()

Returns the Jet Definition Object associated with the instance

Parameters **None** –

Returns Returns the jetdefinition stored as an attribute.

Return type *JetDefinition*

jet_scale_for_algorithm(*data*)

Returns the scale associated with a jet as required for this clustering algorithm (kt² for the kt-algorithm, 1 for the Cambridge algorithm).

Parameters **data** (*awkward.highlevel.Array*) – An Array containing the Jets.

Returns Returns an Awkward Array of the same type as the input.

Return type *awkward.highlevel.Array*

jets()

Allows the user to access the internally stored `_jets()` array, which contains both the initial particles and the various intermediate and final stages of recombination.

Parameters **none** –

Returns Returns an Awkward Array of the same type as the input.

Return type *awkward.highlevel.Array*

n_exclusive_jets(*dcut=0*)

Returns the number of jets (in the sense of the exclusive algorithm) that would be obtained when running the algorithm with the given dcut.

Parameters **dcut** (*float*) – The dcut for the result.

Returns Returns an Awkward Array of the same type as the input.

Return type *awkward.highlevel.Array*

n_exclusive_subjets(*data*, *dcut*=0)

Returns the size of `exclusive_subjets(...)`; still $n \ln n$ with same coefficient, but marginally more efficient than manually taking `len(exclusive_subjets)`

Parameters

- **data** (*awkward.highlevel.Array*) – An Array containing the Jets.
- **dcut** (*float*) – The dcut for the result.

Returns Returns an Awkward Array of the same type as the input.

Return type `awkward.highlevel.Array`

n_particles()

Returns the number of particles that were provided to the clustering algorithm.

Parameters **None** –

Returns Returns an Awkward Array of the same type as the input.

Return type `awkward.highlevel.Array`

unclustered_particles()

Returns the unclustered particles after clustering in the same format as the input awkward array

Parameters **None** –

Returns Returns an Awkward Array of the same type as the input containing the unclustered particles.

Return type `awkward.highlevel.Array`

unique_history_order()

Routine that returns an order in which to read the history such that clusterings that lead to identical jet compositions but different histories (because of degeneracies in the clustering order) will have matching constituents for each matching entry in the `unique_history_order`.

Parameters **None** –

Returns Returns an Awkward Array of the same type as the input.

Return type `awkward.highlevel.Array`

Fastjet is a library for performing Jet-Finding *within* the Scikit-HEP ecosystem. The library includes the classic interface, and a new interface built to perform clustering on multi-event Awkward Array objects.

Note: This project is under active development.

DOCUMENTATION

- Python interface - This site.
- [GitHub](#)

INSTALLATION

fastjet can be installed from [pypi](#) using pip:

```
pip install fastjet
```

Most users will get a precompiled binary (wheel) for your operating system and Python version. If not, the above attempts to compile from source.

THE INTERFACES

The fastjet library provides many options for the user to perform clustering on HEP data. The library has been designed keeping in mind the different requirements of users.

The fastjet library contains two interfaces within:

- **The Awkward interface**
- **The Classic interface**

The Awkward interface is the new interface made to handle multi-event data, whereas the classic interface is the same as the [C++ library](#), designed to handle the data in a particle-at-a-time fashion. The tutorials are divided into two to explain how each of the interfaces work. Please take a look at the tutorial section to get started.

12.1 Indices and tables

- [genindex](#)
- [modindex](#)
- [search](#)

C

`childless_pseudojets()` (*fastjet.ClusterSequence* method), 19
ClusterSequence (class in *fastjet*), 19
`constituent_index()` (*fastjet.ClusterSequence* method), 19
`constituents()` (*fastjet.ClusterSequence* method), 19

E

`exclusive_dmerge()` (*fastjet.ClusterSequence* method), 20
`exclusive_dmerge_max()` (*fastjet.ClusterSequence* method), 20
`exclusive_jets()` (*fastjet.ClusterSequence* method), 20
`exclusive_jets_ycut()` (*fastjet.ClusterSequence* method), 20
`exclusive_subdmerge()` (*fastjet.ClusterSequence* method), 20
`exclusive_subdmerge_max()` (*fastjet.ClusterSequence* method), 20
`exclusive_subjets()` (*fastjet.ClusterSequence* method), 21
`exclusive_subjets_up_to()` (*fastjet.ClusterSequence* method), 21
`exclusive_ymerge()` (*fastjet.ClusterSequence* method), 21
`exclusive_ymerge_max()` (*fastjet.ClusterSequence* method), 21

G

`get_child()` (*fastjet.ClusterSequence* method), 21
`get_parents()` (*fastjet.ClusterSequence* method), 21

H

`has_child()` (*fastjet.ClusterSequence* method), 21
`has_parents()` (*fastjet.ClusterSequence* method), 22

I

`inclusive_jets()` (*fastjet.ClusterSequence* method), 22

J

`jet_def()` (*fastjet.ClusterSequence* method), 22
`jet_scale_for_algorithm()` (*fastjet.ClusterSequence* method), 22
`jets()` (*fastjet.ClusterSequence* method), 22

N

`n_exclusive_jets()` (*fastjet.ClusterSequence* method), 22
`n_exclusive_subjets()` (*fastjet.ClusterSequence* method), 22
`n_particles()` (*fastjet.ClusterSequence* method), 23

Q

`Q()` (*fastjet.ClusterSequence* method), 19
`Q2()` (*fastjet.ClusterSequence* method), 19

U

`unclustered_particles()` (*fastjet.ClusterSequence* method), 23
`unique_history_order()` (*fastjet.ClusterSequence* method), 23