Fastjet 3 Tutorial

Grégory Soyez

IPhT, CEA Saclay

with Gavin Salam and Matteo Cacciari

`www.fastjet.fr`

Les Houches — Physics at TeV colliders — June 09 2012

Quickstart

[See also `http://fastjet.fr/quickstart.html`]

Get FastJet: Download from `fastjet.fr`

```
$ wget http://fastjet.fr/repo/fastjet-3.0.4.tar.gz
$ tar zxvf fastjet-3.0.4.tar.gz
$ cd fastjet-3.0.4/
```

Installation:

```
$ ./configure --prefix=$PWD/../fastjet-install
$ make
$ make check # optional
$ make install
```

Compilation/configuration helper

Config tool to get info about your FastJet installation

```
$ fastjet-config --help
...
```

The most useful are

```
  --cxxflags        returns the compilation flags
  --libs            returns the flags to pass to the linker

  --plugins         whether you also want to link the FastJet plugins
                     (default=no)

  --version         prints FastJet version and exits
  --prefix          gets the FastJet installation directory
  --config          shows a summary of how FastJet was configured
  --list-plugins    list all the available plugins
```

Basic FastJet usage

Disclaimer

Obviously impossible (and useless!)
to list all possible features

For extra information, refer to the following:

- FastJet manual (package, website or 1111.6097)
- Doxygen documentation (package, website)
- example programs (package)

Examples

Provoded in `example` in your FastJet sources

01-basic.cc	basic usage example
02-jetdef.cc	changing the jet definition
03-plugin.cc	using plugins
04-constituents.cc	accessing clustering information in a PseudoJet
05-eplus_eminus.cc	using e+e- algorithms
06-area.cc	using jet areas
07-subtraction.cc	subtracting jet background contamination
08-selector.cc	using the Selector tool
09-user_info.cc	adding user information to a fastjet::PseudoJet
10-subjets.cc	extracting subjets
11-filter.cc	use of filtering
12-boosted_higgs.cc	boosted Higgs tagging
13-boosted_top.cc	boosted top tagging
14-groomers.cc	unified use of transformers

A few practical “warming-up” examples

01 - basic usage example

3 fundamental objest of FastJet:

- PseudoJet: 4-momentum
- JetDefinition: the algorithm and its parameters (or a plugin)
- ClusterSequence: handles the clustering

A few practical “warming-up” examples

02 - changing the jet definition

```
JetDefinition(algorithm, R, [recombination_scheme], [strategy])
```

03 - using plugins

```
// define your plugin
JetDefinition(plugin)
```

A few practical “warming-up” examples

06 - using jet areas

Main classes:

- AreaDefinition: defines how areas are computed
- ClusterSequenceArea: clustering with areas
- (GhostedAreaSpec: specifies ghosts (area quanta) placement)

Major changes in FastJet 3

Jets know about their structure

```
ClusterSequence clust_seq(particles, jet_def);
PseudoJet jet = clust_seq.inclusive_jets()[0];

// clustering information:
vector<PseudoJet> constits = jet.constituents();
if (jet.has_parents(parent1, parent2)){...};
vector<PseudoJet> jet.exclusive_subjets(nsub);
vector<PseudoJet> jet.exclusive_subjets_up_to(nsub);
if ((cs = jet.associated_cluster_sequence()) != 0){...}

// and with area:
if (jet.has_area()){...}
jet.area();
jet.area_4vector();
jet.is_pure_ghost();
```

Note: the original cluster sequence must still exist

composite jets

Jets can be joined:

```
// subjets of a top candidate
PseudoJet W1;
PseudoJet W2;
PseudoJet b;

// build the top
PseudoJet W = join(W1,W2); // result is a sensible PseudoJet,
PseudoJet top = join(W,b); // with additions (see pieces() below)

// top constituents:  all the initial particles in the jet
vector<PseudoJet> constituents = top.constituents();

// top pieces:  the b and the W
vector<PseudoJet> pieces = top.pieces();
```

- `jet` from clustering: `jet.pieces()` $\equiv$ parents
- `join(W, b, &recombiner)` works

Jet user-defined extra info

- Fastjet 2.X: only a `PseudoJet::user_index()`
- FastJet 3: optional additional information

```
// user-defined class
class MyInfo :  public PseudoJet::UserInfoBase{
public:
   MyInfo(int id) :  pdg_id(id){}
   int pdg_id;
};

// set the info
particle.set_user_info(new MyInfo(22));

// access the info
int id = particle.user_info<MyInfo>().pdg_id;
```

Selectors

Easy way to apply cuts on `PseudoJets`:

```
#include <fastjet/Selector.hh>

Selector rap_sel = SelectorAbsRapMax(2.5); // y cut
Selector hard_sel = SelectorNHardest(2);   // select 2 hardest

// logical ops:  product *, not !, or ||, and &&
Selector full_sel = hard_sel * rap_sel * (!SelectorIsPureGhost());
Selector acceptance =
   SelectorRapRange(-3.0,-1.0) || SelectorRapRange(1.0,2.0);

// use them like that:
if (rap_sel.pass(particle)){...}
vector<PseudoJet> two_hardest_within_cuts = full_sel(jets);
GhostedAreaSpec gas(acceptance,...); // place ghosts where needed
```

- user-defined selectors possible
- very powerful when combined with user info!

Tools: (i) pileup/background subtraction

$$\rho = \underset{j \in \text{patches}}{\text{median}} \left\{ \frac{p_{t,j}}{A_j} \right\}$$

- patches $\equiv$ jets $^{(*)}$:

```
JetMedianBackgroundEstimator bge(SelectorAbsRapMax(4),
                    JetDefinition(kt_algorithm,0.4), area_def);
```

- patches $\equiv$ grid cells $^{(*)}$:

```
GridMedianBackgroundEstimator bge(ymax, gridsize=0.55);
```

- estimation *per se*:

```
bge.set_particles(particles);
cout << bge.rho() << " " << bge.sigma();
```

$^{(*)}$ both derived from `BackgroundEstimatorBase`

Tools: (i) pileup/background subtraction

Rapidity-dependent background:

- Solution 1: rescaling

$$\rho(\text{jet}) = f(y_j) \operatorname*{median}_{\text{all } j'} \left\{ \frac{p_{t,j'}}{A_{j'}\, f(y_{j'})} \right\}$$

```
FunctionOfPseudoJet<double> rescaling = ...;
bge.set_rescaling_class(&rescaling);
```

- Solution 2: local range (JetMedianBGE only)

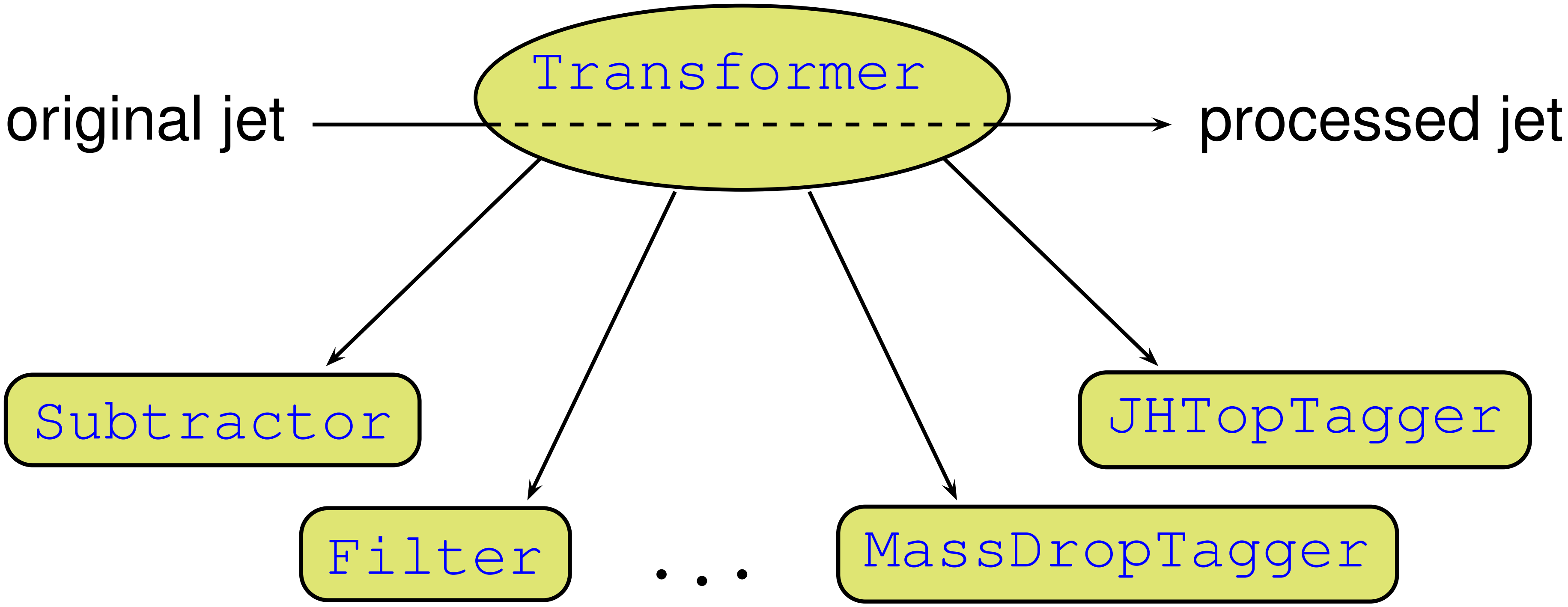
$$\rho(\text{jet}) = \operatorname*{median}_{j' \in \mathcal{R}(\text{j})} \left\{ \frac{p_{t,j'}}{A_{j'}} \right\}$$

```
SelectorRapidityStrip(1.2), SelectorDoughnut(0.4,1.2)
```

- In both cases: ρ and σ depend on the jet:

```
cout << bge.rho(jet) << " " << bge.sigma(jet);
```

Tools: (ii) acting on jets



- Unified interface
- Access to jet substructure
- User-defined transformers possible

Tools: background subtraction

Area-based subtraction:

$$p^\mu_{\text{subtracted}} = p^\mu - \rho A^\mu$$

Use the `Subtractor` transformer:

```
GridMedianBackgroundEstimator bge(...);
// or JetMedianBackgroundEstimator bge(...);

Subtractor subtractor(&bge);

// transformers act on jets or vector<jets>
PseudoJet subtracted_jet = subtractor(jet);
vector<PseudoJet> subtracted_jets =
   subtractor(csa.inclusive_jets());
```

Tools: jet grooming

`Filter` and `Pruner` for filtering, trimming and pruning

```
// filtering
Filter filter(Rfilt,
              SelectorNHardest(3));
PseudoJet groomed_jet = filter(jet);

// the result is a composite jet...
vector<PseudoJet> constituents = groomed_jet.constituents();
vector<PseudoJet> kept_subjets = groomed_jet.pieces();

// a little extra accessed using structure_of<>
vector<PseudoJet> rejected_subjets =
   groomed_jet.structure_of<Filter>().rejected();
```

`jet.structure_of<MyTransformer>()`

accesses the extra structure info of `MyTransformer`

Tools: jet grooming

`Filter` and `Pruner` for filtering, trimming and pruning

```
// filtering
Filter filter(JetDefinition(kt_algorithm, 0.2),
              SelectorNHardest(3));
PseudoJet groomed_jet = filter(jet);

// the result is a composite jet...
vector<PseudoJet> constituents = groomed_jet.constituents();
vector<PseudoJet> kept_subjets = groomed_jet.pieces();

// a little extra accessed using structure_of<>
vector<PseudoJet> rejected_subjets =
   groomed_jet.structure_of<Filter>().rejected();
```

`jet.structure_of<MyTransformer>()`

accesses the extra structure info of `MyTransformer`

Tools: jet grooming

`Filter` and `Pruner` for filtering, trimming and pruning

```
// trimming
Filter trimmer(Rtrim,
               SelectorPtFractionMin(0.05));
PseudoJet groomed_jet = trimmer(jet);

// the result is a composite jet...
vector<PseudoJet> constituents = groomed_jet.constituents();
vector<PseudoJet> kept_subjets = groomed_jet.pieces();

// a little extra accessed using structure_of<>
vector<PseudoJet> rejected_subjets =
   groomed_jet.structure_of<Filter>().rejected();
```

`jet.structure_of<MyTransformer>()`

accesses the extra structure info of `MyTransformer`

Tools: jet grooming + PU subtraction

`Filter` and `Pruner` for filtering, trimming and pruning

```
// trimming
Filter filter(Rfilt, SelectorNHardest(3));
filter.set_subtractor(&subtractor);
PseudoJet groomed_jet = filter(jet);

// the result is a composite jet...
vector<PseudoJet> constituents = groomed_jet.constituents();
vector<PseudoJet> kept_subjets = groomed_jet.pieces();

// a little extra accessed using structure_of<>
vector<PseudoJet> rejected_subjets =
    groomed_jet.structure_of<Filter>().rejected();
```

`jet.structure_of<MyTransformer>()`

accesses the extra structure info of `MyTransformer`

Tools: taggers

Example 1: Higgs mass-drop tagger

```
// declare the tagger
double mu=0.67, y_cut=0.09;
MassDropTagger tagger(mu, y_cut);

// tag a given jet
PseudoJet tagged_jet = tagger(jet); // Higgs candidate

// extract structure
if (tagged_jet != 0) {
   vector<PseudoJet> bjets = tagged_jet.pieces();
   double this_mu = tagged_jet.structure_of<MassDropTagger>().mu();
}
```

- Note again the `jet.structure_of<Transformer>()` usage
- taggers return a zero PseudoJet when unsuccessful

Tools: taggers

Example 1': another Higgs tagger

```
// declare the tagger
double tau2cut=0.08;
RestFrameNSubjettinessTagger tagger(subjet_def, tau2cut);

// tag a given jet
PseudoJet tagged_jet = tagger(jet); // Higgs candidate

// extract structure
if (tagged_jet != 0) {
   vector<PseudoJet> bjets = tagged_jet.pieces();
   double tau2 =
     tagged_jet.structure_of<RestFrameNSubjettinessTagger>().tau2();
}
```

- Note again the `jet.structure_of<Transformer>()` usage
- taggers return a zero PseudoJet when unsuccessful

Tools: taggers

Example 2: the Johns Hopkins top tagger

```
// declare the tagger
double delta_pt=0.1, delta_r=0.19;
JHTopTagger = top_tagger(delta_pt,delta_r);

// cut on the W and top candidates
top_tagger.set_W_selector(SelectorMassRange(65,95));
top_tagger.set_top_selector(SelectorMassRange(150,200));

// tag a given jet
PseudoJet tagged_jet = top_tagger(jet); // top candidate

// extract structure
if (tagged_jet != 0) {
   PseudoJet W = tagged_jet.structure_of<JHTopTagger>().W();
   PseudoJet nonW = tagged_jet.structure_of<JHTopTagger>().non_W();
}
```

Aside: errors and warnings

- Warnings: `LimitedWarning`
  - something strange (but non-critical) going on: check that you know what you are doing!
  - a summary can be obtained using the (static)

```
LimitedWarning::summary();
```

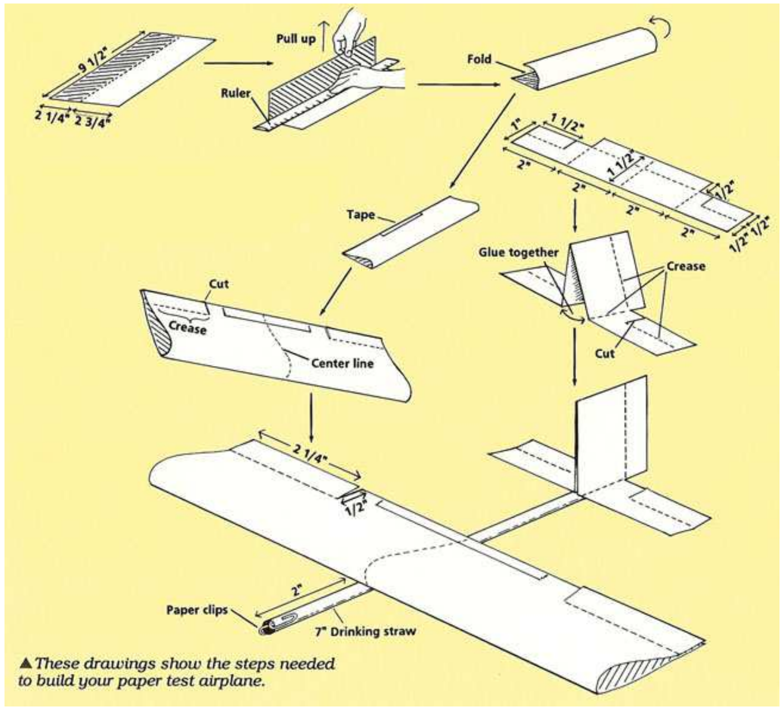
- Errors: `Error` exception thrown
  - You are most likely doing something wrong.
  - Tricky relation to your code ? Try (if available)

```
Error::set_backtrace(true);
```

at the beginning of your code

(`c++filt` and `addr2line` can be helpful to clarify the output)

User-defined tools



▲*These drawings show the steps needed to build your paper test airplane.*

Basic strategy

Inherit from `Transformer`

```
class Transformer : public FunctionOfPseudoJet<PseudoJet>{
    ...

    /// result of the Transformer acting on the 'orig'
    virtual PseudoJet result(const PseudoJet &orig) const=0;

    /// description of the Transformer
    virtual std::string description() const=0;

    /// A typedef for the associated result structure
    /// Used by PseudoJet::structure_of()
    typedef PseudoJetStructureBase StructureType;
};
```

Basic strategy

Inherit from `Transformer`

Where most of the work is:
The action of the transformer (const!)

```
class                                          Jet<PseudoJet>{
    ...

    /// result of the Transformer acting on the 'orig'
    virtual PseudoJet result(const PseudoJet &orig) const=0;

    /// description of the Transformer
    virtual std::string description() const=0;

    /// A typedef for the associated result structure
    /// Used by PseudoJet::structure_of()
    typedef PseudoJetStructureBase StructureType;
};
```

Basic strategy

Inherit from `Transformer`

```
class Transformer : public FunctionOfPseudoJet<PseudoJet>{
    ...

    /// result of the Transformer acting on the 'orig'
    virtual PseudoJet result(const PseudoJet &orig) const=0;

    /// description of the Transformer
    virtual std::string description() const=0;

    /// A typedef for the associated result structure
    /// Used by PseudoJet::structure_of()
    typedef PseudoJetStructureBase StructureType;
};
```

Has to be overloaded

Basic strategy

Inherit from `Transformer`

```
class Transformer : public FunctionOfPseudoJet<PseudoJet>{
    ...

    /// result of the Transformer acting on the 'orig'
    virtual PseudoJet result(const PseudoJet &orig) const=0;


                                                  onst=0;

    /// A typedef for the associated result structure
    /// Used by PseudoJet::structure_of()
    typedef PseudoJetStructureBase StructureType;
};
```

Associated structure with extra info.
derived from PseudoJetStructureBase

Step-by-step example: mass-drop tagger

A step-by-step example: `MassDropTagger`

[Butterworth, Davison, Rubin, Salam, 2008]

1. point to the associated structure

```
class MassDropTagger :  public Transformer{
  ...

  /// A typedef for the associated result structure
  /// Used by PseudoJet::structure_of()
  typedef MassDropTaggerStructure StructureType;
};
```

Step-by-step example: mass-drop tagger

2. implement the action of the transformer

```
PseudoJet MassDropTagger::result(const PseudoJet &jet) const{
  PseudoJet j1, j2, j=jet;
  bool had_parents;

  while ((had_parents = j.has_parents(j1,j2))){
    // make j1 the more massive jet
    if (j1.m2() < j2.m2()) std::swap(j1,j2);

    // exit the loop if we pass the mass-drop condition
    if ((j1.m2() < _mu*_mu*j.m2()) &&
        (j1.kt_distance(j2) > _ycut*j.m2())) break;
    else j = j1;
  }

  if (!had_parents) return PseudoJet();
  else return j;
}
```

Step-by-step example: mass-drop tagger

2. implement the action of the transformer

```
PseudoJet MassDropTagger::result(const PseudoJet &jet) const{
  PseudoJet j1, j2, j=jet;
  bool had_parents;

  while ((had_parents = j.has_parents(j1,j2))){
    // make j1 the more massive jet
    if (j1.m2() < j2.m2()) std::swap(j1,j2);

    // exit the loop if we pass the mass-drop condition
    if ((j1.m2() < _mu*_mu*j.m2()) &&
        (j1.kt_distance(j2) > _ycut*j.m2())) break;
    else j = j1;
  }

  if (!had_parents) return PseudoJet();
  else return j;
}
```

Check conditions on '`jet`'

Associate structure to '`j`'

Step-by-step example: mass-drop tagger

3 What we want in the associated structure

```
class MassDropTaggerStructure :  public WrappedStructure{
public:
   MassDropTaggerStructure(const PseudoJet & result_jet);

   /// parameters that triggered the mass-drop condition
   inline double mu() const {return _mu;}
   inline double y() const {return _y;}

protected:
    double _mu, _y;
    friend class MassDropTagger; // for easier manipulation
};
```

Note: `WrappedStructure(other_structure)` behaves like `other_structure` and so allows for user additions

Step-by-step example: mass-drop tagger

4. create and attach the associated structure

```
PseudoJet MassDropTagger::result(const PseudoJet &jet) const{
   ...

   // create the associated structure
   MassDropTaggerStructure * s = new MassDropTaggerStructure(j);

   // fill it
   s->_mu = sqrt(j1.m2()/j.m2());
   s->_y = j1.kt_distance(j2)/j.m2();

   // attach it to the result
   j.set_structure_shared_ptr(SharedPtr<PseudoJetStructureBase>(s));
   return j;
}

MassDropTaggerStructure(const PseudoJet & res) :
   WrappedStructure(res.structure_shared_ptr()),_mu(0.0), _y(0.0){}
```

Step-by-step example: mass-drop tagger

5. check pre-conditions on `jet`

```
PseudoJet MassDropTagger::result(const PseudoJet &jet) const{
  // MassDrop applies on C/A jets

  // - check that the jet is coming from a clustering
  if (! j.has_associated_cluster_sequence())
    thow Error("...");

  // - and that the algorithm is C/A
  if (j.validated_cs()->jet_def().jet_algorithm()
        != cambridge_algorithm)
    thow Error("...");

  ...
}
```

Step-by-step example: extra note

Internal reclustering is often useful

e.g. filtering

```
PseudoJet result(const PseudoJet &jet) const{
   // check we have constituents
   if (! j.has_constituents())
      thow Error("...");
   ClusterSequence cs(jet.constituents(),
      JetDefinition(cambridge_algorithm, 1000.0));
   PseudoJet j = cs.inclusive_jets()[0];

   ...
}
```

clustering structure of '`j`' lost ('`cs`' goes out of scope)

Step-by-step example: extra note

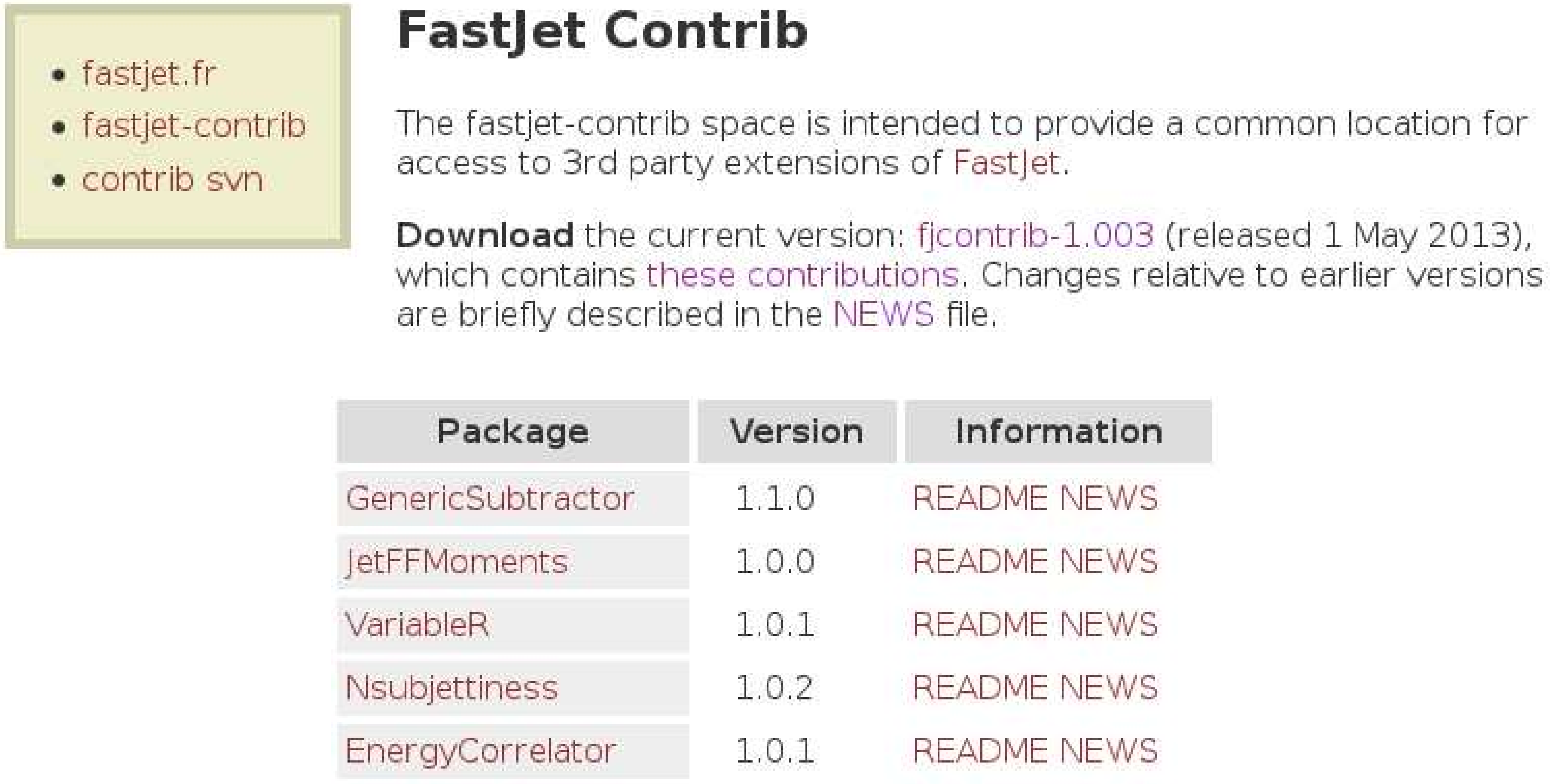
Internal reclustering is often useful

e.g. filtering

```
PseudoJet result(const PseudoJet &jet) const{
   // check we have constituents
   if (! j.has_constituents())
      thow Error("...");
   CS *cs = new ClusterSequence(jet.constituents(),
      JetDefinition(cambridge_algorithm, 1000.0));
   PseudoJet j = cs->inclusive_jets()[0];
   cs->delete_self_when_unused();
   ...
}
```

clustering structure of '`j`' kept (and no memory leak)

FastJet contrib (New: Feb 2013)



- fastjet.fr
- fastjet-contrib
- contrib svn

FastJet Contrib

The fastjet-contrib space is intended to provide a common location for access to 3rd party extensions of FastJet.

Download the current version: fjcontrib-1.003 (released 1 May 2013), which contains these contributions. Changes relative to earlier versions are briefly described in the NEWS file.

Package	Version	Information
GenericSubtractor	1.1.0	README NEWS
JetFFMoments	1.0.0	README NEWS
VariableR	1.0.1	README NEWS
Nsubjettiness	1.0.2	README NEWS
EnergyCorrelator	1.0.1	README NEWS

- a quick and uniform access to 3rd-party code
- contributors are welcome (please contact us)