Jet Physics: Tutorial

Sebastian Sapeta

LPTHE, UPMC, CNRS, Paris

Terascale Monte Carlo School 2011
14-18 March 2011, DESY Hamburg

Analysis framework

Software:

- `FastJet (v3.0alpha2)`
- `HepMC (v2.06.04)`
- `ROOT (v5.28.00a)`

Extra tools:

- `EventRecord.hh` – basic functionality for extracting events from `.hepmc.gz` files
- `SimpleHist.hh` – provides class that we will use for histogramming

Analysis framework

Data files (thanks to Simon Plätzer, Frank Siegert and Stefan Prestel!)

LHC $\sqrt{s} = 7$ TeV

- `dijets-no-ue.hepmc.gz`
  (Pythia 8.1.4.5, no MPI, $p_{t,\min}^{\text{hard proc}} = 60$ GeV)
- `dijets-with-ue.hepmc.gz`
  (Pythia 8.1.4.5, UE tune 3C, $p_{t,\min}^{\text{hard proc}} = 60$ GeV)
- `hz-bbar.hepmc.gz`
  (Herwig++ 2.5, with NLO matching, $Z \to e^+e^-$, $p_{t,H} > 120$ GeV, $m_H =$??? GeV [we will try to find out])
- `z-plus-jets.hepmc.gz`
  (Sherpa 1.2.3, LO with CKKW merging, $Z \to e^+e^-$)

Before starting

Example programs for this tutorial

```
/mc-school/jets-tutorial
```

Before starting

Example programs for this tutorial

```
/mc-school/jets-tutorial
```

Open `README` there and follow the instructions

- set `PATH` and `LD_LIBRARY_PATH` in `.bashrc`
- create link to directory with data files

Before starting

Example programs for this tutorial

```
/mc-school/jets-tutorial
```

Open README there and follow the instructions

- set PATH and LD_LIBRARY_PATH in .bashrc
- create link to directory with data files

In this tutorial, we will write the following programs:

```
algorithms.cc   jets.cc   ue.cc   subtraction.cc   boostedhiggs.cc
```

Before starting

Example programs for this tutorial

```
/mc-school/jets-tutorial
```

Open README there and follow the instructions

- set PATH and LD_LIBRARY_PATH in .bashrc
- create link to directory with data files

In this tutorial, we will write the following programs:

```
algorithms.cc  jets.cc  ue.cc  subtraction.cc  boostedhiggs.cc
```

A lot more can be found

- in the FastJet source directory fastjet-VERSION/example
- online http://www.fastjet.fr (Manual, Doxygen documentation)

Seeing jets

algorithms.cc

Pythia 8, LHC, $\sqrt{s} = 7$ TeV

- a single dijet event (no. 21) from `dijets-with-ue.hepmc.gz`

algorithms.cc

Pythia 8, LHC, $\sqrt{s} = 7$ TeV

- a single dijet event (no. 21) from `dijets-with-ue.hepmc.gz`

Clustering

- anti-k_t, R=0.6

  ```
  JetDefinition jet_def(antikt_algorithm,0.6);
  ```

algorithms.cc

Pythia 8, LHC, $\sqrt{s} = 7$ TeV

- a single dijet event (no. 21) from `dijets-with-ue.hepmc.gz`

Clustering

- anti-k_t, R=0.6

  ```
  JetDefinition jet_def(antikt_algorithm,0.6);
  ```

- additional information on jet areas

  ```
  AreaDefinition area_def(active_area_explicit_ghosts);
  ```

  option `active_area_explicit_ghosts` will allow us to get access to pure ghost jets

algorithms.cc

Pythia 8, LHC, $\sqrt{s} = 7$ TeV

- a single dijet event (no. 21) from `dijets-with-ue.hepmc.gz`

Clustering

- anti-k_t, R=0.6

  ```
  JetDefinition jet_def(antikt_algorithm,0.6);
  ```

- additional information on jet areas

  ```
  AreaDefinition area_def(active_area_explicit_ghosts);
  ```

  option `active_area_explicit_ghosts` will allow us to get access to pure ghost jets

- cluster and get the list of all jets

  ```
  ClusterSequenceArea cs(input_particles, jet_def, area_def);
  vector<PseudoJet> jets = sorted_by_pt(cs.inclusive_jets());
  ```

algorithms.cc

Print list of jets (our own function defined in EventRecord.hh)

```
print_jets(jets);
```

algorithms.cc

Print list of jets (our own function defined in `EventRecord.hh`)

```
print_jets(jets);
```

Visualize jets with ROOT

```
cs.print_jets_for_root(jets, "dijetevent-akt.out");
```

algorithms.cc

Print list of jets (our own function defined in `EventRecord.hh`)

```
print_jets(jets);
```

Visualize jets with ROOT

```
cs.print_jets_for_root(jets, "dijetevent-akt.out");
```

- take a look at the structure of the output file

```
jet-n jet-px jet-py jet-pz jet-E
 ipart eta phi pt
 ipart eta phi pt
```

algorithms.cc

Print list of jets (our own function defined in `EventRecord.hh`)

```
print_jets(jets);
```

Visualize jets with ROOT

```
cs.print_jets_for_root(jets, "dijetevent-akt.out");
```

- take a look at the structure of the output file

  ```
  jet-n jet-px jet-py jet-pz jet-E
   ipart eta phi pt
   ipart eta phi pt
  ```

- produce lego plot with showjets.C

  ```
  root
  root> .L showjets.C+
  root> showjets("dijetevent-akt.out")
  ```

algorithms.cc

Play a bit with parameters and look how results change

- get inclusive jets with $p_t >$ ptmin

  ```
  vector<PseudoJet> jets = sorted_by_pt(cs.inclusive_jets(5.0));
  ```

- see what happens when you go to larger jet radius, e.g. R$= 1.2$
- try other native algorithms: kt_algorithm, cambridge_algorithm
- try an external algorithm: SISCone
  - it comes as a plugin therefore we need to modify the way we set jet definition

    ```
    JetDefinition::Plugin *plugin;
    plugin = new SISConePlugin(0.6, 0.75);
    JetDefinition jet_def(plugin);
    ```

  - for speed reasons, it is good to use large area of ghosts

    ```
    GhostedAreaSpec gaspec(8.0, 1, 0.05);
    AreaDefinition area_def(active_area_explicit_ghosts,gaspec);
    ```

Seeing more jets

jets.cc

Selectors

```
Selector select_pt = SelectorPtMin(70.0);
Selector select_rapidity = SelectorRapRange(0.0,2.0);
Selector full_selection = select_pt && select_rapidity;
```

jets.cc

Selectors

```
Selector select_pt = SelectorPtMin(70.0);
Selector select_rapidity = SelectorRapRange(0.0,2.0);
Selector full_selection = select_pt && select_rapidity;
```

Clustering

```
JetDefinition jet_def(cambridge_algorithm,0.6);
ClusterSequence cs(input_particles, jet_def);
```

jets.cc

Selectors

```
Selector select_pt = SelectorPtMin(70.0);
Selector select_rapidity = SelectorRapRange(0.0,2.0);
Selector full_selection = select_pt && select_rapidity;
```

Clustering

```
JetDefinition jet_def(cambridge_algorithm,0.6);
ClusterSequence cs(input_particles, jet_def);


vector<PseudoJet> jets = cs.inclusive_jets();
vector<PseudoJet> selected_jets
      = sorted_by_pt(full_selection(jets));
if (selected_jets.size() == 0) continue;
PseudoJet hardest_selected_jet = selected_jets[0];
```

jets.cc

Add to histogram

```
hisjetspthardest.add_entry(hardest_selected_jet.perp());
```

Code that you find in `jets.cc` below

```
// write
```

writes the histogram of p_t of the hardest jet into a file.

jets.cc

Add to histogram

```
hisjetspthardest.add_entry(hardest_selected_jet.perp());
```

Code that you find in `jets.cc` below

```
// write
```

writes the histogram of p_t of the hardest jet into a file.

Now...

- change normalization from σ_{tot} to unity: `total_cs` $\rightarrow$ `1.0` in `norm`
- set some suffix for the output file, e.g. `string suffix = "rap02";`
- compile the program, run it and plot the histogram from the `.out` file

jets.cc

Add to histogram

```
hisjetspthardest.add_entry(hardest_selected_jet.perp());
```

Code that you find in `jets.cc` below

```
// write
```

writes the histogram of p_t of the hardest jet into a file.

Now...

- change normalization from σ_{tot} to unity: `total_cs` $\rightarrow$ `1.0` in `norm`
- set some suffix for the output file, e.g. `string suffix = "rap02";`
- compile the program, run it and plot the histogram from the `.out` file

When the above is done...

- change rapidity selection from the range $[0, 2]$ to $[2, 4]$
- change suffix
- run the program and compare results from the two above selections of rapidity ranges

jets.cc

Next, we will try to get *mass distribution of the hardest jet*

- modify the rapidity selection to

  ```
  Selector select_rapidity = SelectorAbsRapMax(4.0);
  ```

- in analogy with p_t histograms: add all code needed for writing histograms of the mass of hardest jet
- produce mass histograms with two p_t cuts: at 50 GeV and 200 GeV

Seeing more than jets: underlying event

ue.cc

Jet based techniques can be also used to measure the underlying event!

- area/median approach [Cacciari, Salam, Soyez (2008)]

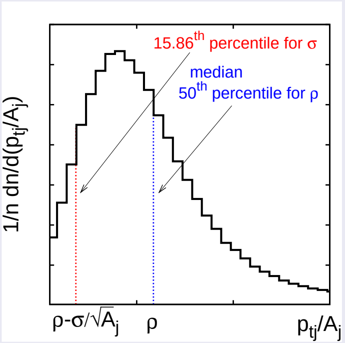
ue.cc

Jet based techniques can be also used to measure the underlying event!

- area/median approach [Cacciari, Salam, Soyez (2008)]

For each event

1. cluster particles with an infrared safe jet finding algorithm (all particles are clustered so we have set of jets ranging from hard to soft)
2. from the list of all jets (no cuts required!) determine
$$\rho = \text{median}\left[\left\{\frac{p_{t,j}}{A_j}\right\}\right]$$
and its uncertainty σ
   - median gives a typical value of p_t/A for a given event
   - using median is a way to **dynamically** separate hard and soft parts of the event



ue.cc

Let us try it with simulated dijet events with the underlying event at the LHC, $\sqrt{s} = 7$ TeV.

ue.cc

Let us try it with simulated dijet events with the underlying event at the LHC, $\sqrt{s} = 7$ TeV.

Set the rapidity range in which you want to measure the UE

```
RangeDefinition rapbin(0.0, 2.0);
```

ue.cc

Let us try it with simulated dijet events with the underlying event at the LHC, $\sqrt{s} = 7$ TeV.

Set the rapidity range in which you want to measure the UE

```
RangeDefinition rapbin(0.0, 2.0);
```

Get the value corresponding to the level of UE per unit area (`rho`) and its uncertainty (`sigma`)

```
double rho,sigma,mean_area;
bool use_area_4vector = true;
cs.get_median_rho_and_sigma(rapbin, use_area_4vector,
                            rho, sigma, mean_area);
```

ue.cc

Let us try it with simulated dijet events with the underlying event at the LHC, $\sqrt{s} = 7$ TeV.

Set the rapidity range in which you want to measure the UE

```
  RangeDefinition rapbin(0.0, 2.0);
```

Get the value corresponding to the level of UE per unit area (`rho`) and its uncertainty (`sigma`)

```
   double rho,sigma,mean_area;
   bool use_area_4vector = true;
   cs.get_median_rho_and_sigma(rapbin, use_area_4vector,
                               rho, sigma, mean_area);
```

Add `rho` to the histogram histogram

```
   hisrho.add_entry(rho);
```

ue.cc

Let us try it with simulated dijet events with the underlying event at the LHC, $\sqrt{s} = 7$ TeV.

Set the rapidity range in which you want to measure the UE

```
RangeDefinition rapbin(0.0, 2.0);
```

Get the value corresponding to the level of UE per unit area (rho) and its uncertainty (sigma)

```
double rho,sigma,mean_area;
bool use_area_4vector = true;
cs.get_median_rho_and_sigma(rapbin, use_area_4vector,
                            rho, sigma, mean_area);
```

Add rho to the histogram histogram

```
hisrho.add_entry(rho);
```

Compile and let it run for a while...

ue.cc

Take a look at the ρ-histogram

- the distribution is broad! underlying event fluctuates a lot from one event to another
- there is always a contribution to "zeroth bin", it comes from quiet events with pure ghost jets constituting more than half of all jets

ue.cc

Take a look at the ρ-histogram

- the distribution is broad! underlying event fluctuates a lot from one event to another
- there is always a contribution to "zeroth bin", it comes from quiet events with pure ghost jets constituting more than half of all jets

Now write the code needed to produce histograms of `sigma` and get results

- σ values are also rather large: underlying event fluctuates a lot from point to point in an event

ue.cc

Take a look at the ρ-histogram

- the distribution is broad! underlying event fluctuates a lot from one event to another
- there is always a contribution to "zeroth bin", it comes from quiet events with pure ghost jets constituting more than half of all jets

Now write the code needed to produce histograms of `sigma` and get results

- σ values are also rather large: underlying event fluctuates a lot from point to point in an event

Change the rapidity range

```
RangeDefinition rapbin(2.0, 4.0);
```

Modify the name of the output file, produce results for `rho` and `sigma` and compare them with those from the rapidity range $[0, 2]$.

ue.cc

Take a look at the ρ-histogram

- the distribution is broad! underlying event fluctuates a lot from one event to another
- there is always a contribution to "zeroth bin", it comes from quiet events with pure ghost jets constituting more than half of all jets

Now write the code needed to produce histograms of `sigma` and get results

- σ values are also rather large: underlying event fluctuates a lot from point to point in an event

Change the rapidity range

```
RangeDefinition rapbin(2.0, 4.0);
```

Modify the name of the output file, produce results for `rho` and `sigma` and compare them with those from the rapidity range $[0, 2]$.

- ρ distribution dependence on the rapidity range where the UE is measured

Seeing jets better

subtraction.cc

What else can we do with ρ and σ?

- ρ may be used e.g. to correct hard jet transverse momentum

  $$p_{t,j}^{(\mathrm{sub})} = p_{t,j} - \rho\, A_j \pm \sigma \sqrt{A_j}$$

  since jet area measures the jet susceptibility to the soft radiation

subtraction.cc

Choose the data sample without UE

```
string ueflag = "no-ue";
```

subtraction.cc

Choose the data sample without UE

```
string ueflag = "no-ue";
```

Get the hardest jet out of those that pass the selection

```
vector<PseudoJet> jets = cs.inclusive_jets();
vector<PseudoJet> selected_jets
    = sorted_by_pt(select_rapidity(jets));
PseudoJet hardest_selected_jet = selected_jets[0];
```

subtraction.cc

Choose the data sample without UE

```
string ueflag = "no-ue";
```

Get the hardest jet out of those that pass the selection

```
vector<PseudoJet> jets = cs.inclusive_jets();
vector<PseudoJet> selected_jets
    = sorted_by_pt(select_rapidity(jets));
PseudoJet hardest_selected_jet = selected_jets[0];
```

Add it to histogram

```
hispthardest.add_entry(hardest_selected_jet.perp());
```

subtraction.cc

Choose the data sample without UE

```
string ueflag = "no-ue";
```

Get the hardest jet out of those that pass the selection

```
vector<PseudoJet> jets = cs.inclusive_jets();
vector<PseudoJet> selected_jets
    = sorted_by_pt(select_rapidity(jets));
PseudoJet hardest_selected_jet = selected_jets[0];
```

Add it to histogram

```
hispthardest.add_entry(hardest_selected_jet.perp());
```

Run the program and produce the results for p_t distribution of the leading jet in the rapidity range $[-4, 4]$ in the case *without the underlying event*.

subtraction.cc

Now we go to the sample with the underlying event

```
string ueflag = "with-ue";
```

subtraction.cc

Now we go to the sample with the underlying event

```
string ueflag = "with-ue";
```

Go for the clustering with area

```
AreaDefinition area_def(active_area_explicit_ghosts);
ClusterSequenceArea cs(input_particles, jet_def, area_def);
```

subtraction.cc

Now we go to the sample with the underlying event

```
string ueflag = "with-ue";
```

Go for the clustering with area

```
AreaDefinition area_def(active_area_explicit_ghosts);
ClusterSequenceArea cs(input_particles, jet_def, area_def);
```

Determine `rho` on the event by event basis

```
 RangeDefinition rapbin(-4.0, 4.0);
 double rho,sigma,mean_area;
 cs.get_median_rho_and_sigma(selected_jets,rapbin, true,
                             rho, sigma, mean_area);
```

subtraction.cc

Subtract the UE contamination and add the result for corrected p_t to the histogram

```
PseudoJet vect_area = hardest_selected_jet.area_4vector();
PseudoJet jet_cor = hardest_selected_jet - rho*vect_area;
hispthardest_cor.add_entry(jet_cor.perp());
```

subtraction.cc

Subtract the UE contamination and add the result for corrected p_t to the histogram

```
PseudoJet vect_area = hardest_selected_jet.area_4vector();
PseudoJet jet_cor = hardest_selected_jet - rho*vect_area;
hispthardest_cor.add_entry(jet_cor.perp());
```

Add the corresponding code for writing `hispthardest_cor` to the file.

subtraction.cc

Subtract the UE contamination and add the result for corrected p_t to the histogram

```
PseudoJet vect_area = hardest_selected_jet.area_4vector();
PseudoJet jet_cor = hardest_selected_jet - rho*vect_area;
hispthardest_cor.add_entry(jet_cor.perp());
```

Add the corresponding code for writing `hispthardest_cor` to the file.

Now we take a look at the results

- compare results for jets without UE with the uncorrected results with UE
- add the corrected result with UE

subtraction.cc

Subtract the UE contamination and add the result for corrected p_t to the histogram

```
PseudoJet vect_area = hardest_selected_jet.area_4vector();
PseudoJet jet_cor = hardest_selected_jet - rho*vect_area;
hispthardest_cor.add_entry(jet_cor.perp());
```

Add the corresponding code for writing `hispthardest_cor` to the file.

Now we take a look at the results

- compare results for jets without UE with the uncorrected results with UE
- add the corrected result with UE

We see that the correction brings the spectrum from the events with UE close to that from the events without UE.

- result still can be improved, e.g. by optimizing R and rapidity range in which UE is determined

Seeing more inside jets: an example of subjet analysis

boostedhiggs.cc

The file /mc-school/data/hz-bbar.hepmc.gz provides 50 000 events for $pp \to ZH \to l^+l^- + b\bar{b}$.

boostedhiggs.cc

The file /mc-school/data/hz-bbar.hepmc.gz provides 50 000 events for $pp \rightarrow ZH \rightarrow l^+l^- + b\bar{b}$.

Get hadrons and leptons from this file (handled by EventRecord)

```
vector<PseudoJet> input_particles = evtrec.hadrons();
vector<PseudoJet> leptons = sorted_by_pt(evtrec.leptons());
```

boostedhiggs.cc

The file `/mc-school/data/hz-bbar.hepmc.gz` provides 50 000 events for $pp \to ZH \to l^+l^- + b\bar{b}$.

Get hadrons and leptons from this file (handled by `EventRecord`)

```
vector<PseudoJet> input_particles = evtrec.hadrons();
vector<PseudoJet> leptons = sorted_by_pt(evtrec.leptons());
```

Select only events with $p_{t,Z} > 200$ GeV and 80 GeV $< m_{l^+l^-} <$ 100 GeV

```
PseudoJet Zboson;
if (leptons.size()>=2)
  Zboson = leptons[0]+leptons[1];
  if (Zboson.perp() < 200 || Zboson.m() < 80 || Zboson.m() >100)
  continue;
```

boostedhiggs.cc

Cluster hadrons and accept only events with $p_{t,\text{hardest jet}} > 200\,\text{GeV}$

```
ClusterSequence cs(input_particles, jet_def);
vector<PseudoJet> jets = sorted_by_pt(cs.inclusive_jets(30.0));
if (jets.size()==0) continue;
PseudoJet hardest_jet = jets[0];
if (hardest_jet.perp() < 200) continue;
```

boostedhiggs.cc

Cluster hadrons and accept only events with $p_{t,\text{hardest jet}} > 200\,\text{GeV}$

```
ClusterSequence cs(input_particles, jet_def);
vector<PseudoJet> jets = sorted_by_pt(cs.inclusive_jets(30.0));
if (jets.size()==0) continue;
PseudoJet hardest_jet = jets[0];
if (hardest_jet.perp() < 200) continue;
```

Remember that our jets were defined with $R = 1.2$. By selecting only the events with high transverse momenta of the lepton pair (i.e. Z boson) and the leading jets we choose configurations with highly collimated products of the Higgs decay $H \to b\bar{b}$. Hence, we believe that b and $\bar{b}$ should end up in our large, leading jet.

boostedhiggs.cc

Add entry to the histogram

```
hismasshardest.add_entry(hardest_jet.m());
```

boostedhiggs.cc

Add entry to the histogram

```
hismasshardest.add_entry(hardest_jet.m());
```

Get the value of σ_{tot}

```
double total_cs = evtrec.total_cross_section();
```

boostedhiggs.cc

Add entry to the histogram

```
hismasshardest.add_entry(hardest_jet.m());
```

Get the value of σ_{tot}

```
double total_cs = evtrec.total_cross_section();
```

Change normalization in `norm`:

`1.0` $\rightarrow$ `total_cs`

boostedhiggs.cc

Add entry to the histogram

```
hismasshardest.add_entry(hardest_jet.m());
```

Get the value of σ_{tot}

```
double total_cs = evtrec.total_cross_section();
```

Change normalization in `norm`:

`1.0` $\rightarrow$ `total_cs`

Compile the program, run it and take a look at the mass distribution...

boostedhiggs.cc

The leading jet may sometimes be something else than the jet form Higgs decay.

boostedhiggs.cc

The leading jet may sometimes be something else than the jet form Higgs decay.

To improve the result we go for the analysis of jet's substructure which consists of the procedures of *mass drop* and *asymmetry cut*:

boostedhiggs.cc

The leading jet may sometimes be something else than the jet form Higgs decay.

To improve the result we go for the analysis of jet's substructure which consists of the procedures of *mass drop* and *asymmetry cut*:

1. decompose your fat jet j into two parent subjets j_1 and j_2

boostedhiggs.cc

The leading jet may sometimes be something else than the jet form Higgs decay.

To improve the result we go for the analysis of jet's substructure which consists of the procedures of *mass drop* and *asymmetry cut*:

1. decompose your fat jet j into two parent subjets j_1 and j_2
2. label them such that $m_{j_1} > m_{j_2}$

boostedhiggs.cc

The leading jet may sometimes be something else than the jet form Higgs decay.

To improve the result we go for the analysis of jet's substructure which consists of the procedures of *mass drop* and *asymmetry cut*:

1. decompose your fat jet j into two parent subjets j_1 and j_2
2. label them such that $m_{j_1} > m_{j_2}$
3. check the **mass drop condition**

$$m_{j_1} < \mu\, m_j \,,$$

   where μ is the mass drop parameter

boostedhiggs.cc

The leading jet may sometimes be something else than the jet form Higgs decay.

To improve the result we go for the analysis of jet's substructure which consists of the procedures of *mass drop* and *asymmetry cut*:

1. decompose your fat jet j into two parent subjets j_1 and j_2
2. label them such that $m_{j_1} > m_{j_2}$
3. check the **mass drop condition**

$$m_{j_1} < \mu \, m_j \, ,$$

   where μ is the mass drop parameter

4. check the **asymmetry condition**

$$\frac{\min(p_{t,j_1}, p_{t,j_2})}{\max(p_{t,j_1}, p_{t,j_2})} \simeq \frac{\min(p_{t,j_1}^2, p_{t,j_2}^2)\Delta_{12}^2}{m_j^2} > y_{\text{cut}}$$

boostedhiggs.cc

The leading jet may sometimes be something else than the jet form Higgs decay.

To improve the result we go for the analysis of jet's substructure which consists of the procedures of *mass drop* and *asymmetry cut*:

1. decompose your fat jet j into two parent subjets j_1 and j_2
2. label them such that $m_{j_1} > m_{j_2}$
3. check the **mass drop condition**

$$m_{j_1} < \mu\, m_j \,,$$

   where μ is the mass drop parameter

4. check the **asymmetry condition**

$$\frac{\min(p_{t,j_1}, p_{t,j_2})}{\max(p_{t,j_1}, p_{t,j_2})} \simeq \frac{\min(p_{t,j_1}^2, p_{t,j_2}^2)\Delta_{12}^2}{m_j^2} > y_{\text{cut}}$$

5. if the two conditions are satisfied close, exit the loop, if not redefine j to be equal to j_1 and go back to step 1

boostedhiggs.cc

To see how it works in practice, uncomment the following lines in the source file

```
// mass drop analysis
double mass_drop_threshold = 0.667;
double rtycut              = 0.3;
PseudoJet parent1, parent2;
bool had_parents;
while ((had_parents = hardest_jet.has_parents(parent1,parent2))) {
  if (parent1.m() < parent2.m()) swap(parent1,parent2);
  if (parent1.m() < mass_drop_threshold * hardest_jet.m() &&
      parent1.kt_distance(parent2) > pow(rtycut,2) * hardest_jet.m2()) {
    break;
  } else {
    hardest_jet = parent1;
  } }
```

boostedhiggs.cc

To see how it works in practice, uncomment the following lines in the source file

```
// mass drop analysis
double mass_drop_threshold = 0.667;
double rtycut              = 0.3;
PseudoJet parent1, parent2;
bool had_parents;
while ((had_parents = hardest_jet.has_parents(parent1,parent2))) {
  if (parent1.m() < parent2.m()) swap(parent1,parent2);
  if (parent1.m() < mass_drop_threshold * hardest_jet.m() &&
      parent1.kt_distance(parent2) > pow(rtycut,2) * hardest_jet.m2()) {
    break;
  } else {
    hardest_jet = parent1;
  } }
```

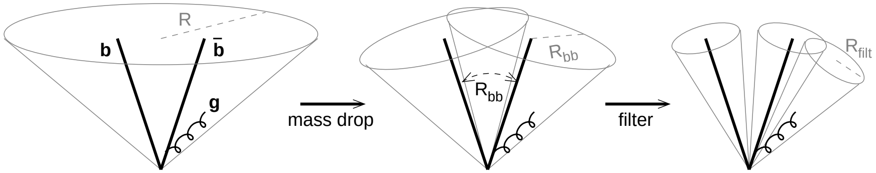
Change the name of the output file, compile the program, run it and compare that result with that without mass drop.

boostedhiggs.cc

To remove contamination from the underlying event and further improved the quality of the peak we add the procedure of *filtering*

boostedhiggs.cc

To remove contamination from the underlying event and further improved the quality of the peak we add the procedure of *filtering*



[from J. M. Butterworth, A. R. Davison, M. Rubin and G. P. Salam, Phys. Rev. Lett. **100** (2008) 242001]

boostedhiggs.cc

Define the new *filter* radius

```
double Rzqq = sqrt( parent1.squared_distance(parent2) );
double Rfilt = min( Rzqq/2.0,0.3);
```

boostedhiggs.cc

Define the new *filter* radius

```
double Rzqq = sqrt( parent1.squared_distance(parent2) );
double Rfilt = min( Rzqq/2.0,0.3);
```

Define the filter: recluster the fat jet with C/A algorithm with `Rfilt` and then take three hardest jets

```
Filter filter(JetDefinition(cambridge_algorithm,Rfilt),
              SelectorNHardest(3));
```

boostedhiggs.cc

Define the new *filter* radius

```
double Rzqq = sqrt( parent1.squared_distance(parent2) );
double Rfilt = min( Rzqq/2.0,0.3);
```

Define the filter: recluster the fat jet with C/A algorithm with `Rfilt` and then take three hardest jets

```
Filter filter(JetDefinition(cambridge_algorithm,Rfilt),
              SelectorNHardest(3));
```

Filter the hardest jet

```
PseudoJet filtered_jet = filter(hardest_jet);
```

boostedhiggs.cc

Define the new *filter* radius

```
double Rzqq = sqrt( parent1.squared_distance(parent2) );
double Rfilt = min( Rzqq/2.0,0.3);
```

Define the filter: recluster the fat jet with C/A algorithm with `Rfilt` and then take three hardest jets

```
Filter filter(JetDefinition(cambridge_algorithm,Rfilt),
              SelectorNHardest(3));
```

Filter the hardest jet

```
PseudoJet filtered_jet = filter(hardest_jet);
```

Check the *ptmin* criterion for the filtered jet and, if satisfied, add entry to the histogram

```
if (filtered_jet.perp() < 200 ) continue;
hismasshardest_filt.add_entry(filtered_jet.m());
```

boostedhiggs.cc

- get the same three histograms for $Z + j$

```
string proc = "z-plus-jets";
```

- compare how $H + Z$ looks on the top of $Z + j$ background
  be careful: the first process is in *nb* the second in *pb*