Table of Contents

Tips and tricks

xAOD migration

Running P-A's trimming script.

.

```
setupATLAS
mkdir dxAOD
cd dxAOD
asetup 19.1.2,here
```

This piece is not necessary for using P-A's script but I would recommend it anyway as you will want to use it later.

```
acmd.py cmt new-pkg JetStuff
cd JetStuff/cmt
cmt config
cd ../src
acmd.py gen-klass --klass TestAlgo --pkg JetStuff --type alg -o TestAlgo
```

At this point you can then do stuff to get jet loop following the tutorial SoftwareTutorialxAODAnalysisInAthena, or you can use my premade TestAlgo.cxx and edit it if you want.

```
cp /afs/cern.ch/user/a/asquith/public/TestAlgo.cxx .
```

Now continue:

```
cd ../cmt
checkreq.py
```

This command above tells you the packages you need to include in requirements for compilation. Open **`requirements`** and paste the following five lines into requirements file after the **`GaudiInterface`** piece"

```
private
use  AthenaBaseComps            AthenaBaseComps-*               Control
use  xAODEventInfo              xAODEventInfo-*                 Event/xAOD
use  xAODJet                    xAODJet-*                       Event/xAOD
end_private
```

Now continue:

```
cmt make
```

Now this bit is for using P-A's trimmiing script

```
cd $TestArea
cmt co -r JetRec-02-02-01 Reconstruction/Jet/JetRec
cd Reconstruction/Jet/JetRec/cmt/
cmt config
cmt make
```

Now open **`../python/JetRecStandardTools.py`** and comment out all references to **`JetLArHVTool`** and **`JetIsolationTool`** (10 lines in total)

Now continue:

```
cd $TestArea/JetStuff
mkdir run
cd run
cp /afs/cern.ch/user/a/asquith/public/PAtrimming.py .
athena.py PAtrimming.py
```

Now you have a dxaod with trimmed jets. Have a look at its contents

```
checkSG.py dxaod.pool.root
```

Substructure bits coming later.

How to do jet area based pileup subtraction on a substructure variable

DISCLAIMER : this is not an official recommendation.

This example will be for jet width.

In your header, make a class:

```
class Width : public FunctionOfPseudoJet<double>{
 public:
  virtual std::string description() const{return "Width";}
  virtual double result(const PseudoJet &jet) const;
};
```

and also decalre the GenericSubtractor in your header:

```
//pileup subtraction for jet shapes
//from  http://fastjet.hepforge.org/svn/contrib/contribs/GenericSubtractor/tags/1.2.0/GenericSubt
class GenericSubtractorInfo{
 public:
  /// returns the unsubtracted shape
  double unsubtracted() const{ return _unsubtracted;}

  /// returns the result of applying the subtraction including only the
  /// first-order correction
  double first_order_subtracted() const{ return _first_order_subtracted;}

  /// returns the result of applying the subtraction including the
  /// first and second-order corrections (this is the default returned
  /// by the subtractions).
  double second_order_subtracted() const{ return _second_order_subtracted;}

  /// returns an estimate of the 3rd order correction. Note that the
  /// third derivative is quite likely to be poorly evaluated in some
  /// cases. For this reason, it is not used by default.
  double third_order_subtracted() const{ return _third_order_subtracted;}
/// returns the first derivative (wrt to rho+rhom). Derivatives
  /// are taken assuming a constant value for the ratio of rhom/rho, as
  /// determined from the background estimates for the jet.
  ///
  /// Note: derivatives are not evaluated for shapes with components,
  /// and therefore set to zero.
  double first_derivative() const{ return _first_derivative;}

  /// returns the second derivative (wrt to rho+rhom). It is not evaluated
  /// for shapes with components, and therefore set to zero.
  double second_derivative() const{ return _second_derivative;}

  /// returns an estimate of the 3rd derivative (wrt to rho+rhom);
```

```
  /// note that the third derivative is quite likely to be poorly
  /// evaluated in some cases. It is not used by default. It is not
  /// evaluated for shapes with components, and therefore set to zero.
  double third_derivative() const{ return _third_derivative;}
/// returns the ghost scale actually used for the computation
  double ghost_scale_used() const{ return _ghost_scale_used;}

  /// return the value of rho and rhom used in the subtraction
  double rho() const  { return _rho;  }
  double rhom() const { return _rhom; }
protected:
  double _unsubtracted;
  double _first_order_subtracted;
  double _second_order_subtracted;
  double _third_order_subtracted;
  double _first_derivative, _second_derivative, _third_derivative;
  double _ghost_scale_used;
  double _rho, _rhom;

  friend class GenericSubtractor;
};
```

In your src, somewhere callable, write down the jet width calculation:

```
double Width::result(const PseudoJet &jet) const{
  double width=-1.;
  if (!jet.has_constituents()){
    cout<<"ERROR! Width can only be applied on jets for which the constituents are known."<<endl;
    return width;
  }
  double jetWidth=0.;
  double jetPtSum=0.;
  vector<PseudoJet> constits = jet.constituents();
  for (vector<PseudoJet>::iterator cit=constits.begin(); cit!=constits.end(); cit++){
    const PseudoJet & cp = *cit;
    double wPt = cp.perp();
    if (wPt > 0) {
      double wdPhi = jet.phi() - cp.phi();
      if(wdPhi>PI) wdPhi=2*PI-wdPhi;
      double wdEta = jet.eta() - cp.eta();
      double wdR=sqrt(wdPhi*wdPhi + wdEta*wdEta);
      jetWidth += wdR * wPt;
      jetPtSum += wPt;
    }
  }
  width = jetWidth / jetPtSum;
  return width;
}
```

In your event loop, pass your jet and your width class to the GenericSubtraction:

```
double rho = Eventshape_rhoKt4LC; // for example
GenericSubtractor gsub(rho,0.);
Width s_width;
double subtracted_width = gsub(s_width, jet );
```

Fastjet

Show fastjet tips and tricks Hide fastjet tips and tricks
If you do not have your favourite groomed jet collections in your DPD, but you do have cluster four-vectors,

it is possible to do approximately anything you like with fastjet 3 manual, doxygen, main page.

Below are a few examples that may be useful.

A **`PseudoJet`** object is like a **`TLorentzVector`** - the most basic use is to just give it the (px,py,pz,e) of the object.

Make a vector of objects corresponding to the clusters associated with the jets in your DPD

Show code snippet Hide code snippet

```
#include "fastjet/PseudoJet.hh"
using namespace fastjet;
...
// Make a vector to hold our PseudoJet objects, which in this case will be positive energy cluste
std::vector<PseudoJet> clusters;
clusters.clear();

// get the constituents associated with the leading jet (this is the ->at(0) bit)
int nclusters_jet0 = jet_AntiKt6LCTopo_constit_index->at(0).size();
  for (int i = 0; i !=nclusters_jet0; i++) {

    // note that the indexed associated clusters are not in pT order
    int cluster_i = (jet_AntiKt6LCTopo_constit_index->at(0)).at(i);
    double e_cluster = cl_had_E->at(cli);
    double pt_cluster = cl_had_pt->at(cli);
    double px_cluster = pt_cluster*cos(cl_had_phi->at(cli);
    double py_cluster = pt_cluster*sin(cl_had_phi->at(cli);
    double pz_cluster = pt_cluster*sinh(cl_had_eta->at(cli);

    //only take clusters with positive energy, as in standard athena jet reconstuction
    if(e_cluster > 0. ){
        PseudoJet this_cluster(px_cluster, py_cluster, pz_cluster, e_cluster);

        // put the cluster into your vector
        clusters.push_back(this_cluster);
   }
}
```

Build a jet from your clusters

Show code snippet Hide code snippet

```
#include "fastjet/PseudoJet.hh"
#include "fastjet/JetDefinition.hh"
#include "fastjet/ClusterSequenceArea.hh"
#include "fastjet/AreaDefinition.hh"
#include "fastjet/Selector.hh"
using namespace fastjet;
...
// this snippet assumes you already have a vector<PseudoJet> clusters from the previous example
// we will make an anti-kt jet with R=0.6
double rad = 0.6;
// the choice of relevant algorithms is: antikt_algorithm, kt_algorithm, cambridge_algorithm -see
JetDefinition jd = JetDefinition(antikt_algorithm,rad);
// the area is defined as active with explicit ghosts
AreaDefinition area_def(active_area_explicit_ghosts,GhostedAreaSpec(SelectorAbsRapMax(4.0)));
ClusterSequenceArea cs(clusters,jd,area_def);
// we make a vector of PseudoJet objects, which in this case are antkt 0.6 jets. The minimum jet
vector<PseudoJet> jets=sorted_by_pt( cs.inclusive_jets(5000.) );
if(jets.size()==0){return 1;}
//access the leading jet
PseudoJet lead = jets[0];
//access the sub-leading jet if there is one
PseudoJet sublead(0.,0.,0.,0.);
```

```
if(jets.size()>1)sublead = jets[1];
```

Basic operations with your new jet collection

Show code snippet Hide code snippet

```
//this snippet assumes you already have a vector<PseudoJet> jets from the previous example.
int Njets = jets.size();

//get the pT of the leading jet
double pt1 = jets[0].perp();
double phi1 = jets[0].phi_std(); // phi_std() returns phi in the range -PI -> PI
double eta1 = jets[0].eta();
double e1 = jets[0].e();

//compare it with the constituent scale (no jet-level calibration) DPD jet_AntiKt6LCTopo_*
// if any of these are non-zero (within precision, noting that the returned value is in MeV), som
double pt_diff = pt1 - jet_AntiKt6LCTopo_constscale_pt->at(0);
double eta_diff = eta1 - jet_AntiKt6LCTopo_constscale_eta->at(0);
double phi_diff = phi1 - jet_AntiKt6LCTopo_constscale_phi->at(0);
double e_diff = e1 - jet_AntiKt6LCTopo_constscale_E->at(0);

//iterate over your vector of PseudoJets
int i=0;
for (vector<PseudoJet>::iterator jit=jets.begin(); jit!=jets.end(); jit++){
        const PseudoJet & jet = *jit;
        cout<<"jet["<<i<<"] has mass: "<<jet.m()   <<" MeV"<<endl;
         i++;
 }
```

Accessing the jet constituents or subjets

Show code snippet Hide code snippet

```
//this snippet assumes you already have a vector<PseudoJet> jets from the previous example.

if (jet.has_constituents()){
   vector<PseudoJet> constits = jets[0].constituents();
}

if (jet.has_pieces()){
   vector<PseudoJet> subjets = jets[0].pieces();
}
```

Calibration issues

Make jets with jet-level calibration applied: (See ApplyJetCalibration2012 for the latest recommendations- this example is for 20th May 2013)

Show code snippet Hide code snippet

```
#include "fastjet/PseudoJet.hh"
...
vector<PseudoJet> holder;
holder.clear();

// Get the rho, mu and npv for calibration
double rho = Eventshape_rhoKt4LC;
double mu = averageIntPerXing;
int npv=0, nv=vx_n;
  for(int i=0; i!=nv; i++){
    if( vx_trk_n->at(i) > 4 ){ // primary vertices are those with >4 tracks
```

```
      npv++;
    }
  }

  int n= jet_AntiKt6LCTopo_pt->size();
   for(int i=0; i!=n; i++){
    double Eraw    = jet_AntiKt6LCTopo_constscale_E->at(i);
    double eta     = jet_AntiKt6LCTopo_constscale_eta->at(i);
    double phi     = jet_AntiKt6LCTopo_constscale_phi->at(i);
    double m       = jet_AntiKt6LCTopo_constscale_m->at(i);
    double Ax      = jet_AntiKt6LCTopo_ActiveAreaPx->at(i);
    double Ay      = jet_AntiKt6LCTopo_ActiveAreaPy->at(i);
    double Az      = jet_AntiKt6LCTopo_ActiveAreaPz->at(i);
    double Ae      = jet_AntiKt6LCTopo_ActiveAreaE->at(i);

    //JES_Full2012dataset_Preliminary_Jan13.config
    TLorentzVector jet = glob.myJES->ApplyJetAreaOffsetEtaJES(Eraw,eta,phi,m,Ax,Ay,Az,Ae,rho,mu,n

    // Take the four momenta of the TLorentzVector jet and make a PseduoJet jet:
    PseudoJet temp(jet.Px(),jet.Py(),jet.Pz(),jet.E());
    holder.push_back(temp);
  }

  // these jets are calibrated - note that they don't have any clusters associated with them
  vector<PseudoJet> calib_jets = sorted_by_pt(holder);
```

Make jets with jet-level calibration applied **and associated clusters** : The point of this is that the jets you made in the example above do not have any associated clusters. If you need the clusters and want the jet to have calibrated four momentum, then you can try this way (note that the clusters are not affected by the calibration):

Show code snippet Hide code snippet

```
//this snippet assumes you already have a vector<PseudoJet> jets made from clusters, and a vector

//iterate over your jets (the ones you made from clusters)
for (vector<PseudoJet>::iterator jit=jets.begin(); jit!=jets.end(); jit++){
        const PseudoJet & jet = *jit;
        cout<<"jet["<<i<<"] has pT: "<<jet.perp()   <<" MeV"<<endl;
        jet.reset_momentum(calib_jets[0]);
        cout<<"Recalibrated jet["<<i<<"] has pT: "<<jet.perp()   <<" MeV"<<endl;
 }
```

Making groomed jets

Show code snippet Hide code snippet

```
//this snippet assumes you already have a vector<PseudoJet> jets from the previous example.


#include "fastjet/tools/Filter.hh"
#include "fastjet/tools/Pruner.hh"
#include "fastjet/Selector.hh"

...
// Trim a jet
Filter trimmer(0.3, SelectorPtFractionMin(0.05));
//apply trimming to the leading pT jet:
PseudoJet trimmed_jet1 =trimmer(jets[0];);
```

```
// Prune a jet :
Pruner pruner(cambridge_algorithm,0.06,0.3);  // zcut=0.06 , Rfilt=0.3
PseudoJet pruned_jet1 =prune(jets[0]);
...

// Filter a jet
Selector sel = SelectorNHardest(3); // how many subjets you require
Filter filter(0.3, sel); // Rfilt=0.3
PseudoJet filtered_jet1 =filter(jets[0]);
```

Doing jet area-based pileup correction on the fly while grooming

Show code snippet Hide code snippet

```
#include "fastjet/tools/Filter.hh"
...
// just like normal trimming, but you are passing in the pT density per unit area, rho
Filter actrimmer(0.3, SelectorPtFractionMin(0.05),rho);
PseudoJet areacro_trimmed_jet1 = actrimmer(jets[0]);
```

This topic: Sandbox > LilyAsquithSandbox
Topic revision: r4 - 2014-09-20 - LilyAsquith