Jet/Etmiss Analysis Tools

Chris Young
On behalf of the Jet/Etmiss group

ATLAS Software Tutorial

October 21, 2019

Supported tools by Jet/Etmiss

- Besides the EDM, an important aspect of the jet software are the jet tools needed in every analysis
- These tools are typically "Dual use" tools, i.e. they work in Athena and EventLoop/similar
- We typically distinguish between two types of tools
  1. **Analysis Task tools** (jet calibration, jet uncertainties, pile-up jet identification, boosted object tagging, ...)
  2. **Reconstruction** (jet finding, jet grooming, add attributes to the jets)
- Will focus today on the first bullet point as reconstruction tasks are typically done at derivation level already for you!
- If you are interested in jet reconstruction, take a look at the rel21 tutorial
- The Jet Software and Validation (JSV) group is responsible for the analysis tools, don't hesitate to send your questions to the mailing list: atlas-cp-jetetmiss-software-and-validation@cern.ch

Jets in xAODs and supported collections

- Jets are stored in JetContainer objects
- Currently supported jet collections by the Jet/Etmiss group
  1. `AntiKt4EMPFlowJets` (default small-R jets)
     - anti-k_t $R = 0.4$ jets reconstructed from particle flow objects
  2. `AntiKt4EMTopoJets`:
     - anti-k_t $R = 0.4$ jets reconstructed from clusters at EM scale
  3. `AntiKt10LCTopoTrimmedPtFrac5SmallR20Jets` (default large-R jets)
     - anti-k_t $R = 1.0$ jets reconstructed from clusters at LC scale
     - A grooming algorithm is applied to mitigate effects from pile-up, soft and wide angle radiation: trimming
     - Other grooming algorithms are actively studied in the Jet/Etmiss group
  4. `AntiKt10TrackCaloClusterTrimmedPtFrac5SmallR20Jets`
     - anti-k_t $R = 1.0$ jets reconstructed from TCC objects and trimmed
     - Perform very well at high p_{T}
- `AntiKt4EMTopoJets` and `AntiKt4EMPFlowJets` are available in xAODs
- Large-radius are reconstructed only at derivation level as well as truth jets and track-jets!!!

Jet cleaning

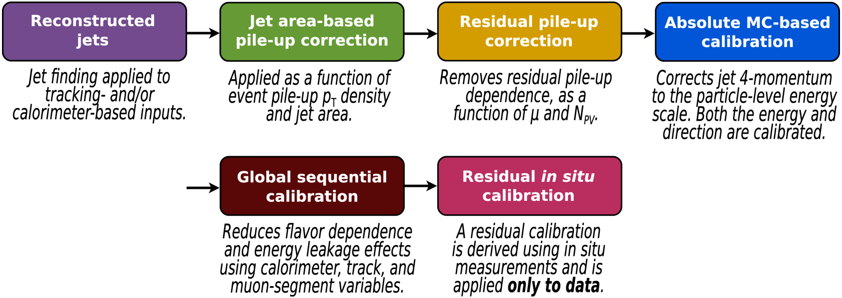
- Need to distinguish real jets arising from *pp* collisions from those originating from a non-collision background (NCB) process (e.g. beam-induced bkg, cosmic ray showers) or calorimeter noise
- Variables to distinguish between good and fake jets are based on the LAr signal pulse shapes, energy ratio variables, and track-based variables
- It is not sufficient to simply remove a fake jet as it can also affect other features of the event, such as the $E_{\mathrm{T}}^{\mathrm{miss}}$ calculation for example
- Therefore the full event should be removed with one or more unclean jets
- An **event cleaning** variable for loose jet cleaning is implemented in the derivations (`DFCommonJets_eventClean_LooseBad`)
    - This cleaning variable should be sufficient for most analyses
- If your analysis is highly affected by NCB, you should apply a tight cleaning criteria to the leading central jet (after overlap removal)

Tight bad jet labelling:

```
((std::fabs(eta)<2.4 && (SumPtTrkPt500[0]/pt)/FracSamplingMax<0.1) || FracSamplingMax<DBL_MIN)
```

Jet calibration chain

- Jets are calibrated to the truth jet energy scale to correct for non-compensation of the calorimeter, inactive material, signal losses, out-of-cone particles, etc
- A residual in situ calibration is derived and applied to data to bring the data to the same scale as MC
- The calibration chain for large-radius jets differs slightly (no pile-up corrections due to applied grooming algorithms, no GSC at the moment)
- **Jets are not fully calibrated in the (D)xAOD so you need to run this at analysis level!**



- The calibration is applied to the jets using JetCalibTools

JetCalibTools - Tool initialisation

Tool initialisation

```
#include "AsgTools/AnaToolHandle.h"
#include "JetCalibTools/IJetCalibrationTool.h"

asg::AnaToolHandle<IJetCalibrationTool> JetCalibrationTool_handle;

m_jetCalibration.setTypeAndName("JetCalibrationTool/myCalibTool");
if( !JetCalibrationTool_handle.isUserConfigured() ){
   ANA_CHECK( ASG_MAKE_ANA_TOOL(m_jetCalibration, JetCalibrationTool) );
   ANA_CHECK( m_jetCalibration.setProperty("JetCollection",jetAlgo.Data()) );
   ANA_CHECK( m_jetCalibration.setProperty("ConfigFile",config.Data()) );
   ANA_CHECK( m_jetCalibration.setProperty("CalibSequence",calibSeq.Data()) );
   ANA_CHECK( m_jetCalibration.setProperty("IsData",isData) );
   ANA_CHECK( m_jetCalibration.retrieve() );
}
```

- `TString jetAlgo = " "; // Name of jet collection`
    - e.g. `AntiKt4EMPFlow` or `AntiKt10LCTopoTrimmedPtFrac5SmallR20`
    - WARNING: when retrieving jet collections from the event store, you need to use `AntiKt4EMPFlowJets` or `AntiKt10LCTopoTrimmedPtFrac5SmallR20Jets`

JetCalibTools - Tool initialisation

Tool initialisation

```
#include "AsgTools/AnaToolHandle.h"
#include "JetCalibTools/IJetCalibrationTool.h"

asg::AnaToolHandle<IJetCalibrationTool> JetCalibrationTool_handle;

m_jetCalibration.setTypeAndName("JetCalibrationTool/myCalibTool");
if( !JetCalibrationTool_handle.isUserConfigured() ){
  ANA_CHECK( ASG_MAKE_ANA_TOOL(m_jetCalibration, JetCalibrationTool) );
  ANA_CHECK( m_jetCalibration.setProperty("JetCollection",jetAlgo.Data()) );
  ANA_CHECK( m_jetCalibration.setProperty("ConfigFile",config.Data()) );
  ANA_CHECK( m_jetCalibration.setProperty("CalibSequence",calibSeq.Data()) );
  ANA_CHECK( m_jetCalibration.setProperty("IsData",isData) );
  ANA_CHECK( m_jetCalibration.retrieve() );
}
```

- `TString config = " "; // Name of config file`
  - Latest information can be found on ApplyJetCalibrationR21 twiki

JetCalibTools - Tool initialisation

Tool initialisation

```
#include "AsgTools/AnaToolHandle.h"
#include "JetCalibTools/IJetCalibrationTool.h"

asg::AnaToolHandle<IJetCalibrationTool> JetCalibrationTool_handle;

m_jetCalibration.setTypeAndName("JetCalibrationTool/myCalibTool");
if( !JetCalibrationTool_handle.isUserConfigured() ){
   ANA_CHECK( ASG_MAKE_ANA_TOOL(m_jetCalibration, JetCalibrationTool) );
   ANA_CHECK( m_jetCalibration.setProperty("JetCollection",jetAlgo.Data()) );
   ANA_CHECK( m_jetCalibration.setProperty("ConfigFile",config.Data()) );
   ANA_CHECK( m_jetCalibration.setProperty("CalibSequence",calibSeq.Data()) );
   ANA_CHECK( m_jetCalibration.setProperty("IsData",isData) );
   ANA_CHECK( m_jetCalibration.retrieve() );
}
```

- `TString calibSeq = ; // Calibration sequence`
  - Tells tool which steps of calibration chain to apply (differs typically between data and MC)
  - e.g. `JetArea_Residual_EtaJES_GSC_Insitu` (for data)
    - Each step corresponds to one of the steps on slide 5

JetCalibTools - Tool initialisation

Tool initialisation

```
#include "AsgTools/AnaToolHandle.h"
#include "JetCalibTools/IJetCalibrationTool.h"

asg::AnaToolHandle<IJetCalibrationTool> JetCalibrationTool_handle;

m_jetCalibration.setTypeAndName("JetCalibrationTool/myCalibTool");
if( !JetCalibrationTool_handle.isUserConfigured() ){
   ANA_CHECK( ASG_MAKE_ANA_TOOL(m_jetCalibration, JetCalibrationTool) );
   ANA_CHECK( m_jetCalibration.setProperty("JetCollection",jetAlgo.Data()) );
   ANA_CHECK( m_jetCalibration.setProperty("ConfigFile",config.Data()) );
   ANA_CHECK( m_jetCalibration.setProperty("CalibSequence",calibSeq.Data()) );
   ANA_CHECK( m_jetCalibration.setProperty("IsData",isData) );
   ANA_CHECK( m_jetCalibration.retrieve() );
}
```

- `bool isData = true/false; // bool describing if the events are data or simulation`

JetUncertainties

- Uncertainties due to calibration procedure are provided by JetUncertainties

Tool initialisation

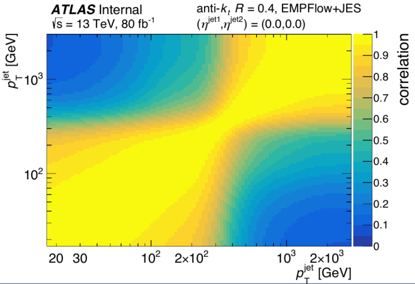
```
// Include the necessary header
#include "JetUncertainties/JetUncertaintiesTool.h"

// Create the tool
JetUncertaintiesTool jesProv("JESProvider");

// Set the properties
ANA_CHECK(jesProv.setProperty("JetDefinition","")); //Jet collection name
ANA_CHECK(jesProv.setProperty("MCType",""));
// MC16 (AFII) for full (fast) simulation
ANA_CHECK(jesProv.setProperty("ConfigFile","")); //Name of config file
ANA_CHECK(jesProv.setProperty("IsData",false));

// Initialize the tool
CHECK(jesProv.initialize());
```

- Take a look at the backup for details on how to apply the jet uncertainties

Nuisance parameter reduction

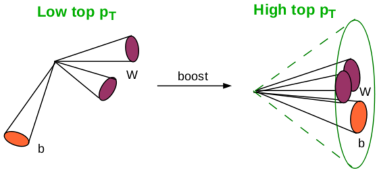
- JetUncertainties handles uncertainties both on the jet energy scale and jet energy resolution
- The in situ JES calibration results in approximately 100 nuisance parameters that need to be propagated to analyses
- Not all analyses are aiming at performing precise jet-dependent measurements or are insensitive to JES correlations
- Therefore various different configurations are provided with a reduced set of nuisance parameters with a minimum correlation loss achieved by using an eigenvalue decomposition
- For the JES, these are:
  - CategoryReduction: ~ 30 NP
  - GlobalReduction: ~ 20 NP (cannot be used for combinations)
  - StrongReduction: sets of $6-7$ NP (cannot be used for combinations)

Full jet-jet correlation matrix



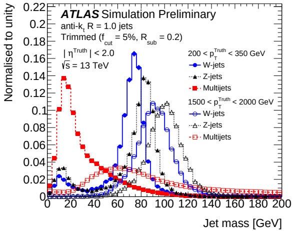
Boosted object identification

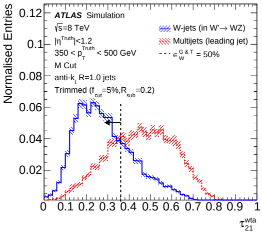
- At high $p_{\mathrm{T}}(\gtrsim 200\ \mathrm{GeV})$, the decay products of W/Z/H bosons or top quarks are Lorentz-boosted and start overlapping $\rightarrow$ reconstruct instead as one large-radius jet ($R \sim 1.0$)



- The inner structure (**substructure (JSS)**) of a quark/gluon initiated jet looks very different from a jet containing the decay products of a heavy object
- Discriminating variables: jet mass, ratios of energy correlation functions and N-subjettiness, or b-tagging information for $H \rightarrow b\bar{b}$ decays

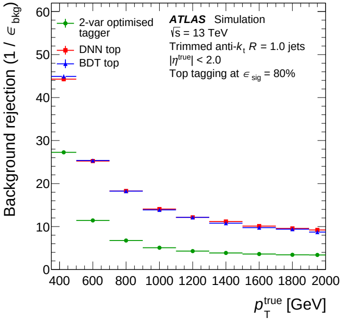
- Simple taggers:
  cut on 2-3 JSS variables
- Multivariate taggers:
  combine several var. in Deep Neural Network

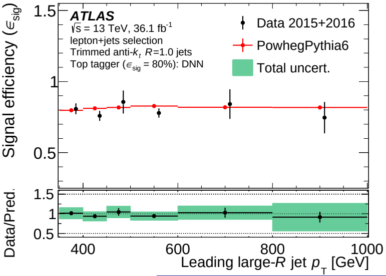




Boosted object tagger and their calibration

- Taggers are either optimised for fixed signal efficiencies (50%, 80%) or to achieve the best significance as a function of jet p_{T}
- Huge improvements observed for top tagging with multivariate techniques
- Supported taggers are implemented in BoostedJetTaggers





- Efficiency of tagger in MC is calibrated to that in data using various final states:
  - $t\bar{t}$ events, V+jets, γ+jets, dijets
- Scale factor (SF):

$$\mathrm{SF} = \frac{\epsilon_{\mathrm{data}}}{\epsilon_{\mathrm{MC}}}$$

BoostedJetTaggers

Tool initialisation

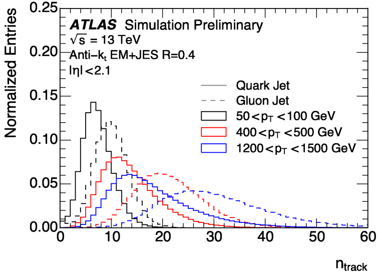
```
#include "AsgTools/AnaToolHandle.h"
#include "JetAnalysisInterfaces/IJetSelectorTool.h"
#include "BoostedJetTaggers/TAGGER_TYPE.h"

// TAGGER_TYPE: e.g. SmoothedWZTagger (simple 3-variable W/Z tagger), JSSWTopTaggerDNN (DNN top tagger)

//declare the tagger
asg::AnaToolHandle<IJetSelectorTool> m_jetTagger; //!

//Initialization
ASG_SET_ANA_TOOL_TYPE( m_jetTagger, TAGGER_TYPE); // same as above
m_jetTagger.setName("TOOL_NAME");
m_jetTagger.setProperty( "ConfigFile", "CONFIG_NAME"); //name of config file
m_jetTagger.setProperty( "CalibArea", "CALIBAREA"); //specify location of config file
m_jetTagger.setProperty( "IsMC", m_IsMC); // boolean
m_jetTagger.retrieve();
```

Detailed tagger information

```
for ( auto *ijet : *my_jets ) {
  const Root::TAccept& taggerResult = m_jetTagger->tag( *ijet ); // can be used as boolean
  //TAccept can be used to receive more detailed information about what cut failed, e.g.:
  bool passMassCutLow = taggerResult.getCutResult("PassMassLow");
  bool passMassCutHigh = taggerResult.getCutResult("PassMassHigh");
  // Retrieve DNN score for 80% top tagger
  ijet->auxdata<float>("DNNTaggerTopQuark80_Score") // Name varies depending on tagger
}
```

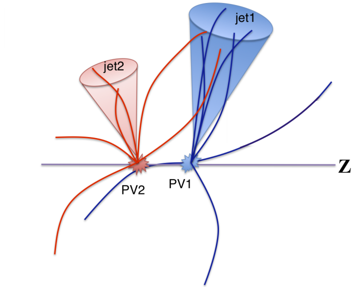
Quark/gluon tagging

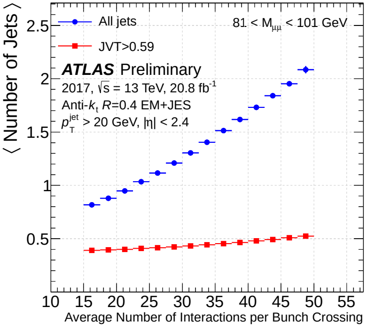
- Distinguishing quark-initiated jets from gluon-initiated jets is beneficial for many measurement and searches
- Gluon jets tend to have more constituents and a broader radiation pattern than quark jets
- Best discriminating variable: number of tracks matched to the jet”
  - Many derivations slim the InDetTrackParticles container, therefore make sure to include q/g tagger already at derivation level
- R&D studies for a BDT tagger are on-going but significant improvements are observed at low p_{T}



Effects of pile-up on jet reconstruction

- Pile-up has two main effects on jet reconstruction:
  1. Energy contribution to hard scattering (HS) objects
     - Biasing total jet energy, and smearing resolution of hard scatter jets
     - These effects are removed in the calibration chain (jet area correction and residual pile-up correction)
  2. Objects originating from pile-up (PU) vertex
     - Increases the number of reconstructed jets



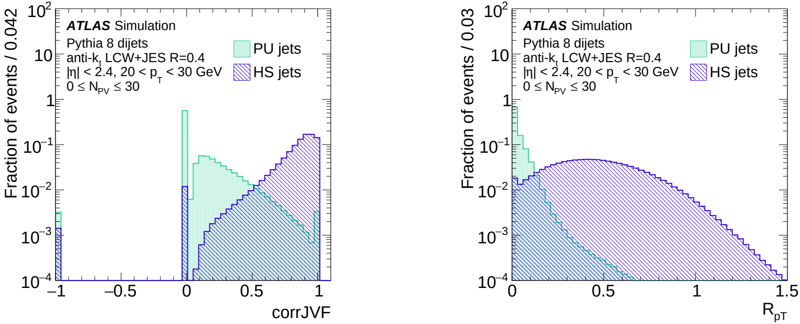


Jet vertex tagger

- Two-dimensional likelihood jet-vertex-tagger (JVT) is constructed from track-based variables to discriminate jets originating from HS or PU vertex
- fJVT constructed in forward region based on jet p_{T} momentum projection on missing transverse momentum per vertex
- Efficiency of JVT criteria in MC is calibrated to match those in data

$$\mathrm{corrJVF} = \frac{\sum_m p_{\mathrm{T},m}^{\mathrm{track}}(\mathrm{PV}_0)}{\sum_\ell p_{\mathrm{T},\ell}^{\mathrm{track}}(\mathrm{PV}_0) + \frac{\sum_{n\geq 1}\sum_\ell p_{\mathrm{T},\ell}^{\mathrm{track}}(\mathrm{PV}_n)}{k \cdot n_{\mathrm{track}}^{\mathrm{PU}}}}$$

$$R_{p_{\mathrm{T}}} = \frac{\sum_k p_{\mathrm{T},k}^{\mathrm{track}}(\mathrm{PV}_0)}{p_{\mathrm{T}}^{\mathrm{jet}}}$$



JVT Tool

- Several working points are available, details can be found on twiki

Jet collection	Working point	η range	p_{T} range	JVT criteria
pFlow	Tight (default)	$\|\eta\| < 2.4$	$20 < p_{\mathrm{T}} < 60$ GeV	JVT > 0.5
	Medium	$\|\eta\| < 2.4$	$20 < p_{\mathrm{T}} < 60$ GeV	JVT > 0.2

Tool initialisation and usage

```
// Include the necessary header
#include "JetJvtEfficiency/JetJvtEfficiency.h"

// Initialize the tool
CP::JetJvtEfficiency *jetsf = new CP::JetJvtEfficiency("jetsf");
jetsf->setProperty("WorkingPoint","Medium");
jetsf->setProperty("SFFile",""); // specify ROOT file here
ANA_CHECK(jetsf->initialize());

// Check if jet passes JVT criteria:
bool passJvt = jetsf->passesJvtCut(*jet);

// Retrieve event-wide MC-to-data scale factor
jetsf->applyAllEfficiencyScaleFactor(jets, sf);
```

Useful Links

Jet/Etmiss main twiki:

https://twiki.cern.ch/twiki/bin/view/AtlasProtected/JetEtMiss

Jet/Etmiss rel21 recommendations:

https://twiki.cern.ch/twiki/bin/view/AtlasProtected/JetEtmissRecommendationsR21

Jet/Etmiss publications:

https://twiki.cern.ch/twiki/bin/view/AtlasPublic/JetEtmissPublicResults

Release 21 jet reconstruction tutorial:

https://gitlab.cern.ch/atlas-jetetmiss/JetRecoTutorial_R21

Available jet attributes in Run 2:

https://twiki.cern.ch/twiki/bin/view/AtlasProtected/Run2JetMoments

Backup

JetCalibTools - how to apply

- There are two ways of applying the calibration to the jets:
  1. Use a calibrated copy
  2. Create shallow copy of jet container

Calibrated copy

```
const xAOD::JetContainer * my_jets = 0;
std::string jetCollectionName = ""; // Name of jet collection ...
// Don't forget "Jets" at the end of container name
ANA_CHECK(m_event->retrieve( my_jets, jetCollectionName));

for ( auto *ijet : *my_jets ) {
    xAOD::Jet * jet = 0;
    m_jetCalibration->calibratedCopy(*ijet,jet);
    // Do some other stuff here with the jet, e.g
    if(jet->pt() < 20000.) continue;
    delete jet;
}
```

JetCalibTools - how to apply - II

- There are two ways of applying the calibration to the jets:
  1. Use a calibrated copy
  2. Create shallow copy of jet container

Shallow copy

```
// Include in header file
#include "xAODCore/ShallowCopy.h"

// In executive function:
const xAOD::JetContainer * my_jets = 0;
std::string jetCollectionName = ""; // Name of jet collection ...
// Don't forget "Jets" at the end of container name
ANA_CHECK(m_event->retrieve( my_jets, jetCollectionName));

//Create a shallow copy of the container:
std::pair<xAOD::JetContainer*,xAOD::ShallowAuxContainer*> my_jets_SC = xAOD::shallowCopyContainer(*my_jets);

for ( auto jet_calib : *my_jets_SC.first ) {
    m_jetCalibration->applyCalibration(*jet_calib);
    // Do some other stuff here with the jet, e.g
    if(jet_calib->pt() < 20000.) continue;
}
```

JetUncertainties - how to apply

- JetUncertainties package supports three ways of applying the systematic variations:

```
CP::CorrectionCode applyCorrection(xAOD::Jet& input);
// Each jet is varied separately
```

```
CP::CorrectionCode correctedCopy(const xAOD::Jet& input, xAOD::Jet*& output);
// A copy of the initial jet is created with the varied four-momentum
```

```
CP::CorrectionCode applyContainerCorrection(xAOD::JetContainer& inputs);
// The whole container is varied instead of individual jets
```

JetUncertainties - how to apply - II

JetUncertainties usage

```
// Get the list of systematics to loop over
const CP::SystematicRegistry& registry = CP::SystematicRegistry::getInstance();
const CP::SystematicSet& recommendedSystematics = registry.recommendedSystematics();
std::vector sysList = CP::make_systematics_vector(recommendedSystematics);
std::vector::const_iterator sysListItr;

// Loop over the systematic variations
for (sysListItr = sysList.begin(); sysListItr != sysList.end(); ++sysListItr) {

  xAOD::JetContainer* jets = ; // get the jets here

  // Tell the tool which systematic to use
  if (jetUncTool.applySystematicVariation(*sysListitr) != CP::SystematicCode::Ok) {
     ERROR("execute()","Cannot configure jet uncertainties tool for systematic");
     continue; // skip this systematic
  }

  // Loop over jets and apply systematic variations
  for (size_t iJet = 0; iJet &lt; jets.size(); ++iJet) {

      if (jetUncTool.aplyCorrection(*jets.at(iJet)) != CP::CorrectionCode::Ok) {
        ERROR("execute()","Cannot apply systematic variation to the jet");
        continue; // skip this jet
      }

      INFO("Successfully applied systematic variation to the jet");

  } // end jet loop
} // end loop over systematic variations
```