Table of Contents

7.10.1 PAT Examples: Tau Example....1
Contents....1
Introduction....1
How to get the code....1
How to run the code....2
Find out more about the details....2
analyzePatTau_fromAOD_cfg.py....2
PatTauAnalyzer.h....3
PatTauAnalyzer.cc....5
Exercises....6
How to get more information....7
Review status....7

7.10.1 PAT Examples: Tau Example

Contents

IMPORTANT This is out of date and replaced by WorkBookPFTauTagging

- Introduction
- How to get the code
- How to run the code
- Find out more about the details
- Exercises
- How to get more information
- Review status

Introduction

The aim of this tutorial is to show you how to access the values of kinematic quantities, tau id. variables and discriminator values from pat::Tau objects. The pat::Tau object represents a tau-jet, the reconstructed particles produced in a hadronic tau decay.

Having a lifetime of about t=290ps, corresponding to a mean distance of ct=87micrometer between their production and decay, tau leptons are different from electrons and muons in that only the decay products of the tau can ever be identified in the detector. The following discussion will be restricted to decays of tau leptons to hadrons, in the majority of cases either 1 or 3 charged pions plus 0, 1 or 2 neutral pions (the latter decay almost immediately to a pair of photons each).

As an example for how to do this, the tutorial will guide you through a simple EDAnalyzer. You can find the source code of the EDAnalyzer and its configuration file in the PatExamples package.

The EDAnalyzer will fill distributions of Pt, eta and phi of the tau-jets before and after cuts representing tau id. requirements are applied. From the ratio of the after/before distributions, a simple ROOT macro will compute tau id. efficiencies.

In summary, this tutorial aims to provide you with the following information:

- how to access information from the pat::Tau.
- how to determine an identification efficiency for tau leptons decaying into hadrons.

Before starting with this tutorial, it is helpful if you have a clear picture of:

- how to write an EDAnalyzer (in principle).
- how to read in parameters from the configuration file.
- how to access a pat::Candidate collection from the event content with an EDAnalyzer.

Note: For more fundamental examples have a look at the WorkBookPATAccessExercise. This example is based on CMSSW_3_3_x.

How to get the code

To run the example check out the latest tags for PAT as given on the PAT Recipes for CMSSW_3_3_x Guide and checkout the following tags in addition:

```
cd CMSSW_3_3_6_patch2/src
cmsenv
cvs co -A PhysicsTools/PatExamples/CMS.BuildFile
cvs co -A PhysicsTools/PatExamples/plugins/CMS.BuildFile
cvs co -A PhysicsTools/PatExamples/plugins/PatTauAnalyzer.h
cvs co -A PhysicsTools/PatExamples/plugins/PatTauAnalyzer.cc
cvs co -A PhysicsTools/PatExamples/test/analyzePatTau_fromAOD_cfg.py
cvs co -A PhysicsTools/PatExamples/test/patTau_idEfficiency.C
```

These files have the following functionality:

- PatTauAnalyzer.h PatTauAnalyzer.cc: the main module to book and fill histograms.
- analyzePatTau_fromAOD_cfg.py: main configuration file for cmsRun
- patTau_idEfficiency.C: root macro to display the efficiency to identify hadronic tau decays

How to run the code

Compile and run the example as given below:

```
scram b
cmsRun PhysicsTools/PhysicsExamples/test/analyzePatTau_fromAOD_cfg.py
```

You can inspect the histograms produced by the PatTauAnalyzer module by executing:

```
root PhysicsTools/PatExamples/test/patTau_Histograms.root
new TBrowser
```

and then click on the folder icon labelled "ROOT Files", followed by clicks on "CMS.PhysicsTools/PatExamples/test/patTau_Histograms.root", "analyzePatTau" and finally one of the histogram icons.

In order to produce plots of the tau id. efficiency as function of Pt of the tau-jet, execute:

```
cd PhysicsTools/PatExamples/test
root patTau_idEfficiency.C
gv tauIdEff_Pt.eps
```

Find out more about the details

Let's have a look at the main configuration file analyzePatTau_fromAOD_cfg.py first:

analyzePatTau_fromAOD_cfg.py

```
process = cms.Process('analyzePatTau')
```

This statement defines the name of the cmsRun instance to be *analyzePatTau* . The histograms will be saved into a subdirectory of that name.

```
process.load('CMS.PhysicsTools.PatAlgos.patSequences_cff')
```

This statement imports definitions needed to produce the PAT objects.

```
process.analyzePatTau = cms.EDAnalyzer("PatTauAnalyzer",
    src = cms.InputTag('cleanLayer1Taus'),
    requireGenTauMatch = cms.bool(True),
    discrByLeadTrack = cms.string("leadingTrackPtCut"),
    discrByIso = cms.string("byIsolation"),
    discrByTaNC = cms.string("byTaNCfrHalfPercent")
)
```

The section above instructs cmsRun to add a module of type PatTauAnalyzer to the process and defines a set of configuration parameters needed by the PatTauAnalyzer module:

- *src*: collection of pat::Taus to be analyzed by the module
- *requireGenTauMatch*: fill histograms with those reconstructed pat::Taus only which match a "true" tau lepton decay on the generator level
- *discrByLeadTrack*, *discrByIso* and *discrByTaNC*: names of "discriminator" used to identify "good" reconstructed pat::Taus, separating "true" hadronic tau decay from a large background of QCD-jets.

Note: The discriminators quantify the likelihood for a reconstructed pat::Tau object to be due to a true hadronic tau decay. They have values between 0 and 1, corresponding to the extreme (hypothetical) cases that a reconstructed pat::Tau is *known not* to to be due/to be due to a true hadronic tau decay. For this tutorial, we will require that a "good" reconstructed pat::Tau contains a high Pt ("leading") track (*discrByLeadTrack*) and to be be either isolated from other (charged) particles in the event (*discrByIso*) or to pass a tau id. discriminator (*discrByTaNC*) computed by a neural network. See this link for more details about the tau identification procedure.

This statement defines the name of the ROOT file in which the histograms get saved:

```
process.TFileService = cms.Service("TFileService",
    fileName = cms.string('patTau_Histograms.root')
)
```

And this statement, located at the end of the main configuration file analyzePatTau_fromAOD_cfg.py, instructs cmsRun what to do with each event, namely to first rerun the tau reconstruction on the AOD level, then produce the PAT objects and finally fill the histograms containing the the values of kinematic quantities, tau id. variables and discriminator values of the reconstructed pat::Tau objects:

```
process.p = cms.Path( process.patDefaultSequence + process.analyzePatTau )
```

Now, let's have a look at the source code of the analyzer module:

PatTauAnalyzer.h

This file contains the *declaration* of the PatTauAnalyzer class:

```
class PatTauAnalyzer : public edm::EDAnalyzer
{
 public:
  explicit PatTauAnalyzer(const edm::ParameterSet&);
  ~PatTauAnalyzer();

//--- methods inherited from EDAnalyzer base-class
  void beginJob(const edm::EventSetup&);
  void analyze(const edm::Event&, const edm::EventSetup&);
  void endJob();

 private:
//--- configuration parameters
```

```
  edm::InputTag src_;

  bool requireGenTauMatch_;

  std::string discrByLeadTrack_;
  std::string discrByIso_;
  std::string discrByTaNC_;

//--- generator level histograms
  TH1* hGenTauEnergy_;
  TH1* hGenTauPt_;
  TH1* hGenTauEta_;
  TH1* hGenTauPhi_;

//--- reconstruction level histograms
  TH1* hTauJetEnergy_;
  TH1* hTauJetPt_;
  TH1* hTauJetEta_;
  TH1* hTauJetPhi_;

  TH1* hNumTauJets_;

  TH1* hTauLeadTrackPt_;

  TH1* hTauNumSigConeTracks_;
  TH1* hTauNumIsoConeTracks_;

  TH1* hTauDiscrByIso_;
  TH1* hTauDiscrByTaNC_;
  TH1* hTauDiscrAgainstElectrons_;
  TH1* hTauDiscrAgainstMuons_;

  TH1* hTauJetEnergyIsoPassed_;
  TH1* hTauJetPtIsoPassed_;
  TH1* hTauJetEtaIsoPassed_;
  TH1* hTauJetPhiIsoPassed_;

  TH1* hTauJetEnergyTaNCpassed_;
  TH1* hTauJetPtTaNCpassed_;
  TH1* hTauJetEtaTaNCpassed_;
  TH1* hTauJetPhiTaNCpassed_;
};
```

The declaration is typical for a physics analysis module that books and fills histograms:

- the class inherits from EDAnalyzer, the base-class for modules which process events without filtering or reconstructing part of the the event content
- the class contains data-members of type edm::InputTag which allow to easily switch the collections of objects to be analyzed by the module by means of changing the values of corresponding parameters in the python configuration file
- and, of course, the class contains data-members for the histograms that are to be filled

The histograms filled in this tutorial contain the values of the following kinematic quantities, tau id. variables and discriminator values:

- *hTauJetEnergy_*, *hTauJetPt_*, *hTauJetEta_*, *hTauJetPhi_*: the energy, Pt, eta and phi of those reconstructed pat::Tau objects which match a "true" tau lepton on the generator level
- *hNumTauJets_*: the number of all tau-jet candidates in the event (without the generator level matching applied)
- *hTauLeadTrackPt_*: the transverse momentum of the highest Pt ("leading") track within the tau-jet
- *hTauNumSigConeTracks_*, *hTauNumIsoConeTracks_*: the number of tracks reconstucted within the *signal* and *isolation* cones of the tau-jet.

- *hTauDiscrByIso_*, *hTauDiscrByTaNC_*, *hTauDiscrAgainstElectrons_*, *hTauDiscrAgainstMuons_*: the values of tau id. discriminators against QCD-jets, based on track isolation (*discrByIso*) and neural network tau id. (*discrByTaNC*), and against misidentified electrons (*discrAgainstElectrons*) and muons (*discrAgainstMuons*)
- *hTauJetEnergyIsoPassed_*, *hTauJetPtIsoPassed_*, *hTauJetEtaIsoPassed_*, *hTauJetPhiIsoPassed_*: the energy, Pt, eta and phi for the subset of reconstructed pat::Taus which match a "true" tau lepton on the generator level and pass the combination of the *discrByLeadTrack* && *discrByIso* discriminators
- *hTauJetEnergyTaNCpassed_*, *hTauJetPtTaNCpassed_*, *hTauJetEtaTaNCpassed_*, *hTauJetPhiTaNCpassed_*: the energy, Pt, eta and phi for the subset of reconstructed pat::Taus which match a "true" tau lepton on the generator level and pass the combination of tau id. discriminators *discrByLeadTrack* && *discrByTaNC*

The latter two sets of histograms are used to compute a tau id. efficiency.

PatTauAnalyzer.cc

This file contains the actual *implementation* of the PatTauAnalyzer module:

Like the declaration, the implementation of the PatTauAnalyzer module is typical for that of an analysis module:

- in the constructor of the module, the parameter values are read in from the python configuration file
- in the beginJob() method of the module, which gets called by cmsRun exactly once, all the histograms are booked
- the histograms are filled in the analyze() method of the module, which gets called by cmsRun once per event

The only part of the implementation which is slightly non-standard is the matching of the reconstructed pat::Taus with "true" tau leptons leptons on the generator level:

```
 for ( pat::TauCollection::const_iterator patTau = patTaus->begin();
   patTau != patTaus->end(); ++patTau ) {

    const reco::GenParticle* genTau = getGenTau(*patTau);
    if ( requireGenTauMatch_ && !genTau ) continue;
```

In case the *requireGenTauMatch* configuration parameters is set to true (default), the *if* statement causes all reconstructed pat::Taus that don't match a generated tau lepton to be ignored.

The evaluation of whether or not a reconstructed pat::Tau matches a generated tau lepton is implemented in the *getGenTau* function:

```
const reco::GenParticle* getGenTau(const pat::Tau& patTau)
{
  std::vector<reco::GenParticleRef> associatedGenParticles = patTau.genParticleRefs();
  for ( std::vector<reco::GenParticleRef>::const_iterator it = associatedGenParticles.begin();
   it != associatedGenParticles.end(); ++it ) {
    if ( it->isAvailable() ) {
      const reco::GenParticleRef& genParticle = (*it);
      if ( genParticle->pdgId() == -15 || genParticle->pdgId() == +15 ) return genParticle.get();
    }
  }
  return 0;
}
```

For the pat::Tau passed as function argument the *getGenTau* function returns the pointer to the generator level tau lepton if the match succeeds and a *NULL* pointer otherwise. In the implementation of the *getGenTau*

function the genParticleRefs() method, inherited from the pat::PATObject base-class of the pat::Tau, is utilized. This method returns the list of generator level particles which have been matched to the pat::Tau during the PAT production sequence (per default, each generated tau lepton is matched to the reconstructed pat::Tau object closest in dR = sqrt(dEta^2 - dPhi^2)).

Please click here to see the complete source code of the module.

Exercises

The following exercises will help you getting more familiar with the way histograms are booked and filled and how the values of kinematic quantities, tau id. variables and discriminator values of pat::Tau are accessed.

Please open the file PatTauAnalyzer.cc in your preferred editor.

In the source code of the *PatTauAnalyzer::beginJob()* method you will find a string "TO-DO". Add code to book the histograms

- *hTauJetEta_*
- *hTauJetPt_*

at this position in the source code. (These histograms are already defined in PatTauAnalyzer.h)

Note: It is important that you assign the names "TauJetEta" and "TauJetPhi" to these histograms, because those are the names expected by the ROOT macro which computes the tau id. efficiency. The name of the histogram is defined by the first parameter passed when calling the *fs->make()* function.

In the source code of the *PatTauAnalyzer::analyze()* method, add code to fill the histograms

- *hTauJetEta_* and *hTauJetPt_*: with eta, phi of the tau-jet
- *hTauLeadTrackPt_*: with the transverse momentum of the highest Pt ("leading") track within the tau-jet
- *hTauDiscrAgainstElectrons_*: with the values of the discriminator against misidentified electrons

You can find the names of the methods for accessing the pseudo-rapidity and azimuthal angle of the pat::Tau in the header file of its reco::Particle base-class.

In the header file of the pat::Tau class you can find the methods for accessing the leading track and for the discriminator value.

Note: Not all tau-jet candidates actually have an associated leading track. You need to check for the existence of the leading track by checking that *pat::Tau::leadTrack().isAvailable() == true* before filling the histogram with its Pt value.

Note: In order to find the value of the string that needs to be passed to the *pat::Tau::tauId()* method in order to access the *discrAgainstElectrons* value, please follow this link.

Once you have finished implementing booking and filling of these histograms, save your changes, recompile and rerun the PatTauAnalyzer module:

```
scramv1 b
cmsRun PhysicsTools/PatExamples/test/patTau_Histograms.root
```

Have a look at the *hTauJetEta_*, *hTauJetPt_*, *hTauLeadTrackPt_* and *hTauDiscrAgainstElectrons_* in the TBrowser.

Open the ROOT macro patTau_idEfficiency.C in your editor and remove all occurences of the strings "/*" and "*/" in the macro. When you execute

```
cd PhysicsTools/PatExamples/test
root patTau_idEfficiency.C
gv tauIdEff_Eta.eps
gv tauIdEff_Phi.eps
```

now, you will see the plots of the tau id. efficiency as a function of eta and phi of the tau-jet.

At this point, the tutorial on pat::Taus is finished.

For further information on taus in CMS, please have a look at the links below.

How to get more information

You can get handouts of the PAT tau tutorial from here.

The following links may help you to find-out more about taus in CMS:

- Tau id. wiki
- Particle flow & tau id. hypernews
- CMS analysis note on particle flow based tau id.

and about how taus are used in different physics analyses:

- EWK tau wiki
- Higgs tau wiki

Review status

Reviewer/Editor and Date (copy from screen)	Comments
RogerWolf - 13 May 2009	Created the template page
ChristianVeelken - 12 June 2009	Added content (based on CMSSW_2_2_x)
ChristianVeelken - 18 December 2009	Updated tutorial to CMSSW_3_3_x

Responsible: ChristianVeelken
Last reviewed by: ChristianVeelken - 18 Dec 2009

This topic: CMSPublic > WorkBookPATExampleTau
Topic revision: r14 - 2017-10-04 - MargueriteTonjes