Table of Contents

PAT Exercise 08: Tools for MC truth matching in PAT

Contents

Objectives

- Learn about the support of any kind of matching in the Analysis Tools (AT) Group.
- Learn how PAT exploits these matching tools for Monte Carlo truth matching.
- Learn how to configure Monte Carlo truth matching in PAT and how to access it.

Note:

This web course is part of the PAT Tutorial, which takes regularly place at cern and in other places. When following the PAT Tutorial the answers of questions marked in RED should be filled into the exercise form that has been introduced at the beginning of the tutorial. Also the solutions to the Exercises should be filled into the form. The exercises are marked in three colours, indicating whether this exercise is basic (obligatory), continuative (recommended) or optional (free). The colour coding is summarized in the table below:

Color Code	Explanation
	Basic exercise, which is obligatory for the PAT Tutorial.
	Continuative exercise, which is recommended for the PAT Tutorial to deepen what has been learned.
	Optional exercise, which shows interesting applications of what has been learned.

Basic exercises ( ) are obliged and the solutions to the exercises should be filled into the exercise form during the PAT Tutorial.

Introduction

If you work on FNAL you can find instructions how to setup the environment here: WorkBookRemoteSiteSpecifics.

The matching between two different physics objects is a common task in analyses. To do this the Analysis Tools (AT) group provides an evolved architecture that can be used for many purposes. This exercise should make you familiar with the potential of these tools, how PAT exploits of them and how to use it in your analysis. You will learn the following:

- How to access the matching information starting from a *pat::Candidate*.
- What different kinds of matching are supported by PAT (AT).

- How to choose and configure the PAT-Matching to your needs.
- How to store multiple matches and access.
- What dependencies do exist when skimming of the event content.

At the end you can find some exercises to consolidate your knowledge.

Setting up of the environment

We assume that you are logged in on `lxplus` and are in your work directory. If not you can follow the instruction given here.

```
mkdir exercise08
cd exercise08
cmsrel CMSSW_7_4_1_patch4
cd CMSSW_7_4_1_patch4/src
cmsenv
git cms-addpkg PhysicsTools/PatAlgos
git cms-addpkg FWCore/GuiBrowsers
git cms-merge-topic -u CMS-PAT-Tutorial:CMSSW_7_4_1_patTutorial
scram b -j 4
```

How to run the code

Run the example by typing

```
cd PhysicsTools/PATExamples/test
cmsRun mypatmatching_Task0_cfg.py | tee runLogGen.log
cmsRun mypatmatching_Task1_cfg.py
cmsRun mypatmatching_Task2_cfg.py
```

For this example we produce PAT on the fly from a RECO input file, which means that we will not make the created pat::Candidate collections persistent. We analyse the data with a dedicated EDAnalyzer in the same process and investigate the PAT Monte Carlo matching with some basic histograms saved in the output file `histo.root`.

Sources

You can use

```
## Source
process.source.fileNames = ['root://eoscms//eos/cms/store/relval/CMSSW_7_4_1/RelValTTbar_13/GEN-S
4367CFB-9BEC-E411-90A3-0025905A60C6.root']
```

These event sample was generated as RelVal sample using CMSSW_7_4_1.

Find out more about the details

(Task-0) genParticles candidates

Do you know how to access genParticles candidates collection? We can start from *PhysicsTools/PATExamples/src/MyPatMatchingTask0.cc*.

```
edm::Handle<GenParticleCollection> genParticles;
```

```
  iEvent.getByLabel("genParticles", genParticles);
  //
  if(genDebug_==true){
    for(size_t i = 0; i < genParticles->size(); ++ i) {
      if(i>200) break;
      const GenParticle & ParticleCand = (*genParticles)[i];
      cout<<i<<") Particle = "<<ParticleCand.pdgId()<<", Status = "<<ParticleCand.status()<<endl
      // Daughter
      const GenParticleRefVector& daughterRefs = ParticleCand.daughterRefVector();
      for(reco::GenParticleRefVector::const_iterator idr = daughterRefs.begin(); idr!= daughterR
      " cout<<"Daughter "<<(*idr).key()<<" "<<(*idr)->pdgId()<<endl;
      }
      // Mother
      const GenParticleRefVector& motherRefs = ParticleCand.motherRefVector();
      for(reco::GenParticleRefVector::const_iterator imr = motherRefs.begin(); imr!= motherRefs.
      " cout<<"Mother "<<(*imr).key()<<" "<<(*imr)->pdgId()<<endl;
      }
    }
  }
```

When you do `cmsRun mypatmatching_Task0_cfg.py | & tee runLogGen.log`, you can see on the screen and also in the `runLogGen.log`, particles are produced step by step, starting from proton (pdgId = 2212), and status values correspond to the following meanings:

status	meaning
1	stable particle (handed over to Geant4 for detector simulation)
2	particle after parton showering and ISR/FSR
3	particle before parton showering and ISR/FSR (i.e. as it comes from matrix element calculation)

Let's see how the generator produce ttbar event. Note that, the status `1` does not mean `stable particle` in nature, but it means that particle has long lifetime enough to enter to the detector (interact with material). In this case, the detector simulation software (i.e. Geant4) will take care for them.

Question 8 a) What EventContent(s) does contain `genParticles collection`?

Some Hints... Hide
You can see comments in EventContent_cff.py

Question 8 b) How do you know that you ROOT data file contains genParticle collection?

Some Hints... Hide
edmDumpEventContent yourData.root | grep genParticles

(Task-1) Create your own MC Matching

To do this, you need 3 steps:
(1) Define your matching.
(2) Add matching information to your related objects and make it accessible.
(3) Enjoy your analysis.

(1.1) Define your matching.

In the configuration `mypatmatching_Task1_cfg.py`, you will see the following syntaxes:

```
process.muMatch1 = process.muonMatch.clone(mcStatus = [1])
process.muMatch3 = process.muonMatch.clone(mcStatus = [3])
```

In this syntax, we clone the EDProducer `muonMatch` of the *patDefaultSequence* and change the parameter *mcStatus* to different values. Do you remember status `1` and `3` represent?

Note: There is an alternative way to do the cloning and parameter replacement in two steps:

Example Hide
```
process.muMatch3 = process.muonMatch.clone()
process.muMatch3.mcStatus = [3]
```

Question 8 c) How would you proceed to check the parameters of the original *muonMatch* module?

Hints Hide
There is quite a few possibilities. We list three of them here:

- Make use of interactive *python* and check for the configuration of *muonMatch*:

```
python -i mypatmatching_Task1_cfg.py
```

```
>>> process.muonMatch
     cms.EDProducer("MCMatcher",
         src = cms.InputTag("muons"),
         maxDPtRel = cms.double(0.5),
         mcPdgId = cms.vint32(13),
         mcStatus = cms.vint32(1),
         resolveByMatchQuality = cms.bool(False),
         maxDeltaR = cms.double(0.5),
         checkCharge = cms.bool(True),
         resolveAmbiguities = cms.bool(True),
         matched = cms.InputTag("genParticles")
     )
```

Try `process.muMatch1`, and `process.muMatch3`.

- Use the ConfigBrowser.

```
edmConfigBrowser mypatmatching_Task1_cfg.py
```

- Have a look to the github. You can find the proper file in the [[mcMatchLayer0 directory of the *PatAlgos* package.

Remember: To define your own matching, you can clone from existings (as we show above) or you modify defaults.

Question 8 d) Where does the default MC matching configurations stay?

Some Hints... Hide
```
PhysicsTools/PatAlgos/python/mcMatchLayer0
```

(1.2) Add matching information to your related objects and make it accessible.

Now, we add our newly created modules to the *patDefaultSequence*. Using the member function `replace` of the class Sequence the new matching modules substitute the the old ones.

```
process.patDefaultSequence.replace(process.muonMatch,
                                   process.muMatch1 +
                                   process.muMatch3
)
```

Note: You can use the *python* interpreter to find out which member function the class object of cms has. Just type `python -i` in your shell and load the ParameterSet of the Framework as you usually do in the first

line of your cfg-files. Then use dir(class_you_are_interested_in) and help(function_of_this_class) to find out how to use a specific function. For the above example a summary of these steps is given below:

Example Hide
```
cmsenv
python -i
import FWCore.ParameterSet.Config as cms
dir(cms)
dir(cms.Sequence)
help(cms.Sequence.replace)
q
```

Use `Ctrl-d` to exit python interpreter.

As the matching information should be accessible via the *pat::Muon* the module *patMuons* holds a parameter `genParticleMatch`. Note that this parameter is of type *std::vector<edm::InputTag>*, so you can store more than one match, we will replace the default configuration by the two matches we created beforehand:

```
process.patMuons.genParticleMatch = cms.VInputTag(
    cms.InputTag("muMatch3"),
    cms.InputTag("muMatch1")
)
```

For this we had to make sure of course, that both modules were produced before the *patMuon* will appear in the sequence. Now that we have adapted the *patDefaultSequence* let's add it to the *process.Path*:

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

Before we concentrate on the EDAnalyzer to convince ourselves that all changes work fine make sure you can answer the following questions:

Question 8 e) Where in the Pat Workflow is the matching performed?

Some Hints... Hide
Check out the WorkBookPATWorkflow and use WorkBookConfigEditor (or just github or lxr) for browsing the cff/cfi-chain of the *patDefaultSequence*

Question 8 f) How can you configure the matching? What other parameters than *mcStatus* do exist and what are they good for?

Some Hints... Hide
Have a look at SWGuidePATMCMatching. Inspect the configuration file with `python -i mypatmatching_Task1_cfg.py` or simply look it up in github or lxr. The meaning of the parameters will become clearer in the next section.

Question 8 g) How is the matching information embedded into the pat::Candidates? Which parameters of the Object Producer modules concern the matching and what are they good for?

Some Hints... Hide
You can for instance find the module definitions in the PhysicsTools/PatAlgos/python/producersLayer1 sub-directory of the *PatAlgos* package on github.

(1.3) Enjoy your analysis.

Open the EDAnalyzer plugin definition *PhysicsTools/PATExamples/src/MyPatMatchingTask1.cc*. You can find more information about the rest of the module at WorkBookPATAccessExercise. We will directly move on to the member function *analyze*. First we read in a common *edm::Handle* to access the *pat::Muons*.

```
edm::Handle<edm::View<pat::Muon> > muons;
iEvent.getByLabel(muonSrc_,muons);

for(edm::View<pat::Muon>::const_iterator muon=muons->begin(); muon!=muons->end(); ++muon){
```

Since we have stored more than one Monte Carlo match we loop over them.

```
for(uint i = 0 ; i < muon->genParticleRefs().size() ; ++i ){
          switch( muon->genParticle(i)->status() ){
```

To illustrate the effect of the two different matches, histograms are created which show the resolution in deltaR. What do you expect for the resolution in deltaR and the number of possible matches for the status 1 muons with respect to the status 3 muons? To see if you are right have a look at them with the *root* Browser:

```
root -l histo.root
new TBrowser
```

Note: ***Don't get nervous*** if one of the histograms that you produced is indeed empty. Go through the configuration of the matching modules once again.

Question 8 h) Do you understand why one set of histograms is empty?

Some Hints... Hide
Compare the definition of the matching plot in the `MyPatMatchingTask1.cc` module implementation.

Before moving to the exercises make sure you are able to answer the following questions:

Question 8 i) How can you access matching information starting from a PAT candidate?

Some Hints... Hide
Look it up in the Doxygen pages of the according Candidate (e.g. for muons here)

Question 8 j) How can you distinguish the different matches you embedded?

Some Hints... Hide
For example with their status or their pdgId.

(Task-2) "On the fly" particles.

As we discussed in Task-0, in the generator level, we doesn't pass only stable particles to the detector simulation, we also pass some exotic particles which have long life time enough to interact with detector before their decay, to the detector simulation. An example of this setting is:

```
'MSTJ(22)=2   ! Decay those unstable particles',
'PARJ(71)=10 .! for which ctau 10 mm',
```

With this syntax, it tells PYTHIA6 to decay unstable particles but not for particles which have lifetime (in c*tau unit) longer than 10 mm.

Exercises

(Task-0):

Exercise 8 a): From the example, you can see how to get information directly from genParticle candidates, so it means you can study at the parton level. There is no exercise here, but it will be important to you when

you want to show results at the parton level, i.e. differential cross section of the highest jet pt or charge lepton asymmetry in some SUSY models, produced by generator compare with the data or detector simulation results.

(Task-1):

Exercise 8 b): Accessing the matching information: Instead of looping over the different matches pick the match directly with an appropriate member function which is able to distinguish the status and pdgIds of the matches.
How is this member function called?

The according member function is genParticleById(). We show an example of the modified loop over the muons below:

```
for(edm::View<pat::Muon>::const_iterator muon=muonCollection->begin(); muon!=muonCollection->end(
     if(muon->genParticleById(0,1).isNonnull() ){
       histContainer_["DR_status1Match"]->Fill( ROOT::Math::VectorUtil::DeltaR(muon->p4() , (muon
       histContainer_["DPt_status1Match"]->Fill(muon->pt() - (muon->genParticleById(0,1))->pt() )
     }
}
```

Note: The following might be of help for you:

- The Doxygen page of the *pat::Muon* you can find here.
- Make sure the Ref to the match is nonNull. See here how to check this.

Exercise 8 c): Skimming the event content: Imagine that you dropped the *genParticle* collection from the *EventContent* in a skimming step where you also run PAT:
What would you have to change to have still access to matching information afterwards?

You would need to embed the matching information in the production step of the pat::Candidates. The following parameters will do the trick:

```
addGenMatch      = cms.bool(True),
embedGenMatch    = cms.bool(True),
genParticleMatch = cms.InputTag("muonMatch")
```

Exercise 8 d): Apply different kinds of matching: Match by deltaPt instead of deltaR and compare by plotting deltaPt for both matchings.

The matching module is named *MCMatcherByPt* instead of *MCMatcher. The functionality is identical, since both matchers are instances of the same templated code. The DPt histogram should become slightly narrower when matching by DPt instead of DR but you probably won't see an effect in our example. We show the modified definition of the _muonMatch* module:

```
process.muonMatchByPt = cms.EDProducer("MCMatcherByPt", # cut on deltaR, deltaPt/Pt; pick best by
    src     = cms.InputTag("muons"), # RECO objects to match
    matched = cms.InputTag("genParticles"),   # mc-truth particle collection
    mcPdgId     = cms.vint32(13), # one or more PDG ID (13 = muon); absolute values (see below)
    checkCharge = cms.bool(True), # True = require RECO and MC objects to have the same charge
    mcStatus = cms.vint32(1),     # PYTHIA status code (1 = stable, 2 = shower, 3 = hard scatteri
    maxDeltaR = cms.double(0.5),  # Maximum deltaR for the match
    maxDPtRel = cms.double(0.5),  # Maximum deltaPt/Pt for the match
    resolveAmbiguities = cms.bool(True),     # Forbid two RECO objects to match to the same GEN o
    resolveByMatchQuality = cms.bool(False), # False = just match input in order; True = pick low
)
```

Note: The following might be of help for you:

- Here you can find the corresponding module.

Exercise 8 e): Configure the PAT Monte Carlo matching: Try to change some parameters of the matching module and visualise the effect:
Change the parameters *maxDeltaR* and *maxDPtRel* to some sensible values and check the impact on the histograms. What are the default values?

How does the number of matches change if you set the parameter *checkCharge* in the definition of the *muonMatch* module to `False`? How often does a muon of different charge give a match?

You probably won't find any match to a different charge because of the small statistics of the example. But you should have switched the parameter *checkCharge = False* and made use of the muon member function *genParticleById*.

Resolve ambiguities by match quality: Does the resolution in DR change?

It should become better since the amibiguities are solved by DR.

(Task-2):

Exercise 8 f): Store multiple matches: Create a new collection of *inFlightMuons*. Note that the module to produce muons from decay in flight from the Monte Carlo truth is already declared and configured in the *mypatmatching_Task2_cfg.py* file.

```
process.load("SimGeneral.HepPDTESSource.pythiapdt_cfi")
process.inFlightMuons = cms.EDProducer("PATGenCandsFromSimTracksProducer",
        src           = cms.InputTag("famosSimHits"),   ## use "famosSimHits" for FAMOS, "g4SimHi
        setStatus     = cms.int32(-1),
        particleTypes = cms.vstring("mu+"),           ## picks also mu-, of course
        filter        = cms.vstring("pt > 0.5"),      ## just for testing
        makeMotherLink = cms.bool(True),
        writeAncestors = cms.bool(True),              ## save also the intermediate GEANT ancestor
        genParticles   = cms.InputTag("genParticles"),
)
```

You have to put it into a proper place in the process path. Next create another clone of the *muonMatch* module and configure it accordingly. Note that you have to change the parameters *mcStatus* and *matched* accordingly to do the matching. What are sensible values for *maxDeltaR* and *maxDPtRel* in this case? How many muons do not have a status 1 or status 3 but a *inFlightMatch*?

See here for further information.

Note: The following might be of help for you:

- Here you can find the appropriate section in the Pat MC Matching SWGuide.

Solutions

A set of solutions will be added here within the next days.

Note:

In case of problems don't hesitate to contact the SWGuidePAT#Support. Having successfully finished **Exercise 8** you might want to proceed to **Exercise 9** of the SWGuidePAT to learn more about the complex world of trigger at CMS. For an overview you can go back to the WorkBookPATTutorial entry page.

Review status

Show Hide

Reviewer/Editor and Date (copy from screen)	Comments
RogerWolf - 17 March 2012	Added color coding. But this needs some more reordering and an adaptation to the general scheme
FrancescaRicciTam - 3 July 2013	Removed version numbers from addpkg commands.

Responsible: YvonneKuessel

This topic: CMSPublic > SWGuidePATMCMatchingExercise
Topic revision: r95 - 2015-07-02 - PhatSrimanobhas