3 September 2021

Preparing example code for CMS open data

Andro Petković
Faculty of Science, Split, Croatia

Supervisor: Kati Lasilla-Perini

Summary

This document summarizes the work done during my CERN Summer Student internship. I was involved in CMS Open Data Workshop 2021 by contributing to Physics Objects Extractor Tool (POET) - an example code for extracting information from Analysis Data Object (AOD) files. I also prepared code for matching of generated Monte Carlo particles with physical objects based on their angular separation. With the end of the workshop and following the transfer of higher level analysis MiniAOD files from Data Agrregation System to the eospublic area, I started testing docker images needed for the corresponding analysis and extracting relevant information from MiniAOD files. At the end of my stay, I created MiniAOD version of POET together with the instructions for users. Example code prepared in this project will be implemented in CMS Open Data Guide.

Contents

1 Introduction

Since 2014, the CMS (Compact Muon Solenoid) collaboration has released a significant amount of LHC (Large Hadron Collider) research data for public use. In 2020, the collaboration organised CMS Open Data Workshop where theorists, phenomenologists, and other interested researchers could learn how to analyze publicly available datasets. I was involved in the second iteration of this workshop (CMS Open Data Workshop 2021, repository available here), mostly by contributing to the Physics Objects Extractor Tool (POET). Example code prepared in this project will also be implemented in CMS Open Data Guide.

In CMS, the raw data is processed in a series of reconstruction and selection steps to reduce its size and increase usability. [1]. In the first step, tracking, vertexing, and particle flow are performed, and basic physics object collections are created [1]. In the next step, only some hits and other detector-level info are kept, prioritizing physics object collections, and the reconstructed data is reduced to Analysis Object Data (AOD) [1].

This report is organised in three sections. Physics Objects Extractor Tool section describes the modular structure of the POET. Section 3 describes the matching of reconstructed particles with Monte Carlo objects. Section 4 introduces Mini Analysis Object Data (MiniAOD) together with a MiniAOD version of POET. The above-mentioned sections contain short snippets of the code, as well as links to the full code I have prepared during the internship.

2 Physics Objects Extractor Tool

POET provides an example code for analyzing AOD files from the year 2011/2012, which are the current public data for Run 1 (the first operational run of LHC). At CMS, physics objects refer to particles that can be identified by the signals from the detector. In AOD files, they are stored in modules - data structures that act as containers for multiple instances of the same C++ class. POET (repository available here) has the individual analyzers for the following objects:

- Electrons
- Muons
- Taus
- Photons
- Jets
- Missing transverse momentum
- Tracks
- Vertices
- Triggers

- Generator-level (Monte Carlo) particles

I have prepared an example C++ code for tracks, vertices, and generator particles (TrackAnalyzer, VetrexAnalyzer, and GenParticleAnalyzer). Data is analyzed with Scientific Linux CERN 6 (slc6) operating system using CMSSW - a collection of software libraries that the CMS experiment uses to acquire, produce, process, and analyze its data. Docker image with CMSSW_5_3_32 installed is used, since that is the official CMSSW version for 2011/2012 data.

2.1 POET configuration

In CMS, all information about the collision of particles is stored in an event. POET is an Event Data Analyzer. It is written in C++, but its configuration is manipulated using Python language. A short snippet of POET configuration file is shown in Listing 1 (the full configuration is available at this link).

```
...
if isData:
  sourceFile='root://eospublic.cern.ch//eos/opendata/cms/Run2012B/
   DoubleMuParked/AOD/22Jan2013-v1/10000/1EC938EF-ABEC-E211-94E0-90
   E6BA442F24.root'
else:
  sourceFile='root://eospublic.cern.ch//eos/opendata/cms/MonteCarlo2012/
   Summer12_DR53X/TTbar_8TeV-Madspin_aMCatNLO-herwig/AODSIM/
   PU_S10_START53_V19-v2/00000/000A9D3F-CE4C-E311-84F8-001E673969D2.root'
process.source = cms.Source("PoolSource",
    fileNames = cms.untracked.vstring(sourceFile)
...
process.mypvertex = cms.EDAnalyzer('VertexAnalyzer')

process.mytracks= cms.EDAnalyzer('TrackAnalyzer')

process.mygenparticle= cms.EDAnalyzer('GenParticleAnalyzer',
              #---- Collect particles with specific "pdgid:status"
                                    #---- Check PDG ID in the PDG.
              #---- if 0:0, collect them all
              input_particle = cms.vstring("1:11","1:13","1:22","2:15")
              )
...
process.TFileService = cms.Service(
  "TFileService", fileName=cms.string("myoutput.root"))
process.p = cms.Path(...process.mypvertex+process.mytracks+process.
   mygenparticle+...)
```

Listing 1: POET configuration file. This code first loads the file for analysis. Individual processes are then linked to Event Data Analyzers. At the end, processes of interest are selected and the outpoot file is defined.

At the beginning, the source file of interest (collision data or simulated Monte Carlo data) can be selected. This file is then analyzed using Event Data Analyzers (EDAnalyzer). For generator particles, the user can collect particles with a specific Data Group ID numbers and generator status codes [2, 3]. Upon the execution of the configuration, an output ROOT

file (myoutpoot.root) is created. ROOT is a software framework for data analysis and I/O developed at CERN [6]. POET output file stores the variables extracted using individual EDAnalyzers. Since ROOT provides a graphical user interface, their distributions can now be graphically viewed remotely, e.g., using VNC application installed in the CMSSW_5_3_32 container.

2.2 POET Event Data Analyzers

As an example of individual Event Data Analyzers, a snippet of a GenParticleAnalyzer is shown in the Listing 2:

```
...
class GenParticleAnalyzer : public edm::EDAnalyzer {
...
      TTree *mtree;
      std::vector<float> GenPart_pt;
      std::vector<float> GenPart_eta;
      ...
}
GenParticleAnalyzer::GenParticleAnalyzer(const edm::ParameterSet& iConfig)
   :particle(iConfig.getParameter<std::vector<std::string> >("
   input_particle"))
{
    mtree->Branch("numGenPart",&numGenPart);
    mtree->GetBranch("numGenPart")->SetTitle("number of generator
         particles");
    mtree->Branch("GenPart_pt",&GenPart_pt);
    mtree->GetBranch("GenPart_pt")->SetTitle("generator particle
         transverse momentum");
}
...
void GenParticleAnalyzer::analyze(const edm::Event& iEvent, const edm::
   EventSetup& iSetup)
{
    Handle<reco::GenParticleCollection> gens;
    iEvent.getByLabel("genParticles", gens);
    ...
    itGenPart=reco::GenParticleCollection::const_iterator
    for (itGenPart=gens->begin(); itGenPart!=gens->end(); ++itGenPart)
        {
          for(i=0;i<particle.size();i++)
             {
               if((status_parsed[i]==itGenPart->status() &&
               pdgId_parsed[i]==itGenPart->pdgId())||
               (status_parsed[i]==0 && pdgId_parsed[i]==0))
             {
                  GenPart_pt.push_back(itGenPart->pt());
                  GenPart_eta.push_back(itGenPart->eta());
                  ...
             }...
```

Listing 2: Generator Particle Analyzer. After declaration of variables of interest, code loops through Generator Particle collection and stores them in branches of mtree object.

ROOT provides the TTree class to store large quantities of the same class objects, using branches to hold a list of variables. In this example, from Listing 2, interesting variables (transverse momentum, pseudorapidity,...) are stored in a mtree object. After the addition of branches to a tree, GenParticleAnalyzer parses the desired particle/status pairs and checks each generated particle from Generator Particle Collection against these conditions. Finally, it stores variables from interesting particles in branches of the mtree object.

3 Matching Analysis

For the needs of the CMS Open Data Workshop 2021, I also prepared a code that matches the reconstructed particles with MC particles. It opens a POET output file (myoutpoot.root) and matches the generated particles with electrons, photons, muons, and taus. A snippet of the code is shown in the Listing 3 (full code is available here).

```
void BestMatch(float object_eta, float object_phi, int object_pdgId,
   Long64_t event){
...
tevent->AddFriend(tgenparticles);

  tevent->GetEntry(event);
  for (Int_t j=0; j<numGenPart;++j)
    {
       r=deltaR(GenPart_eta->at(j),GenPart_phi->at(j),object_eta,
                object_phi);
       if (r < minDeltaR && abs(GenPart_pdgId->at(j))==object_pdgId)
       {
        minDeltaR = r;
        j_matched=j;
       }
    }
...
int main (){
...
for (Long64_t i=0;i<nentries;i++)
  {
    tevent->GetEntry(i);
    for (Int_t j=0; j<numelectron;++j)
    {
      BestMatch(electron_eta->at(j),electron_phi->at(j),11,i);
    }
```

Listing 3: Matching Analysis. This code loops over electrons and matches them with MC particles based on their angular separation.

The code in Listing 3 provides an example of looping over reconstructed objects (this snippet shows electrons) and finding the generated particle of the same type that minimizes their angular separation $\Delta R = \sqrt{(\Delta\eta)^2 + (\Delta\phi)^2}$, where $\Delta\eta$ and $\Delta\phi$ represent the difference between the pseudorapidity and azimuthal angle of the reconstructed objects and MC particles.

4 MiniAOD POET

The MiniAOD is a high-level data tier introduced in Spring 2014 [4]. It is derived from the AOD-level objects which are passed through additional selections and standard algorithms (such as jet energy corrections and identification criteria algorithms) [1]. Main contents of this format are [4]:

- high level physics objects (leptons, photons, jets, missing transverse momentum)
- the full list of particles reconstructed by the ParticleFlow
- MC Truth information: a subset of the genParticles
- trigger information

High level physics objects in MiniAOD are saved in the PAT (Physics Analysis Toolkit) dataformats [4]. PAT is a high-level analysis layer providing access to the algorithms developed by Physics Objects Groups (POGs) in the framework of the CMSSW offline software [5]. I prepared MiniAOD version of POET which is available in MiniAOD branch of my fork. It is used for the analysis of the 2015 MiniAOD files and tested in preparation for their release. Example code is analogous to the one in AOD POET, with the biggest difference being that the information is now extracted from the slimmed PAT objects. At this level, standard analysis algorithms are already applied and there is no need to implement them (e.g., electron id criteria) manually, which is illustrated in the snippet code in Listing 4.

```
electron_isLoose.push_back(el.electronID("cutBasedElectronID-PHYS14-
                           PU20bx25-V2-standalone-loose"));
electron_isMedium.push_back(el.electronID("cutBasedElectronID-PHYS14
                           -PU20bx25-V2-standalone-medium"));
electron_isTight.push_back(el.electronID("cutBasedElectronID-PHYS14-
                           PU20bx25-V2-standalone-tight"));
```

Listing 4: MiniAOD Electron Analyzer. Electron variables are filled based on loose, medium, and tight identification criteria which are implemented in PAT dataformat.

I tested MiniAOD POET using slc6_CMSSW_7_6_7 docker image and documented usage instructions here.

5 Conclusion

With the AOD files available at the eospublic area, POET provides an example code for their analysis. In preparation of the 2015 data release, MiniAOD version of POET is created. Possible improvements to this version include the addition of corrected transverse jet momentum with jet energy corrections and resolution, shifted up and down, as well as the addition of B-Tagging event weights, shifted up and down, to the MiniAOD Jet Analyzer.

6 Acknowledgements

I would like to thank my supervisor Kati Lasilla-Perrini for her guidance through this project. Her mentorship has been incredibly valuable, and I have learned a lot working with her. I would also like to thank Clemens Lange for preparing the docker images needed for this project.

References

[1] CMS Open Data Workshop 2021, CMS Data Flow, https://cms-opendata-workshop.github.io/workshop-lesson-presel-skim/01-introduction/index.html

[2] C-J Lin and Lbnl and S. Navas, Monte Carlo Particle Numbering Scheme, 2018

[3] Generator Event Format in AOD, https://twiki.cern.ch/twiki/bin/view/CMSPublic/WorkBookGenParticleCandidate

[4] MiniAOD Analysis Documentation, https://twiki.cern.ch/twiki/bin/view/CMSPublic/WorkBookMiniAOD2015#Example_code_accessing_all_high

[5] WorkBookPATGlossary, https://twiki.cern.ch/twiki/bin/view/CMSPublic/WorkBookPATGlossary#GlosP

[6] ROOT, https://root.cern/