Table of Contents

UmassNtupleAnalysis

Current Status

This analysis package is not currently up to date. See https://twiki.cern.ch/twiki/bin/view/AtlasProtected/WZJratioAnalysis for an up to date analysis framework written for WZ common ntuples and used for Standard Model WZ + jets measurements. Please email me if you have any questions/problems (ameade@physicsNOSPAMPLEASE.umass.edu).

UMAnalysis

UMAnalysis is an Ntuple analysis framework framework. It has two basic features:

- **Generalized Ntuple Reader:** Takes a list of branches as input, loading those branches only. It performs the following checks: that the branch exists, that the type is expected, and the the variable is loaded into the analysis correctly. Errors are stored and printed. Should work with any ntuple, and does not crash if the ntuple used changes. (of course make sure to read what branches are not loaded)
- **Internal Event Data Model (EDM):** Its own simple particle classes, particle selection, hlstograming, etc. The goal is to have tools/structures to build your analysis that can be independent of changes in the data/ntuple format used.

The next two sections will be a howto, describing how to start using the package, and then a more thorough description of the analysis package and its features.

How To Download and Use the Framework

The package can be downloaded from anywhere you have access to SVN, probably hal or lxplus. This tutorial assumes you are at lxplus, and uses a sample file in my public area, which is of the common D3PDmaker format made by the WZ Standard Model groups. See below for other ntuple types, available ntuples, and ntuple makers. The SVN Group Area ($SVNGRP) is set when you setup athena, so you must do this first. If you are not at lxplus, you will also need to kinit. For more information on setting up cmt and you work area, see the WorkBook.

```
source ~/cmthome/setup.sh -tag=15.5.1
svn co $SVNGRP/Institutes/UMass/UMAnalysis/trunk UMAnalysis
cd UMAnalysis/run
root -n -b -q wzLoadAnalysis.C  //other files load other ntuples, for example oxLoadAnalysis.C fo
```

When you check out the package, the last argument is the name of the directory you create. The wzLoadAnalysis.C macro compiles and executes all classes necessary for the analysis, and by default runs 10,000 events on the default file, writing output histograms to testoutput.root. To checkout a specific version of the framework, and run on your own file, do:

```
svn co $SVNGRP/Institutes/UMass/UMAnalysis/tag/UMAnalysis-XX-YY-ZZ UMAnalysis
cd UMAnalysis/run
root -n -b -q 'oxLoadAnalysis.C("/path/to/my/file/my.D3PD.root","outputName.root", eventsToProces
```

The option -n tell root not to load the log on script, this isn't necessary in most cases, but earlier versions of the framework clashed with some scripts.

Structure of the Analysis Package

The following are the files you will want to look at and alter, in order of importance to the end user:

- **MyAnalysis.cxx**: This is where the final analysis is done on internal particle classes (muons,jets,etc), which are accessed through m_data, for example m_data->preMuons gives you a pointer to the vector of muons before selection. This has evolved so that most of the analysis is now done in Analysis Modules (see below), which are loaded by this event loop.

- **wzConfig.cxx**: This configures both the branch variables, as well as analysis level configuration such as cuts. So you can change for example the minimum p_T of muons you will select, or what jet algorithms to consider. Note that because branch name information is set here, so there is a different config file for each ntuple type. Each of them set a number of static variables in myConfig.h, that are then picked up by the other classes. Example:

  ```
  myConfig::MuonBranchName
  ```

  This gives the prefix for muon variables, perhaps "muidMuon_",

- **DataManager.cxx**: This class stores the particle vectors in the struct particleData. It fills the internal particle classes from the branch information, and selects a subset of these particles that you actually want to use in your analysis (for example m_data->selMuons), based on a number of cuts on momentum, angular distribution and quality criteria (for example isolation).

- **particle.cxx**: These classes define particles (muon, egamma, jet, etc). Each contains a momentum TLorenzVector and some amount of specific info such as vertex info and quality info. Each cxx file defines the ranges for all the cuts on that particle (though which cuts are actually used are defined in DataManager::selectParticles()). Each particle has a method selectParticle() which returns a particleSelection object.

- **ParticleSelection.h**: These classes define an integer in which each bit corresponds to a cut. There is an enumeration for each cut. So when selectParticle() is called, the particleSelection object contains which cuts the particle passes. This object also contains a number of methods to manipulate and evaluate these cuts, most important being passesCuts(bits), where bits is an integer specifying which cuts you want the particle to pass. If the particle passes all the cuts, it returns true. Examples are in

dataManager::selectParticles(). Note that in the bitword, 0 specifies a bit that IS passed.

- **AnyTree.C/.h**: This is a custom TSelector class: it allocates the data vectors to which the branches are writen, and then defines an event loop in which all the other classes are called. It is very different from your typical makeSelector() class, in that it only reads the branches specified in the config file, and all other branches are turned off, which improves performance. In addition, it checks for the existence of the branches before reading them/allocating memory, and as such avoids the crashes that often plague this type of ntuple analysis.

Advanced Features and Utilities

The above functionality should remain constant, however, new features and utilities are expected to be added as time goes on

- **HistoMgr**: This class allows you to avoid having to declare, initialize, and fill histograms in different files. Instead, you just call its "fill" function, and it does the declaration/initialization for you. Because of its implementation (as a singleton) it can be used across any number of files, and is already setup in myAnalysis, dataManager, and WZanalyMod, using the following:

```
HistoMgr * m_histoMgr = HistoMgr::Instance();
//...
m_histoMgr->fillHisto(histoName, histoTitle, nBins, xLow, xHigh, fillValue, weight);
```

- **AnalysisModules**: These are classes to contain blocks of self-contained analysis code, that can be then included and called from myAnalysis. They only take particleData (m_data) as input. This allows us to save our own code to the EWAnalysis package without changing myAnalysis, and also allows us to easily turn-on/turn-off blocks of analysis code. Right now only the following exist: myModule.cxx, a dummy analyisis, and WZacceptMod.cxx, a module I wrote to look at systematics for WZ studies.

- **CutflowMgr**: This class keeps track of particle and event cuts you make in your analysis, and then prints a summary of the efficiencies of each of these cuts. It is configured automatically for particle selection cuts. For event level cuts, it is called using the following:

```
m_cutMgr->applyCut("objectBeingCut","nameOfCut",passesCut);
//... more cuts ...
m_cutMgr->cutsDoneFor("objectBeingCut");
```

This last command lets the tool know the event/particle/other-object is done being cut. After the analysis is over, myAnalysis calls m_cutMgr->printCuts(). A sample of the output is attached here: cutflowOutput.txt. The sequential efficiency is out of events

passing the cuts up till that point, the raw efficiency is out of all events, and the exclusive efficiency is out of events where all other cuts are passed.

Supported Ntuples

Right now, D3PDMaker ntuples and Oxford Analysis Ntuples are supported. physicsConfig.cxx is set for the AODToPhysicsD3PD.py nuptles (with default options), and wzConfig.cxx is set for the common SM WZ ntuples that are being centrally produced (all three are D3PDmaker based). The MCP ntuple (harvardConfig.cxx) is being developed by Martin, but the scope of the ntuple (detailed hit information, etc), is a bit more detailed than UMAnalysis on the moment, so the reader will probably only access some of the ntuples content (or the framework will grow). In addition, I have all the information to support EWPA ntuples, let me know if you need this functionality.

Adding Modules

Copy AnalyModules/MyModule.cxx and rename it as you wish. In MyAnalysis.cxx, add an include statement for your module at the top, and add the following in the MyAnalysis constructor:

```
AnalyModule * yourModule = new YourModule(m_data);
m_analysisModuleList.push_back(yourModule);
```

Finally add your module to the list of classes to be compiled in run/compileAll.C

```
srcFiles.push_back("AnalyModules/YourModule.cxx");
```

That should be it!

Adding Variables

If you want a variable added, say muon chi2 information, you just need to do the following: (1) add it to yourConfig.cxx initMuonMap (where the postfixes are added), (2) add the variable to muon.h, and then go (3) dataManager::makeMuon() and add a line to fill the variable.

```
(1)  muonMap["fitChi2"] = "trackfitchi2"; // in your config file
(3)   aMuon->fitChi2 =  Get<double>("muon_fitChi2",i);    //in dataManager::makeMuon()
```

In these maps, the first string corresponds to the variable that will get filled (of the muon object), and the second string is the postfix of the branch where the data is located. (So that if the MuonBranchName is set to be "MuidMuon_", the branch that will be searched for is MuidMuon_trackfitchi2.) Make sure that the type of the variable you add is the same type as the branch. Get will attempt to cast the branch found to that type, which could cause problems if you are not careful.

Other Ntuple Types

It should be very easy to alter this for your favorate ntuple type. Just change the branch name prefixes (ex MuidMuon) and postfixes (ex. _px) listed in oxfordConfig.cxx to your own! That should be all. If you do, let me know, and put the config file in the package to share! Contact me if there are any problems.

Available Ntuples

As mentioned on lxatlasumass2 a selection of oxford ntuples exist at /data/atlas/data/oxNtuple. D3PD ntuples will be stored somewhere soon... To make your own ntuples, see one of the following pages:

- D3PDMaker
- EWPA

Planned Improvements

The framework is now in a period of growth, and so I consider it time to evaluate what areas of the framework structure could use improvement. The below will be addressed within the following weeks:

- There are a number of old variables that the particle classes contain and attempt to fill based on a defunct ntuple format. These should be cleaned up. This would get rid of most of the printouts of "empty variables" (which are not used anyways).
- Selection needs to be improved, right now the muon object defines the quality cuts that are then manipulated by a MuSelections object. Instead I would like to have the MuSelections object to define the cuts (possibly being initialized by a muon). I would also like to add a MuSelections::selectMuons() method that could take in a vector of muon pointers and return the selected muons, which would clean up the DataManager class as well.
- MyAnalysis is now not needed. All modules should be loaded in AnyTree.C instead. MyAnalysis existed to have an event loop which could be independent of different TSelector classes from different tried, but now AnyTree deals with all of them! It is now a superfluous (but harmless) layer.
- In AnyTree.C the readTree() function is awkwardly placed, but I'm not sure where else it should go.
- The DataManager.cxx class has many ways it could improve performance, by reducing unnecessary data copying and more importantly, fewer map operations (costly as they involve string comparisons). The structure of the DataManager is also a bit bloated, as it manages all filling and selection. Some of its functionality could be moved into the Selection classes, and/or new modules or classes.
- Configuration needs to be improved. The method of passing configuration is not ideal, and there also should be a separation of tree dependent configuration (branch names, etc) and tree-independent configuration (pt cuts, modules to load).

- Histograms output should be changed so that a directory is created in the output file for each module, in order to keep all the histograms organized.

-- AndrewMeade - 18-Jan-2010

- cutflowOutput.txt: sample output from CutflowMgr
