Muon p_T scale at high energies in the CMS detector

Grace E. Cummings
Virginia Commonwealth University, SIST Intern
07 Aug 2015

Advisers Pushpa C. Bhat and Leonard G. Spiegel
Fermi National Accelerator Laboratory

Abstract

For searches beyond the Standard Model with the Compact Muon Solenoid (CMS) detector, a detailed understanding of systematic uncertainty in the transverse momentum (p_T) scale of the detector is necessary. From cosmic ray muon studies, it is apparent that the barrel of CMS has a 5% uncertainty in the p_T scale for muons at 1000 GeV/c. This scale is unobserved at low or moderate p_T values, and therefore does not affect the reconstruction of the Z boson's dimuon decay channel. The 5% uncertainty becomes important in the search for massive new particles (e.g., Z'). This uncertainty is believed to arise from a weak mode within the detector. For the Run 2 CMS detector the uncertainty in the scale for high p_T muons and its effect on reconstructed dimuon invariant mass has not yet been studied. Dimuon invariant mass is the mass of a parent particle with $\mu^+$and μ^- decay products. Minimum dimuon mass is used to impose lower mass thresholds (cuts) on event counting studies in particle detection. Through our study, we have developed a parameterization for the effect of p_T scaling on event count within CMS as a function of dimuon minimum mass.

1. The Compact Muon Solenoid Experiment

1.1 The Detector

The Compact Muon Solenoid (CMS) detector is one of two general purpose detectors at the LHC [1]. The detector is made of several sub-detector systems. The collision point lies at the nominal center of the detector, and the layers of the detector cylindrically extend outward, with each detector sub-system designed to measure different properties of collision events. The innermost layer of the detector, closest to the collision point, is the silicon tracker, which measures the paths of particles as they leave the collision. Following the tracker is the electromagnetic calorimeter (ECAL). This detector layer stops and measures the energy contained in electrons and photons. Radially out, the hadron calorimeter (HCAL) measures the energy of hadronic collision products. The last layer of detecting material is the muon system, a further stratified system with alternating layers of muon chamber and steel flux-return yoke for the magnetic field. The muon system is the last detector subsystem. Muons are the only particles (besides neutrinos) that barely interact in matter and live long enough to travel into outer layers of the detector.

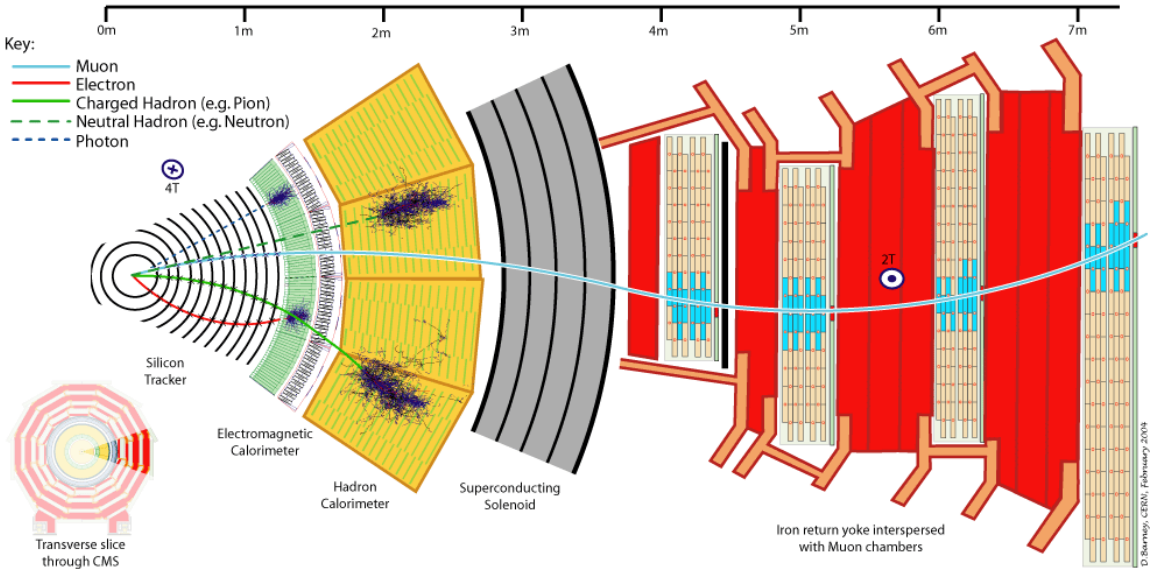

Figure 1: Zoomed in image of a radial section of the CMS detector illustrating particle paths and interactions. [2]

Between the HCAL and the muon system lies the 4 Tesla superconducting solenoid magnet. The magnetic field bends the paths of charged particles within the detector, allowing for momentum measurement. Past the solenoid, the muons change the direction of their curvature due to the returning magnetic field outside of the magnet. For high p_T muons, the muon system dominates the resolution of muons.

1.2 p_T scaling in the CMS detector

The measurement of the transverse momentum (p_T), η, and ϕ parameters of particles within the CMS detector allows for the reconstruction of the parent particle's mass. For muons which have little interaction in the calorimeter, the p_T measurement is the only method to determine the energy of the particle, a quantity necessary for the parent mass calculation. At high p_T, a systematic uncertainty enters the measurement [3], believed to be from a weak mode within the muon system. For a p_T value of 1 TeV, the uncertainty in the muon p_T scale is ~5%, and this scaling is assumed to increase with increasing p_T. Cosmic ray muon studies, used for the commissioning and alignment of the detector, provided this 5% value.

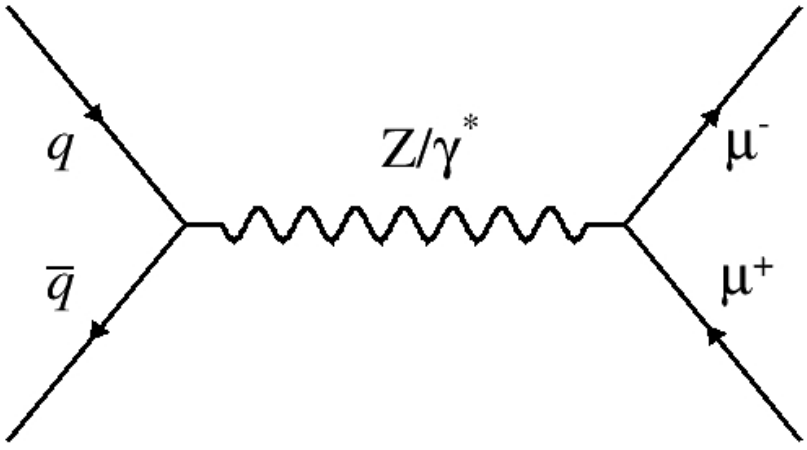

Figure 2: Drell-Yan process Feynman diagram produced using ROOT.

For the first run of the LHC, the Aachen CMS research group analyzed the effects of this scaling for 8 TeV collisions [4]. The goal of their study was to understand how the p_T scaling affects the dimuon invariant mass calculation, and therefore affects the number of particles counted above a certain mass threshold. The scaling could result in an artificial over-counting of events above a minimum mass threshold, potentially misleading particle searches. We performed this study to look into this effect on the Drell-Yan process with dimuon decay at 13 TeV.

The Drell-Yan process is the quark/antiquark annihilation that produces either a Z boson or a virtual photon, which then decays to two oppositely charged leptons (Fig. 2) [5]. Dimuon production is the focus of this study. Drell-Yan is the primary background for many beyond the Standard Model searches with dilepton final states, such as the search for the theorized Z' (a massive analog of the Z boson); thus an accurate count of the background events is paramount.

1.3 CMSSW, the user data handling software

For the standardization of analysis across the experiment, the CMS experiment has developed an analysis framework known as CMSSW [6]. The framework includes generation, simulation, and analysis systems for the CMS experiment. CMSSW is accessed through release packages installed either through cmslpc at FNAL, lxplus at CERN, or other computing centers associated with the CMS collaboration. Once the release has been installed, aspects of CMSSW can be called through the command line.

CMSSW integrates many aspects of high energy physics analysis, forging links between traditionally standalone programs without the need for explicit user input. This efficiency streamlines complex workflows. The framework serves to standardize code, allowing for easier communication across the collaboration; however, the standardization of the code and

implementation of links reduces user control through a dependence on automatically generated script skeletons and dense file downloads [7]. The CMSSW framework is also harder to initially learn due to its complexity and documentation.

2. Monte Carlo Generation with PYTHIA 8

To model the effects of p_T scaling on the Drell-Yan process, the process is generated using Monte Carlo techniques. Monte Carlo is an experimental technique used to generate outcomes based on defined initial parameters and random generation based on probability functions. In the case of high energy particle physics, this produces collision events and their respective measurable characteristics, such as particle energy or momentum.

PYTHIA 8 is a Monte Carlo generator used to produce high energy physics events. It can model collisions of electron beams, electron/positron beams, proton beams, and proton/antiproton beams [8, 9]. The default parton distribution library used in PYTHIA 8 is CTEQ5L [10]. PYTHIA 8 can run in two ways: standalone and within the CMSSW framework. The standalone version runs with C++ main programs, and can produce its own initial analysis. To do more sophisticated analysis, PYTHIA 8 output is often merged with ROOT in order to produce a ROOT formatted output file. In principle, the PYTHIA 8 and ROOT standalone libraries can be linked, but this is difficult in practice. The CMSSW framework streamlines the link between PYTHIA 8 and ROOT. Within CMSSW, PYTHIA 8 is called with a Python script that loads issued support files allowing the Python language to read the PYTHIA 8 commands. Regardless of the mode of usage, PYTHIA only generates the initial events. Within the framework, the Python configuration file can be changed to run the events through the CMS detector. In our project, we focused only on generator-level production, with no CMS detector simulation.

In this study, PYTHIA generated high p_T dimuons through the Drell-Yan process. Table 1 contains the PYTHIA 8 cards used for Drell-Yan event generation. Additional cards were also used to set the tune of the generation, but are not directly related to the Drell-Yan process. PYTHIA 8 allows both $2 \rightarrow 1$ and $2 \rightarrow 2$ interaction possibilities for the Drell-Yan process. For the generation of $Z/\gamma^* \rightarrow \mu^+\mu^-$, PYTHIA 8 uses a combination of electroweak physics and QCD physics, to account for the whole process of quark/antiquark annihilation and Z boson decay [8].

Table 1: PYTHIA 8 Cards for Drell-Yan Process

PYTHIA 8 Card	Process Description
WeakZ0:gmZmode = 0	Default, allows for γ^* and Z boson decay with interference
WeakSingleBoson:ffbar2gmz = on	Turns on Drell-Yan production
23:onMode = off	Turns off decays from Z^0/γ^*
23:onIfAny = 13	Turns on Z/γ^* dimuon decay channel
PhaseSpace:mHatMin = 'str+(options.minMass)	Set minimum generated $\sqrt{\hat{s}}$ (parton center-of-mass energy) through command line
PartonLevel:FSR = on	Default, turns on final state radiation

3. Analysis

The analysis of the generated data takes two steps: a ntuple generation step, to reduce size and content of the original generation file, and a second analysis step, applying the p_T scaling and mass reconstruction.

3.1 Initial Analysis

The event files were generated within CMSSW, thus the file's structure and content are dictated by the Event Data Model, featured in the program package. Due to the EDM, the event data is named and called through standardized headings. Each datum is assigned four variables, a C++ data type (or handle), a module label, a product instance label, and a process name [11]. To access data types within a file, there is an analyzer class within CMSSW. EDAnalyzer, the CMSSW class, allows for the analysis of a data file while preserving the original structure and content. To work within the EDAnalyzer class, CMSSW has a utility to generate skeleton analyzers. These skeletons are customized to perform the desired tasks, and are called through a C++ plugin in a Python script.

The initial ntupler, calls the "recoGenParticleCollection" C++ handle, to begin selecting for muons. The ntupler is a C++ plugin called through a second Python configuration script. The generation file has no detector simulation, thus the muons have not been reconstructed and assigned a new data type. They are selected manually. Using the PID value for muons, 13, the analyzer loops through all particles generated. PYTHIA 8 generates and stores all of the particles produced, including intermediate particles. The plugin code was adapted to collect the last daughter particles using a recursive function. Only collecting last daughters accounts for final

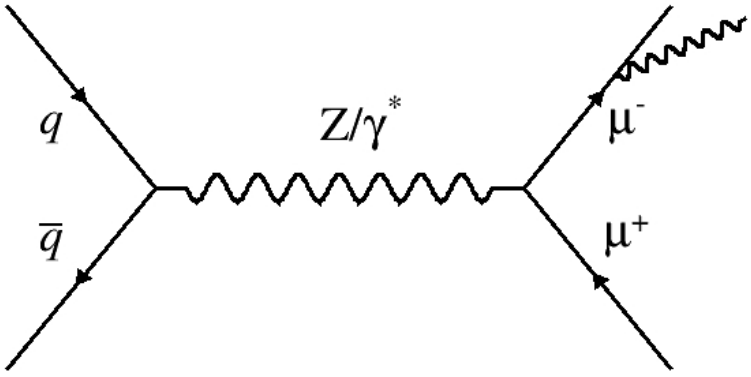

Figure 3: Drell-Yan process Feynman diagram including final state radiation.

state radiation (Fig. 3). This radiation results in dimuons that have a reconstructed mass that may be less than the generated "minimum mass."

Once the muons are selected, their η, ϕ, and p_T values are collected and written to a ROOT tree stored in a new file, forming the ntuple for further analysis. A Python script executed with the CMSSW command *cmsRun* calls the plugin that selects for muons and fills the tree. The Python executable applies the plugin to the initial generation file and writes the output ROOT file containing the tree.

3.2 ROOT Analysis

To apply the p_T scaling and calculate the dimuon invariant mass, the study employed a ROOT class structure [12]. The TTree:MakeClass() command executes within the ROOT command line, and generates two C++ files, a header file and an executable, tailored to the contents of a ROOT tree. The branches of the tree, which contain the event data, are assigned to variables in the header file. The executable contains an automatically generated loop that when called for the variables defined in the header file retrieves all of the values housed at the tree branch address. Once the event data can be accessed through variables, manipulation can be done through the executable C++ macro. A class was made from the tree produced within the output of our ntuple. To view my ROOT macros, please see Appendix II. The original tree had branches for each final state muon daughter of the Drell-Yan process. For each muon, the p_T had to be scaled separately. When the p_T is in GeV, the scale factor used for p_T is

$$scale\ factor = 1.0\ \pm 0.05\left(\frac{p_T}{1000}\right). \qquad (1)$$

This factor accounts for the observed 5% scaling of p_T for $p_T = 1000\ \mathrm{GeV/c}$. For the dimuon mass calculations, a scaling was calculated for a both a 5% increase and decrease at 1000 GeV/c. The Aachen study at 8 TeV assumed the same scale factor as show in Equation (1).

Using the scaled p_T, η, and ϕ, the macro calculates the dimuon invariant mass. Since the mass of the muon, roughly 105.7 MeV/c^2, is very small in comparison to the masses in our study (800 GeV/c^2 and larger) we ignored the muon mass in our mass calculation. Dropping the muon mass must be done carefully to find the invariant mass expression. The following expression

$$M_{\mu 1\mu 2} = \sqrt{2p_{T1}p_{T2}(\cosh(\eta_1 - \eta_2) - \cos(\phi_1 - \phi_2))}, \quad (2)$$

was used to calculate the invariant dimuon mass. For the complete derivation of this expression, please see Appendix I. The pseudorapidity, η, within the mass expression is given as

$$\eta = -\ln\left(\tan\left(\frac{\theta}{2}\right)\right). \quad (3)$$

This term is used instead of θ because $\Delta\eta$ is Lorentz invariant.

For particle discoveries, an accurate count of the number of events above a certain threshold is critical. To transfer the scaling experienced through the p_T to an effect on the number of particles detected, the difference between the count of entries produced due to the scaled mass and count of true entries must be calculated. Starting at 1000 GeV/c^2, the number of entries greater than or equal to the selected lower mass threshold are calculated, in increments of 20 GeV/c^2. Then, the relative difference, given as

$$reldiff = \left|\frac{(entries_{scaled} \geq M_{min}) - (entries_{true} \geq M_{min})}{(entries_{true} \geq M_{min})}\right|,$$

is calculated. This expression gives the fractional difference between the number of true entries and the entries that would be expected due to scaling. All of this is included in the executable ROOT macro.

4. Results

The study consisted of three 100k event root files containing Drell-Yan produced events. The first file had a minimum generated $\sqrt{\hat{s}}$ (parton center-of-mass energy) of 800 GeV/c^2, the second 1300 GeV/c^2, and the final file 1800 GeV/c^2. These files were then run through the analysis work flow, and produced shifted mass values and relative difference distributions. To examine the mass, the ROOT macro produced four graphs (Fig. 4). These graphs served as a check for the correctness of our analyzer. To achieve better statistics, the relative difference calculations from the three files were merged to achieve one relative difference distribution (Fig. 5). The 800 GeV/c^2 minimum dimuon mass generation filled the relative difference distribution from 1000 to 1499 GeV/c^2 minimum mass threshold, the 1300 GeV/c^2 minimum dimuon mass from 1500 to 1999 GeV/c^2 minimum mass threshold, and the 1800 GeV/c^2 minimum dimuon mass distribution from 2000 to 2399 GeV/c^2 minimum mass threshold.

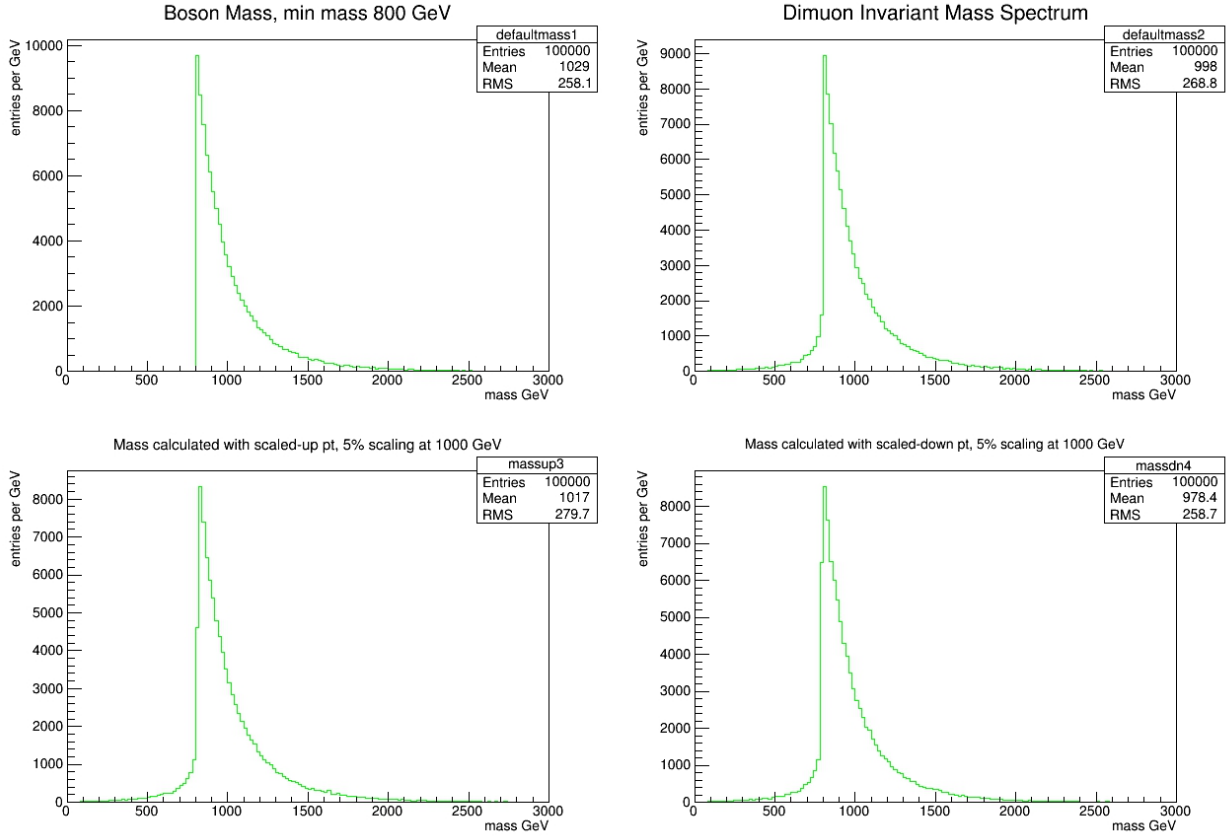

Figure 4: Mass plots for 100,000 event generation, minimum dimuon mass 800 GeV/c^2. One of three sets (800 GeV/c^2, 1300 GeV/c^2, and 1800 GeV/c^2). Top Left: Boson mass generated without final state radiation. Top Right: Dimuon mass generated with final state radiation. Bottom Left: Dimuon mass with FSR calculated with 5% up-scaling in p_T at 1000 GeV/c. Bottom Right: Dimuon mass with FSR calculated with 5% down-scaling in p_T at 1000 GeV/c. Scaling dependent on mass.

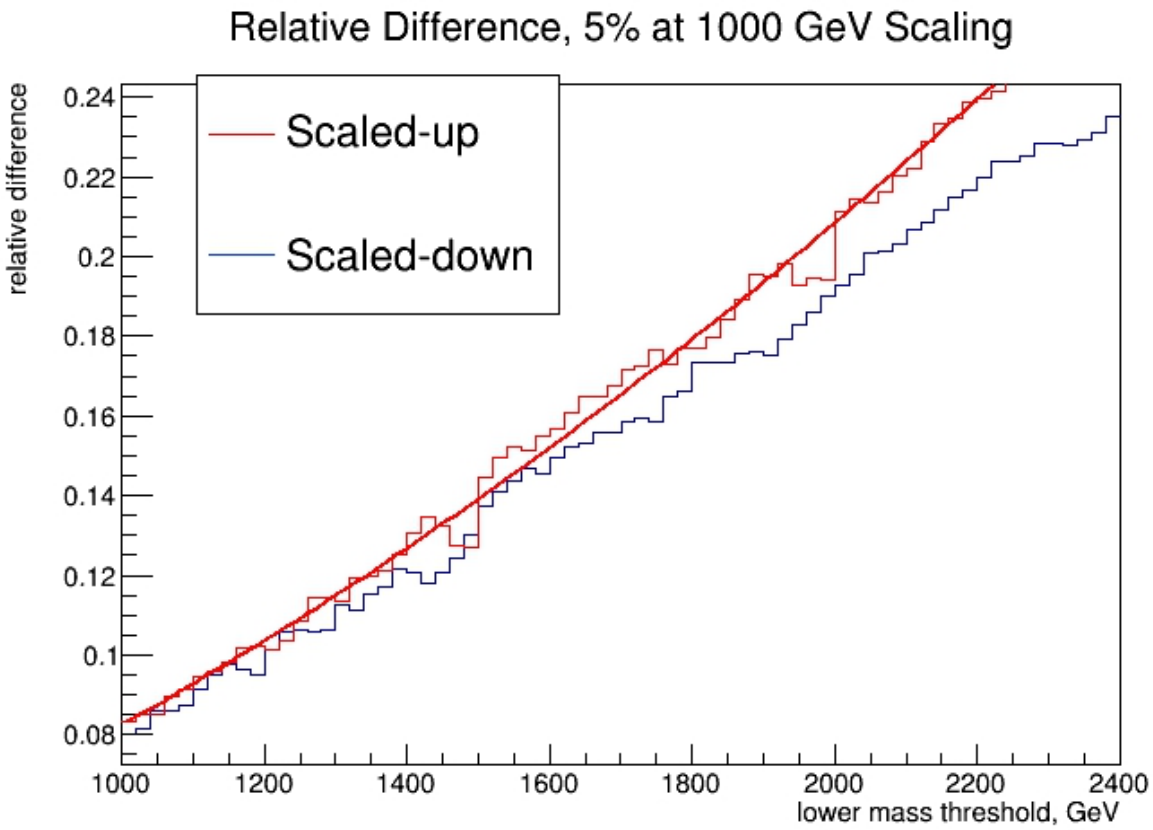

Figure 5: Relative difference plots. Red scaled-up, blue scaled-down, 5% scaling in both. Scale up features a quadratic fit.

A quadratic fit was applied to the scaled-up relative difference distribution to make a parameterization of the fractional difference in entries as a function of minimum mass. The quadratic fit was chosen to coincide with the Aachen study. This fractional difference can be taken as the uncertainty in the event count as a function of minimum mass. The fit is applied to the scaled up graph to account for the worst-case scenario. The fit produced is given as

$$uncertainty_{events} = 6.90 \times 10^{-3} + 5.03 \times 10^{-5} m_{min} + 2.52 \times 10^{-8} m_{min}^2.$$

The parameters and their errors are given in Table 2.

Table 2:Fit Parameters	
p0	0.00689805 $\pm$ 2.29103
p1	5.03351e-05 $\pm$ 0.00279879
p2	2.52364e-08 $\pm$ 8.18564e-07

5. Conclusions

Knowledge of the effects of p_T scaling on the count of mass dependent events is crucial to our understanding of the uncertainty associated with the search for beyond the standard model physics. Through Monte Carlo generation and analysis, we have developed a preliminary parameterization of the uncertainty in the entry count as a function of minimum dimuon mass. With our scripts, we have constructed the basis to extend the study to incorporate full detector simulation, to refine the parameterization.

6. Acknowledgements and Thanks

I would like to acknowledge Shawn Zaleski (Wayne State University) for his EDAnalyzer plugin and Danny Nooman (Florida Institute of Technology) for the addition of FSR sensitivity. This study would have been much harder without the guidance and motivation of Graham Stoddard (Northern Illinois University) and the beginner coding assistance from Shannon Massey (University of Notre Dame). Finally, I want to thank my advisers for their optimism, and the entire SIST committee for allowing me the chance to work with my heroes.

References

1. L. Taylor, http://cms.web.cern.ch/news/cms-detector-design (2011)
2. https://www.phys.ksu.edu/reu2014/wabehn/index_files/image005.gif
3. G. Abbiendi, https://twiki.cern.ch/twiki/bin/viewauth/CMS/MuonReferenceResolution
4. T. Pook, https://twiki.cern.ch/twiki/bin/viewauth/CMS/AachenADD
5. S. Drell and T. Yan, Phys. Rev. Lett. 25(1970)316, doi: 10.1103/PhysRevLett.25.316
6. S. Malik, https://twiki.cern.ch/twiki/bin/view/CMSPublic/WorkbookMoreOnCMSSWFramwwork (2012)
7. G. Cummings, Inct. Swag. 05(1994)05
8. T. Sjöstrand, S. Mrenna and P. Skands, JHEP05 026
9. T. Sjöstrand et al, Comput. Phys. Commun. 191 (2015) 159 [arXiv:1410.3012[hep-ph]]
10. D. Dagenhart, https://twiki.cern.ch/twiki/bin/view/CMSPublic/SWGuideEDMGetDataFromEvent (2014)
11. Root User's Manual, https://root.cern.ch/root/htmldoc/guides/users-guide/ROOTUsersGuideA4.pdf

Appendix I

Derivation of $M = \sqrt{2\, p_{T1}\, p_{T2}(\cosh(\eta_1 - \eta_2) - (\cos(\phi_1 - \phi_2))}$.

Begin with the four-vector method of calculating invariant mass, and start to simplify, producing

$$M^2 = \left(p_1^{\mu} + p_2^{\mu}\right)\left(p_{1\,\mu} + p_{2\,\mu}\right),$$

$$= \left(\begin{pmatrix} E_1 \\ p_{x1} \\ p_{y1} \\ p_{z1} \end{pmatrix} + \begin{pmatrix} E_2 \\ p_{x2} \\ p_{y2} \\ p_{z2} \end{pmatrix}\right)\left(\begin{pmatrix} E_1 & p_{x1} & p_{y1} & p_{z1} \end{pmatrix} + \begin{pmatrix} E_2 & p_{x2} & p_{y2} & p_{z2} \end{pmatrix}\right),$$

$$= \begin{pmatrix} E_1 + E_2 \\ p_{x1} + p_{x2} \\ p_{y1} + p_{y2} \\ p_{z1} + p_{z2} \end{pmatrix}\begin{pmatrix} E_1 + E_2 & p_{x1} + p_{x2} & p_{y1} + p_{y2} & p_{z1} + p_{z2} \end{pmatrix},$$

$$= (E_1 + E_2)^2 - (p_{x1} + p_{x2})^2 - \left(p_{y1} + p_{y2}\right)^2 - (p_{z1} + p_{z2})^2.$$

First, we will simplify the momentum terms

$$-(p_{x1} + p_{x2})^2 - \left(p_{y1} + p_{y2}\right)^2 - (p_{z1} + p_{z2})^2 =$$
$$-\left({p_{x1}}^2 + 2\, p_{x1}\, p_{x2} + {p_{x2}}^2\right) - \left({p_{y1}}^2 + p_{y1}\, p_{y2} + {p_{y2}}^2\right) - \left({p_{z1}}^2 + p_{z1}\, p_{z2} + {p_{z2}}^2\right),$$
$$= -\left({p_{x1}}^2 + {p_{y1}}^2 + {p_{z1}}^2\right) - \left({p_{x2}}^2 + {p_{y2}}^2 + {p_{z2}}^2\right) - 2\,\vec{p}_1 \bullet \vec{p}_2.$$

Next, we will simplfy the energy term

$$(E_1 + E_2)^2 = \left(\sqrt{{m_1}^2 + \left({p_{x1}}^2 + {p_{y1}}^2 + {p_{z1}}^2\right)} + \sqrt{{m_2}^2 + \left({p_{x2}}^2 + {p_{y2}}^2 + {p_{z2}}^2\right)}\right)^2$$
$$= {m_1}^2 + \left({p_{x1}}^2 + {p_{y1}}^2 + {p_{z1}}^2\right) +$$
$$2\sqrt{{m_1}^2 + \left({p_{x1}}^2 + {p_{y1}}^2 + {p_{z1}}^2\right)}\ \sqrt{{m_2}^2 + \left({p_{x2}}^2 + {p_{y2}}^2 + {p_{z2}}^2\right)} + {m_2}^2 + \left({p_{x2}}^2 + {p_{y2}}^2 + {p_{z2}}^2\right)$$
$$= {m_1}^2 + {m_2}^2 + \left({p_{x1}}^2 + {p_{y1}}^2 + {p_{z1}}^2\right) + \left({p_{x2}}^2 + {p_{y2}}^2 + {p_{z2}}^2\right) + 2\, E_1\, E_2.$$

Combining the expressions gives

$$M^2 = (E_1 + E_2)^2 - (p_{x1} + p_{x2})^2 - \left(p_{y1} + p_{y2}\right)^2 - (p_{z1} + p_{z2})^2,$$
$$= {m_1}^2 + {m_2}^2 + \left({p_{x1}}^2 + {p_{y1}}^2 + {p_{z1}}^2\right) + \left({p_{x2}}^2 + {p_{y2}}^2 + {p_{z2}}^2\right) +$$
$$2\, E_1\, E_2 - \left({p_{x1}}^2 + {p_{y1}}^2 + {p_{z1}}^2\right) - \left({p_{x2}}^2 + {p_{y2}}^2 + {p_{z2}}^2\right) - 2\,\vec{p}_1 \bullet \vec{p}_2,$$
$$= {m_1}^2 + {m_2}^2 + 2\, E_1\, E_2 - 2\,\vec{p}_1 \bullet \vec{p}_2.$$

We now can factor out the 2, and since the muon mass is negligible, we can set m_1 and m_2 equal to 0, giving

$$M^2 = 2\left(E_1\, E_2 - \vec{p}_1 \bullet \vec{p}_2\right).$$

We can simplfy the above expression using the transverse energy, E_T, equals $E_T = {p_x}^2 + {p_y}^2 + m^2$. Using the relation $E_1\, E_2 = E_{T1}\, E_{T2} \cosh\eta_1 \cosh\eta_2$, we get

$$M^2 = 2\left(E_{T1}\, E_{T2} \cosh\eta_1 \cosh\eta_2 - \vec{p}_1 \bullet \vec{p}_2\right).$$

Next, simplify the dot product and put it in terms of p_T, producing

$$\begin{aligned}\vec{p}_1 \bullet \vec{p}_2 &= p_{x1}\, p_{x2} + p_{y1}\, p_{y2} + p_{z1}\, p_{z2},\\ &= p_{T1}\cos\phi_1\, p_{T2}\cos\phi_2 + p_{T1}\sin\phi_1\, p_{T2}\sin\phi_2 + E_1 \sinh\eta_1\, E_2 \sinh\eta_2,\\ &= p_{T1}\, p_{T2}\big(\cos(\phi_1 - \phi_2) + E_{T1}\, E_{T2} \sinh\eta_1 \sinh\eta_2.\end{aligned}$$

When substituting back into our expression, we have

$$\begin{aligned}M^2 &= 2\left(E_{T1}\, E_{T2}\cosh\eta_1 \cosh\eta_2 - \vec{p}_1 \bullet \vec{p}_2\right),\\ &= 2\left(E_{T1}\, E_{T2}\cosh\eta_1 \cosh\eta_2 - \left(p_{T1}\, p_{T2}\big(\cos(\phi_1 - \phi_2) + E_{T1}\, E_{T2} \sinh\eta_1 \sinh\eta_2\big)\right)\right),\\ &= 2\left(E_{T1}\, E_{T2}\cosh\eta_1 \cosh\eta_2 - p_{T1}\, p_{T2}\big(\cos(\phi_1 - \phi_2) - E_{T1}\, E_{T2} \sinh\eta_1 \sinh\eta_2\right).\end{aligned}$$

Using hyperbolic trigonometric identities, we receive

$$\begin{aligned}M^2 &= 2\left(E_{T1}\, E_{T2}\cosh\eta_1 \cosh\eta_2 - p_{T1}\, p_{T2}\big(\cos(\phi_1 - \phi_2) - E_{T1}\, E_{T2} \sinh\eta_1 \sinh\eta_2\right)\\ &= 2\, E_{T1}\, E_{T2}\big(\cosh\eta_1 \cosh\eta_2 - \sinh\eta_1 \sinh\eta_2\big) - 2\, p_{T1}\, p_{T2}\big(\cos(\phi_1 - \phi_2),\\ &= 2\, E_{T1}\, E_{T2}\big(\cosh\eta_1 \cosh\eta_2 - \sinh\eta_1 \sinh\eta_2\big) - 2\, p_{T1}\, p_{T2}(\cos(\phi_1 - \phi_2),\\ &= 2\, E_{T1}\, E_{T2}\cosh(\eta_1 - \eta_2) - 2\, p_{T1}\, p_{T2}(\cos(\phi_1 - \phi_2)).\end{aligned}$$

Since the mass of the muon is assumed to be 0, the E_T goes to p_T, leaving

$$\begin{aligned}M^2 &= 2\, E_{T1}\, E_{T2}\cosh(\eta_1 - \eta_2) - 2\, p_{T1}\, p_{T2}(\cos(\phi_1 - \phi_2)),\\ &= 2\, p_{T1}\, p_{T2}(\cosh(\eta_1 - \eta_2) - \ (\cos(\phi_1 - \phi_2)).\end{aligned}$$

Taking the square root,

$$M = \sqrt{2\, p_{T1}\, p_{T2}(\cosh(\eta_1 - \eta_2) - \ (\cos(\phi_1 - \phi_2))}\ .$$

Appendix II

DimuonClass.C

```
//DimuonClass.C ROOT macro

#define DimuonClass_cxx
#include "DimuonClass.h"
#include <TH1.h>
#include <TStyle.h>
#include <TCanvas.h>
#include <TMath.h>
#include <iostream>
#include <TLorentzVector.h>
#include <TString.h>
#include <TTree.h>
#include <TLegend.h>
#include <TLatex.h>

void DimuonClass::Loop()
{
   if (fChain == 0) return;

   Long64_t nentries = fChain->GetEntriesFast();
   Long64_t nbytes = 0, nb = 0;

   //The File
   TFile *fl = new TFile("outputfilename.root","NEW");

   //The Canvas
   TCanvas *c6 = new TCanvas("c6","Relative Diffference, Arrays",25,33,700,400);
   c6->Divide(1,1);

   TCanvas *c8 = new TCanvas("c8","Mass Graphs",23,33,1400,900);
   c8->Divide(2,2);

   TCanvas *c9 = new TCanvas("c9","PT Difference",33,50,700,400);
   c9->Divide(1,1);

   //Canvas for z
   TCanvas *c10 = new TCanvas("c10","Mass vs. Massup",33,50,700,400);
   c10->Divide(1,1);

   //Create Histogram for Zmass
   Int_t range       = 3000;
   Int_t step        = 20;
   Int_t lowestcut   = 1000;
   Int_t bins        = range/step;
   Int_t lowthreshbin = lowestcut/step+1;
   Int_t binsrel     = (2400-lowestcut)/step;

   TH1F *defaultmass = new TH1F("defaultmass","Dimuon Invariant Mass
Spectrum",bins,0,range);//FSR
   defaultmass->GetXaxis()->SetTitle("mass GeV");
```

```
  defaultmass->GetYaxis()->SetTitle("entries per GeV");
  defaultmass->GetYaxis()->SetTitleOffset(1.5);
  defaultmass->SetLineColor(kBlue);
  defaultmass->SetFillColor(kBlue);

  TH1F *bosonmass = new TH1F("bosonmass","True mass",bins,0,range);//no FSR
  bosonmass->GetXaxis()->SetTitle("mass GeV");
  bosonmass->GetYaxis()->SetTitle("entries per GeV");
  bosonmass->GetYaxis()->SetTitleOffset(1.5);


  TH1F *massup = new TH1F("massup","Mass calculated with scaled-up P_{T}, 5% scaling at
1000 GeV",bins,0,range);
  massup->GetXaxis()->SetTitle("mass GeV");
  massup->GetYaxis()->SetTitle("entries per GeV");
  massup->GetYaxis()->SetTitleOffset(1.5);
  massup->SetLineColor(kRed);
  massup->SetFillColor(kRed);

  TH1F *massdn = new TH1F("massdn", "Mass calculated with scaled-down P_{T}, 5% scaling at
1000 GeV",bins,0,range);
  massdn->GetXaxis()->SetTitle("mass GeV");
  massdn->GetYaxis()->SetTitle("entries per GeV");
  massdn->GetYaxis()->SetTitleOffset(1.5);

  TH1F *arrayrelup = new TH1F("arrayrelup","Relative difference in scaled-up 5%
P_{T}",binsrel,1000,2400);
  arrayrelup->SetLineColor(kRed);
  arrayrelup->SetStats(kFALSE);
  arrayrelup->GetXaxis()->SetTitle("Lower mass threshold in GeV");
  arrayrelup->GetYaxis()->SetTitle("Relative Difference in P_{T}");
  arrayrelup->GetYaxis()->SetTitleOffset(1.5);

  TH1F *arrayreldn= new TH1F("arrayreldn","Relative difference in scaled-down 5%
pt",binsrel,1000,2400);
  arrayreldn->SetLineColor(kAzure);
  arrayreldn->SetStats(kFALSE);
  arrayreldn->GetXaxis()->SetTitle("Lower mass threshold in GeV");
  arrayreldn->GetYaxis()->SetTitle("Relative Difference in pt");
  arrayreldn->GetYaxis()->SetTitleOffset(1.5);

  TH1F *ptup = new TH1F("ptup", "P_{T} of muon 1, 800 minimum mass",100,0,1200);
  ptup->SetLineColor(kRed);
  ptup->GetXaxis()->SetTitle("P_{T}, GeV");
  ptup->GetYaxis()->SetTitle("entires per bin");
  ptup->SetStats(kFALSE);

  TH1F *ptdn = new TH1F("ptdn", "P_{T} of muon 1, 800 minimum mass",100,0,1200);
  ptdn->SetLineColor(kAzure);
  ptdn->GetXaxis()->SetTitle("P_{T}, GeV");
  ptdn->GetYaxis()->SetTitle("entires per bin");
  ptdn->SetStats(kFALSE);\

  TH1F *ptmuon1 = new TH1F("ptmuon1", "pt, muon 1",100,0,1200);
  ptup->GetXaxis()->SetTitle("P_{T}, GeV");
```

```
  ptup->GetYaxis()->SetTitle("entires per bin");
  ptup->SetStats(kFALSE);



  //Loop for tree
  for (Long64_t jentry=0; jentry<nentries;jentry++) {
    Long64_t ientry = LoadTree(jentry);
    if (ientry < 0) break;

    nb = fChain->GetEntry(jentry);   nbytes += nb;
    // if (Cut(ientry) < 0) continue;


    //pt multiplying factor
    Float_t momtmu1   = decay1P4_pt;
    Float_t mu1ptup   = momtmu1*(1.0+.05*(momtmu1/1000.0));
    Float_t mu1ptdn   = momtmu1*(1.0-.05*(momtmu1/1000.0));
    Float_t momtmu2   = decay2P4_pt;
    Float_t mu2ptup   = momtmu2*(1.0+.05*(momtmu2/1000.0));
    Float_t mu2ptdn   = momtmu2*(1.0-.05*(momtmu2/1000.0));

    Float_t imup      = TMath::Sqrt(2*mu1ptup*mu2ptup*(TMath::CosH(decay1P4_eta-
decay2P4_eta)-TMath::Cos(decay1P4_phi-decay2P4_phi)));//mass pt up
    Float_t imdn      = TMath::Sqrt(2*mu1ptdn*mu2ptdn*(TMath::CosH(decay1P4_eta-
decay2P4_eta)-TMath::Cos(decay1P4_phi-decay2P4_phi)));//mass w/ pt dn
    Float_t mass      = TMath::Sqrt(2*momtmu1*momtmu2*(TMath::CosH(decay1P4_eta-
decay2P4_eta)-TMath::Cos(decay1P4_phi-decay2P4_phi)));//true invariant masszx

   //Fill Z Histogram
    massup->Fill(imup);
    massdn->Fill(imdn);
    defaultmass->Fill(mass);
    ptup->Fill(mu1ptup);
    ptdn->Fill(mu1ptdn);
    ptmuon1->Fill(decay1P4_pt);
    bosonmass->Fill(bosonP4_mass);

      }


  //Arrays for relative difference analysis

  Double_t massdefault[binsrel];
  Double_t upmass[binsrel];
  Double_t dnmass[binsrel];
  Float_t  upreldiff[binsrel];
  Float_t  dnreldiff[binsrel];

  for (Int_t i = 0; i<binsrel; i++)
    {
    massdefault[i]=defaultmass->Integral(i+lowthreshbin,bins);
    upmass[i]=massup->Integral(i+lowthreshbin,bins);
    dnmass[i]=massdn->Integral(i+lowthreshbin,bins);
    upreldiff[i]=((upmass[i]-massdefault[i])/massdefault[i]);
```

```
  dnreldiff[i]=((dnmass[i]-massdefault[i])/massdefault[i]);
  arrayrelup->AddBinContent(i+1,upreldiff[i]);
  arrayreldn->AddBinContent(i+1,TMath::Abs(dnreldiff[i]));
    }



TLegend *leg = new TLegend(0.16,0.63,0.45,0.91);
leg->AddEntry(arrayrelup, "Scaled-up","l");
leg->AddEntry(arrayreldn, "Scaled-down","l");

TLegend *leg1 = new TLegend(0.75,0.75,0.95,0.95);
leg1->AddEntry(ptmuon1,"true P_{T}","l");
leg1->AddEntry(ptup,"P_{T}, scaled-up 5%","l");
leg1->AddEntry(ptdn,"P_{T}, scaled-down 5%","l");

//Draw
c6->cd(1);
arrayrelup->Draw();
arrayreldn->Draw("SAME");
leg->Draw();

c9->cd(1);
ptdn->Draw();
ptmuon1->Draw("SAME");
ptup->Draw("SAME");
leg1->Draw();

c10->cd(1);
massup->Draw();
defaultmass->Draw("SAME");
c10->Write();

//Draw
c8->cd(1);
bosonmass->Draw();
c8->cd(2);
defaultmass->Draw();
c8->cd(3);
massup->Draw();
c8->cd(4);
massdn->Draw();

//c6->Write();
bosonmass->Write();
c8->Write();
massup->Write();
massdn->Write();
defaultmass->Write();
ptup->Write();
ptdn->Write();
ptmuon1->Write();
arrayrelup->Write();
arrayreldn->Write();
c9->Write();
```

```
}
```

DimuonClass.h

```
//DimuonClass.h

//////////////////////////////////////////////////////////
// This class has been automatically generated on
// Mon Jul 13 14:04:41 2015 by ROOT version 6.02/05
// from TTree pdfTree/PDF Tree
// found on file: DYoutput_cut.root
//////////////////////////////////////////////////////////

#ifndef DimuonClass_h
#define DimuonClass_h

#include <TROOT.h>
#include <TChain.h>
#include <TFile.h>
#include <TH1.h>

 // Header file for the classes stored in the TTree if any.

class DimuonClass {
public :
   TTree          *fChain;   //!pointer to the analyzed TTree or TChain
   Int_t           fCurrent; //!current Tree number in a TChain

// Fixed size dimensions of array or collections stored in the TTree if any.

   // Declaration of leaf types
   Float_t         bosonP4_energy;
   Float_t         bosonP4_et;
   Float_t         bosonP4_eta;
   Float_t         bosonP4_phi;
   Float_t         bosonP4_pt;
   Float_t         bosonP4_mass;
   Float_t         bosonP4_theta;
   Float_t         decay1P4_energy;
   Float_t         decay1P4_et;
   Float_t         decay1P4_eta;
   Float_t         decay1P4_phi;
   Float_t         decay1P4_pt;
   Float_t         decay1P4_mass;
   Float_t         decay1P4_theta;
   Float_t         decay2P4_energy;
   Float_t         decay2P4_et;
   Float_t         decay2P4_eta;
   Float_t         decay2P4_phi;
   Float_t         decay2P4_pt;
   Float_t         decay2P4_mass;
   Float_t         decay2P4_theta;
   Int_t           decay1PID;
```

```
   Int_t          decay2PID;
   Int_t          bosonPID;

   // List of branches
   TBranch        *b_bosonP4;   //!
   TBranch        *b_decay1P4;   //!
   TBranch        *b_decay2P4;   //!
   TBranch        *b_decay1PID;   //!
   TBranch        *b_decay2PID;   //!
   TBranch        *b_bosonPID;   //!

   DimuonClass(TTree *tree=0);
   virtual ~DimuonClass();
   virtual Int_t    Cut(Long64_t entry);
   virtual Int_t    GetEntry(Long64_t entry);
   virtual Long64_t LoadTree(Long64_t entry);
   virtual void     Init(TTree *tree);
   virtual void     Loop();
   virtual Bool_t   Notify();
   virtual void     Show(Long64_t entry = -1);
};

#endif

#ifdef DimuonClass_cxx
DimuonClass::DimuonClass(TTree *tree) : fChain(0)
{
// if parameter tree is not specified (or zero), connect the file
// used to generate this class and read the Tree.
   if (tree == 0) {
     TFile *f = (TFile*)gROOT->GetListOfFiles()->FindObject("ntuple.root");
      if (!f || !f->IsOpen()) {
         f = new TFile("ntuple.root");
      }
      TDirectory * dir = (TDirectory*)f->Get("ntuple.root");
      dir->GetObject("pdfTree",tree);

   }
   Init(tree);
}

DimuonClass::~DimuonClass()
{
   if (!fChain) return;
   delete fChain->GetCurrentFile();
}

Int_t DimuonClass::GetEntry(Long64_t entry)
{
// Read contents of entry.
   if (!fChain) return 0;
```

```
   return fChain->GetEntry(entry);
}
Long64_t DimuonClass::LoadTree(Long64_t entry)
{
// Set the environment to read one entry
   if (!fChain) return -5;
   Long64_t centry = fChain->LoadTree(entry);
   if (centry < 0) return centry;
   if (fChain->GetTreeNumber() != fCurrent) {
      fCurrent = fChain->GetTreeNumber();
      Notify();
   }
   return centry;
}

void DimuonClass::Init(TTree *tree)
{
   // The Init() function is called when the selector needs to initialize
   // a new tree or chain. Typically here the branch addresses and branch
   // pointers of the tree will be set.
   // It is normally not necessary to make changes to the generated
   // code, but the routine can be extended by the user if needed.
   // Init() will be called many times when running on PROOF
   // (once per file to be processed).

   // Set branch addresses and branch pointers
   if (!tree) return;
   fChain = tree;
   fCurrent = -1;
   fChain->SetMakeClass(1);

   fChain->SetBranchAddress("bosonP4", &bosonP4_energy, &b_bosonP4);
   fChain->SetBranchAddress("decay1P4", &decay1P4_energy, &b_decay1P4);
   fChain->SetBranchAddress("decay2P4", &decay2P4_energy, &b_decay2P4);
   fChain->SetBranchAddress("decay1PID", &decay1PID, &b_decay1PID);
   fChain->SetBranchAddress("decay2PID", &decay2PID, &b_decay2PID);
   fChain->SetBranchAddress("bosonPID", &bosonPID, &b_bosonPID);
   Notify();
}

Bool_t DimuonClass::Notify()
{
   // The Notify() function is called when a new file is opened. This
   // can be either for a new TTree in a TChain or when when a new TTree
   // is started when using PROOF. It is normally not necessary to make changes
   // to the generated code, but the routine can be extended by the
   // user if needed. The return value is currently not used.

   return kTRUE;
}
```

```
void DimuonClass::Show(Long64_t entry)
{
// Print contents of entry.
// If entry is not specified, print current entry
   if (!fChain) return;
   fChain->Show(entry);
}
Int_t DimuonClass::Cut(Long64_t entry)
{
// This function may be called from Loop.
// returns  1 if entry is accepted.
// returns -1 otherwise.
   return 1;
}
#endif // #ifdef DimuonClass_cxx
```

runDimuonClass.C

```
//runDimuonClass.C

#include "DimuonClass.C"

void runDimuonClass()
{
  TChain *chain = new TChain("Dimuon/pdfTree","");
  chain->Add("ntuple.root");
  DimuonClass _DimuonClass(chain);
  _DimuonClass.Loop();
}
```

Mergereldiff.C, program to merge relative difference histograms produced by the DimuonClass macros

```
//mergereldiff.C

{

  Int_t step      = 20;
  Int_t lowestcut = 1000;
  Int_t range     = 2400;
  Int_t cut1      = 1500;
  Int_t cut2      = 2000;
  Int_t cut3      = 2400;
  Int_t bins      = (range-lowestcut)/step;

TH1F *hd       = new TH1F("hd","Relative Difference, 5% at 1000 GeV
Scaling",bins,lowestcut,range);
hd->SetStats(kFALSE);
 hd->GetXaxis()->SetTitle("lower mass threshold, GeV");
 hd->GetYaxis()->SetTitle("relative difference");
 hd->GetYaxis()->SetTitleOffset(1.5);
 hd->SetMaximum(0.24);

TH1F *hu       = new TH1F("hu","Relative Difference",bins,lowestcut,range);
 hu->SetLineColor(kRed);
 hu->SetStats(kFALSE);
 hu->SetMaximum(0.24);

TFile f800("rebintest800_scaledby5perc.root");
TH1F  *h800up  = (TH1F*)gDirectory->Get("arrayrelup");
TH1F  *h800dn  = (TH1F*)gDirectory->Get("arrayreldn");

TFile f1300("rebintest1300_scaledby5perc.root");
TH1F  *h1300up = (TH1F*)gDirectory->Get("arrayrelup");
TH1F  *h1300dn = (TH1F*)gDirectory->Get("arrayreldn");

TFile f1800("rebintest1800_scaledby5perc.root");
TH1F  *h1800up = (TH1F*)gDirectory->Get("arrayrelup");
TH1F  *h1800dn = (TH1F*)gDirectory->Get("arrayreldn");

TLegend *leg = new TLegend(0.16,0.63,0.45,0.91);
   leg->AddEntry(h800up, "Scaled-up","l");
   leg->AddEntry(h800dn, "Scaled-down","l");

   for (Int_t i = 1;i<=bins; i++)
     {
       if(i<=((cut1-lowestcut)/step)){
          hd->AddBinContent(i,h800dn->GetBinContent(i));
          hu->AddBinContent(i,h800up->GetBinContent(i));
       }
       else if (i<=((cut2-lowestcut)/step)){
          hd->AddBinContent(i,h1300dn->GetBinContent(i));
```

```
        hu->AddBinContent(i,h1300up->GetBinContent(i));
      }
      else if (i<=((cut3-lowestcut)/step)) {
        hd->AddBinContent(i,h1800dn->GetBinContent(i));
        hu->AddBinContent(i,h1800up->GetBinContent(i));
      }
    }


TCanvas *c9 = new TCanvas("MyCanvas","Merged Hists",700,500);
 hd->Draw();
 hu->Draw("SAME");
 leg->Draw();

TFile *fnew = new TFile("mergedanalysis.root","NEW");
/
                    c9->Write();
}
```