Matthias Schott

Analysis of AOD-Files with Athena

What we will learn today?

- Where is help?
- Getting started
  - Check out of an Analysis Package
  - Overview of the Package Structure
  - Run it
- Playing with AOD Files
  - How many myons are reconstructed? What is the pt of the reconstructed myons? And what the hell is “storegate”?
  - A simple analysis example: What is the Z-mass?
  - What is the pt-resolution of the reconstructed myons.
- I want my own Athena Package

Where is help?

Links

- **Hilfe zur Analyse mit AOD-Files**
  - Athena Twiki
    http://uimon.cern.ch/twiki/bin/view/Atlas
  - Working with AOD-Files:
    http://www.usatlas.bnl.gov/PAT/analysis_on_aod.html
  - Source-Reference of Athena
    http://reserve02.usatlas.bnl.gov/lxr/source/atlas

Immer Überprüfen mit welcher Version man arbeitet und welche Pakete bzw. Paketnummern man auschecken will

- Zunächst muss die Athena Umgebung aufgesetzt werden (vgl. Günther)
  - cp /software/atlas/dist-kit/AtlasReleases/setup_10.0.3.sh Verzeichnis
  - nedit setup_10.0.3.sh
  - Änderung: cernusername = mschott
  - source setup_10.0.3.sh
- Es gibt Athena Versionen wie Sand am Meer. In München ist zur Zeit Version 10.0.1 installiert. Wenn wir ein Athena-Paket auschecken wollen, müssen wir wissen, welche Version wir haben wollen. Dies kann unter (http://atlas.web.cern.ch/Atlas/GROUPS/SOFTWARE/OO/Development/) überprüft werden.
- cmt co -r UserAnalysis-00-02-10 PhysicsAnalysis/AnalysisCommon/UserAnalysis
- Wenn wir die allerneuste Version auschecken wollen, verwenden wir einfach
- cmt co PhysicsAnalysis/AnalysisCommon/UserAnalysis

The usage of *cmt br* may be omitted in our analysis

- cd PhysicsAnalysis/AnalysisCommon/UserAnalysis/ UserAnalysis-00-02-10
- Inhalt des Verzeichnis
  - \run : Hier starten wir athena.py
  - \cmt : Hier befinden sich Konfigurationsdatein
  - \src : Hier befindet sich der Quellcode AnalysisSkeleton.cxx
  - \share : Hier befinden sich diverse JobOptionFiles, die
  - \UserAnalysis : Hier befindet sich das Header-File AnalysisSkeleton
- Initialisierung (Die Verwendung „cmt br“ wird erst bei der Entwicklung wichtig)
  - cd cmt
  - cmt br cmt config
  - source setup.sh
  - cmt br gmake

The file *share/myTopOptionFile.py* should give you a starting point for your own analysis

- Wir gehen in das /run/ Verzeichnis. Um Athena zu starten brauchen wir ein JopOptionFile. Sollte sich beim Packet kein File im run/ Verzeichnis befinden, kopieren wir
- cp ../share/myTopOptions.py ./jobOptions.py
- Betrachten wir zunächst das File jobOptions.py (nedit jobOptions.py)
  - Zunächst werden mit include(“Beispiel_jobOptions.py") diverse Services gesetzt. Das interessiert und nicht weiter …
  - Mit EventSelector.InputCollections = [“Beispiel.AOD.root”] setzt man das File, das analysiert werden soll. Dazu kommen wir später
  - Durch theApp.Dlls += [“ImportantPackage”] werden Bibliotheken initialisiert, die unsere Anwendung benötigt.
  - Der Algorithmus, bei dem die Funktionen initialize(), execute() und finalize() für jedes Event aufgerufen werden wird durch theApp.TopAlg += ["AnalysisSkeleton"] und AnalysisSkeleton = Algorithm("AnalysisSkeleton") festgelegt
  - Natürlich muss noch bestimmt werden, wieviele Events betrachtet werden sollen. Dies funktioniert durch theApp.EvtMax = 10000
  - Der Rest interessiert uns im Moment noch nicht.

Don’t forget to register your input-files with pool_insertFileToCatalog

- Damit der Athena das jobOption-File und den zugehörigen Quellcode bei diesem Beispiel korrekt ausführen kann muss noch eine Bibliothek hinzugefügt werden
  - theApp.Dlls += ["TruthParticleAlgs"]
- Damit kann Athena gestartet werden: Wir sind im /run/ Verzeichnis:
  - athena.py [jobOptions.py]
- *ACHTUNG: Wir erhalten eine Fehlermeldung!*
- Dies liegt natürlich daran, das wir kein korrektes InputAOD-File angegeben haben. Daher ändern wir im jobOptionFile
  - EventSelector.InputCollections = ["/VERZEICHNIS/Dateiname1.AOD.pool.root", "/VERZEICHNIS/Dateiname2.AOD.pool.root"]
- Bei Athena Versionen vor 10.3.0 müssen die AOD-Files noch in einem XML-Pool-File registriert werden. Wir fügen diese im /run/ Verzeichnis wie folgt hinzu
  - pool_insertFileToCatalog /VERZEICHNIS/Dateiname1.AOD.pool.root
  - pool_insertFileToCatalog /VERZEICHNIS/Dateiname2.AOD.pool.root

We want to start from scratch and therefore delete all unimportant content

- Damit können wir unsere Analyse erneut mit athena.py jobOptions.py starten. Im jobOptions.py-File wurde das Ausgabefile mit HistogramPersistencySvc.OutputFile = "AnalysisSkeleton.root“ festgelegt.
- Nach erfolgreichem Programmabschluss von Athena schauen wir uns das Ausgabe-File an und stellen fest, dass dort nichts brauchbares drinsteht. Dies ist kein Wunder, da unsere Beispiel Analyse "AnalysisSkeleton.cxx“ nicht für unsere Eingabe-Daten geschrieben wurde.
- Um einige Erfahrungen mit Athena zu sammeln, reduzieren wir "AnalysisSkeleton.cxx“ und "AnalysisSkeleton.h“ auf ein Minimum an Funktionalität und beginnen von da an unsere eigene Analyse.

Changes in the Header-File

Änderungen im Header-File: AnalysisSkeleton.h

```
//...oberhalb soll alles unverändert bleiben

class AnalysisSkeleton : public Algorithm
{

 public:

   AnalysisSkeleton(const std::string& name, ISvcLocator* pSvcLocator);
   ~AnalysisSkeleton();

   StatusCode initialize();
   StatusCode finalize();
   StatusCode execute();

 private:

    IAnalysisTools * m_analysisTools;

    StoreGateSvc* m_storeGate;
    std::string m_muonContainerName;

    std::string m_truthParticleContainerName;

};

#endif // ANALYSIS_SKELETON_H
```

Changes in the Source-File

Änderungen im Source-File AnalysisSkeleton.cxx: Konstruktur

```
AnalysisSkeleton::AnalysisSkeleton(const std::string& name,  ISvcLocator* pSvcLocator) :
    Algorithm(name, pSvcLocator),  m_analysisTools(0)
{
}
```

Änderungen im Source-File AnalysisSkeleton.cxx: initialize()

```
StatusCode AnalysisSkeleton::initialize() {

  MsgStream mLog( messageService(), name() );

//…hier bleibt alles

IAlgTool *tmp_analysisTools;
  sc = toolSvc->retrieveTool("AnalysisTools",tmp_analysisTools);
  if (StatusCode::SUCCESS != sc) {
     mLog << MSG::ERROR << "Can't get handle on analysis tools" << endreq;
     return StatusCode::FAILURE;
  }
  m_analysisTools=dynamic_cast<IAnalysisTools *>(tmp_analysisTools);

//…Entfernen von allen histoSvc()->book Operationen

return StatusCode::SUCCESS;
}
```

Out first Athena Error

Änderungen im Source-File AnalysisSkeleton.cxx: execute()

```
StatusCode AnalysisSkeleton::execute()
{
    MsgStream mLog( messageService(), name() );
    mLog << MSG::DEBUG << "execute()" << endreq;
    StatusCode sc = StatusCode::SUCCESS;

    mLog << MSG::INFO << "Atlas funktioniert nicht, wir haben keine Daten" <<   endreq;

    return StatusCode::SUCCESS;
}
```

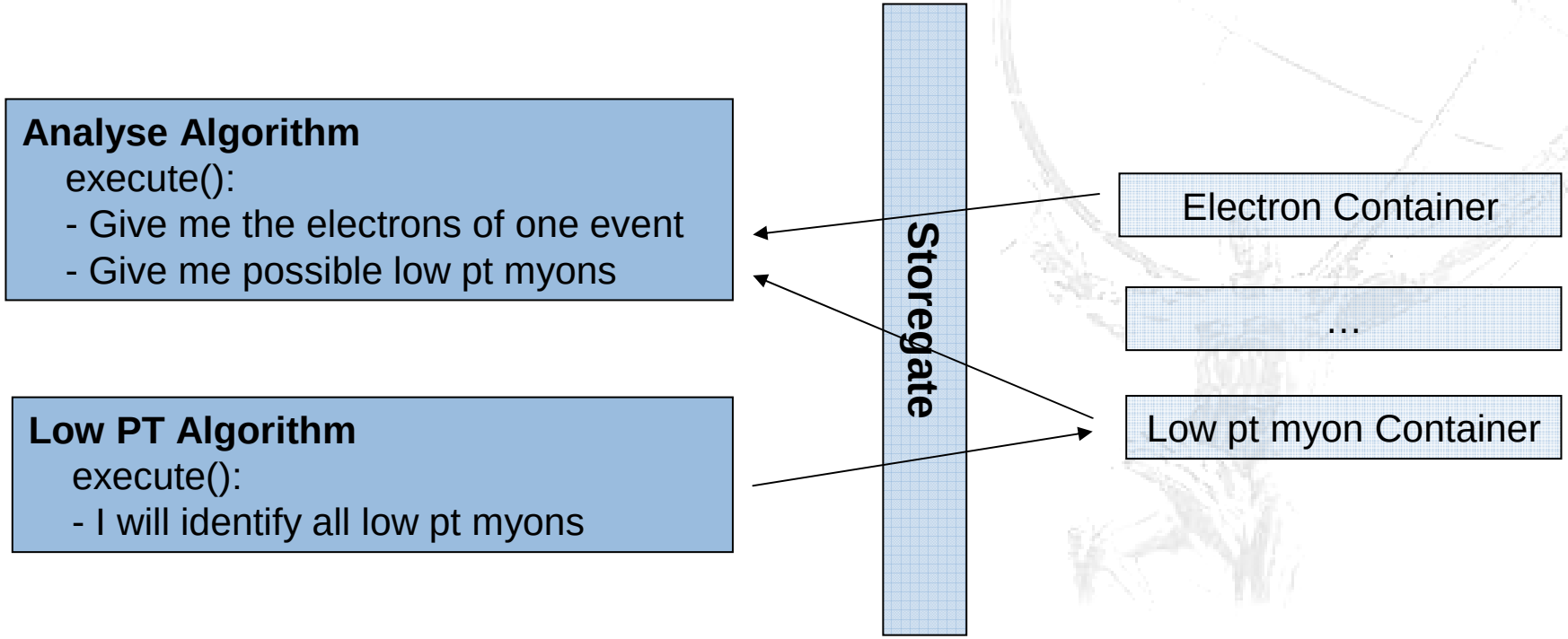
- Nun können wir testen, was unser Algorithmus macht
  - cd ../cmt
  - cmt br gmake
  - cd ../run
  - athena.py jobOptions.py

- Wir haben (hoffentlich) unseren ersten Athena Fehler, den wir selbst lösen

For our purposes, it is not important how Storegate is working

- Wir wollen die pt-Verteilung der Myonen abfragen. Woher bekommt man diese Informationen?

- Das Storegate-Prinzip



We want to access the myons

Änderungen im Source-File AnalysisSkeleton.cxx: execute()

- Abfragen des Muon-Containers

```
StatusCode AnalysisSkeleton::execute()
{
    MsgStream mLog( messageService(), name() );
    mLog << MSG::DEBUG << "execute()" << endreq;
    StatusCode sc = StatusCode::SUCCESS;

    const MuonContainer* muonTDS=0;
    sc=m_storeGate->retrieve( muonTDS, "MuonCollection");
    if( sc.isFailure()  ||  !muonTDS )
    {
            mLog << MSG::WARNING << "No AOD muon container of muons found in
    TDS"<< endreq;
            return StatusCode::SUCCESS;
    }
    mLog << MSG::DEBUG << "MuonContainer successfully retrieved" << endreq;

    return StatusCode::SUCCESS;
}
```

We want to get the PT of all reconstructed myons

Änderungen im Source-File AnalysisSkeleton.cxx: execute()

- Iteration über alle rekonstruierten Myonen pro Event im Container
- Speichern des PT der Myonen in einem Histogramm

```
StatusCode AnalysisSkeleton::execute()
{
    // alles wie bisher
    mLog << MSG::DEBUG << "MuonContainer successfully retrieved" << endreq;
    MuonContainer::const_iterator muonItr  = muonTDS->begin();
    MuonContainer::const_iterator muonItrE = muonTDS->end();
    for (; muonItr != muonItrE; ++muonItr)
    {
            m_h_muon_pt->fill( (*muonItr)->pt(), 1.);
    }
    return StatusCode::SUCCESS;
}
```

Natürlich muss das Histogramm m_h_muon_pt in der initialize()-Funktion initialisiert werden und im Header-File deklariert werden

- AnalysisSkeleton.h: IHistogram1D* m_h_m_muon_pt;
- AnalysisSkeleton.cxx: initialize(): m_h_muon_pt= histoSvc()->book("/stat","Muon PTs","N el",50,0,200.*GeV);

We access the Storegate with m_storeGate->retrieve(…)

- **Was haben wir bisher gemacht?**
  - Der Container aller Myonen wird mit sc=m_storeGate->retrieve(muonTDS, "MuonCollection"); aus dem Storegate geholt.
  - Wir definieren einen Iterator auf diesem Container MuonContainer:: const_iterator muonItr = muonTDS->begin();und interieren über alle Myonen im Container
  - Mit (*muonItr)->pt() greifen wir auf dem Transveralimpuls der Myonen zu
- Nun können wir wieder **testen**, wie unser Algorithmus arbeiten
  - cd ../cmt, cmt br gmake, cd ../run, athena.py jobOptions.py
  - root …
- **Sehr kleine Übung**
  - Es soll ein Histogramm mit der Anzahl der rekonstruierten Myonen pro Event erzeugt werden

This is only one possible implementation.

Änderungen im Source-File AnalysisSkeleton.cxx: execute()

- Überprüfen aller möglichen Kombinationen von Myonen und Schnitte auf die Myon-PTs
- Erzeugen eines kombinierten Teilchens

```
StatusCode AnalysisSkeleton::execute()
{
  // alles wie bisher
  if (nMuons > 1)  {
    AnalysisUtils::Combination<const MuonContainer> comb(muonTDS,2);
    std::vector<const MuonContainer::base_value_type*> muonPair;
    while (comb.get(muonPair))  {
     bool test1 = muonPair[0]->charge() == -(muonPair[1]->charge());
     bool test2 = (muonPair[0]->pt() > m_MuonPtCut) && (muonPair[1]->pt() > m_MuonPtCut);
     if (test1 && test2)   {
                CompositeParticle Zmm;
                Zmm.add(muonPair[0], muonPair[1]);
                Zmm.set_charge(0);
                Zmm.set_pdgId(PDG::Z0);
                double invMass = Zmm.m();
                m_h_Zmass->fill(invMass);
    }
     }
  }
  return StatusCode::SUCCESS;
}
```

Don’t forget to initialize and declare the histograms

- **Was ist die Grundlegende Idee?**
  - Die rekonstruierten Myonen sollten entgegen gesetzte Ladung (->charge()) haben
  - Das pt eines Myons (->pt()) sollte größer als eine vorgegebene Schwelle sein
- **Kombinationen von Myonen und Zusammengesetzte Teilchen**
  - Natürlich könnte man bei dieser Untersuchung einfach über alle im Container befindlichen Myonen iterieren, wie wir es schon gemacht haben. Eine elegantere Möglichkeit ist die Verwendung der Funktion comb.get(muonPair), die bei jedem Aufruf eine neue Kombination von Myonen zurück gibt.
  - Wenn dieses Myonpaar unsere Voraussetzungen erfüllt, dann kreieren wir ein neues Teilchen CompositeParticle Zmm aus den beiden Myonen Zmm.add(muonPair[0], muonPair[1])
  - Natürlich müssen auch hier unsere Histogramme initialisiert und deklariert werden

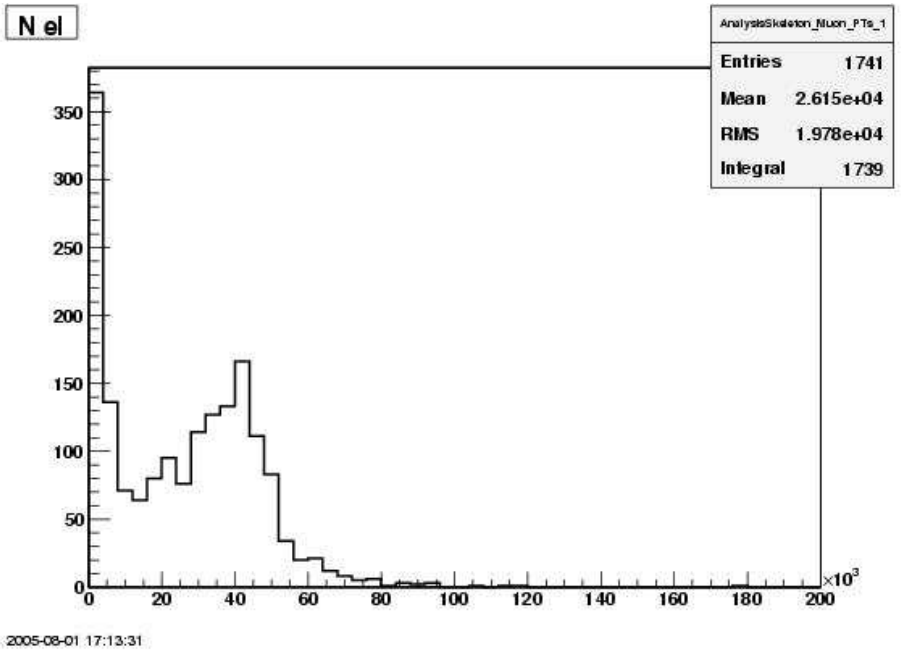
We can make a variable accessible from the jobOptionfile through the method declareProperty()

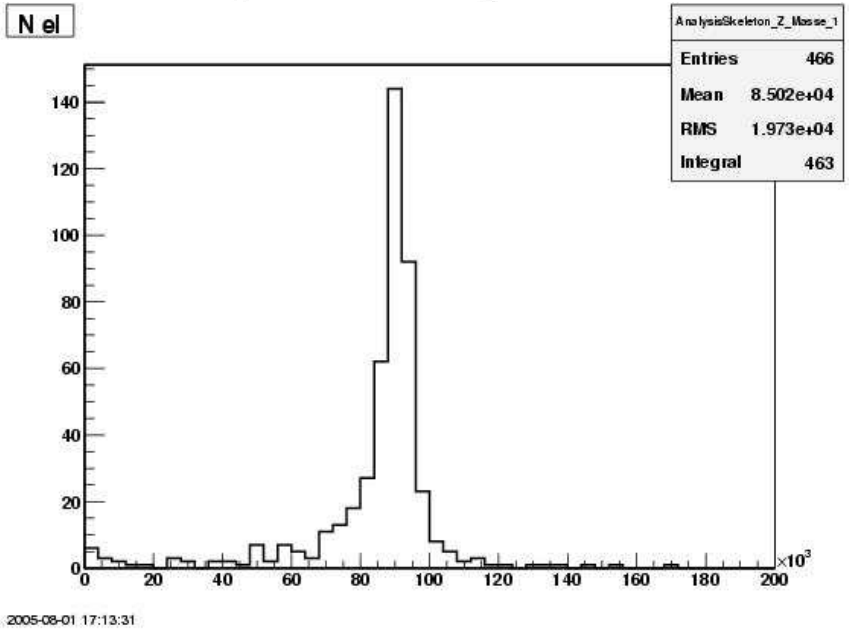
- Wir würden gerne den Cut auf den pt der Myonen erst im JobOptionfile angeben.
  - Dazu definieren wir im Headerfile eine Variable
    double m_MuonPtCut;
  - Im Source-File schreiben wir in der initialize()-Methode
    declareProperty("MuonPtCut", m_MuonPtCut = 10.0*GeV);
  - Durch die Funktion declareProperty() wird die Variable m_MuonPtCut sichtbar für das Joboptionfile gemacht. Der Standardwert von m_MuonPtCut ist 10GeV. Der Zugriff im JobOptionFile erfolgt mit
    AnalysisSkeleton.MuonPtCut = 12.0*GeV

Our first results

The Z-Mass

- Nach dem Kompilieren und ausführen unserer Analyse erhalten wir – mit etwas Glück – die folgenden Histogramme





The access of the Truth-information is analog to the Myons

Getting the resolution (1)

Änderungen im Source-File AnalysisSkeleton.cxx: execute()

- Zunächst benötigen wir wieder den Container, der die True-Informationen der Simulation enthält

```
StatusCode AnalysisSkeleton::execute()
{
  MsgStream mLog( messageService(), name() );
  mLog << MSG::DEBUG << "execute()" << endreq;
  StatusCode sc = StatusCode::SUCCESS;

  const TruthParticleContainer*  mcpartTES = 0;
  sc=m_storeGate->retrieve( mcpartTES,m_“SpclMC“);
  if( sc.isFailure()  ||  !mcpartTES )
  {
    mLog << MSG::WARNING<< "No AOD MC truth particle container found in TDS"<< endreq;
    return StatusCode::SUCCESS;
  }
  mLog <<MSG::DEBUG << "MC Truth Container Successfully Retrieved" << endreq;

  // Alles wie bisher
  // ...
}
```

Only some small changes until the end

Änderungen im Source-File AnalysisSkeleton.cxx: execute()

- Zunächst benötigen wir wieder den Container, der die True-Informationen der Simulation enthält

```
StatusCode AnalysisSkeleton::execute()
{
  // Alles wie bisher
  Int index = -1; double deltaRMatch;  double m_deltaRMatchCut = 0.2;  double m_maxDeltaR = 0.9999;
  MuonContainer::const_iterator muonItr  = muonTDS->begin();
  MuonContainer::const_iterator muonItrE = muonTDS->end();

  for (; muonItr != muonItrE; ++muonItr) {
    m_h_muon_pt->fill( (*muonItr)->pt(), 1.);
    const TruthParticleContainer * truthContainer = mcpartTES;
    bool findAMatch = m_analysisTools->matchR((*muonItr),
                      truthContainer, index, deltaRMatch, (*muonItr)->pdgId());
    if (findAMatch) {
              const TruthParticle*  muonMCMatch = (*mcpartTES)[index];
              double res = (*muonItr)->pt() / muonMCMatch->pt();
              deltaRMatch = (deltaRMatch > m_maxDeltaR) ? m_maxDeltaR : deltaRMatch;
              if ( deltaRMatch < m_deltaRMatchCut) {
                        const TruthParticle*  electronMCMatch = (*mcpartTES)[index];
                        double res = (*muonItr)->pt() / electronMCMatch->pt();
                        m_h_mcmuonPtResolution->fill(res,1.);
              }
    }
  }
  // Alles wie bisher }
```

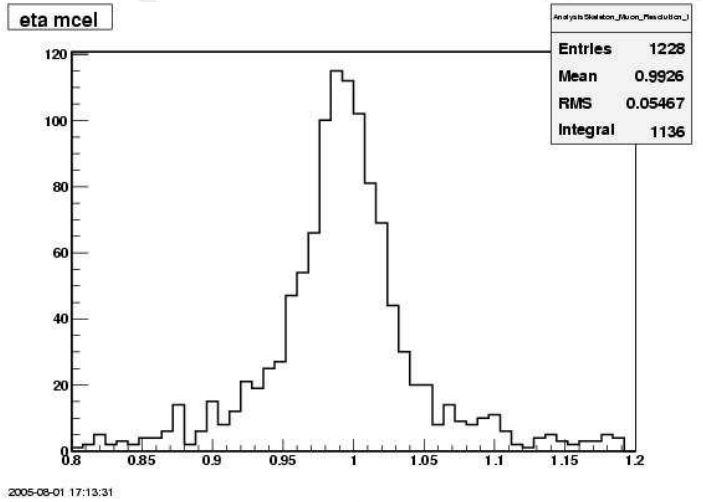
The function m_analysisTools->matchR() finds a candidate-particle that may match to the reconstructed particle

Was haben wir gemacht / Was ist neu?

- Das Anbinden des TrueInformation Containers durch das Storegate erfolgt wie schon bei den Myonen: sc=m_storeGate->retrieve(mcpartTES,m_“SpclMC“);
- Die Funktion m_analysisTools->matchR((*muonItr), truthContainer, index, deltaRMatch, (*muonItr)->pdgId()) überprüft, ob ein rekonstruiertes Myon zu einem Myon im True-Container passend ist und gibt den Abstand mit deltaRMatch zu diesem Myon an
- Wenn deltaRMatch innerhalb unseres Cuts m_deltaRMatchCut liegt, berechnen wir die Auflösung

Was ist das Ergebnis?

- Nach Kompilieren und Ausführen sollten wir nebenstehende Auflösung erhalten:



Sorry for this very small tutorial for creating an own package, but my train arrives now at Munich Main Station…

- Follow the instructions given at https://uimon.cern.ch/twiki/bin/view/Atlas/WorkBookCreatingNewPackage
- Modify our cmt/requirements and run/jobOptions.py according to the chosen name of your package