Histograms, Ntuples, Root and Batch Queues!

Laura Fields

2007 CLEO 101
Outline

- What are Histograms and Ntuples?
- Making Histograms and Ntuples in Suez
  - Simple/Suez Method
  - Root Method
- The EventStore
- Batch Queues
- Running Root
  - Getting Access to your Histograms and Ntuples
  - Turning Ntuples into Histograms
  - Manipulating Histograms
  - Chaining Trees and Looping Over Trees
What are Histograms

Suppose you want to discover a particle:

![Figure: A Histogram from the discovery of $h_c$](image)

Well, okay you might not do that, but whatever you do in particle physics, you will likely do it it with the aid of histograms.

- Histograms allow us to see how our data are distributed
- The x-axis: a variable that you are interested in, divided into bins
- The y-axis: the number of entries that fall into that bin
- A histogram is essentially an array: a list of the number of entries in each bin
What are Ntuples?

- Storing Histograms directly is somewhat limiting, so we turn to NTuples
  - essentially an array of arrays, e.g. a list of events with a set of variables associated with each event
- Storing Ntuples allows us to:
  - Make histograms of a variable and easily change the histogram endpoints and binning
  - Make two or three dimensional histograms
  - Make histograms of a certain subset of events

Laura Fields

Histograms, Ntuples, Root and Batch Queues

4 / 37
You will start with **Analysis Quality Data** (also known as “physics grade” in the EventStore)

- Other people have taken CLEO Raw Data and formed basic tracks and showers for you

You want to find events with basic qualities that you want in the data, using **Suez**

- Make a ROOT file containing Histograms or Ntuples

Use ROOT to analyze and fit the data in your histograms and Ntuples
The Two Ways of Making Root Files

- Suez was originally designed to work with older ROOT alternatives called PAW and MN_FIT
  - Not C++ based
  - Variables in histograms and ntuples have to be floating point variables

- There are two ways of making Suez create root files
  - Simply tell Suez to make a ROOT file rather than a PAW/MN_FIT file.
    - This means all of your variables must be of type “float”
  - Use ROOT within Suez to create files
    - Now the variables in your ntuples can be classes. Woohoo!

- All of the following examples can be found in my example processor Cleo101Day2, which you can get from the Cleo101 Directory or /home/ljf26/workshop/Cleo101Day2
Simple Histograms in Suez

Declare Histogram in your Processor’s .h file

```cpp
#include "HistogramInterface/HistogramPackage.h"
...
HIHist1D *m_trackMom;
```

Define Histogram in the hist_book() method of your Processor

```cpp
void RootExampleProc::hist_book( HIHistoManager& iH )
{
    m_trackMom = iHistoManager.histogram("track momentum (GeV)",50, 0, 1.5);
}
```

Fill Histogram in event() method of your Processor

```cpp
// fill a "simple histogram" with track momentum
m_trackMom->fill(momentum);
```

Load Conversion Module in .tcl script

```tcl
module sel RootHistogramModule
root file root_suez_style_example_data31.root
root init
```
Declare Ntuple in your Processor’s .h file

```cpp
#include "HistogramInterface/HistogramPackage.h"
#include "HistogramInterface/HINtupleVarNames.h"
...
HINtuple *m_trackTuple;
```

Make an enum of variable names

```cpp
enum{knhe,
    khit_fraction,
kcos_theta,
kmomentum,
kpx,
kpy,
kpz,
kd0,
kz0,
knum_vars};
```
Simple Ntuples in Suez

Define Ntuple in hist_book()

// make an object that holds the names of the variables in your ntuple
HINtupleVarNames knames(knum_vars);
knames.addVar(knhe,"nhe");
knames.addVar(khit_fraction,"hitfrac");
knames.addVar(kcos_theta,"costheta");
knames.addVar(kmomentum,"momentum");
knames.addVar(kpx,"px");
knames.addVar(kpy,"py");
knames.addVar(kpz,"pz");
knames.addVar(kd0,"d0");
knames.addVar(kz0,"z0");
m_trackTuple = iHistoManager.ntuple(1,"tracktuple",knum_vars,50000,knames.names());

Fill Ntuple in event() method

float trackTuple[knum_vars];
trackTuple[knhe] = num_hits_expected;
trackTuple[khit_fraction] = hit_fraction;
trackTuple[kcos_theta] = cos_theta;
trackTuple[kmomentum] = momentum;
trackTuple[kpx] = px;
trackTuple[kpy] = py;
trackTuple[kpz] = pz;
trackTuple[kd0] = d0;
trackTuple[kz0] = z0;
m_trackTuple->fill(trackTuple);
That’s it for simple histograms/ntuples. Now we’ll learn how to make Root Histograms/Ntuples directly in Suez.

Root is a collection of C++ classes. Root Classes generally start with “T”:

- Histograms all inherit from TH1:
  - TH1F - a histogram containing floats
  - TH1D - a histogram containing doubles
  - TH2F - a 2-Dimensional histogram of floats
  - ...

- Ntuples are called TTrees in Root
  - The variables in the Trees are stored in objects called TBranch

TTree and TH1 have loads of useful member functions for playing with the contents of the ntuples/histograms. We’ll see some of these later. You can also investigate:

Declare Histograms in your Processor’s .h file

```c++
#include "TFile.h"
#include "TH1F.h"
...
TH1F *m_hist;
TFile *m_file;
```

Initialize Histogram in init() method of Processor

```c++
m_file = new TFile("Cleo101Day2_roothist.root","RECREATE");
m_hist = new TH1F("h1","track momentum",50,0,1.5);
```

Fill in event() method

```c++
m_hist->Fill(momentum);
```

Write and Close in terminate() method

```c++
m_file->Write();
m_file->Close();
```
To use Root within Suez, you have to make sure all of your libraries are linked correctly in your MakeFile. See

- see /home/ljf26/workshop/Cleo101Day2/Makefile for an example
- these are the things I had to change:

### Changes to your Makefile

```
OTHRLIB:= $(C3_OTHER)/lib -->
    OTHRLIB:= $(C3_OTHER)/lib $(shell exec $(C3_OTHER)/Root/bin/root-config --libdir)

#OTHR_INCS :=$(C3_OTHER) -->
    OTHR_INCS :=$(C3_OTHER)/Root/include

USER_LFLAGS:= $(XTRA_LFLAGS) -->
    USER_LFLAGS:= $(XTRA_LFLAGS) $(shell exec $(C3_OTHER)/Root/bin/root-config --libs)
```
There are different ways of filling Root TTrees from within Suez.
This is what we will do:
- Write a class that holds the variables that will form the tree (ExampleNtuple)
- Write another class that declares tree branches, fills the tree, and writes the tree to a file (FillExampleTree)
- Use the processor to access data and fill tree via the helper classes

For more detailed examples, see last year’s talk:
class ExampleNtuple 
{
    public:
    // -- constants, enums and typedefs
    enum {MAXNTRK = 100};

    // -- Constructors and Destructor --
    ExampleNtuple();
    virtual ~ExampleNtuple();

    // Track data
    int ntrk;
    float nhe[MAXNTRK];
    float hit_fraction[MAXNTRK];
    float cos_theta[MAXNTRK];
    float momentum[MAXNTRK];
    float px[MAXNTRK];
    float py[MAXNTRK];
    float pz[MAXNTRK];
    float d0[MAXNTRK];
    float z0[MAXNTRK];

    // ----------- member functions ---------
    void init();

    // ----------- const member functions ---
    ...

    // constructors and destructor
    ExampleNtuple::ExampleNtuple()
    {
        init();
    }

    ExampleNtuple::~ExampleNtuple()
    {
    }

    // member functions
    void ExampleNtuple::init()
    {
        // initialize all Ntuple elements
        ntrk = -999;
        for(int i = 0; i<MAXNTRK; i++) {
            nhe[i] = -999;
            hit_fraction[i] = -999;
            cos_theta[i] = -999;
            momentum[i] = -999;
            px[i] = -999;
            py[i] = -999;
            pz[i] = -999;
            d0[i] = -999;
            z0[i] = -999;
        }
    }
```cpp
void FillExampleTree::init(std::string filename, ExampleNtuple* ntuplePtr) {
    m_ntuple = ntuplePtr;
    m_file = new TFile( filename.c_str(), "RECREATE" );
    m_tree = new TTree("trkinfo","Root Example Ntuple",1);
    m_tree->Branch("ntrk", &m_ntuple->ntrk, "ntrk/I");
    m_tree->Branch("nhe", &m_ntuple->nhe, "nhe/F");
    m_tree->Branch("hit_fraction", &m_ntuple->hit_fraction, "hit_fraction/F");
    m_tree->Branch("cos_theta", &m_ntuple->cos_theta, "beamE/F");
    m_tree->Branch("momentum", &m_ntuple->momentum, "momentum/F");
    m_tree->Branch("px", &m_ntuple->px, "px/F");
    m_tree->Branch("py", &m_ntuple->py, "py/F");
    m_tree->Branch("pz", &m_ntuple->pz, "pz/F");
    m_tree->Branch("d0", &m_ntuple->px, "d0/F");
    m_tree->Branch("z0", &m_ntuple->px, "z0/F");
    m_tree->Print();
}

void FillExampleTree::fill() {
    m_tree->Fill();
}

void FillExampleTree::finalize() {
    m_file->Write();
    m_file->Close();
}
```
In your Processor's .h file

```c
#include "Cleo101Day2/FillExampleTree.h"
...
ExampleNtuple m_exampleNtuple;
FillExampleTree m_fillExample;
```

In your Processor's init() method

```c
m_fillExample.init( "/home/ljf26/workshop/Cleo101Day2/Cleo101Day2_rootntuple.root",
&m_exampleNtuple );
```

In your Processor's terminate() method

```c
// Finalize the Root Ntuple
m_fillExample.finalize();
```
Fill Tree in event() method

m_exampleNtuple.init();

int i = 0;

FATable< NavTrack >::const_iterator trackTableBegin = trackTable.begin();
FATable< NavTrack >::const_iterator trackTableEnd = trackTable.end();
// Loop over tracks.
for ( FATable< NavTrack >::const_iterator trackItr = trackTableBegin;
    trackItr != trackTableEnd ;
    ++trackItr )
{
    ...
    // fill a root tree with track variables
    // Bail if we have too many tracks
    if( i > 99 ) continue;
    m_exampleNtuple.ntrk = trackTable.size();
    m_exampleNtuple.nhe[i] = num_hits_expected;
    m_exampleNtuple.hit_fraction[i] = hit_fraction;
    m_exampleNtuple.cos_theta[i] = cos_theta;
    m_exampleNtuple.momentum[i] = momentum;
    m_exampleNtuple.px[i] = px;
    m_exampleNtuple.py[i] = py;
    m_exampleNtuple.pz[i] = pz;
    m_exampleNtuple.d0[i] = d0;
    m_exampleNtuple.z0[i] = z0;
    i++;
}
m_fillExample.fill();
Our data is stored in the EventStore
Access it at runtime from the tcl script’:

```tcl
module sel EventStoreModule
eventstore in 20070501 dtag dataset 31
```

dtag gives analysis quality data that has been dskimmed: it has at least one set of particles consistent with a hadronic D decay
20070501: Timestamp - different timestamps may have different versions of the data, but if you stick with the same timestamp, you should always get the same version

Much more info about the EventStore is available at: https://www.lepp.cornell.edu/restricted/CLEO/CLEO3/soft/howto/howto_UseEventStore.html
Batch Queues

- Short jobs can be run interactively from a terminal window.
- For longer jobs, you’ll want to use the **Batch Queues**:
  - Submit shell scripts from Linux machines via:
    
    ```
    qsub -l arch=lx24-x86 your_script_name
    ```
  - See `/home/ljf26/workshop/Cleo101Day2/Cleo101Day2.scr` for an example script.
  - You can set environment variables with “-v”:
    
    ```
    qsub -l arch=lx24-x86 -v VARIABLE=value your_scripts_name
    ```
  - This is particularly useful for long Suez jobs, but you can run ROOT in batch mode too:
    
    ```
    c3root -b -f your_root_script
    ```
To Run Root Interactively, open a terminal and type ‘c3root’
- This gives you the version of root associated with the software release you are using
- Use 20070330_FULL for Cleo101 Exercises

You can also create a script of root commands, all enclosed in {} and run it via

```
.x myRootScript.C
```

This interprets your script line-by-line

There are a ways few running scripts. See the “CINT the C++ Interpreter” section of the ROOT user’s guide:

You can also compile your scripts - often safer and faster!
Getting Access to your histograms

Run root from the directory containing your suez histograms:

```
[ljf26@lnx6223 Cleo101Day2]$ c3root -l
root [0] TFile f1(``Cleo101Day2_simple.root’’);
root [1] f1.ls();
TFile** Cleo101Day2_simple.root
TFile* Cleo101Day2_simple.root
KEY: TDirectory Cleo101Day2;1 Cleo101Day2
root [2] TH1F* hist1 = (TH1F*)f1.Get(``Cleo101Day2/h1’’);
root [3] hist1->SetFillColor(4);
```

In this case, the histograms are in a directory with the same name as your processor, not the case for Root-style histograms:

```
[ljf26@lnx6223 Cleo101Day2]$ c3root -l
root [0] TFile f2(``Cleo101Day2_roothist.root’’)
root [1] f2.ls();
TFile** Cleo101Day2_roothist.root
TFile* Cleo101Day2_roothist.root
KEY: TH1F h1;1 track momentum
root [2] TH1F* hist2 = (TH1F*)f2.Get(``h1’’);
root [3] hist2->SetFillColor(4);
root [4] hist2->Draw();
```
Getting Access to your histograms

The result: in either case!

track momentum (GeV)

<table>
<thead>
<tr>
<th>h1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entries: 27739</td>
</tr>
<tr>
<td>Mean: 0.3964</td>
</tr>
<tr>
<td>RMS: 0.2619</td>
</tr>
</tbody>
</table>
Getting Access to your Ntuples

- Run root from the directory containing your .root file:

  ```
  root [1] TFile f1("Cleo101Day2_simple.root");
  root [2] f1->ls();
  TFile** Cleo101Day2_simple.root
  TFile* Cleo101Day2_simple.root
  KEY: TDirectory Cleo101Day2;1 Cleo101Day2
  ```

- And for root-style ntuples:

  ```
  root [0] TFile f2("Cleo101Day2_rootntuple.root");
  root [1] f2.ls();
  TFile** Cleo101Day2_rootntuple.root
  TFile* Cleo101Day2_rootntuple.root
  KEY: TTree trkinfo;1 Root Example Ntuple
  root [2] TTree* tree2 = (TTree*)f2.Get("trkinfo");
  root [3] tree2->Draw("momentum");
  ```
The result:

Not so useful!
To Manipulate the way your Ntuples appear, put them into Histograms. Let’s do it with a script:

```c++
{
// Example script to put info from an
// Ntuple into a Histogram run with
// .x NtupleToHistogram.C

// Get our ntuple
TFile f1(“Cleo101Day2_simple.root”);
TTree* tree1 =
    (TTree*)f1.Get(“Cleo101Day2/nt1”);

// Define a histogram
TH1F *hist =
    new TH1F(“khist”, “Track Momentum”,
            50, 0, 1.5);

// Draw variable from ntuple using >>
tree1->Draw(“momentum>>khist”);
hist->SetFillColor(4);
hist->Draw();
}
```

Histograms, Ntuples, Root and Batch Queues!
Manipulating Histograms: Changing Colors and Lines

ColorLineFill.C

```c
{ // Make a canvas to draw on
  mycanvas =
    new TCanvas("mycanvas","My Canvas",0,0,500,750);
  mycanvas->Divide(1,3);
  mycanvas->cd(1);

  // Get our ntuple and make a histogram
  TFile f1("Cleo101Day2_simple.root");
  TTree* tree1 =
    (TTree*)f1.Get("Cleo101Day2/nt1");
  TH1F *hist1 =
    new TH1F("khist1","Track Momentum",50,0, 1.5);
  tree1->Draw("momentum>>khist1");
  hist1->DrawClone("e"); // draw error bars
  mycanvas->cd(2);
  hist1->SetFillColor(8);
  hist1->SetFillStyle(3001);
  hist1->DrawClone();
  mycanvas->cd(3);
  hist1->SetFillStyle(1001);
  hist1->SetLineColor(2);
  hist1->SetFillColor(4);
  hist1->SetLineWidth(3);
  hist1->DrawClone();
}
```
AxesTitleStat.C

void main()
{
  // Make a canvas to draw on
  mycanvas = new TCanvas("mycanvas","",0,0,500,400);

  TFile f1("Cleo101Day2_simple.root");
  TTree* treel =
    (TTree*)f1.Get("Cleo101Day2/nt1");
  TH1F *hist1 =
    new TH1F("khist1","Track Momentum",50,0, 1.5);
  treel->Draw("momentum>>khist1");

  // Change the title
  hist1->SetTitle("Titles can have latex style
    characters: \(\pi\), \(\alpha_{\beta}\), \(\rightarrow\), etc");

  // Fool around with the Axes
  hist1->GetXaxis()->SetLabelSize(0.03);
  hist1->GetXaxis()->SetTitle("P_{\text{Track}} \ (GeV)"낱
  hist1->GetXaxis()->SetTitleSize(0.04);
  hist1->GetXaxis()->CenterTitle();
  hist1->GetYaxis()->SetLabelSize(0.03);
  hist1->GetYaxis()->SetTitle("Num Tracks per 30 MeV"");
  hist1->GetYaxis()->SetTitleSize(0.04);
  hist1->GetYaxis()->CenterTitle();
  hist1->SetFillColor(4);
  hist1->SetStats(kFALSE); // turn off stats box
  gPad->SetLogy(kTRUE); // make y-axis log scale
}

Titles can have latex style characters: \(\pi\), \(\alpha_{\beta}\), \(\rightarrow\), etc

Number of Tracks per 30 MeV

0 0.2 0.4 0.6 0.8 1 1.2 1.4

P_{\text{Track}} \ (GeV)
Manipulating Histograms: 2D Plots

2DPlots.C

{ // Make a canvas to draw on
mycanvas = new TCanvas("mycanvas","",0,0,500,700);
mycanvas->Divide(1,3);
mycanvas->cd(1);

// Get our ntuple and make a histogram
TFile f1("Cleo101Day2_simple.root");
TTree* tree1 = (TTree*)f1.Get("Cleo101Day2/nt1");
TH2F *scatter = new TH2F( "h2", "d0 vs z0, scatter",
 20, -0.05, 0.05, 20, -0.005, 0.005 );
tree1.Draw( "d0:z0 >> h2" );
scatter->DrawClone();
mycanvas->cd(2);
scatter->SetTitle("d0 vs. z0, box");
scatter->SetFillColor(30);
scatter->DrawClone("box");
mycanvas->cd(3);
scatter->SetTitle("d0 vs. z0, lego1");
scatter->SetFillColor(30);
scatter->DrawClone("lego1");
}

Track d0 vs z0 (m), scatter
Entries 27739
Mean x -0.001003
Mean y -2.249e-05
RMS x 0.01033
RMS y 0.001058
Track d0 vs z0 (m), box
Entries 27739
Mean x -0.001003
Mean y -2.249e-05
RMS x 0.01033
RMS y 0.001058
Track d0 vs z0 (m), lego1
Entries 27739
Mean x -0.001003
Mean y -2.249e-05
RMS x 0.01033
RMS y 0.001058

Laura Fields ()
Histograms, Ntuples, Root and Batch Queues
Manipulating Histograms: Cuts!

```c
// Make a canvas to draw on
mycanvas = new TCanvas("mycanvas","",0,0,500,600);
mycanvas->Divide(1,2);
mycanvas->cd(1);

// Get our ntuple and make a histogram
TFile f1("Cleo101Day2_simple.root");
TTree* treel =
   (TTree*)f1.Get("Cleo101Day2/nt1");
TH1F *hist1 = new TH1F("h1", "momentum, no cuts", 50,0,1.5);
treel.Draw( "momentum >> h1" );
hist1->SetFillColor(4);
hist1->Draw();

mycanvas->cd(2);

TH1F *hist2 = new TH1F("h2", "momentum, |d0|<.002", 50,0,1.5);
treel.Draw( "momentum >> h2", "abs(d0)<.002" );
hist2->SetFillColor(4);
hist2->Draw();
```

Histograms, Ntuples, Root and Batch Queues!
// Get our ntuple and make a histogram
TFile f1("Cleo101Day2_simple.root");
TTree* tree1 =
    (TTree*)f1.Get("Cleo101Day2/nt1");
TH1F *hist1 = new TH1F( "h1","momentum, |d0|<.002", 50,0,1.5);
tree1.Draw( "momentum >> h1","TMath::Abs(d0)<.002");
TH1F *hist2 = new TH1F( "h2","momentum, |d0|>=.002", 50, 0, 1.5);
tree1.Draw( "momentum >> h2","TMath::Abs(d0)>=.002");

// overlay
hist1->SetFillStyle(3001);hist2->SetFillStyle(3001);
hist1->SetFillColor(2); hist2->SetFillColor(4);
hist2->Draw();
hist1->Draw("same");

mycanvas->cd(2);

//stack
THStack *mystack = new THStack("mystack","momentum");
mystack->Add(hist1);
mystack->Add(hist2);
mystack->Draw();

// legend
TLegend *leg = new TLegend(.65,.5,.85,.8);
leg->AddEntry(hist1,"|d0|<.002", "f");
leg->AddEntry(hist2,"|d0|>=.002", "f");
leg->Draw();
Manipulating Histograms: Functions of Variables

Functions.C

{
   // Make a canvas to draw on
   mycanvas = new TCanvas("mycanvas","",0,0,500,600);
   mycanvas->Divide(1,2);
   mycanvas->cd(1);

   // Get our ntuple and make a histogram
   TFile f1("Cleo101Day2_simple.root");
   TTree* tree1 =
      (TTree*)f1.Get("Cleo101Day2/nt1");
   TH1F *hist1 = new TH1F( "h1", "pz/p",
      50,-1,1);
   tree1.Draw("pz/momentum >> h1");
   hist1->SetFillColor(30);
   hist1->Draw();

   mycanvas->cd(2);
   TH1F *hist2 = new TH1F( "h2", "acos(pz/p)",
      50,0,3.15);
   tree1.Draw("TMath::ACos(pz/momentum) >> h2");
   hist2->SetFillColor(30);
   hist2->Draw();

   mycanvas->Print("rootexample.eps","eps");
}
Chaining and Looping over Trees

- You’ll probably want to make plots using data from several .root files
  - e.g. separate files for data31, data32, etc.

```c
TChain mydata("MyProcName/nt1");
mydata.Add("31.root");
mydata.Add("32.root");
mydata.Add("33.root");
mydata.Add("35.root");
mydata.Add("36.root");
mydata.Add("37.root");
```

- TChain inherits from TTree, so use it like you would TTree
- You might also want to loop over all of the entries in a Tree
  - Use the MakeClass method of TTree
  - See http://root.cern.ch/root/html/TTree.html#TTree:MakeClass
Contextual Menus

Menus provide interface to member functions of objects
What You Should Know how to Do

- Make Histograms and Ntuples in Suez
- Use the EventStore
- Use the Batch Queues
- Load Histograms and Ntuples in Root
- Run Scripts in Root
- Make and Manipulate Histograms in Root
- Chain Trees together and Loop over Trees
Resources

- The Official Root Documentation: http://root.cern.ch
    - “histograms” section has lots of info on drawing options
    - “Graphics and GUI” section has lists of the numbers you can set
color, fillStyle, MarkerStyles etc and what they mean
  - Also: HowTo’s, Tutorials
- Previous Years Cleo101:
  - http://www.lns.cornell.edu/public/CLEO/CLEO101/
- Matt Shepherd’s Lunch Talk:
- My example Processor, tcl and root scripits:
  - /home/ljf26/workshop/Cleo101Day2/
Exercises

- Make a simple processor that looks for showers (extract showers like tracks with NavTrack->NavShower)
  - Fill a histogram with shower energy
  - Fill an ntuple with the following shower parameters, which can be accessed via the attributes() method of NavShower: e9oe25, e9oe25Unf, goodBarrel, goodEndCap, energy, hot, phi, theta, distance, and any other attributes you think might be interested. See a list at:
  - Also include the noTrackMatch() member of NavShower, which tells you whether the shower is matched to a track.
- Run this processor over your favorite dataset, running over 5,000 events.
- Try doing this interactively and using the batch queues.
Exercises

- Use root to look at your histogram.
  - Play with fill/color attributes
  - Try saving the histogram as a .eps file using the interactive GUI
  - Try using the “p” histogram draw option,
  - Play with the marker using the SetMarkerStyle() and SetMarkerSize() members of TH1F.

- Using your ntuple
  - Plot shower energy for all showers, then for showers that aren’t from hot crystals. You’ll probably want to make this cut for the rest of the exercise too.
  - Make separate histograms of e9oe25 for matched and unmatched showers, then plot them stacked. e9oe25 is the ratio of energy in the innermost 9 crystals to the innermost 25 crystals of the shower. What conclusions can you draw from these plots?
  - Compare e9oe25 and e9oe25Unf.
  - Compare various attributes for goodBarrel and goodEndCap showers.