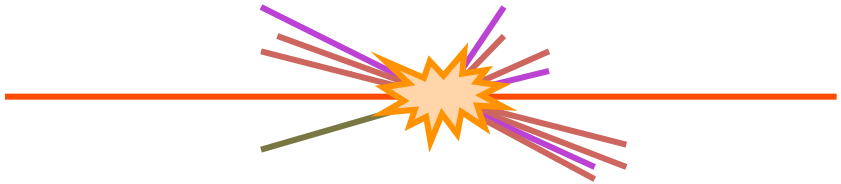
ROOT – lecture 2

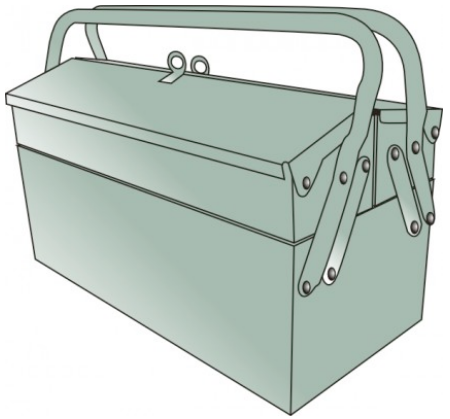


W. H. Bell
Université de Genève

I. van Vulpen
UvA/Nikhef

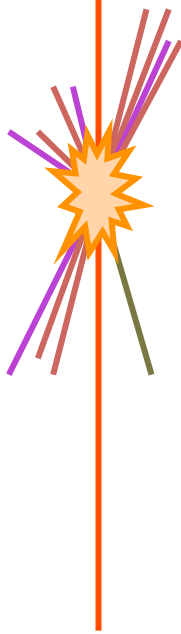
Overview

- Philosophy
- Other usages
  - Compiling against ROOT
  - Python interface
- Data storage (TTree, TFile)
  - Additional STL instantiations
  - New classes
- Quick analysis with TTree::Draw()
- Debugging
- Other packages
  - RooFit, TMVA, RooUnfold



C++ overview, not just for ROOT: http://wbell.web.cern.ch/wbell/HepCppIntro/

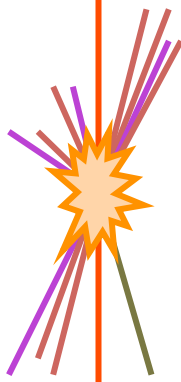
Data analysis philosophy



- Remaking summary data + MC (80TBytes)
  - 1 week running on the GRID, including failures
- Remaking mini-ntuples (~600GBytes)
  - 12 hrs on a pbs system
- Remaking histograms
  - 4hrs on a multicore PC

- Try to allow common tasks later in chain.
  - Minimise total time integral, but allow some flexibility

Compiling against ROOT



- Compiled code is:
  - Faster than CINT and Python
  - More robust than ACLiC
  - Slightly easier to debug
- Setup for compilation:
  - Require all header files are included
  - Need to specify include/ and lib/ directories and library list.

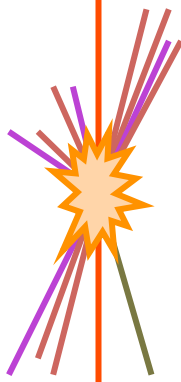
Compiling against ROOT

Simple Makefile, where main() is in writeHistogram.cxx

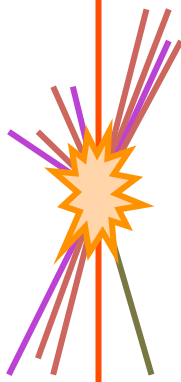
```
CC=g++
TARGET=writeHistogram
OBJECTS=writeHistogram.o

INCFLAGS = -I$(shell root-config --incdir) -O
ROOTLIBS = $(shell root-config --libs)

LIBS = $(ROOTLIBS) -ldl

$(TARGET): $(OBJECTS)
    @echo "** Linking Executable"
    $(CC) $(OBJECTS) $(LIBS) -o $(TARGET)

clean:
    @rm -f *.o *~

veryclean: clean
    @rm -f $(TARGET)

%.o: %.cxx
    @echo "** Compiling C++ Source"
    $(CC) -c $(INCFLAGS) $<
```

ROOT include directory

ROOT library directory and libraries



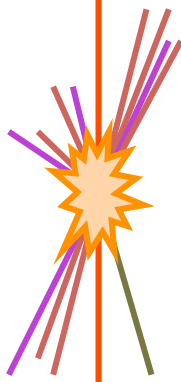
Python interface



- The Python interface relies on ROOT dictionaries – similar to CINT library use.
- Python interface provides:
  - Access to all ROOT classes
  - Direct access to TTree branches, but access is slow.
  - Access to other C++ classes via dictionary generation.

Python interface

```
import ROOT
from ROOT import TH1F,TFile

h = TH1F("h_name",";x-axis;y-axis",11,-0.5,10.5)
h.Fill(2.1,1.0)
h.Fill(3.3,1.0)
h.Fill(1.9,1.0)

f = TFile.Open("python_hist.root","RECREATE")
if f == None:
  exit(1)
f.WriteTObject(h)
f.Close()
```



Null C++ pointers are expressed as None.
This is mostly true except some cases, e.g. when a null pointer is passed as an argument.

TTree – arrays and objects

- TTrees can be used to store arrays, vectors or other objects.
- Arrays
  - Dynamic array length within output ROOT file
  - Constant length within C++ code
  - Faster data access than vectors or other objects
  - No need to generate dictionaries

TTree arrays

Writing

```
Int_t array_n;
Float_t array[100];

TFile *file = TFile::Open("tree.root","RECREATE");
TTree *tree = new Ttree("tree","Test tree");

tree->Branch("array_n",&array_n,"array_n/I");
tree->Branch("array",&array,"array[array_n]/F");
```

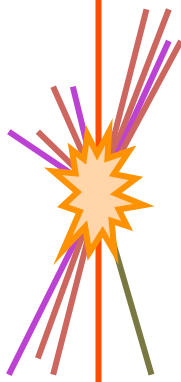
Then call TTree Fill() as normal.

Reading

```
Int_t array_n;
Float_t array[100];

TFile *file = TFile::Open("tree.root");
TTree *tree = (TTree*)file->Get("tree");
tree->SetBranchAddress("array_n",&array_n);
tree->SetBranchAddress("array",array);
```

Once branches are associated call GetEntry as normal.

TTrees - vectors



- STL vectors can be stored in TTrees or directly within a TFile.
- Writing – Objects must be created on the stack.
- Reading – Pass null pointer.
- Storage of non-ROOT classes requires dictionary generation.
  - The dictionaries for some STL instances are already present in ROOT.
  - Other dictionaries can be generated using rootcint.

TTrees - vectors

Writing

```
int particle_n = 0; // Number of particles
std::vector<float>* particle_px = new std::vector<float>();
tree->Branch("particle_n", &m_particle_n,"particle_n/I");
tree->Branch("particle_px", &particle_px);
```

Assuming TFile and TTree instantiation beforehand.
Then call TTree Fill() as normal. Objects associated with a TTree must not be deleted.

Reading

```
int particle_n = 0; // Number of particles
std::vector<float>* particle_px = 0; // Null pointer
tree->SetBranchAddress("particle_n", &particle_n);
tree->SetBranchAddress("particle_px", &particle_px);
```

Assuming TFile and TTree instantiation beforehand.
Then call TTree GetEntry() as normal. Pointers associated with a TTree must be null.

When dictionaries are needed

```
Warning in <TClass::TClass>: no dictionary for class Particle is available
```

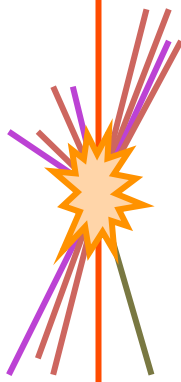
When a file is opened before the dictionary is loaded, ROOT will produce a warning.

To access this class, a dictionary is needed.
This is the case for some nested STL types and for any additional objects.

TTrees – other STL types

Need to generate dictionaries to write or read other STL types.
Dictionaries are generated from files with the suffix LinkDef.h

```
#ifdef __MAKECINT__
#pragma extra_include "vector";
#pragma link C++ class std::vector<vector<int> >+;
#pragma link C++ class std::vector<vector<float> >+;
#pragma link C++ class std::vector<vector<unsigned int> >+;
#pragma link C++ class vector<vector<string> >+;
#endif
```



Then generate dictionary with rootcint.
The dictionary can be linked into an executable or added to a shared library.
Warning: shared library production is not the same on LINUX and OSX

LINUX version

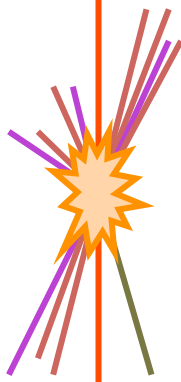
```
rootcint -f dict.cxx -c LinkDef.h
g++ -fPIC -c dict.cxx -I$(root-config --incdir)
g++ -shared dict.o -o libMyStlDict.so
```

Use $ROOTSYS/etc/Makefile.arch within Makefile to pickup correct options for OSX etc..
(version of ROOT before 5.32 use $ROOTSYS/test/Makefile.arch)

TTrees – other STL types

Can then load the definitions into CINT,

```
gSystem->Load("libMyStlDict.so")
```



or Python,

```
import ROOT
from ROOT import gSystem
gSystem.Load("libMyStlDict.so")
from ROOT import
```

The dictionaries can then be used in Python, to define objects.

```
ff = std.vector(std.vector(float))
ss = std.vector(std.vector(std.string))
```

TTrees – other objects

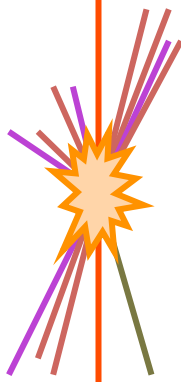
The declaration of a simple C++ class

```
#ifndef PARTICLE_H
#define PARTICLE_H

class Particle {
 public:
  Particle(float px_=0., float py_=0., float pz_=0., float E_=0);
  float px;
  float py;
  float pz;
  float E;
};

#endif
```

The implementation of the C++ class

```
#include "Particle/Particle.h"

Particle::Particle(float px_, float py_, float pz_, float E_):
 px(px_), py(py_), pz(pz_), E(E_){
}
```

TTrees – other objects



```
#include "TFile.h"
#include "TTree.h"
#include "Particle/Particle.h"

int main() {
  TFile *f = TFile::Open("particles.root","RECREATE");
  Particle *p = new Particle(10.,5.,15.,18.71);
  TTree *t = new TTree("tree","test");
  t->Branch("particle",&p);
  t->Fill(); // Once per event
  t->Write(); // At the end, flush the data to a file
  f->Close();
  return 0;
}
```

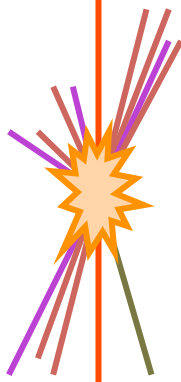
Other objects are attached as TTree branches in the same way as vectors.
Call Fill() for each entry, where the value of the Particle object has been modified

TTrees – other objects

```
#ifdef __CINT__
#include "Particle/Particle.h"
#pragma link off all globals;
#pragma link off all classes;
#pragma link off all functions;
#pragma link C++ class Particle+;
#endif
```

To write or read the objects of Particle class type, a dictionary is needed.
This dictionary can be generated in a similar way as additional STL containers.

TTrees – other objects



Dictionary can be loaded into CINT,

```
gSystem->Load("lib/libParticle.so");
```

Loaded into Python,

```
import ROOT
from ROOT import gSystem
gSystem.Load("lib/libParticle.so");
from ROOT import Particle
```

Or linked into an executable.

Storing objects in TFiles

Sometimes it is useful to add meta data to each file.
This data might contain the sum of the event weights,
or the name of the selection cuts, etc..

Writing

```
#include "TFile.h"
#include <vector>

int writeTFile() {
  TFile *f = TFile::Open("file-objects.root", "RECREATE");
  if(!f) return 1;
  std::vector<double> *w = new std::vector<double>();
  w->push_back(2.0);  // append one element
  w->push_back(4.0);  // append another element
  f->WriteObject(w,"weights");  // write the vector to the TFile
  f->Close();
  return 0;
}
```

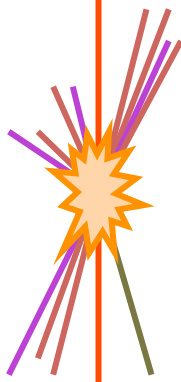
Reading objects from TFiles

The Get() function is used to retrieve the object.
Need to cast the object (in C++), since Get() returns TObject pointer

Reading

```
#include "TFile.h"
#include <vector>
#include <iostream>
int readTFile() {
  TFile *f = TFile::Open("file-objects.root");
  if(!f) return 1;
  std::vector<double> *w = 0;
  w = (std::vector<double>*)f->Get("weights");
  std::vector<double>::iterator itr = w->begin();
  std::vector<double>::iterator itr_end = w->end();
  for(;itr!=itr_end;++itr) {
    std::cout << (*itr) << std::endl;
  }
  f->Close();
  return 0;
}
```

TTree Draw



The Draw() function allows quick plots of observables.

```
root –l TTbar_PowHeg_Pythia_P2011C.root
particles->Draw("part_jet_pt[0]");  // Draw leading jet pT
```

Can use the Draw() function to fill a histogram.
The function arguments are <plot command>, <cuts>, <drawing options>

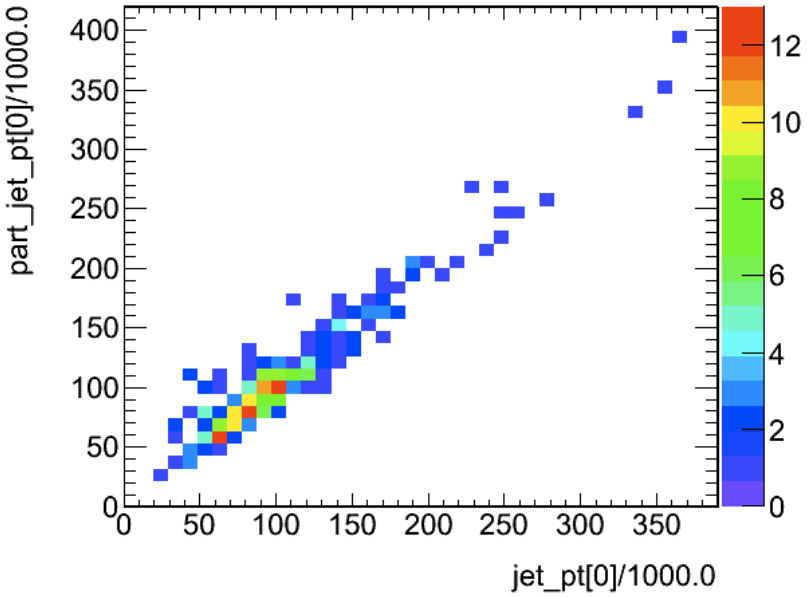
```
root –l TTbar_PowHeg_Pythia_P2011C.root
TH1F *h_part_jet_pt = new TH1F("h_part_jet_pt",";jet p_{T}   [GeV];Events/[GeV]",
100,0.,200.);
particles->Draw("part_jet_pt[0]/1000.0 >> h_part_jet_pt",
"pass_part_ejet || pass_part_mujet");
```

TTree Draw

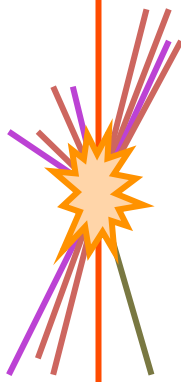
An event weight can be used within the cut selection.

```
TFile *f = TFile::Open("TTbar_PowHeg_Pythia_P2011C.root");
TTree *event = (TTree*)f->Get("event");
TTree *reco = (TTree*)f->Get("nominal/reco");
TTree *particles = (TTree*)f->Get("particles");
event->AddFriend(reco);
event->AddFriend(particles);
event->Draw("part_jet_pt[0]/1000.0:jet_pt[0]/1000.0",
"eventWeight*(pass_part_ejet && pass_ejet)", "COLZ");
```

Three TTrees are combined as one TTree, using the AddFriend function [problem].



Debugging



- Possible to debug using ACLiC using ++g,

```
.x solve.C++g()
```

- Compiled programs can be debugged using gdb.
  - Compile and link C++ code with –g flag.
  - Run by typing,

```
gdb executableName
r executableName [<command line options>]
```

Memory management

- ROOT objects are assigned memory within the present working directory.
  - Objects are instantiated with `new` are not on the stack!
  - The name of the object must be unique, since it is used to store the object.
    - Two new function calls with the same name will result in an error.
    - Can have the same name, but within two separate directories.
- If the present working file is closed, the associated objects will be cleared.
  - `SetDirectory(0)` puts histograms into unassociated memory, allowing the associated file to be closed.
- If several files are being used, must change directory to directory associated with TTree before calling `Write()`.

Warning: string comparisons can use up a lot of CPU.

RooFit

- Easily formulate probability distribution functions
- Unbinned maximum likelihood fits, using:
  - non-trivial functions
  - multidimensional functions

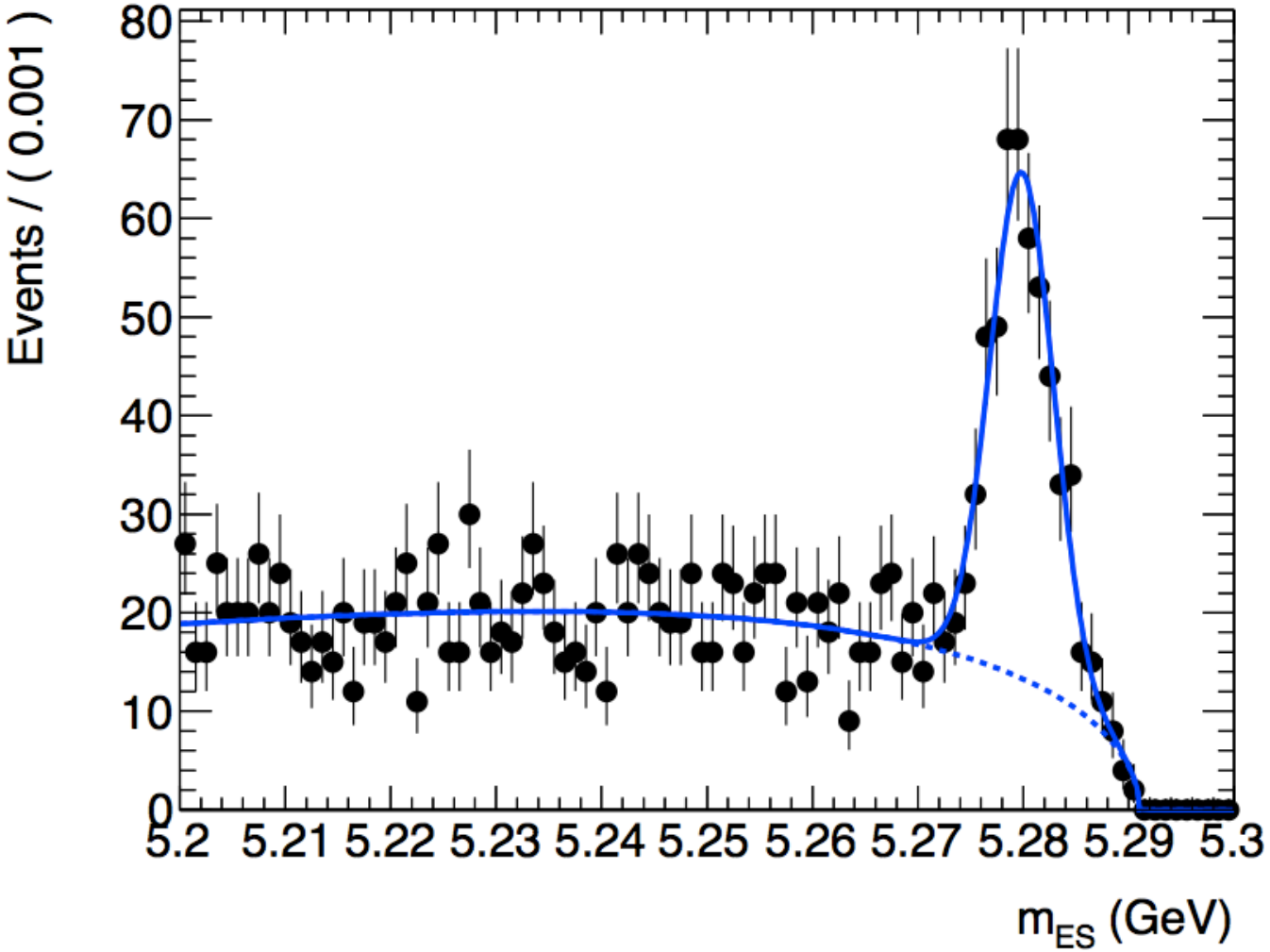
Concept	Math Symbol	RooFit class name
Variable	x, p	RooRealVar
Function	$f(\vec{x})$	RooAbsReal
PDF	$F(\vec{x}; \vec{p}, \vec{q})$	RooAbsPdf
Space point	$\vec{x}$	RooArgSet
Integral	$\int_{\vec{x}_{min}}^{\vec{x}_{max}} f(\vec{x}) d\vec{x}$	RooRealIntegal
Derivative	dF/dx	RooDerivative
-log(Likelihood)	$-\sum_{data} \log(F(x_i, \vec{p})$	RooNLLVar
List of space points	$\vec{x}_k$	RooAbsData

RootFit – $B^{\pm}$ -> $D^{0}K^{\pm}$

```
// Observable
RooRealVar mes("mes","m_{ES} (GeV)",5.20,5.30);

// Build Gaussian signal PDF
RooRealVar sigmean("sigmean","B^{#pm} mass",5.28,5.20,5.30);
RooRealVar sigwidth("sigwidth","B^{#pm} width",0.0027,0.001,1.);
RooGaussian gauss("gauss","gaussian PDF",mes,sigmean,sigwidth);

// Build Argus background PDF
RooRealVar argpar("argpar","argus shape parameter",-20.0,-100.,-1.);
RooRealVar mass("mass","B^{#pm}",5.291);
RooArgusBG argus("argus","Argus PDF",mes,mass,argpar);

// Construct signal+background PDF
RooRealVar nsig("nsig","#signal events",200,0.,10000);
RooRealVar nbkg("nbkg","#background events",800,0.,10000);
RooAddPdf sum("sum","g+a",RooArgList(gauss,argus),RooArgList(nsig,nbkg));

// Generate a toyMC sample from composite PDF
RooDataSet *data = sum.generate(mes,2000);

// Perform extended ML fit of composite PDF to toy data
sum.fitTo(*data,Extended());

// Plot toy data and composite PDF overlaid
RooPlot* mesframe = mes.frame();
data->plotOn(mesframe);
sum.plotOn(mesframe);
sum.plotOn(mesframe,Components(argus),LineStyle(kDashed));
```

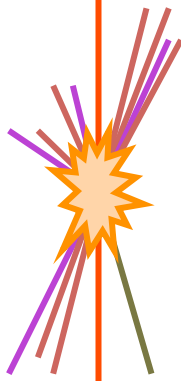
W. Verkerke, D. Kirkby

RootFit – $B^{\pm}$ -> $D^{0}K^{\pm}$



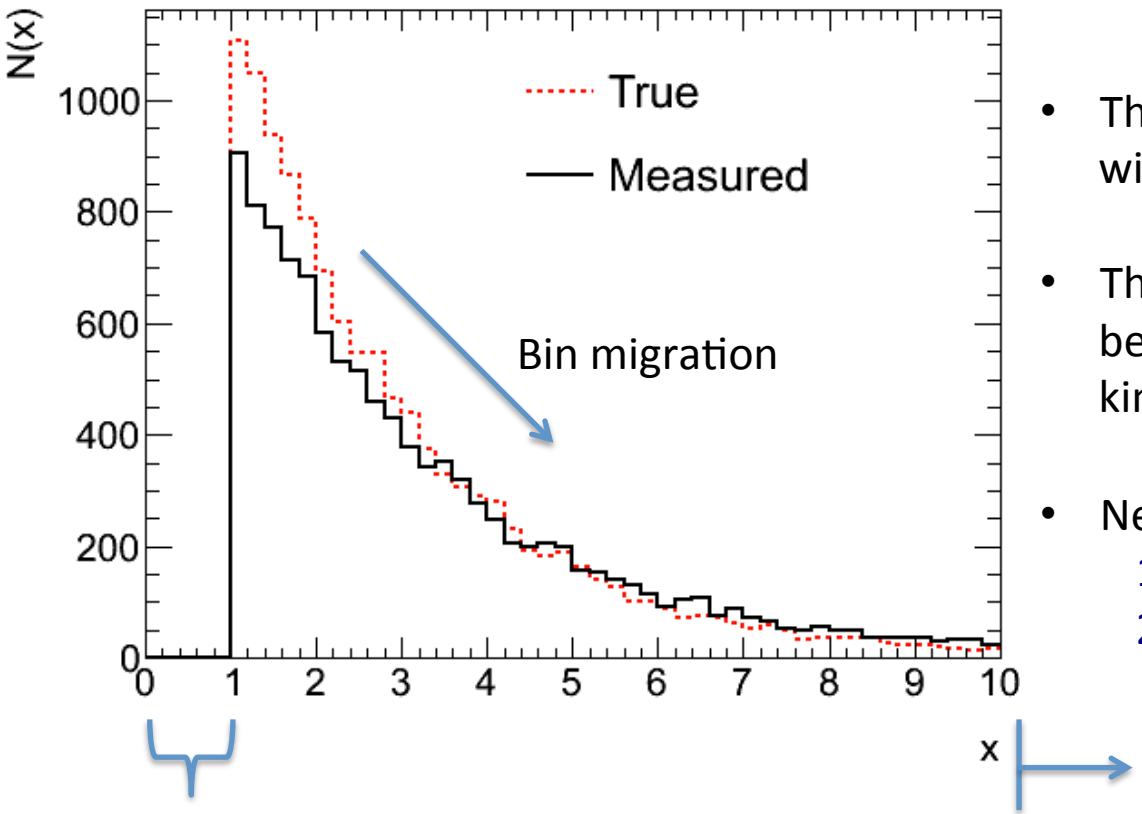
W. Verkerke, D. Kirkby

TMVA - Toolkit for Multivariate Data Analysis with ROOT



- Rectangular cut optimisation
- Multi-dimensional likelihood estimation
- Artificial neural networks
- Boosted decision trees
- Predictive learning
- and other multivariate techniques

Unfolding: illustration



- The distribution is measured within kinematic range.
- The true distribution has been truncated to match this kinematic range.
- Need to correct for:
  1. event migration
  2. bin migration

Unfolding: methodology

- Data is typically binned in histograms.
- Define a response matrix which relates the true distribution to the measured distribution.
- Each slice along x or y-axis of the response matrix describes a PDF for the given multiplicity.
  - $X_{meas} = M_{ij} X_{true}$
- With continuous distributions this would be written as

$$f_{meas}(x) = \int R(x \mid y) f_{true}(y)\, dy$$

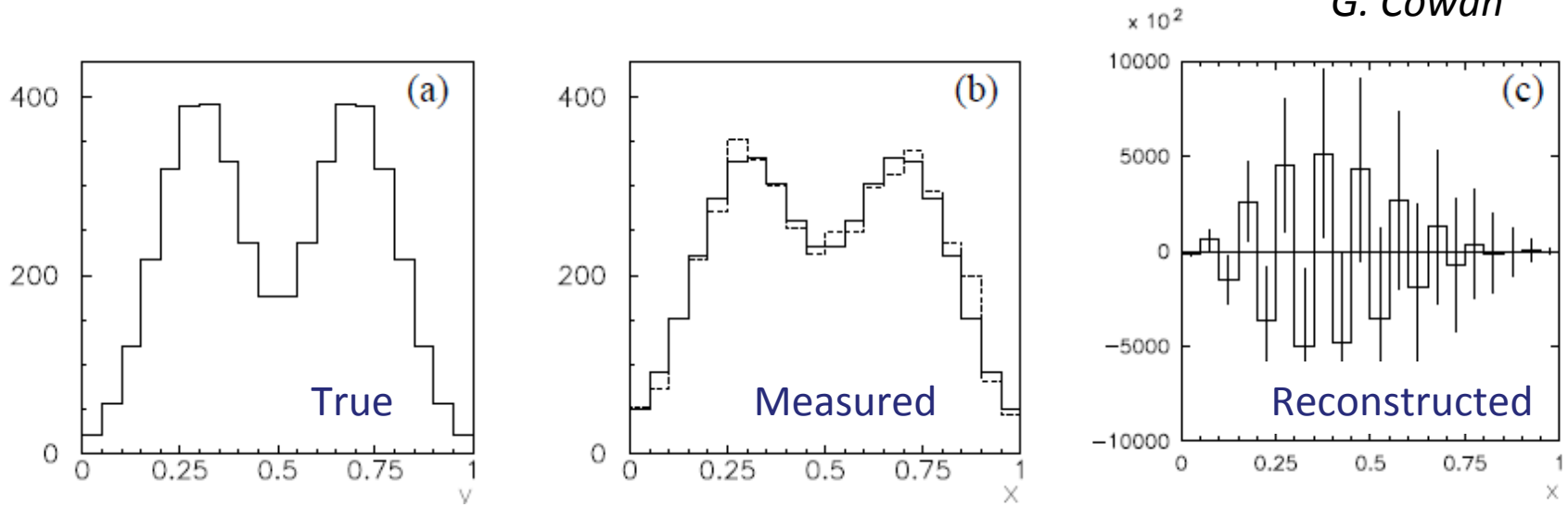
where R(x|y) is the response function, the continuous version of M_{ij}.

- Warning: background has not been included.

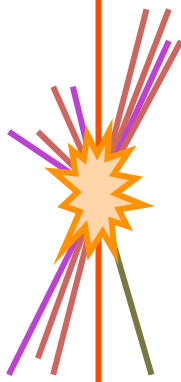
Unfolding: complexities and concerns

- Cannot simply invert the response matrix due to statistical nature of response matrix.
  - Inverting the response matrix causes large non-physical fluctuations.

G. Cowan



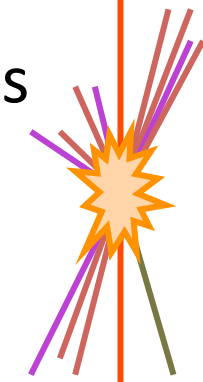
Unfolding: example



- Download RooUnfold from
  - http://hepunx.rl.ac.uk/~adye/software/unfold/RooUnfold.html
- Compile and run the example program.
  - Comment on the results obtained from the different unfolding methods available.
  - Tim explains how to compile and run

    `RooUnfold-1.1.1/examples/RooUnfoldExample.cxx` in the `README` file.

Problem



- The file TTbar_PowHeg_Pythia_P2011C.root contains particle and reconstructed objects for ttbar events.
  - Require pass_ejet or pass_mujet
  - Require jet_n >= 4 (at least four jets)
  - Require jet_ntags >= 2 (two b-tags)
- Assemble a pseudo-top:
  - Select two hardest b-tagged jets.
  - Select remaining two hardest jets as the W.
  - Select best match between hardest b-jet and W with respect to the top mass of 172.9 GeV.
- Plot the p_T and pseudorapidity spectrum of the pseudotop.

Problem

Use MakeClass() to generate header file. Then edit to minimum.
Use AddFriend() to associate the three TTrees.

```
TFile *f = TFile::Open("TTbar_PowHeg_Pythia_P2011C.root");
TTree *event = (TTree*)f->Get("event");
TTree *reco = (TTree*)f->Get("nominal/reco");
TTree *particles = (TTree*)f->Get("particles");
event->AddFriend(reco);
event->AddFriend(particles);
```

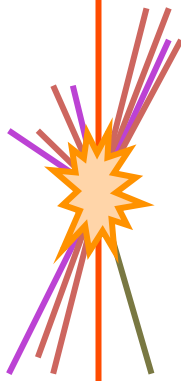
Just use the jet branches. The "part" versions refer to the particle selection, whereas the versions without "part" refer to the reconstructed selection.

part_jet_n : part_jet_n/I – Number of particle jets in the event
part_jet_pt : part_jet_pt[part_jet_n]/F – Transverse momentum (highest first)
part_jet_eta : part_jet_eta[part_jet_n]/F – Pseudorapidity
part_jet_phi : part_jet_phi[part_jet_n]/F – Azimuthal angle
part_jet_E : part_jet_E[part_jet_n]/F – The energy of the jet
part_jet_ntags : part_jet_ntags/I – Number of b-tagged jets in the event
part_jet_tagged : part_jet_tagged[part_jet_n]/O – Jet is tagged (true/false)
pass_part_ejet : pass_part_ejet/O – Event passes selection (true/false)
pass_part_mujet : pass_part_mujet/O – Event passes selection (true/false)

Problem

Instantiate TLorentzVectors from the jet objects

```
TLorentzVector part_jet;
part_jet.SetPtEtaPhiE(part_jet_pt[i], part_jet_eta[i],
                      part_jet_phi[i], part_jet_E[i]);
```



Combine the TLorentzVectors together

```
TLorentzVector part_Wboson;
part_Wboson = part_jet+part_jet_1;
```

Plot the mass of the W-boson candidate.

Then combine the W-boson with each leading b-tagged jet and produce the pseudo-top four vector.