MadGraph and ROOT output from the Standard Model background processes $pp \to t\bar{t}$, $W + \text{jets}$, $Z^0 + \text{jets}$

Eshwen Bhal

November 11, 2016

Contents

Figures

Tables

Input, source code and header files

Abstract

Standard Model process were computed using MADGRAPH with PYTHIA and PGS runs. The data were analysed in ROOT using C++ macros and histogram stacks were plotted after cuts (not necessarily to gain any specific information, but to showcase the techniques and syntax required to produce these types of results). As an example, the counts were plotted as a function of the psuedorapidity of the leading jet (sometimes written as Jet.Eta or Jet η) after applying a transverse momentum cut (Jet.PT, or Jet p_{T}) of 200 GeV/c.

0.1 MadGraph Input

The input files for MadGraph are detailed below. They were run, with several files (some being GBs in size) being produced. In each run (one run for each process) 100,000 events were simulated. A lepton momentum cut of 60 GeV/c was included in each input file.

0.1.1 $pp \to t\bar{t}$

```
#************************************************************
#*                        MadGraph 5                        *
#*                                                          *
#*                *                       *                 *
#*                  *        * *        *                   *
#*                    * * * * 5 * * * *                     *
#*                  *        * *        *                   *
#*                *                       *                 *
#*                                                          *
#*                                                          *
#*    The MadGraph Development Team - Please visit us at    *
#*    https://server06.fynu.ucl.ac.be/projects/madgraph     *
#*                                                          *
#************************************************************
#*                                                          *
#*               Command File for MadGraph 5                *
#*                                                          *
#*     run as ./bin/mg5  filename                           *
#*                                                          *
#************************************************************
import model sm
# Define multiparticle labels
define p = g u c d s u~ c~ d~ s~
define j = g u c d s u~ c~ d~ s~
define l+ = e+ mu+
define l- = e- mu-
define vl = ve vm vt
define vl~ = ve~ vm~ vt~
# Specify process(es) to run
generate p p > t t~
# Output processes to MadEvent directory

output sm_test_ppttbar
launch
pythia=ON
pgs=ON
set nevents 100000
set ptl 60
```

Listing 1: MadGraph input file for $pp \to t\bar{t}$

0.1.2 $pp \to W + \text{jets}$

```
#************************************************************
#*                        MadGraph 5                        *
#*                                                          *
#*                *                       *                 *
#*                  *        * *        *                   *
#*                    * * * * 5 * * * *                     *
#*                  *        * *        *                   *
#*                *                       *                 *
#*                                                          *
#*                                                          *
#*    The MadGraph Development Team - Please visit us at    *
#*    https://server06.fynu.ucl.ac.be/projects/madgraph     *
#*                                                          *
#************************************************************
#*                                                          *
#*               Command File for MadGraph 5                *
#*                                                          *
#*     run as ./bin/mg5  filename                           *
#*                                                          *
#************************************************************
import model sm
# Define multiparticle labels
define p = g u c d s u~ c~ d~ s~
define j = g u c d s u~ c~ d~ s~
define l+ = e+ mu+
define l- = e- mu-
define vl = ve vm vt
define vl~ = ve~ vm~ vt~
# Specify process(es) to run
generate p p > W+ j @1
# Output processes to MadEvent directory

output sm_test_ppWjets
launch
pythia=ON
pgs=ON
set nevents 100000
set ptl 60
```

Listing 2: MadGraph input file for $pp \to W + \text{jets}$

0.1.3 $pp \to Z^0 + \text{jets}$

```
#**************************************************************
#*                        MadGraph 5                          *
#*                                                            *
#*                *                        *                  *
#*                  *         * *         *                   *
#*                    * * * * 5 * * * *                       *
#*                  *         * *         *                   *
#*                *                        *                  *
#*                                                            *
#*                                                            *
#*    The MadGraph Development Team - Please visit us at      *
#*    https://server06.fynu.ucl.ac.be/projects/madgraph       *
#*                                                            *
#**************************************************************
#*                                                            *
#*               Command File for MadGraph 5                  *
#*                                                            *
#*     run as ./bin/mg5  filename                             *
#*                                                            *
#**************************************************************
import model sm
# Define multiparticle labels
define p = g u c d s u~ c~ d~ s~
define j = g u c d s u~ c~ d~ s~
define l+ = e+ mu+
define l- = e- mu-
define vl = ve vm vt
define vl~ = ve~ vm~ vt~
# Specify process(es) to run
generate p p > Z j @1
# Output processes to MadEvent directory

output sm_test_ppZjets
launch
pythia=ON
pgs=ON
set nevents 100000
set ptl 60
```

Listing 3: MadGraph input file for $pp \to Z^0 + \text{jets}$

0.2 MadGraph output

Of the many files produced in the output, only the .lhco file was used in this analysis, so I converted it to a .root file (with the only tree being LHCO) containing branches about the jet, muons, electrons, taus, photons, and the events in general.

0.2.1 $pp \to t\bar{t}$

The cross section for the process was 504.9 pb. The subprocesses were $gg \to t\bar{t}$, and $q\bar{q} \to t\bar{t}$, where q is a quark.

0.2.2 $pp \to W + \text{jets}$

The cross section for the process was 2.144×10^4 pb. The subprocesses were $gu \to W^+d$, $gc \to W^+s$, $g\bar{d} \to W^+\bar{u}$, $g\bar{s} \to W^+\bar{c}$, $u\bar{d} \to W^+g$, and $c\bar{s} \to W^+g$.

0.2.3 $pp \to Z^0 + \text{jets}$

The cross section for the process was 1.166×10^4 pb. The subprocesses were $gX \to Z^0X$, $g\bar{X} \to Z^0\bar{X}$, and $X\bar{X} \to Z^0g$, where X is $\{u, d, c, s\}$.

0.3 ROOT output

Histogram stacks were created using C++ macros. Using the ROOT command 'MakeClass', a header and source file for each .root file were obtained. They were edited to suit the needs of the analysis (filling histograms with the jet η variable, applying a cut, and then plotting histogram stacks). The cut applied when filling the histograms was jet $p_\text{T} > 200$ GeV/c. Each histogram was normalized according to the luminosity. The scale factor $s.f.$ was calculated using the simple, standard formula

$$N = \sigma\mathcal{L} \tag{1}$$

where N is the number of events, σ is the interaction cross section, and $\mathcal{L}$ is the luminosity. These values are detailed in Table 1. Then the scale factor is

$$s.f. = \sigma\mathcal{L}/N \tag{2}$$

and is dimensionless, since [σ] = pb and [$\mathcal{L}$] = pb^{-1}.

Property	$pp \to t\bar{t}$	$pp \to W + \text{jets}$	$pp \to Z^0 + \text{jets}$
Number of Events (MadGraph)	100 000	100 000	100 000
Cross section (pb)	504.9	2.144×10^4	1.166×10^4
Assumed luminosity (pb^{-1})	20 000	20 000	20 000
Jet η events before cut	99 999	99 992	99 996
Jet η events after cut (Jet p_T > 200)	4 609	371	470
Efficiency (%)	4.6	0.37	0.47

Table 1: The properties of the processes that were simulated.

Then, what I dub, a "daddy macro" was written to create the canvas, execute the functions from each source file, and apply aesthetics to each pad of the canvas before saving it. I also created a header file to store global variables and constants, etc., and the "include" declarations the other files would need. The graphs from this analysis are displayed in Figure 1.

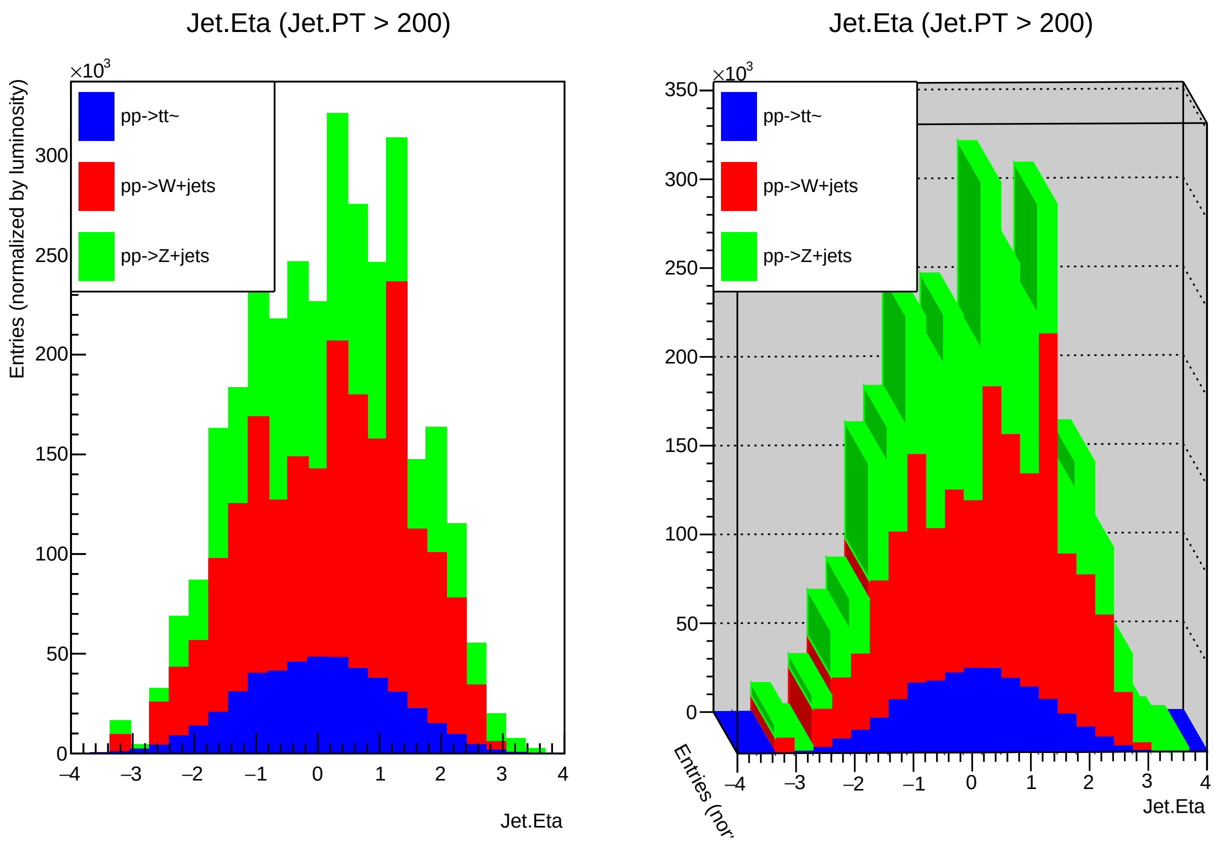

Figure 1: Histogram stacks for several standard model process depicting the normalized number of entries vs. the η variable of the respective leading jet. The left-most histogram is the default stack, whilst the right-most graph is a "lego" plot.

The source code for the "daddy macro":

```
// Script to run the macros and plot histograms from multiple root files

// Most things (canvas, legend, etc.) can be initialised in this file, but the
// first .C file called must initialise the histogram stack (THStack)

#include "ppttbar.C"
#include "ppWjets.C"
#include "ppZjets.C"
#include "global.h"

void execComparejets() {

      // Create canvas of width w and height h (in pixels), then split into two
      // columns
      TCanvas* c1 = new TCanvas("c1");
      Int_t w = 900, h = 600;
      c1->SetCanvasSize(w,h);
      c1->Divide(2,1);

      // Run main loops to fill and draw histograms
      ppttbar t;
      t.Loop();
      ppWjets u;
      u.Loop();
      ppZjets v;
      v.Loop();

      // Set y-axis range of plots
      Double_t ymin = stackedhists->GetMinimum(), ymax = stackedhists->GetMaximum();
      stackedhists->SetMinimum(ymin);
      stackedhists->SetMaximum(ymax);

      // Set axes labels and offset (in % of pad width) so they don't overlap
      // with axis ticks
      stackedhists->GetXaxis()->SetTitle("Jet.Eta");
      stackedhists->GetXaxis()->SetTitleOffset(1.4);
      stackedhists->GetYaxis()->SetTitle("Entries (normalized by luminosity)");
      stackedhists->GetYaxis()->SetTitleOffset(1.5);

      // Set aesthetics for lego plot
      c1->cd(2);
      gPad->SetFrameFillColor(17);
      gPad->SetTheta(3.77);
      gPad->SetPhi(2.9);

      // Add the same legend to both plots
      for (Int_t i = 1; i < 3; ++i) {
            c1->cd(i);
            gPad->BuildLegend(0.1,0.65,0.43,0.9,"");
            // Range of the legend box (x1,y1,x2,y2). Origin at bottom left
      }
      // Save as pdf because png/bmp doesn't work properly at high
```

```
    resolution
      c1->SaveAs("comparejets.pdf");
}

// FIGURE OUT HOW TO SET THE Z-AXIS LABEL (AND REMOVE THE Y-AXIS) FOR THE
    LEGO
// PLOT

/* TRY AND TIDY UP AND STREAMLINE SOURCE FILES. SEE IF I CAN PUT HISTOGRAM
    NAMES
   AS STRINGS AND REFERENCE THE STRING AT EACH USE (SO THAT I ONLY HAVE TO
    EDIT
   ONE LINE OF CODE IF I WANT TO PLOT A DIFFERENT HISTOGRAM). SEE IF I CAN
    DO
   THE SAME WITH THE CUTTING VARIABLE. OR JUST TRY AND PLOT ALL HISTOGRAMS
    AND
   SAVE THEM IN AN ARRAY, BUT THEN CALL AND DISPLAY ONLY SPECIFIC ONES.
*/
```

Listing 4: Daddy macro

The global header file:

```
// Global variables/constants/include files are defined here

// ROOT header files
#include <TROOT.h>
#include <TChain.h>
#include <TFile.h>
#include <TH2.h>
#include <TStyle.h>
#include <TCanvas.h>
#include <TString.h>
#include <TLegend.h>
#include <THStack.h>
#include <TPad.h>

// Header file for the classes stored in the TTree if any.
#include "TClonesArray.h"
#include "TObject.h"

// C++ header files
#include <iostream.h>

// Constants for histogram initialization
const int N_BINS = 25;
const int X_MIN = -4;
const int X_MAX = 4;

// Beam properties. N_EVENTS is defined in MadGraph input file,
    luminosity_pb
// is assumed
const double N_EVENTS = 100000;
const double luminosity_pb = 20000;

// Declare the type of histogram to draw
TString histType_pad1 = "";
TString histType_pad2 = "lego1";
```

Listing 5: Global header file

0.3.1 $pp \to t\bar{t}$

The header file:

```
//////////////////////////////////////////////////////////
// This class has been automatically generated on
// Tue Nov  1 16:18:19 2016 by ROOT version 6.02/02
// from TTree LHCO/ppttbar tree
// found on file: ppttbar.root
//////////////////////////////////////////////////////////

// Header file to include declarations, classes, variables for ppttbar.C
   macro

#ifndef ppttbar_h
#define ppttbar_h

#include "global.h"

class ppttbar {
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
   .
//   const Int_t kMaxEvent = 1;
//   const Int_t kMaxPhoton = 3;
//   const Int_t kMaxElectron = 2;
//   const Int_t kMaxMuon = 2;
//   const Int_t kMaxTau = 3;
//   const Int_t kMaxJet = 11;
//   const Int_t kMaxMissingET = 1;

   // Declaration of leaf types
   Int_t           Event_;
   UInt_t          Event_fUniqueID[1];   //[Event_]
   UInt_t          Event_fBits[1];   //[Event_]
   Long64_t        Event_Number[1];   //[Event_]
   Int_t           Event_Trigger[1];   //[Event_]
   Int_t           Event_size;
   Int_t           Photon_;
   UInt_t          Photon_fUniqueID[3];   //[Photon_]
   UInt_t          Photon_fBits[3];   //[Photon_]
   Double_t        Photon_PT[3];   //[Photon_]
   Double_t        Photon_Eta[3];   //[Photon_]
   Double_t        Photon_Phi[3];   //[Photon_]
   Double_t        Photon_EhadOverEem[3];   //[Photon_]
   Int_t           Photon_size;
   Int_t           Electron_;
   UInt_t          Electron_fUniqueID[3];   //[Electron_]
   UInt_t          Electron_fBits[3];   //[Electron_]
   Double_t        Electron_PT[3];   //[Electron_]
   Double_t        Electron_Eta[3];   //[Electron_]
   Double_t        Electron_Phi[3];   //[Electron_]
   Double_t        Electron_Charge[3];   //[Electron_]
   Double_t        Electron_Ntrk[3];   //[Electron_]
   Double_t        Electron_EhadOverEem[3];   //[Electron_]
   Int_t           Electron_size;
   Int_t           Muon_;
   UInt_t          Muon_fUniqueID[3];   //[Muon_]
   UInt_t          Muon_fBits[3];   //[Muon_]
   Double_t        Muon_PT[3];   //[Muon_]
   Double_t        Muon_Eta[3];   //[Muon_]
   Double_t        Muon_Phi[3];   //[Muon_]
   Double_t        Muon_Charge[3];   //[Muon_]
   Double_t        Muon_Ntrk[3];   //[Muon_]
   Double_t        Muon_PTiso[3];   //[Muon_]
   Double_t        Muon_ETiso[3];   //[Muon_]
   Int_t           Muon_JetIndex[3];   //[Muon_]
   Int_t           Muon_size;
   Int_t           Tau_;
   UInt_t          Tau_fUniqueID[4];   //[Tau_]
   UInt_t          Tau_fBits[4];   //[Tau_]
   Double_t        Tau_PT[4];   //[Tau_]
   Double_t        Tau_Eta[4];   //[Tau_]
   Double_t        Tau_Phi[4];   //[Tau_]
   Double_t        Tau_Charge[4];   //[Tau_]
   Double_t        Tau_Ntrk[4];   //[Tau_]
   Double_t        Tau_EhadOverEem[4];   //[Tau_]
   Int_t           Tau_size;
   Int_t           Jet_;
   UInt_t          Jet_fUniqueID[17];   //[Jet_]
   UInt_t          Jet_fBits[17];   //[Jet_]
   Double_t        Jet_PT[17];   //[Jet_]
   Double_t        Jet_Eta[17];   //[Jet_]
   Double_t        Jet_Phi[17];   //[Jet_]
   Double_t        Jet_Mass[17];   //[Jet_]
   Double_t        Jet_Ntrk[17];   //[Jet_]
   Double_t        Jet_BTag[17];   //[Jet_]
   Double_t        Jet_EhadOverEem[17];   //[Jet_]
   Int_t           Jet_Index[17];   //[Jet_]
   Int_t           Jet_size;
   Int_t           MissingET_;
   UInt_t          MissingET_fUniqueID[1];   //[MissingET_]
   UInt_t          MissingET_fBits[1];   //[MissingET_]
   Double_t        MissingET_MET[1];   //[MissingET_]
   Double_t        MissingET_Phi[1];   //[MissingET_]
   Int_t           MissingET_size;

   // List of branches
   TBranch        *b_Event_;   //!
   TBranch        *b_Event_fUniqueID;   //!
   TBranch        *b_Event_fBits;   //!
   TBranch        *b_Event_Number;   //!
   TBranch        *b_Event_Trigger;   //!
   TBranch        *b_Event_size;   //!
   TBranch        *b_Photon_;   //!
   TBranch        *b_Photon_fUniqueID;   //!
   TBranch        *b_Photon_fBits;   //!
   TBranch        *b_Photon_PT;   //!
   TBranch        *b_Photon_Eta;   //!
   TBranch        *b_Photon_Phi;   //!
   TBranch        *b_Photon_EhadOverEem;   //!
   TBranch        *b_Photon_size;   //!
   TBranch        *b_Electron_;   //!
   TBranch        *b_Electron_fUniqueID;   //!
```

```
   TBranch        *b_Electron_fBits;   //!
   TBranch        *b_Electron_PT;   //!
   TBranch        *b_Electron_Eta;   //!
   TBranch        *b_Electron_Phi;   //!
   TBranch        *b_Electron_Charge;   //!
   TBranch        *b_Electron_Ntrk;   //!
   TBranch        *b_Electron_EhadOverEem;   //!
   TBranch        *b_Electron_size;   //!
   TBranch        *b_Muon_;   //!
   TBranch        *b_Muon_fUniqueID;   //!
   TBranch        *b_Muon_fBits;   //!
   TBranch        *b_Muon_PT;   //!
   TBranch        *b_Muon_Eta;   //!
   TBranch        *b_Muon_Phi;   //!
   TBranch        *b_Muon_Charge;   //!
   TBranch        *b_Muon_Ntrk;   //!
   TBranch        *b_Muon_PTiso;   //!
   TBranch        *b_Muon_ETiso;   //!
   TBranch        *b_Muon_JetIndex;   //!
   TBranch        *b_Muon_size;   //!
   TBranch        *b_Tau_;   //!
   TBranch        *b_Tau_fUniqueID;   //!
   TBranch        *b_Tau_fBits;   //!
   TBranch        *b_Tau_PT;   //!
   TBranch        *b_Tau_Eta;   //!
   TBranch        *b_Tau_Phi;   //!
   TBranch        *b_Tau_Charge;   //!
   TBranch        *b_Tau_Ntrk;   //!
   TBranch        *b_Tau_EhadOverEem;   //!
   TBranch        *b_Tau_size;   //!
   TBranch        *b_Jet_;   //!
   TBranch        *b_Jet_fUniqueID;   //!
   TBranch        *b_Jet_fBits;   //!
   TBranch        *b_Jet_PT;   //!
   TBranch        *b_Jet_Eta;   //!
   TBranch        *b_Jet_Phi;   //!
   TBranch        *b_Jet_Mass;   //!
   TBranch        *b_Jet_Ntrk;   //!
   TBranch        *b_Jet_BTag;   //!
   TBranch        *b_Jet_EhadOverEem;   //!
   TBranch        *b_Jet_Index;   //!
   TBranch        *b_Jet_size;   //!
   TBranch        *b_MissingET_;   //!
   TBranch        *b_MissingET_fUniqueID;   //!
   TBranch        *b_MissingET_fBits;   //!
   TBranch        *b_MissingET_MET;   //!
   TBranch        *b_MissingET_Phi;   //!
   TBranch        *b_MissingET_size;   //!

   ppttbar(TTree *tree=0);
   virtual ~ppttbar();
   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 ppttbar_cxx
ppttbar::ppttbar(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("ppttbar.root"
   );
      if (!f || !f->IsOpen()) {
         f = new TFile("ppttbar.root");
      }
      f->GetObject("LHCO",tree);

   }
   Init(tree);
}

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

Int_t ppttbar::GetEntry(Long64_t entry)
{
// Read contents of entry.
   if (!fChain) return 0;
   return fChain->GetEntry(entry);
}
Long64_t ppttbar::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 ppttbar::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("Event", &Event_, &b_Event_);
fChain->SetBranchAddress("Event.fUniqueID", Event_fUniqueID, &
b_Event_fUniqueID);
fChain->SetBranchAddress("Event.fBits", Event_fBits, &b_Event_fBits);
fChain->SetBranchAddress("Event.Number", Event_Number, &b_Event_Number);
fChain->SetBranchAddress("Event.Trigger", Event_Trigger, &
b_Event_Trigger);
fChain->SetBranchAddress("Event_size", &Event_size, &b_Event_size);
fChain->SetBranchAddress("Photon", &Photon_, &b_Photon_);
fChain->SetBranchAddress("Photon.fUniqueID", Photon_fUniqueID, &
b_Photon_fUniqueID);
fChain->SetBranchAddress("Photon.fBits", Photon_fBits, &b_Photon_fBits);
fChain->SetBranchAddress("Photon.PT", Photon_PT, &b_Photon_PT);
fChain->SetBranchAddress("Photon.Eta", Photon_Eta, &b_Photon_Eta);
fChain->SetBranchAddress("Photon.Phi", Photon_Phi, &b_Photon_Phi);
fChain->SetBranchAddress("Photon.EhadOverEem", Photon_EhadOverEem, &
b_Photon_EhadOverEem);
fChain->SetBranchAddress("Photon_size", &Photon_size, &b_Photon_size);
fChain->SetBranchAddress("Electron", &Electron_, &b_Electron_);
fChain->SetBranchAddress("Electron.fUniqueID", Electron_fUniqueID, &
b_Electron_fUniqueID);
fChain->SetBranchAddress("Electron.fBits", Electron_fBits, &
b_Electron_fBits);
fChain->SetBranchAddress("Electron.PT", Electron_PT, &b_Electron_PT);
fChain->SetBranchAddress("Electron.Eta", Electron_Eta, &b_Electron_Eta);
fChain->SetBranchAddress("Electron.Phi", Electron_Phi, &b_Electron_Phi);
fChain->SetBranchAddress("Electron.Charge", Electron_Charge, &
b_Electron_Charge);
fChain->SetBranchAddress("Electron.Ntrk", Electron_Ntrk, &
b_Electron_Ntrk);
fChain->SetBranchAddress("Electron.EhadOverEem", Electron_EhadOverEem, &
b_Electron_EhadOverEem);
fChain->SetBranchAddress("Electron_size", &Electron_size, &
b_Electron_size);
fChain->SetBranchAddress("Muon", &Muon_, &b_Muon_);
fChain->SetBranchAddress("Muon.fUniqueID", Muon_fUniqueID, &
b_Muon_fUniqueID);
fChain->SetBranchAddress("Muon.fBits", Muon_fBits, &b_Muon_fBits);
fChain->SetBranchAddress("Muon.PT", Muon_PT, &b_Muon_PT);
fChain->SetBranchAddress("Muon.Eta", Muon_Eta, &b_Muon_Eta);
fChain->SetBranchAddress("Muon.Phi", Muon_Phi, &b_Muon_Phi);
fChain->SetBranchAddress("Muon.Charge", Muon_Charge, &b_Muon_Charge);
fChain->SetBranchAddress("Muon.Ntrk", Muon_Ntrk, &b_Muon_Ntrk);
fChain->SetBranchAddress("Muon.PTiso", Muon_PTiso, &b_Muon_PTiso);
fChain->SetBranchAddress("Muon.ETiso", Muon_ETiso, &b_Muon_ETiso);
fChain->SetBranchAddress("Muon.JetIndex", Muon_JetIndex, &
b_Muon_JetIndex);
fChain->SetBranchAddress("Muon_size", &Muon_size, &b_Muon_size);
fChain->SetBranchAddress("Tau", &Tau_, &b_Tau_);
fChain->SetBranchAddress("Tau.fUniqueID", Tau_fUniqueID, &
b_Tau_fUniqueID);
fChain->SetBranchAddress("Tau.fBits", Tau_fBits, &b_Tau_fBits);
fChain->SetBranchAddress("Tau.PT", Tau_PT, &b_Tau_PT);
fChain->SetBranchAddress("Tau.Eta", Tau_Eta, &b_Tau_Eta);
```

```
   fChain->SetBranchAddress("Tau.Phi", Tau_Phi, &b_Tau_Phi);
   fChain->SetBranchAddress("Tau.Charge", Tau_Charge, &b_Tau_Charge);
   fChain->SetBranchAddress("Tau.Ntrk", Tau_Ntrk, &b_Tau_Ntrk);
   fChain->SetBranchAddress("Tau.EhadOverEem", Tau_EhadOverEem, &
   b_Tau_EhadOverEem);
   fChain->SetBranchAddress("Tau_size", &Tau_size, &b_Tau_size);
   fChain->SetBranchAddress("Jet", &Jet_, &b_Jet_);
   fChain->SetBranchAddress("Jet.fUniqueID", Jet_fUniqueID, &
   b_Jet_fUniqueID);
   fChain->SetBranchAddress("Jet.fBits", Jet_fBits, &b_Jet_fBits);
   fChain->SetBranchAddress("Jet.PT", Jet_PT, &b_Jet_PT);
   fChain->SetBranchAddress("Jet.Eta", Jet_Eta, &b_Jet_Eta);
   fChain->SetBranchAddress("Jet.Phi", Jet_Phi, &b_Jet_Phi);
   fChain->SetBranchAddress("Jet.Mass", Jet_Mass, &b_Jet_Mass);
   fChain->SetBranchAddress("Jet.Ntrk", Jet_Ntrk, &b_Jet_Ntrk);
   fChain->SetBranchAddress("Jet.BTag", Jet_BTag, &b_Jet_BTag);
   fChain->SetBranchAddress("Jet.EhadOverEem", Jet_EhadOverEem, &
   b_Jet_EhadOverEem);
   fChain->SetBranchAddress("Jet.Index", Jet_Index, &b_Jet_Index);
   fChain->SetBranchAddress("Jet_size", &Jet_size, &b_Jet_size);
   fChain->SetBranchAddress("MissingET", &MissingET_, &b_MissingET_);
   fChain->SetBranchAddress("MissingET.fUniqueID", MissingET_fUniqueID, &
   b_MissingET_fUniqueID);
   fChain->SetBranchAddress("MissingET.fBits", MissingET_fBits, &
   b_MissingET_fBits);
   fChain->SetBranchAddress("MissingET.MET", MissingET_MET, &
   b_MissingET_MET);
   fChain->SetBranchAddress("MissingET.Phi", MissingET_Phi, &
   b_MissingET_Phi);
   fChain->SetBranchAddress("MissingET_size", &MissingET_size, &
   b_MissingET_size);
   Notify();
}

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

```
    return 1;
}
#endif // #ifdef ppttbar_cxx
```

Listing 6: Header file for $pp \to t\bar{t}$

The source file:

```
#define ppttbar_cxx

#include "ppttbar.h"
#include "global.h"

void ppttbar::Loop() {

//   In a ROOT session, you can do:
//      Root > .L ppttbar.C
//      Root > ppttbar t
//      Root > t.GetEntry(12); // Fill t data members with entry number 12
//      Root > t.Show();       // Show values of entry 12
//      Root > t.Show(16);     // Read and show values of entry 16
//      Root > t.Loop();       // Loop on all entries
//

//     This is the loop skeleton where:
//    jentry is the global entry number in the chain
//    ientry is the entry number in the current Tree
//  Note that the argument to GetEntry must be:
//    jentry for TChain::GetEntry
//    ientry for TTree::GetEntry and TBranch::GetEntry
//
//       To read only selected branches, Insert statements like:
// METHOD1:
//    fChain->SetBranchStatus("*",0);  // disable all branches
//    fChain->SetBranchStatus("branchname",1);  // activate branchname
// METHOD2: replace line
//    fChain->GetEntry(jentry);       //read all branches
//by  b_branchname->GetEntry(ientry); //read only this branch

      if (fChain == 0)
            return;
      Long64_t nentries = fChain->GetEntriesFast();

      // Create histogram to be filled
      TH1F* ttbar_jet_eta = new TH1F("jet_eta", "pp->tt~", N_BINS, X_MIN,
   X_MAX);

      Long64_t nbytes = 0, nb = 0;

      // Loop over entries and fill histograms
      for (Long64_t jentry = 0; jentry < nentries; jentry++) {
            Long64_t ientry = LoadTree(jentry);
            if (ientry < 0)
                  break;

            nb = fChain->GetEntry(jentry);
            nbytes += nb;

            // Loop to apply a cut
            if (Jet_PT[0] > 200)
                  ttbar_jet_eta->Fill(Jet_Eta[0]);
      // if (Cut(ientry) < 0) continue;
      }

      // Normalize histogram by luminosity
```

```
     Double_t cross_sec = 504.9;
     ttbar_jet_eta->Scale( cross_sec * luminosity_pb / N_EVENTS );

     // Set aesthetics
     ttbar_jet_eta->SetLineColor(kBlue);
     ttbar_jet_eta->SetFillColor(kBlue);
     ttbar_jet_eta->SetMarkerStyle(21);
     ttbar_jet_eta->SetMarkerColor(kBlue);

     // Create a histogram stack
     THStack *stackedhists = new THStack("stackedhists", "Jet.Eta (Jet.PT
   > 200)");
     stackedhists->Add(ttbar_jet_eta);

    // Canvas is initialised in execComparejets.C before calling this file
    c1->cd(1);
    // Draw 1D plot
    stackedhists->Draw(histType_pad1);

    c1->cd(2);
    // Draw lego plot
    stackedhists->Draw(histType_pad2);
}
```

Listing 7: Source file for $pp \rightarrow t\bar{t}$

0.3.2 $pp \to W + \text{jets}$

The header file:

```
//////////////////////////////////////////////////////////
// This class has been automatically generated on
// Wed Nov  2 14:23:58 2016 by ROOT version 5.34/36
// from TTree LHCO/Analysis tree
// found on file: ppWjets.root
//////////////////////////////////////////////////////////

// Header file to include declarations, classes, variables for ppttbar.C
   macro

#ifndef ppWjets_h
#define ppWjets_h

#include "global.h"

class ppWjets {
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
   .
//   const Int_t kMaxEvent = 1;
//   const Int_t kMaxPhoton = 3;
//   const Int_t kMaxElectron = 2;
//   const Int_t kMaxMuon = 2;
//   const Int_t kMaxTau = 3;
//   const Int_t kMaxJet = 11;
//   const Int_t kMaxMissingET = 1;

   // Declaration of leaf types
   Int_t           Event_;
   UInt_t          Event_fUniqueID[1];   //[Event_]
   UInt_t          Event_fBits[1];   //[Event_]
   Long64_t        Event_Number[1];   //[Event_]
   Int_t           Event_Trigger[1];   //[Event_]
   Int_t           Event_size;
   Int_t           Photon_;
   UInt_t          Photon_fUniqueID[3];   //[Photon_]
   UInt_t          Photon_fBits[3];   //[Photon_]
   Double_t        Photon_PT[3];   //[Photon_]
   Double_t        Photon_Eta[3];   //[Photon_]
   Double_t        Photon_Phi[3];   //[Photon_]
   Double_t        Photon_EhadOverEem[3];   //[Photon_]
   Int_t           Photon_size;
   Int_t           Electron_;
   UInt_t          Electron_fUniqueID[3];   //[Electron_]
   UInt_t          Electron_fBits[3];   //[Electron_]
   Double_t        Electron_PT[3];   //[Electron_]
   Double_t        Electron_Eta[3];   //[Electron_]
   Double_t        Electron_Phi[3];   //[Electron_]
   Double_t        Electron_Charge[3];   //[Electron_]
   Double_t        Electron_Ntrk[3];   //[Electron_]
   Double_t        Electron_EhadOverEem[3];   //[Electron_]
   Int_t           Electron_size;
   Int_t           Muon_;
   UInt_t          Muon_fUniqueID[3];   //[Muon_]
   UInt_t          Muon_fBits[3];   //[Muon_]
   Double_t        Muon_PT[3];   //[Muon_]
   Double_t        Muon_Eta[3];   //[Muon_]
   Double_t        Muon_Phi[3];   //[Muon_]
   Double_t        Muon_Charge[3];   //[Muon_]
   Double_t        Muon_Ntrk[3];   //[Muon_]
   Double_t        Muon_PTiso[3];   //[Muon_]
   Double_t        Muon_ETiso[3];   //[Muon_]
   Int_t           Muon_JetIndex[3];   //[Muon_]
   Int_t           Muon_size;
   Int_t           Tau_;
   UInt_t          Tau_fUniqueID[4];   //[Tau_]
   UInt_t          Tau_fBits[4];   //[Tau_]
   Double_t        Tau_PT[4];   //[Tau_]
   Double_t        Tau_Eta[4];   //[Tau_]
   Double_t        Tau_Phi[4];   //[Tau_]
   Double_t        Tau_Charge[4];   //[Tau_]
   Double_t        Tau_Ntrk[4];   //[Tau_]
   Double_t        Tau_EhadOverEem[4];   //[Tau_]
   Int_t           Tau_size;
   Int_t           Jet_;
   UInt_t          Jet_fUniqueID[17];   //[Jet_]
   UInt_t          Jet_fBits[17];   //[Jet_]
   Double_t        Jet_PT[17];   //[Jet_]
   Double_t        Jet_Eta[17];   //[Jet_]
   Double_t        Jet_Phi[17];   //[Jet_]
   Double_t        Jet_Mass[17];   //[Jet_]
   Double_t        Jet_Ntrk[17];   //[Jet_]
   Double_t        Jet_BTag[17];   //[Jet_]
   Double_t        Jet_EhadOverEem[17];   //[Jet_]
   Int_t           Jet_Index[17];   //[Jet_]
   Int_t           Jet_size;
   Int_t           MissingET_;
   UInt_t          MissingET_fUniqueID[1];   //[MissingET_]
   UInt_t          MissingET_fBits[1];   //[MissingET_]
   Double_t        MissingET_MET[1];   //[MissingET_]
   Double_t        MissingET_Phi[1];   //[MissingET_]
   Int_t           MissingET_size;

   // List of branches
   TBranch        *b_Event_;   //!
   TBranch        *b_Event_fUniqueID;   //!
   TBranch        *b_Event_fBits;   //!
   TBranch        *b_Event_Number;   //!
   TBranch        *b_Event_Trigger;   //!
   TBranch        *b_Event_size;   //!
   TBranch        *b_Photon_;   //!
   TBranch        *b_Photon_fUniqueID;   //!
   TBranch        *b_Photon_fBits;   //!
   TBranch        *b_Photon_PT;   //!
   TBranch        *b_Photon_Eta;   //!
   TBranch        *b_Photon_Phi;   //!
   TBranch        *b_Photon_EhadOverEem;   //!
   TBranch        *b_Photon_size;   //!
   TBranch        *b_Electron_;   //!
   TBranch        *b_Electron_fUniqueID;   //!
```

```
   TBranch        *b_Electron_fBits;   //!
   TBranch        *b_Electron_PT;   //!
   TBranch        *b_Electron_Eta;   //!
   TBranch        *b_Electron_Phi;   //!
   TBranch        *b_Electron_Charge;   //!
   TBranch        *b_Electron_Ntrk;   //!
   TBranch        *b_Electron_EhadOverEem;   //!
   TBranch        *b_Electron_size;   //!
   TBranch        *b_Muon_;   //!
   TBranch        *b_Muon_fUniqueID;   //!
   TBranch        *b_Muon_fBits;   //!
   TBranch        *b_Muon_PT;   //!
   TBranch        *b_Muon_Eta;   //!
   TBranch        *b_Muon_Phi;   //!
   TBranch        *b_Muon_Charge;   //!
   TBranch        *b_Muon_Ntrk;   //!
   TBranch        *b_Muon_PTiso;   //!
   TBranch        *b_Muon_ETiso;   //!
   TBranch        *b_Muon_JetIndex;   //!
   TBranch        *b_Muon_size;   //!
   TBranch        *b_Tau_;   //!
   TBranch        *b_Tau_fUniqueID;   //!
   TBranch        *b_Tau_fBits;   //!
   TBranch        *b_Tau_PT;   //!
   TBranch        *b_Tau_Eta;   //!
   TBranch        *b_Tau_Phi;   //!
   TBranch        *b_Tau_Charge;   //!
   TBranch        *b_Tau_Ntrk;   //!
   TBranch        *b_Tau_EhadOverEem;   //!
   TBranch        *b_Tau_size;   //!
   TBranch        *b_Jet_;   //!
   TBranch        *b_Jet_fUniqueID;   //!
   TBranch        *b_Jet_fBits;   //!
   TBranch        *b_Jet_PT;   //!
   TBranch        *b_Jet_Eta;   //!
   TBranch        *b_Jet_Phi;   //!
   TBranch        *b_Jet_Mass;   //!
   TBranch        *b_Jet_Ntrk;   //!
   TBranch        *b_Jet_BTag;   //!
   TBranch        *b_Jet_EhadOverEem;   //!
   TBranch        *b_Jet_Index;   //!
   TBranch        *b_Jet_size;   //!
   TBranch        *b_MissingET_;   //!
   TBranch        *b_MissingET_fUniqueID;   //!
   TBranch        *b_MissingET_fBits;   //!
   TBranch        *b_MissingET_MET;   //!
   TBranch        *b_MissingET_Phi;   //!
   TBranch        *b_MissingET_size;   //!

   ppWjets(TTree *tree=0);
   virtual ~ppWjets();
   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 ppWjets_cxx
ppWjets::ppWjets(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("ppWjets.root"
   );
      if (!f || !f->IsOpen()) {
         f = new TFile("ppWjets.root");
      }
      f->GetObject("LHCO",tree);

   }
   Init(tree);
}

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

Int_t ppWjets::GetEntry(Long64_t entry)
{
// Read contents of entry.
   if (!fChain) return 0;
   return fChain->GetEntry(entry);
}
Long64_t ppWjets::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 ppWjets::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("Event", &Event_, &b_Event_);
fChain->SetBranchAddress("Event.fUniqueID", Event_fUniqueID, &
b_Event_fUniqueID);
fChain->SetBranchAddress("Event.fBits", Event_fBits, &b_Event_fBits);
fChain->SetBranchAddress("Event.Number", Event_Number, &b_Event_Number);
fChain->SetBranchAddress("Event.Trigger", Event_Trigger, &
b_Event_Trigger);
fChain->SetBranchAddress("Event_size", &Event_size, &b_Event_size);
fChain->SetBranchAddress("Photon", &Photon_, &b_Photon_);
fChain->SetBranchAddress("Photon.fUniqueID", Photon_fUniqueID, &
b_Photon_fUniqueID);
fChain->SetBranchAddress("Photon.fBits", Photon_fBits, &b_Photon_fBits);
fChain->SetBranchAddress("Photon.PT", Photon_PT, &b_Photon_PT);
fChain->SetBranchAddress("Photon.Eta", Photon_Eta, &b_Photon_Eta);
fChain->SetBranchAddress("Photon.Phi", Photon_Phi, &b_Photon_Phi);
fChain->SetBranchAddress("Photon.EhadOverEem", Photon_EhadOverEem, &
b_Photon_EhadOverEem);
fChain->SetBranchAddress("Photon_size", &Photon_size, &b_Photon_size);
fChain->SetBranchAddress("Electron", &Electron_, &b_Electron_);
fChain->SetBranchAddress("Electron.fUniqueID", Electron_fUniqueID, &
b_Electron_fUniqueID);
fChain->SetBranchAddress("Electron.fBits", Electron_fBits, &
b_Electron_fBits);
fChain->SetBranchAddress("Electron.PT", Electron_PT, &b_Electron_PT);
fChain->SetBranchAddress("Electron.Eta", Electron_Eta, &b_Electron_Eta);
fChain->SetBranchAddress("Electron.Phi", Electron_Phi, &b_Electron_Phi);
fChain->SetBranchAddress("Electron.Charge", Electron_Charge, &
b_Electron_Charge);
fChain->SetBranchAddress("Electron.Ntrk", Electron_Ntrk, &
b_Electron_Ntrk);
fChain->SetBranchAddress("Electron.EhadOverEem", Electron_EhadOverEem, &
b_Electron_EhadOverEem);
fChain->SetBranchAddress("Electron_size", &Electron_size, &
b_Electron_size);
fChain->SetBranchAddress("Muon", &Muon_, &b_Muon_);
fChain->SetBranchAddress("Muon.fUniqueID", Muon_fUniqueID, &
b_Muon_fUniqueID);
fChain->SetBranchAddress("Muon.fBits", Muon_fBits, &b_Muon_fBits);
fChain->SetBranchAddress("Muon.PT", Muon_PT, &b_Muon_PT);
fChain->SetBranchAddress("Muon.Eta", Muon_Eta, &b_Muon_Eta);
fChain->SetBranchAddress("Muon.Phi", Muon_Phi, &b_Muon_Phi);
fChain->SetBranchAddress("Muon.Charge", Muon_Charge, &b_Muon_Charge);
fChain->SetBranchAddress("Muon.Ntrk", Muon_Ntrk, &b_Muon_Ntrk);
fChain->SetBranchAddress("Muon.PTiso", Muon_PTiso, &b_Muon_PTiso);
fChain->SetBranchAddress("Muon.ETiso", Muon_ETiso, &b_Muon_ETiso);
fChain->SetBranchAddress("Muon.JetIndex", Muon_JetIndex, &
b_Muon_JetIndex);
fChain->SetBranchAddress("Muon_size", &Muon_size, &b_Muon_size);
fChain->SetBranchAddress("Tau", &Tau_, &b_Tau_);
fChain->SetBranchAddress("Tau.fUniqueID", Tau_fUniqueID, &
b_Tau_fUniqueID);
fChain->SetBranchAddress("Tau.fBits", Tau_fBits, &b_Tau_fBits);
fChain->SetBranchAddress("Tau.PT", Tau_PT, &b_Tau_PT);
fChain->SetBranchAddress("Tau.Eta", Tau_Eta, &b_Tau_Eta);
```

```
   fChain->SetBranchAddress("Tau.Phi", Tau_Phi, &b_Tau_Phi);
   fChain->SetBranchAddress("Tau.Charge", Tau_Charge, &b_Tau_Charge);
   fChain->SetBranchAddress("Tau.Ntrk", Tau_Ntrk, &b_Tau_Ntrk);
   fChain->SetBranchAddress("Tau.EhadOverEem", Tau_EhadOverEem, &
   b_Tau_EhadOverEem);
   fChain->SetBranchAddress("Tau_size", &Tau_size, &b_Tau_size);
   fChain->SetBranchAddress("Jet", &Jet_, &b_Jet_);
   fChain->SetBranchAddress("Jet.fUniqueID", Jet_fUniqueID, &
   b_Jet_fUniqueID);
   fChain->SetBranchAddress("Jet.fBits", Jet_fBits, &b_Jet_fBits);
   fChain->SetBranchAddress("Jet.PT", Jet_PT, &b_Jet_PT);
   fChain->SetBranchAddress("Jet.Eta", Jet_Eta, &b_Jet_Eta);
   fChain->SetBranchAddress("Jet.Phi", Jet_Phi, &b_Jet_Phi);
   fChain->SetBranchAddress("Jet.Mass", Jet_Mass, &b_Jet_Mass);
   fChain->SetBranchAddress("Jet.Ntrk", Jet_Ntrk, &b_Jet_Ntrk);
   fChain->SetBranchAddress("Jet.BTag", Jet_BTag, &b_Jet_BTag);
   fChain->SetBranchAddress("Jet.EhadOverEem", Jet_EhadOverEem, &
   b_Jet_EhadOverEem);
   fChain->SetBranchAddress("Jet.Index", Jet_Index, &b_Jet_Index);
   fChain->SetBranchAddress("Jet_size", &Jet_size, &b_Jet_size);
   fChain->SetBranchAddress("MissingET", &MissingET_, &b_MissingET_);
   fChain->SetBranchAddress("MissingET.fUniqueID", MissingET_fUniqueID, &
   b_MissingET_fUniqueID);
   fChain->SetBranchAddress("MissingET.fBits", MissingET_fBits, &
   b_MissingET_fBits);
   fChain->SetBranchAddress("MissingET.MET", MissingET_MET, &
   b_MissingET_MET);
   fChain->SetBranchAddress("MissingET.Phi", MissingET_Phi, &
   b_MissingET_Phi);
   fChain->SetBranchAddress("MissingET_size", &MissingET_size, &
   b_MissingET_size);
   Notify();
}

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

```
   return 1;
}
#endif // #ifdef ppWjets_cxx
```

Listing 8: Header file for $pp \to W + \text{jets}$

The source file:

```
#define ppWjets_cxx

#include "ppWjets.h"
#include "global.h"

void ppWjets::Loop() {

//   In a ROOT session, you can do:
//      Root > .L ppWjets.C
//      Root > ppWjets t
//      Root > t.GetEntry(12); // Fill t data members with entry number 12
//      Root > t.Show();       // Show values of entry 12
//      Root > t.Show(16);     // Read and show values of entry 16
//      Root > t.Loop();       // Loop on all entries
//

//     This is the loop skeleton where:
//    jentry is the global entry number in the chain
//    ientry is the entry number in the current Tree
//  Note that the argument to GetEntry must be:
//    jentry for TChain::GetEntry
//    ientry for TTree::GetEntry and TBranch::GetEntry
//
//       To read only selected branches, Insert statements like:
// METHOD1:
//    fChain->SetBranchStatus("*",0);  // disable all branches
//    fChain->SetBranchStatus("branchname",1);  // activate branchname
// METHOD2: replace line
//    fChain->GetEntry(jentry);       //read all branches
//by  b_branchname->GetEntry(ientry); //read only this branch

      if (fChain == 0)
            return;
      Long64_t nentries = fChain->GetEntriesFast();

      // Create histogram to be filled
      TH1F* Wjets_jet_eta = new TH1F("jet_eta", "pp->W+jets", N_BINS, X_MIN
    , X_MAX);

      Long64_t nbytes = 0, nb = 0;

      // Loop over entries and fill histograms
      for (Long64_t jentry = 0; jentry < nentries; jentry++) {
            Long64_t ientry = LoadTree(jentry);
            if (ientry < 0)
                  break;

            nb = fChain->GetEntry(jentry);
            nbytes += nb;

            // Loop to apply a cut
            if (Jet_PT[0] > 200)
                  Wjets_jet_eta->Fill(Jet_Eta[0]);
      // if (Cut(ientry) < 0) continue;
      }

      // Normalize histogram by luminosity
```

```
        Double_t cross_sec = 2.144e+04;
        Wjets_jet_eta->Scale( cross_sec * luminosity_pb / N_EVENTS );

        // Set aesthetics
        Wjets_jet_eta->SetLineColor(kRed);
        Wjets_jet_eta->SetFillColor(kRed);
        Wjets_jet_eta->SetMarkerStyle(21);
        Wjets_jet_eta->SetMarkerColor(kRed);

        stackedhists->Add(Wjets_jet_eta);

       // Canvas is initialised in execComparejets.C before calling this file
       c1->cd(1);
       // Draw 1D plot
       stackedhists->Draw(histType_pad1);

       c1->cd(2);
       // Draw lego plot
       stackedhists->Draw(histType_pad2);
}
```

Listing 9: Source file for $pp \rightarrow W + \text{jets}$

0.3.3 $pp \to Z^0 + \text{jets}$

The header file:

```
//////////////////////////////////////////////////////////
// This class has been automatically generated on
// Tue Nov  1 16:18:19 2016 by ROOT version 6.02/02
// from TTree LHCO/ppZjets tree
// found on file: ppZjets.root
//////////////////////////////////////////////////////////

// Header file to include declarations, classes, variables for ppZjets.C
    macro

#ifndef ppZjets_h
#define ppZjets_h

#include "global.h"

class ppZjets {
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
    .
//   const Int_t kMaxEvent = 1;
//   const Int_t kMaxPhoton = 3;
//   const Int_t kMaxElectron = 2;
//   const Int_t kMaxMuon = 2;
//   const Int_t kMaxTau = 3;
//   const Int_t kMaxJet = 11;
//   const Int_t kMaxMissingET = 1;

   // Declaration of leaf types
   Int_t           Event_;
   UInt_t          Event_fUniqueID[1];   //[Event_]
   UInt_t          Event_fBits[1];   //[Event_]
   Long64_t        Event_Number[1];   //[Event_]
   Int_t           Event_Trigger[1];   //[Event_]
   Int_t           Event_size;
   Int_t           Photon_;
   UInt_t          Photon_fUniqueID[3];   //[Photon_]
   UInt_t          Photon_fBits[3];   //[Photon_]
   Double_t        Photon_PT[3];   //[Photon_]
   Double_t        Photon_Eta[3];   //[Photon_]
   Double_t        Photon_Phi[3];   //[Photon_]
   Double_t        Photon_EhadOverEem[3];   //[Photon_]
   Int_t           Photon_size;
   Int_t           Electron_;
   UInt_t          Electron_fUniqueID[3];   //[Electron_]
   UInt_t          Electron_fBits[3];   //[Electron_]
   Double_t        Electron_PT[3];   //[Electron_]
   Double_t        Electron_Eta[3];   //[Electron_]
   Double_t        Electron_Phi[3];   //[Electron_]
   Double_t        Electron_Charge[3];   //[Electron_]
   Double_t        Electron_Ntrk[3];   //[Electron_]
   Double_t        Electron_EhadOverEem[3];   //[Electron_]
   Int_t           Electron_size;
   Int_t           Muon_;
   UInt_t          Muon_fUniqueID[3];   //[Muon_]
   UInt_t          Muon_fBits[3];   //[Muon_]
   Double_t        Muon_PT[3];   //[Muon_]
   Double_t        Muon_Eta[3];   //[Muon_]
   Double_t        Muon_Phi[3];   //[Muon_]
   Double_t        Muon_Charge[3];   //[Muon_]
   Double_t        Muon_Ntrk[3];   //[Muon_]
   Double_t        Muon_PTiso[3];   //[Muon_]
   Double_t        Muon_ETiso[3];   //[Muon_]
   Int_t           Muon_JetIndex[3];   //[Muon_]
   Int_t           Muon_size;
   Int_t           Tau_;
   UInt_t          Tau_fUniqueID[4];   //[Tau_]
   UInt_t          Tau_fBits[4];   //[Tau_]
   Double_t        Tau_PT[4];   //[Tau_]
   Double_t        Tau_Eta[4];   //[Tau_]
   Double_t        Tau_Phi[4];   //[Tau_]
   Double_t        Tau_Charge[4];   //[Tau_]
   Double_t        Tau_Ntrk[4];   //[Tau_]
   Double_t        Tau_EhadOverEem[4];   //[Tau_]
   Int_t           Tau_size;
   Int_t           Jet_;
   UInt_t          Jet_fUniqueID[17];   //[Jet_]
   UInt_t          Jet_fBits[17];   //[Jet_]
   Double_t        Jet_PT[17];   //[Jet_]
   Double_t        Jet_Eta[17];   //[Jet_]
   Double_t        Jet_Phi[17];   //[Jet_]
   Double_t        Jet_Mass[17];   //[Jet_]
   Double_t        Jet_Ntrk[17];   //[Jet_]
   Double_t        Jet_BTag[17];   //[Jet_]
   Double_t        Jet_EhadOverEem[17];   //[Jet_]
   Int_t           Jet_Index[17];   //[Jet_]
   Int_t           Jet_size;
   Int_t           MissingET_;
   UInt_t          MissingET_fUniqueID[1];   //[MissingET_]
   UInt_t          MissingET_fBits[1];   //[MissingET_]
   Double_t        MissingET_MET[1];   //[MissingET_]
   Double_t        MissingET_Phi[1];   //[MissingET_]
   Int_t           MissingET_size;

   // List of branches
   TBranch        *b_Event_;   //!
   TBranch        *b_Event_fUniqueID;   //!
   TBranch        *b_Event_fBits;   //!
   TBranch        *b_Event_Number;   //!
   TBranch        *b_Event_Trigger;   //!
   TBranch        *b_Event_size;   //!
   TBranch        *b_Photon_;   //!
   TBranch        *b_Photon_fUniqueID;   //!
   TBranch        *b_Photon_fBits;   //!
   TBranch        *b_Photon_PT;   //!
   TBranch        *b_Photon_Eta;   //!
   TBranch        *b_Photon_Phi;   //!
   TBranch        *b_Photon_EhadOverEem;   //!
   TBranch        *b_Photon_size;   //!
   TBranch        *b_Electron_;   //!
   TBranch        *b_Electron_fUniqueID;   //!
   TBranch        *b_Electron_fBits;   //!
   TBranch        *b_Electron_PT;   //!
   TBranch        *b_Electron_Eta;   //!
   TBranch        *b_Electron_Phi;   //!
   TBranch        *b_Electron_Charge;   //!
   TBranch        *b_Electron_Ntrk;   //!
   TBranch        *b_Electron_EhadOverEem;   //!
   TBranch        *b_Electron_size;   //!
   TBranch        *b_Muon_;   //!
   TBranch        *b_Muon_fUniqueID;   //!
   TBranch        *b_Muon_fBits;   //!
   TBranch        *b_Muon_PT;   //!
   TBranch        *b_Muon_Eta;   //!
   TBranch        *b_Muon_Phi;   //!
   TBranch        *b_Muon_Charge;   //!
   TBranch        *b_Muon_Ntrk;   //!
   TBranch        *b_Muon_PTiso;   //!
   TBranch        *b_Muon_ETiso;   //!
   TBranch        *b_Muon_JetIndex;   //!
   TBranch        *b_Muon_size;   //!
   TBranch        *b_Tau_;   //!
   TBranch        *b_Tau_fUniqueID;   //!
   TBranch        *b_Tau_fBits;   //!
   TBranch        *b_Tau_PT;   //!
   TBranch        *b_Tau_Eta;   //!
   TBranch        *b_Tau_Phi;   //!
   TBranch        *b_Tau_Charge;   //!
   TBranch        *b_Tau_Ntrk;   //!
   TBranch        *b_Tau_EhadOverEem;   //!
   TBranch        *b_Tau_size;   //!
   TBranch        *b_Jet_;   //!
   TBranch        *b_Jet_fUniqueID;   //!
   TBranch        *b_Jet_fBits;   //!
   TBranch        *b_Jet_PT;   //!
   TBranch        *b_Jet_Eta;   //!
   TBranch        *b_Jet_Phi;   //!
   TBranch        *b_Jet_Mass;   //!
   TBranch        *b_Jet_Ntrk;   //!
   TBranch        *b_Jet_BTag;   //!
   TBranch        *b_Jet_EhadOverEem;   //!
   TBranch        *b_Jet_Index;   //!
   TBranch        *b_Jet_size;   //!
   TBranch        *b_MissingET_;   //!
   TBranch        *b_MissingET_fUniqueID;   //!
   TBranch        *b_MissingET_fBits;   //!
   TBranch        *b_MissingET_MET;   //!
   TBranch        *b_MissingET_Phi;   //!
   TBranch        *b_MissingET_size;   //!

   ppZjets(TTree *tree=0);
   virtual ~ppZjets();
   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 ppZjets_cxx
ppZjets::ppZjets(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("ppZjets.root"
   );
      if (!f || !f->IsOpen()) {
         f = new TFile("ppZjets.root");
      }
      f->GetObject("LHCO",tree);

   }
   Init(tree);
}

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

Int_t ppZjets::GetEntry(Long64_t entry)
{
// Read contents of entry.
   if (!fChain) return 0;
   return fChain->GetEntry(entry);
}
Long64_t ppZjets::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 ppZjets::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("Event", &Event_, &b_Event_);
fChain->SetBranchAddress("Event.fUniqueID", Event_fUniqueID, &
b_Event_fUniqueID);
fChain->SetBranchAddress("Event.fBits", Event_fBits, &b_Event_fBits);
fChain->SetBranchAddress("Event.Number", Event_Number, &b_Event_Number);
fChain->SetBranchAddress("Event.Trigger", Event_Trigger, &
b_Event_Trigger);
fChain->SetBranchAddress("Event_size", &Event_size, &b_Event_size);
fChain->SetBranchAddress("Photon", &Photon_, &b_Photon_);
fChain->SetBranchAddress("Photon.fUniqueID", Photon_fUniqueID, &
b_Photon_fUniqueID);
fChain->SetBranchAddress("Photon.fBits", Photon_fBits, &b_Photon_fBits);
fChain->SetBranchAddress("Photon.PT", Photon_PT, &b_Photon_PT);
fChain->SetBranchAddress("Photon.Eta", Photon_Eta, &b_Photon_Eta);
fChain->SetBranchAddress("Photon.Phi", Photon_Phi, &b_Photon_Phi);
fChain->SetBranchAddress("Photon.EhadOverEem", Photon_EhadOverEem, &
b_Photon_EhadOverEem);
fChain->SetBranchAddress("Photon_size", &Photon_size, &b_Photon_size);
fChain->SetBranchAddress("Electron", &Electron_, &b_Electron_);
fChain->SetBranchAddress("Electron.fUniqueID", Electron_fUniqueID, &
b_Electron_fUniqueID);
fChain->SetBranchAddress("Electron.fBits", Electron_fBits, &
b_Electron_fBits);
fChain->SetBranchAddress("Electron.PT", Electron_PT, &b_Electron_PT);
fChain->SetBranchAddress("Electron.Eta", Electron_Eta, &b_Electron_Eta);
fChain->SetBranchAddress("Electron.Phi", Electron_Phi, &b_Electron_Phi);
fChain->SetBranchAddress("Electron.Charge", Electron_Charge, &
b_Electron_Charge);
fChain->SetBranchAddress("Electron.Ntrk", Electron_Ntrk, &
b_Electron_Ntrk);
fChain->SetBranchAddress("Electron.EhadOverEem", Electron_EhadOverEem, &
b_Electron_EhadOverEem);
fChain->SetBranchAddress("Electron_size", &Electron_size, &
b_Electron_size);
fChain->SetBranchAddress("Muon", &Muon_, &b_Muon_);
fChain->SetBranchAddress("Muon.fUniqueID", Muon_fUniqueID, &
b_Muon_fUniqueID);
fChain->SetBranchAddress("Muon.fBits", Muon_fBits, &b_Muon_fBits);
fChain->SetBranchAddress("Muon.PT", Muon_PT, &b_Muon_PT);
fChain->SetBranchAddress("Muon.Eta", Muon_Eta, &b_Muon_Eta);
fChain->SetBranchAddress("Muon.Phi", Muon_Phi, &b_Muon_Phi);
fChain->SetBranchAddress("Muon.Charge", Muon_Charge, &b_Muon_Charge);
fChain->SetBranchAddress("Muon.Ntrk", Muon_Ntrk, &b_Muon_Ntrk);
fChain->SetBranchAddress("Muon.PTiso", Muon_PTiso, &b_Muon_PTiso);
fChain->SetBranchAddress("Muon.ETiso", Muon_ETiso, &b_Muon_ETiso);
fChain->SetBranchAddress("Muon.JetIndex", Muon_JetIndex, &
b_Muon_JetIndex);
fChain->SetBranchAddress("Muon_size", &Muon_size, &b_Muon_size);
fChain->SetBranchAddress("Tau", &Tau_, &b_Tau_);
fChain->SetBranchAddress("Tau.fUniqueID", Tau_fUniqueID, &
b_Tau_fUniqueID);
fChain->SetBranchAddress("Tau.fBits", Tau_fBits, &b_Tau_fBits);
fChain->SetBranchAddress("Tau.PT", Tau_PT, &b_Tau_PT);
fChain->SetBranchAddress("Tau.Eta", Tau_Eta, &b_Tau_Eta);
```

```
   fChain->SetBranchAddress("Tau.Phi", Tau_Phi, &b_Tau_Phi);
   fChain->SetBranchAddress("Tau.Charge", Tau_Charge, &b_Tau_Charge);
   fChain->SetBranchAddress("Tau.Ntrk", Tau_Ntrk, &b_Tau_Ntrk);
   fChain->SetBranchAddress("Tau.EhadOverEem", Tau_EhadOverEem, &
    b_Tau_EhadOverEem);
   fChain->SetBranchAddress("Tau_size", &Tau_size, &b_Tau_size);
   fChain->SetBranchAddress("Jet", &Jet_, &b_Jet_);
   fChain->SetBranchAddress("Jet.fUniqueID", Jet_fUniqueID, &
    b_Jet_fUniqueID);
   fChain->SetBranchAddress("Jet.fBits", Jet_fBits, &b_Jet_fBits);
   fChain->SetBranchAddress("Jet.PT", Jet_PT, &b_Jet_PT);
   fChain->SetBranchAddress("Jet.Eta", Jet_Eta, &b_Jet_Eta);
   fChain->SetBranchAddress("Jet.Phi", Jet_Phi, &b_Jet_Phi);
   fChain->SetBranchAddress("Jet.Mass", Jet_Mass, &b_Jet_Mass);
   fChain->SetBranchAddress("Jet.Ntrk", Jet_Ntrk, &b_Jet_Ntrk);
   fChain->SetBranchAddress("Jet.BTag", Jet_BTag, &b_Jet_BTag);
   fChain->SetBranchAddress("Jet.EhadOverEem", Jet_EhadOverEem, &
    b_Jet_EhadOverEem);
   fChain->SetBranchAddress("Jet.Index", Jet_Index, &b_Jet_Index);
   fChain->SetBranchAddress("Jet_size", &Jet_size, &b_Jet_size);
   fChain->SetBranchAddress("MissingET", &MissingET_, &b_MissingET_);
   fChain->SetBranchAddress("MissingET.fUniqueID", MissingET_fUniqueID, &
    b_MissingET_fUniqueID);
   fChain->SetBranchAddress("MissingET.fBits", MissingET_fBits, &
    b_MissingET_fBits);
   fChain->SetBranchAddress("MissingET.MET", MissingET_MET, &
    b_MissingET_MET);
   fChain->SetBranchAddress("MissingET.Phi", MissingET_Phi, &
    b_MissingET_Phi);
   fChain->SetBranchAddress("MissingET_size", &MissingET_size, &
    b_MissingET_size);
   Notify();
}

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

```
   return 1;
}
#endif // #ifdef ppZjets_cxx
```

Listing 10: Header file for $pp \to Z^0 + \mathrm{jets}$

The source file:

```
#define ppZjets_cxx

#include "ppZjets.h"
#include "global.h"

void ppZjets::Loop() {

//   In a ROOT session, you can do:
//      Root > .L ppZjets.C
//      Root > ppZjets t
//      Root > t.GetEntry(12); // Fill t data members with entry number 12
//      Root > t.Show();       // Show values of entry 12
//      Root > t.Show(16);     // Read and show values of entry 16
//      Root > t.Loop();       // Loop on all entries
//

//     This is the loop skeleton where:
//    jentry is the global entry number in the chain
//    ientry is the entry number in the current Tree
//  Note that the argument to GetEntry must be:
//    jentry for TChain::GetEntry
//    ientry for TTree::GetEntry and TBranch::GetEntry
//
//       To read only selected branches, Insert statements like:
// METHOD1:
//    fChain->SetBranchStatus("*",0);  // disable all branches
//    fChain->SetBranchStatus("branchname",1);  // activate branchname
// METHOD2: replace line
//    fChain->GetEntry(jentry);       //read all branches
//by  b_branchname->GetEntry(ientry); //read only this branch

      if (fChain == 0)
            return;
      Long64_t nentries = fChain->GetEntriesFast();

      // Create histogram to be filled
      TH1F* Zjets_jet_eta = new TH1F("jet_eta", "pp->Z+jets", N_BINS, X_MIN
    , X_MAX);

      Long64_t nbytes = 0, nb = 0;

      // Loop over entries and fill histograms
      for (Long64_t jentry = 0; jentry < nentries; jentry++) {
            Long64_t ientry = LoadTree(jentry);
            if (ientry < 0)
                  break;

            nb = fChain->GetEntry(jentry);
            nbytes += nb;

            // Loop to apply a cut
            if (Jet_PT[0] > 200)
                  Zjets_jet_eta->Fill(Jet_Eta[0]);
      // if (Cut(ientry) < 0) continue;
      }

      // Normalize histogram by luminosity
```

```
     Double_t cross_sec = 1.166e+04;
     Zjets_jet_eta->Scale( cross_sec * luminosity_pb / N_EVENTS );

     // Set aesthetics
     Zjets_jet_eta->SetLineColor(kGreen);
     Zjets_jet_eta->SetFillColor(kGreen);
     Zjets_jet_eta->SetMarkerStyle(21);
     Zjets_jet_eta->SetMarkerColor(kGreen);

     stackedhists->Add(Zjets_jet_eta);

    // Canvas is initialised in execComparejets.C before calling this file
    c1->cd(1);
    // Draw 1D plot
    stackedhists->Draw(histType_pad1);

    c1->cd(2);
    // Draw lego plot
    stackedhists->Draw(histType_pad2);
}
```

Listing 11: Source file for $pp \rightarrow Z^0 + \text{jets}$