CutLang[*]V2 introductory guide

S. Sekmen[1] and G. Unel[2] —01/31/20

[1]Kyungpook National University, Physics Dept., Daegu, South Korea
[2]University of California at Irvine, Department of Physics and Astronomy, Irvine, USA

Abstract

This note presents a quick introduction to the Version 2 implementation of CutLang, a domain specific language for cut based high energy physics data analysis description and its interpretation framework. CutLang allows to write analyses as a set of commands in human readable text files (adl files), which are interpreted by the framework at run time. It is implemented in C++ on top of ROOT classes and operates on particles based on Lorentz vectors. The version 2 uses the Lex/Yacc based approach for adl file processing. Main features of CutLang V2 are described, and two analysis examples are presented to demonstrate its usage. The initial experience with CutLang has shown that run time interpretation of human readable analysis description is a highly practical alternative in analysis writing that is worth exploring further.

1 Introduction

The steep learning curve for general purpose programming languages and experiment specific analysis frameworks erect a barrier between data and the physicist who may simply wish to try an analysis idea. These difficulties could be overcome by using analysis description language (ADL). It is a human readable, declarative domain specific language, capable of describing the analysis flow in a standard and unambiguous way, independent of any computing framework. An ADL decouples the mathematical and logical algorithm of a physics analysis from computing operations. Being declarative, it eliminates programming difficulties and errors, consequently allowing the analysis to be more efficient and flawless. CutLang V2 is such a language, processing events from ROOT files, using algorithms described in human readable analysis description files.

2 The ADL file structure

The analysis description text file is referred commonly as an `adl` file. The mass, energy and momentum are all written in Giga Electron Volt (GEV) and angles in radians. User comments and explanations should be preceded by a hash (#) sign. The command execution order is top to bottom. In this file, there are five possible sections, of which one is mandatory:
[initializations] [definitions1] [objects] [definitions2] commands .

initializations: This section contains so far 3 possible keywords and values separated by an equal sign. The last two lines in the same table refer to the lepton (electron or muon) triggers. Although their utilization will become apparent in the next subsection, it is worth noting at this point that MC simulation weights are not taken into account when the trigger value is set to data.

definitions1: This section has the potential to render the following analysis commands more understandable by introducing shortcuts like `Zhreco` for a hadronically reconstructed Z boson. It can only use the predefined keywords and particle definitions. It is also possible to define values like mH.

objects: This section can be used to define new objects based on predefined physics objects and shorthand notations declared in definitions1.

definitions2: This section has the potential to render the following analysis commands even more understandable by introducing more shortcuts. It can use the newly defined objects in the previous section and predefined particles.

commands: This section is mandatory with a name and at least one command. A command is either a selection criteria or a special instruction to include Monte Carlo weight factors or to fill histograms. Currently both one and two dimensional histograms are accepted.

[*]The version 2 of this software also contains contributions by A.M. Toon, A. Paul, B. Gokturk, N. Ravel, B.Orgen

Table 1: initialization keywords and possible values

Keyword	Explanation
SkipHistos	Display (=1) or not (=0) the histograms in final efficiency table
TRGm	0=no trigger, 1=trigger for data for muons, 2=Monte Carlo trigger for muons
TRGe	0=no trigger, 1=trigger for data for electrons, 2=Monte Carlo trigger for electrons

3 Predefined physics objects

Some basic physics objects and their properties are already defined in *CutLang*. The particles are initially sorted per decreasing transverse momentum and their indices start at zero. Both python type and latex type notations are accepted, the former with square brackets and the latter with an underline character. The base particle types and notation examples are given in Table 2. There are two object types that merit special attention: The lepton and the neutrino cases. The `LEP` keyword is generic and reduced to an electron or to a muon depending on the trigger choice made in Table 1. This helps the physicist avoiding two algorithm sections, one for electron and other muon triggered analyses. The second object type is more delicate and it is related to the taming of the neutrino escaping from the detector. It has been shown that at LHC energies and beyond, for which this tool is intended, the W bosons are generally produced with a sufficient boost such that in the leptonic decays, the pseudorapidity of the charged lepton is not very different from the chargeless one [R]. Therefore this particular physics object benefits from this approximation to define a massless and chargeless particle with transverse momentum and azimuthal angle (ϕ) values extracted from the Missing Transverse Energy (MET) measurements. The pseudorapidity, however, is taken equal to that of the charged lepton with the same particle index.

Table 2: Basic physics object nomenclature in *CutLang*

Name	Keyword	First object		Second object		$j+1^{th}$ object
Electron	ELE	ELE[0]	ELE_0	ELE[1]	ELE_1	ELE_j
Muon	MUO	MUO[0]	MUO_0	MUO[1]	MUO_1	MUO_j
Tau	TAU	TAU[0]	TAU_0	TAU[1]	TAU_1	TAU_j
Lepton	LEP	LEP[0]	LEP_0	LEP[1]	LEP_1	LEP_j
Photon	PHO	PHO[0]	PHO_0	PHO[1]	PHO_1	PHO_j
Jet	JET	JET[0]	JET_0	JET[1]	JET_1	JET_j
Fat Jet	FJET	FJET[0]	FJET_0	FJET[1]	FJET_1	FJET_j
b-tagged Jet	BJET	BJET[0]	BJET_0	BJET[1]	BJET_1	BJET_j
light Jet	QGJET	QGJET[0]	QGJET_0	QGJET[1]	QGJET_1	QGJET_j
Neutrino	NUMET	NUMET[0]	NUMET_0	NUMET[1]	NUMET_1	NUMET_j
MET	METLV	METLV[0]	METLV_0	—	—	—
generator particle	GEN	GEN[0]	GEN_0	GEN[1]	GEN_1	GEN_j

4 Predefined functions

The functions are mostly case insensitive, however they are written here in a certain syntax for reading clarity. External functions can be downloaded and added to CutLang library. Such an operation is described in Appendix A.

4.1 Particle related functions

There are two classes of functions in CutLang V2, the ones directly related to Lorentz Vectors such as Mass, Rapidity etc, and the ones related to other variables found in some commonly used ntuples. In both cases, both the function syntax with parentheses and the attribute syntax with curly braces can be used. The syntaxes are given in Tables 3 and 4. One should note that in CutLang V2, it is not necessary to use a + sign to add two particles, but it is also an accepted notation. Additionally, the last three functions in Table 3 require two comma separated particles.

4.2 Mathematical functions

Trigonometric and logarithmic functions are implemented with their usual meanings. There are few functions to pay attention to, such as the minimization and maximization functions which are discussed in subsection 4.5.

Table 3: Lorentz vector function nomenclature in *CutLang*.

Meaning	Syntax 1	Syntax 2
Mass of	`m( )`	`{ }m`
Charge of	`q( )`	`{ }q`
Phi of	`Phi( )`	`{ }Phi`
Eta of	`Eta( )`	`{ }Eta`
Absolute value of Eta of	`AbsEta( )`	`{ }AbsEta`
Rapidity of	`Rep( )`	`{ }Rep`
Pt of	`Pt( )`	`{ }Pt`
Pz of	`Pz( )`	`{ }Pz`
Energy of	`E( )`	`{ }E`
Momentum of	`P( )`	`{ }P`
Angular distance between	`dR( )`	`{ }dR`
Phi difference between	`dPhi( )`	`{ }dPhi`
Eta difference between	`dEta( )`	`{ }dEta`

Table 4: Ntuple variable function nomenclature in *CutLang*.

Meaning	Syntax 1	Syntax 2
Missing transverse energy in the event	`MET`	–
sum of jet transverse momenta	`HT( )`	–
PDGID of a particle	`PDGID( )`	`{ }PDGID`
Charge of a particle	`btagDeepB( )`	`{ }btagDeepB`
is the jet b tagged?	`bTag( )`	`{ }bTag`
Soft Drop mass of a jet	`msoftdrop( )`	`{ }msoftdrop`
N-subjetiness variable 1	`tau1( )`	`{ }tau1`
N-subjetiness variable 2	`tau2( )`	`{ }tau2`
N-subjetiness variable 3	`tau3( )`	`{ }tau3`
partitioning of jets into two hemi-spheres	`fmegajets( )`	`{ }fmegajets`
Razor variable MR	`fMR( )`	`{ }fMR`
Razor variable MTR	`fMTR( )`	`{ }fMTR`
Leptonic diTau invariant mass	`fMTauTau( )`	`{ }fMTauTau`
transverse impact parameter	`dxy( )`	`{ }dxy`
longitudinal impact parameter	`dz( )`	`{ }dz`
lepton identification variable (CMS NanoAOD only)	`softId( )`	`{ }softId`
relative isolation for leptons (CMS NanoAOD only)	`miniPFRelIsoAll( )`	`{ }miniPFRelIsoAll`
MVA based tau ID (CMS NanoAOD only)	`dMVAnewDM2017v2( )`	`{ }dMVAnewDM2017v2`

The Heaviside step function or the unit step function, is a discontinuous function, named after Oliver Heaviside, whose value is zero for negative arguments and one for positive arguments. The size / count function returns the number of elements of a given set, such as the number of electrons.

Table 5: mathematical and logical operators

Meaning	Operator	Meaning	Operator
number of	Size( ) Count() NumOf()	absolute value	abs()
tangent	tan()	hyperbolic tangent	tanh()
sine	sin()	hyperbolic sine	cosh()
cosine	cos()	hyperbolic cosine	sinh()
natural exponential	exp()	natural logarithm	log()
square root	sqrt()	Heaviside step function	hstep()
as close as possible	~=	usual meaning	+ - / *
as far away as possible	!=	to the power	^

4.3 Comparators, range and logical functions

CutLang understands the basic mathematical comparison expressions and logical operations. Therefore `C/C++` operator notations and their Fortran counterparts are recognized and correctly interpreted. Additionally square braces are used to define inclusive or exclusive ranges. The available comparators can be found in Table 6.

Table 6: Range comparators in *CutLang*

Keywords	Explanation
`> >= == <= <`	usual meaning
`GT GE EQ LE LT`	usual meaning
`<> NE`	not equal
`[ ]`	in the interval
`] [`	not in the interval
`NOT`	logical not
`AND and &&`	logical and
`OR or ||`	logical or

4.4 Ternary operator

Application of conditional selection criteria is available, including nested statements. The C++ syntax is assumed: condition ? true-case : false-case . An example is given below, if the number of muonsVeto particles is one, than the MTm quantity should be less than 100 otherwise MTe quantity should be less than 100.

```
Size(muonsVeto) == 1 ?  MTm < 100 :  MTe < 100
```

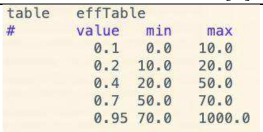
4.5 χ^2 minimization

In an analysis with a multitude of objects of the same type, the analyst could search for the best combination defined by some criterion. A typical example, used in fully hadronic $t\bar{t}$ reconstruction would be to find the jet combination that would yield the best W boson mass, or to find the two charged leptons that would result in the best Z boson mass. The need for such a search can be expressed in *CutLang* using two special comparison operators: `~=` and `!=` . The former is used in the sense of "as close as possible to" whereas the latter for calculation "as far as possible from". These two operators can be used to express χ^2 like operations. The indices of the particles in such a search are to be given as negative. For example, the statement "find two leptons with a combined invariant mass as close to 90.1 GeV" has its equivalent in *CutLang* notation as `{ LEP_-1 LEP_-1 }m ~= 90.1` . In such cases *CutLang* finds such a pair of particles and stores per event for possible later use. However the analyzer should not use negative indices directly inside the algorithm section. It is a much better practice that improves readability to define a new object such as `define ZLepRec = LEP[-1] LEP[-1]`. This definition can be used in histograms or in other selection criteria such as selecting the charge of the found lepton pair etc. If another particle of the same type (e.g. another lepton) is to be found, it is necessary to use a different but still negative index value. This concept will be further explained in section 7.2 with two examples.

5 Shortcuts and other declarations

Keywords can be assigned to constants (e.g. Z boson mass) or variables (e.g. angular variables between objects, mass of the Z boson reconstructed from two leptons, etc.). Simple explanations and examples are given in Table 7. Similarly, objects can be defined from already existing objects (e.g. defining b-tagged jets from high transverse momentum jets) or from derived objects. The necessary keywords and their usage are discussed in Table 8.

Table 7: Simple definitions

Keywords	argument1	symbol[1]	argument1	Example
`def define`	name	: =	value	define mZprime = 500
`def define`	name	: =	function	define mTop1 : m(Top1)
`def define`	name	: =	particle(s)	define Zreco : ELE[0] ELE[1]
`table`	name	list of value min max triplets		

```
table   effTable
#       value   min     max
          0.1   0.0    10.0
          0.2  10.0    20.0
          0.4  20.0    50.0
          0.7  50.0    70.0
          0.95 70.0    1000.0
```

Table 8: Object definitions

Keywords	argument1	symbol[2]			argument1	Example
`obj object`	new object name	`:`	`using`	`take`	base object name	`object goodEle : ELE`

5.1 Defining new objects

New objects can be declared using the syntax in Table 8, however this is usually not enough. Some selection commands are also needed to clarify the difference between the base object class and the derived objects. A typical example is seen in algorithm box 1.

Algorithm 1 A simple example for new physics object definition

```
object AK4jets take JET
    select {JET_}Pt > 30
    select {JET_}AbsEta < 2.4
```

It is also possible to create a group out of different leptons or their derived objects. This is achieved using the `Union` keyword, the utilization example is given in algorithm box 2. This particular case of new object creation doesn't use any selection as in the previous box.

Algorithm 2 Creation of a united object

```
object leps :  Union( MUO , ELE, TAU) #add all leptons into a set
object gleps :  Union( goodEle , goodMuo ) #add all good electrons and good muons into
another set
```

5.2 All possible combinations

Sometimes in an analysis algorithm it might be necessary to request all possible combinations of objects and select the final "good" combination(s) based on certain criteria that should be calculated at run time. CutLang V2 provides this functionality. This is best explained using an example problem. Therefore lets assume that we have an event with 5 jets and we would like to reconstruct hadronic Z bosons, hZs. What are the combinations? Numbering the jets from 1 to 5, some possibilities are given in Table 9 left side. It is obvious that not all possibilities are listed, and finally only one possibility can be true: after all a jet can not be used to reconstruct two different Z bosons. On top of this, other requirements might be applied to further restrict the possible Z candidates. For example there might be a pseudorapidity range limit on each candidate, the transverse momentum of the jets forming the Z boson could be limited, the angular separation between the hadronic Z candidate and the first constituent jet might be limited, and finally the invariant mass of the Z candidate might be requested to be in a certain range. After all these restrictions the same initial set might be reduced to Table 9 right side. The candidates that didn't pass the requirements are shown as stroked out.

Table 9: Combination example

possibility ID	Zhadronic	Zhadronic
1	12	34
2	12	35
3	12	45
4	13	24
5	13	25
6	13	45
...	...	...

possibility ID	Zhadronic	Zhadronic
1	12	~~34~~
2	12	35
3	12	~~45~~
4	13	24
5	13	25
6	13	45
...	...	...

[1]both : and = can be used interchangeably
[2]each be used interchangeably

Algorithm 3 Combinatorics object example

```
object hZs :  COMB( jets[-1] jets[-2] ) alias ahz # the candidate is temporarily called ahz
    select { ahz }AbsEta < 3.0
    select {jets[-2] }Pt > 2.1
    select {jets[-1] }Pt > 5.1
    select {jets[-1], ahz }dR < 0.6 # dR between ahz and its constituent 1, apply to all
    select { ahz }m [] 10 200
```

This example can be written in CutLang using the ADL notation in algorithm box 3. In order to activate this new object, and eliminate the combinations that do not satisfy the requirements, one has to put a selection command into the running algorithm (or region), this could be, for example, to have at least two hadronic Z candidates per event:

```
algo testCombinations
  select Size(jets) >= 2 #we need at least 2 jets for a Z boson
  select Count(hZs) >= 2 #the object name is used, not the temporary alias.
```

As indicated by Table 9 right side, there are still multiple possibilities, such as rows 2, 4 and 5. To further reduce these by killing the overlapping candidates and leave a single valid one, some sort of ideal condition should be specified. This can be achieved using the previously discussed χ^2 minimization. As an example case, lets require the masses of the both candidates to be as close as possible to the known Z mass. Therefore, the final algorithm is given in algorithm box 4.

Algorithm 4 Combinatorics final example

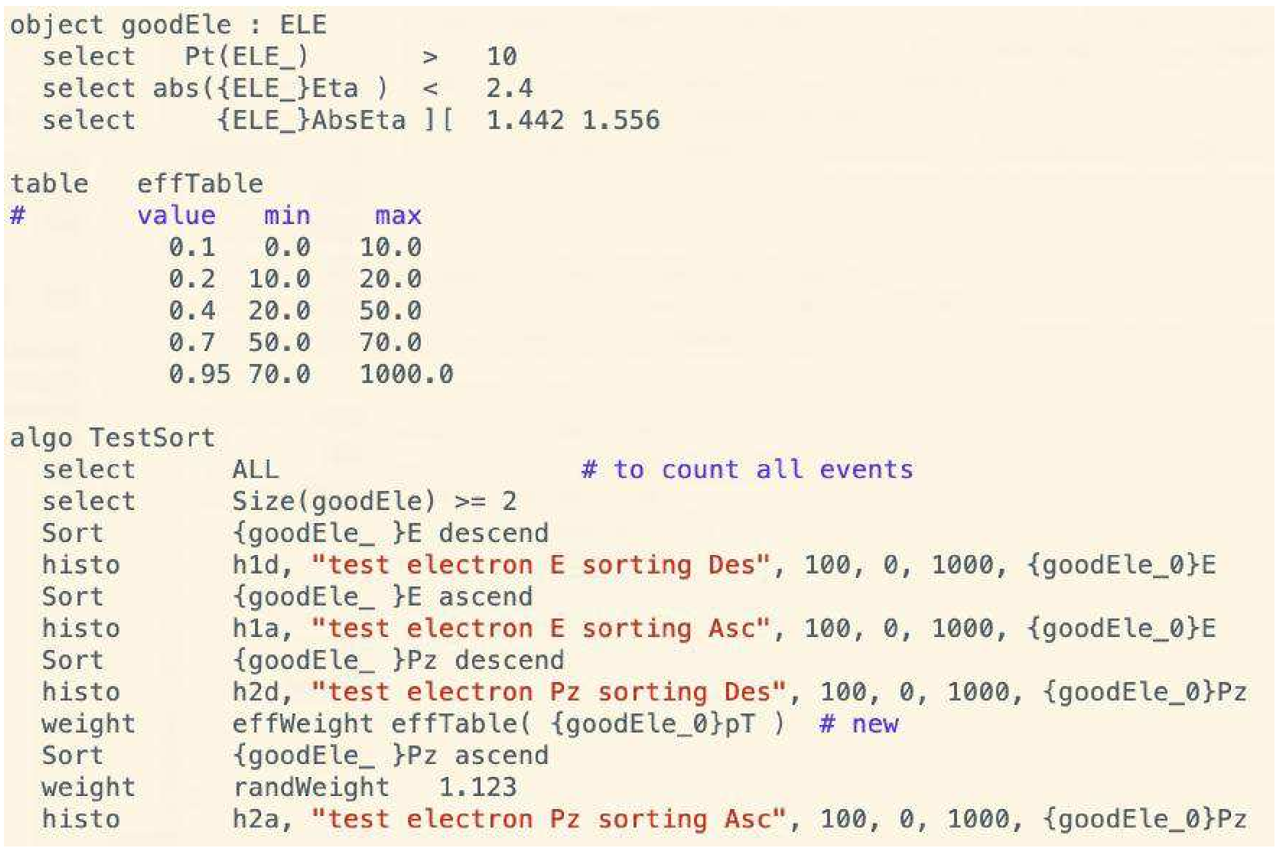
```
object hZs :  COMB( jets[-1] jets[-2] ) alias ahz # the candidate is temporarily called ahz
    select { ahz }AbsEta < 3.0
    select {jets[-2] }Pt > 2.1
    select {jets[-1] }Pt > 5.1
    select {jets[-1], ahz }dR < 0.6 # dR between ahz and its constituent 1, apply to all
    select { ahz }m [] 10 200

define zham :  {hZs[-1]}m
define zhbm :  {hZs[-2]}m
define chi2 :  (zham - 91.2)^2 + (zhbm - 91.2)^2

Algo testCombinations
    select ALL # to count all events # count number size are all the same.
    select Size(jets) >= 2 # we need at least 2 jets for a Z boson
    select Count(hZs) >= 2 # we need at least 2 Zhad candidates.
    select chi2 ~= 0 # we kill here overlapping candidates.  .
```

6 Selection and histogramming

The histogramming follow closely the ROOT notation. A simple 1D histogram should have a name, like h1mReco, and a list of parameters separated by commas. The explanation of the histogram contents should be given in quotation marks, "Z candidate mass (GeV)", the number of bins, lower and upper limits as numbers e.g. 100, 0, 200, and finally the quantity to histogram e.g. {ELE_0 ELE_1}m . A similar syntax is also used for the 2D histograms: hmTopWH2 , "Top W correlation (GeV)", 50, 50, 150, 70, 0, 700, m(WH2), mTop2 . An example algorithm showing sorting and histogramming commands applied on a user defined object is available in Figure 1.

Table 10: Selection and related keywords

Keywords	argument	Explanation
`cmd cut select command`	condition	accepts the events according to the condition
`reject`	condition	rejects the events according to the condition
`algo region algorithm`	name	starts a new algorithm / region with the name given
`histo`	name details	fills the named histogram according to given details
`weight`	name table(function)	applies a user weight according to the given function
`Sort`	function Descend or Ascend	sorts the particle set in the function in selected way
`save`	filename[3]	saves the surviving events in LVL0 format



```
object goodEle : ELE
  select   Pt(ELE_)        >   10
  select abs({ELE_}Eta )   <   2.4
  select     {ELE_}AbsEta ][   1.442 1.556

table   effTable
#       value   min    max
          0.1   0.0   10.0
          0.2  10.0   20.0
          0.4  20.0   50.0
          0.7  50.0   70.0
          0.95 70.0   1000.0

algo TestSort
  select      ALL                   # to count all events
  select      Size(goodEle) >= 2
  Sort        {goodEle_ }E descend
  histo       h1d, "test electron E sorting Des", 100, 0, 1000, {goodEle_0}E
  Sort        {goodEle_ }E ascend
  histo       h1a, "test electron E sorting Asc", 100, 0, 1000, {goodEle_0}E
  Sort        {goodEle_ }Pz descend
  histo       h2d, "test electron Pz sorting Des", 100, 0, 1000, {goodEle_0}Pz
  weight      effWeight effTable( {goodEle_0}pT )  # new
  Sort        {goodEle_ }Pz ascend
  weight      randWeight   1.123
  histo       h2a, "test electron Pz sorting Asc", 100, 0, 1000, {goodEle_0}Pz
```

Figure 1: An example algorithm including sorting and 1D histogram examples on a user defined object

6.1 Conditions

The conditions based on which an event can be accepted or rejected are usually written in the form of functions applied on particles complemented by a comparison operator and a limit value. An example would be "`select Size(goodEle) >= 2`" . However there are some special keywords that require further discussion. These are explained in Table 11.

Table 11: Special Conditions in *CutLang*

Keywords	Example	Explanation
`ALL`	`select ALL`	accept all events
`LEPsf`	`cmd LEPsf`	apply leptonic scale factor to MC events
`bTagSF`	`cmd bTagSF`	apply b-jet tagging scale factor to MC events

7 Two analysis examples

7.1 Z-boson reconstruction

The first simple example is the reconstruction of the Z boson mass from two charged leptons. This simple algorithm in *CutLang* notation is given in Figure 2 left side. The first section of the analysis defines new objects based on electrons and muons for their proper observability. The first command, `ALL`, will accept all events and provides a simple way for selection efficiency calculation and normalization as an internal histogram is filled automatically after each *CutLang* command. The first real selection is requiring events with only two leptons, needed for a simple Z boson reconstruction. Among the reconstructed Z boson candidates, only opposite charged ones are selected as the reconstructed particle is chargeless, and only those in the mass window of 70 to 120 GeV are accepted. Finally the invariant mass of the surviving lepton pairs are filled in a histogram which can be found in the same figure, right side. There are two regions or algorithms which are executed in parallel, one for electrons and the other for muons.

[3]without the . and the extension name

```
object goodEle using ELE
  select  Pt(ELE)        >   10
  select abs({ELE}Eta)   <   2.4
  select     {ELE}AbsEta ][  1.442 1.556

object myMu : MUO
  select Pt(MUO) > 20
  select abs({MUO}Eta)   <   2.5

define ZrecE : goodEle[0] goodEle[1]
define ZrecM :    myMu[0]    myMu[1]

region   testE
  select      ALL              # to count all events
  select      Size(ELE)     >= 2  # events with 2 or more electrons
  select      Size(goodEle) >= 2  # events with 2 or more electrons
  select      {ZrecE}q == 0       # Z is neutral
  select      {ZrecE}m [] 70 120  # mass window
  histo       h1mRecE, "Z ee candidate mass (GeV)", 100, 0, 200, {ZrecE}m

region   testM
  select      ALL              # to count all events
  select      Size(MUO) >= 2  # events with 2 or more electrons
  select      Size(myMu) >= 2  # events with 2 or more electrons
  select      {ZrecM}q == 0       # Z is neutral
  select      {ZrecM}m [] 70 120  # mass window
  histo       h1mRecM, "Z mm candidate mass (GeV)", 100, 0, 200, {ZrecM}m
```

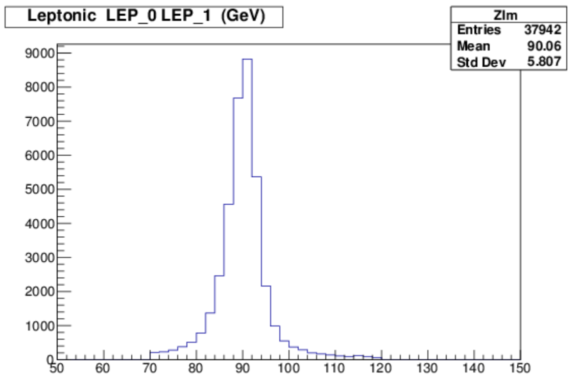


Figure 2: Z boson reconstruction simple example

The next example is very similar, with the exception that it is illustrating the utilization of the search method. The reconstruction algorithm allows any event with two or more leptons and searches the pair that would give an invariant mass close to that of a Z boson. In this particular sample, there are about 3000 events with more than two electrons. As the search command doesn't reject any events by itself, a mass window could be defined as an additional selection criterion. The rest of the algorithm is as before and the resulting histogram is shown in the same figure, right hand side.

```
object goodEle using ELE
  select  Pt(ELE)        >   10
  select abs({ELE}Eta)   <   2.4
  select     {ELE}AbsEta ][  1.442 1.556

define ZrecE : goodEle[-1] goodEle[-1]

region   testE
  select      ALL                    # to count all events
  select      Size(ELE)     >= 2    # events with 2 or more electrons
  select      Size(goodEle) >= 2    # events with 2 or more electrons
  select      {ZrecE}m      ~= 91.2 # find two best candidates
  select      {ZrecE}q      == 0    # Z is neutral
  histo       h1mRecE, "Z ee candidate mass (GeV)", 100, 0, 200, {ZrecE}m
```

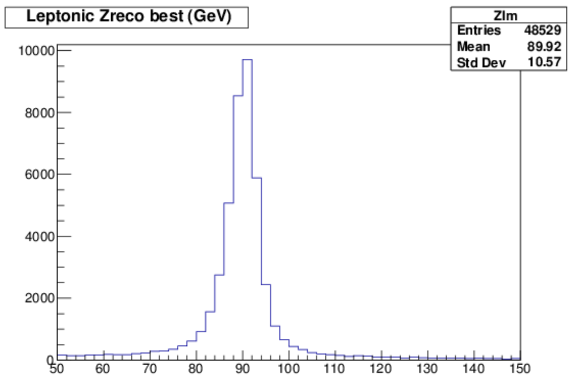


Figure 3: Z boson reconstruction example with automatic search

7.2 Top quark pair production

$t\bar{t}$ production is one of the main processes we work with at LHC. The final states in the fully hadronic mode has at least 6 jets, two of which could be b-tagged. Although it is possible to reconstruct top quark masses using b-tagging, it is not often desirable because of the low tagging efficiency. However, it is possible to reconstruct the top quark invariant masses without using b-tagging. This method consists of minimizing the χ^2 defined by equation (1). It simply is the difference of the W's mass difference from the known value, squared added to the squared difference between the two top quark candidates.

$$\chi^2 = \frac{(m_{b_1 j_1 j_2} - m_{b_2 j_3 j_4})^2}{\sigma^2_{\Delta m_{bjj}}} + \frac{(m_{j_1 j_2} - m_W^{MC})^2}{\sigma^2_{\Delta m_W^{MC}}} + \frac{(m_{j_3 j_4} - m_W^{MC})^2}{\sigma^2_{\Delta m_W^{MC}}} \tag{1}$$

This algorithm can be implemented in CutLang as presented in Figure 4. The resulting plots can be seen in figure, 5.

```
define   WH1  : JET[-1] JET[-1] # - index means I do not know yet which jets to take
define   WH2  : JET[-3] JET[-3] # -3 here is different than -1, so these are two OTHER jets.
# --------------> -3 & -3 repeated indices in the same hadronic W
# --------------> this means the order is NOT important
# --------------> Jet1 + Jet2 = Jet2 + Jet1

###      chi2 for W finder
define   Wchi2 : (({WH1}m - 80.4)/2.1)^2 + (({WH2}m - 80.4)/2.1)^2

## top quarks without b tagging
define   Top1 : WH1 JET[-2] # take another jet
#----> 12 3 toplamakla, 13 2 toplamak ayni degil. cunku ilkinde bj candidate 3 ikincisinde 2 numarali jetler.
define   Top2 : WH2 JET[-8]
define   mTop1 : m(Top1)
define   mTop2 : m(Top2)
###      chi2 for top finder
define   topchi2 : ((mTop1 - mTop2)/4.2)^2
define   totchi : Wchi2 + topchi2

algo besttop
  select   ALL                 # to count all events
  select   Size(JET) >= 6      # at least 6 jets
  select   MET < 100           # no large MET
  select   totchi  ~= 0  # find the tops and ws
  histo    hchi , "Histo of chi2", 50, 0, 500, totchi
  select   totchi < 10.5 # small than a certain value of chi2
  histo    hmWH1 , "Hadronic W reco (GeV)", 50, 50, 150, m(WH1)
  histo    hmWH2 , "Hadronic W reco (GeV)", 50, 50, 150, m(WH2)
  histo    hmTop1 , "Hadronic top reco (GeV)", 70, 0, 700, mTop1
  histo    hmTop2 , "Hadronic top reco (GeV)", 70, 0, 700, mTop2
  histo    hWbR1   , "Angular distance between W1 and bjet1", 70, 0, 7,  dR(WH1, JET[-2])
  histo    hWbR2   , "Angular distance between W2 and bjet2", 70, 0, 7,  dR(WH2, JET[-8])
  select   dR(WH1 , JET[-2] ) > 0.6
  select   dR(WH2 , JET[-8] ) > 0.6
  histo    hmASTop1 , "Hadronic top reco after cut(GeV)", 70, 0, 700, mTop1
  histo    hmAsTop2 , "Hadronic top reco after cut(GeV)", 70, 0, 700, mTop2
```

Figure 4: Fully hadronic top quark pair example with automatic search

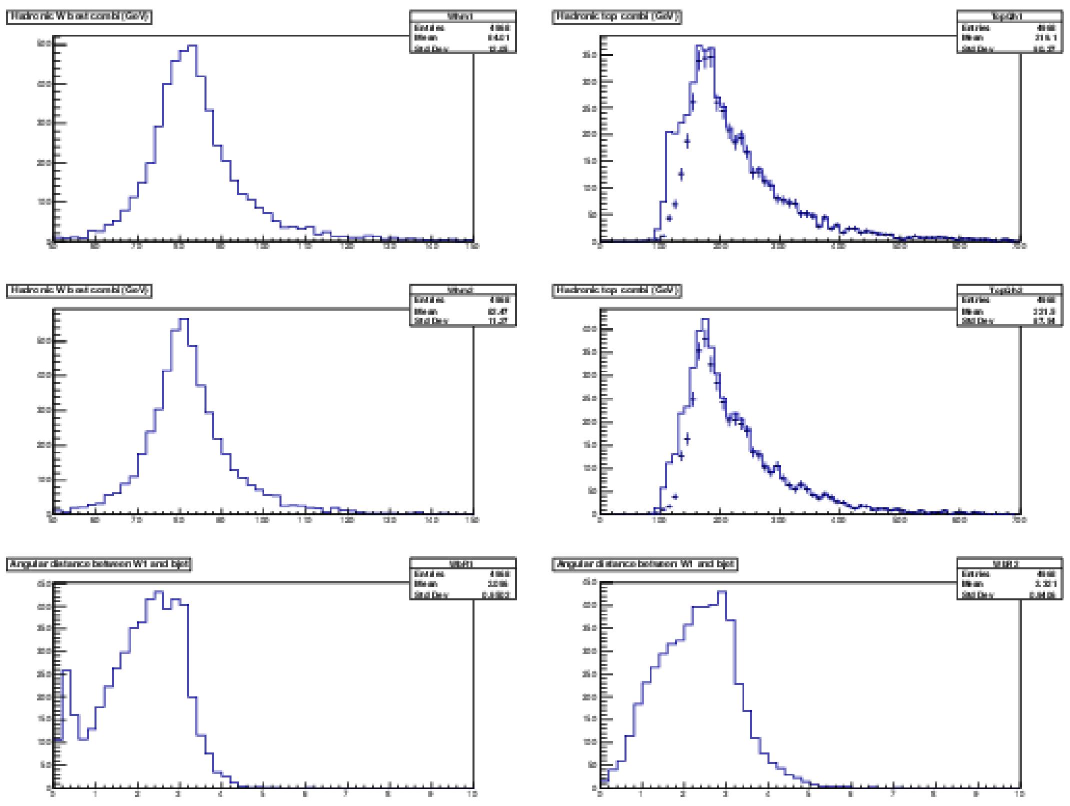


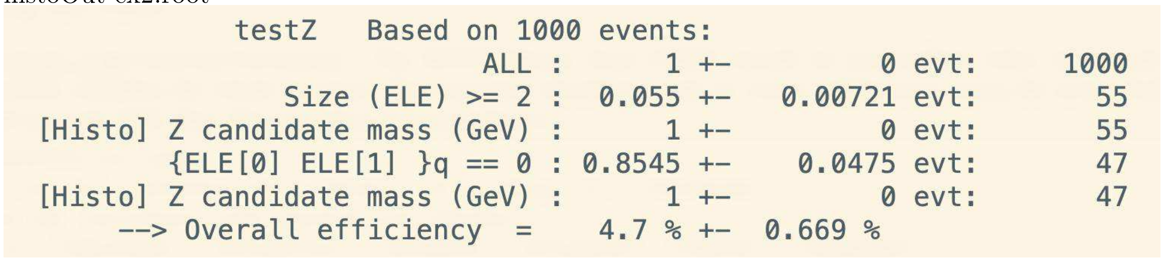
Figure 5: Fully hadronic boson reconstruction results

8 Compiling and Running the code

The code lives in `https://github.com/unelg/CutLang`. The ROOT library from CERN should be preinstalled. After downloading the source code, the `make` command executed in `CLA` subdirectory compiles the whole program. The execution should be made in `runs` subdirectory using the script `CLA.sh` . In this subdirectory there are a multitude of example ADL files. The execution command line is

```
./CLA.sh ROOTfile_name ROOTfile_type
          [-h] -i|--inifile -e|--events -s|--start -h|--help -v|--verbose
where ROOT file type can be (without the quotation marks):
       "LHCO" "FCC" "LVL0" "DELPHES" "ATLASVLL" "ATLMIN" "ATLASOD" "CMSOD" "CMSNANO"
```

histoOut-ex2.root



```
              testZ   Based on 1000 events:
                             ALL :      1 +-        0 evt:     1000
                    Size (ELE) >= 2 :  0.055 +-  0.00721 evt:       55
  [Histo] Z candidate mass (GeV) :      1 +-        0 evt:       55
        {ELE[0] ELE[1] }q == 0 : 0.8545 +-   0.0475 evt:       47
  [Histo] Z candidate mass (GeV) :      1 +-        0 evt:       47
       --> Overall efficiency  =    4.7 % +-  0.669 %
```

Appendix A: Adding an external function

–To be written–