

Documentation of the project: ISR jet tagging

Author:

Andrés Felipe García Albarracín

Advisor:

Juan Carlos Sanabria, Ph.D.

July 14, 2015

Contents

1	Introduction	1
2	Simulation chain	3
2.1	Usage of MadGraph 5.2	4
2.2	Usage of Pythia 8.2	7
2.2.1	Code Usage	7
2.2.2	The code	10
2.2.3	Pythia ntuple generation	11
2.3	Usage of Delphes 3.2	12
2.4	Integration of MadGraph 5.2 + Pythia 8.2 + Delphes 3.2 . . .	13
2.5	Example of the integration scripts	15
3	Analysis programs	17
3.1	Preparation of the codes	17
3.2	The ISR jet tagging method	20
3.2.1	The method	20

3.2.2	From probability density functions to normalized histograms	22
3.2.3	The Algorithm	23
3.2.4	The code	25
3.3	Matching algorithm	26
3.4	ISR jet analysis code	28
3.5	Histograms code	29
3.6	Example of the usage of the codes	30
4	Software tools	31
4.1	C++ and ROOT Makefile	32
4.2	Using Eclipse	33
	Appendices	41
A	Simulation codes	43
A.1	Pythia code	43
A.2	Integration scripts	49
A.2.1	Configuration script: <code>config_Integration.ini</code>	49
A.2.2	Execution script: <code>script_Integration.sh</code>	51
B	Analysis codes	55
B.1	Tagging algorithm	55
B.2	Matching algorithm	78

<i>CONTENTS</i>	iii
B.3 ISR jet analysis	90
B.4 Creating histograms	122
Bibliography	138

Chapter 1

Introduction

During the last semester of 2014, I made my Undergraduate Thesis Project entitled “*Design of algorithms to identify high momentum Initial State Radiation (ISR) Jets in proton – proton collision events*”, under the supervision of Juan Carlos Sanabria, Ph.D.. As the name suggests, the project consisted in the proposal of an algorithm to identify ISR jets. Due to the promising results, I was employed during the first semester of 2015 under the charge “Joven Investigador” of COLCIENCIAS in order to improve the initially obtained results. Throughout this time, several codes and programs were developed. To encourage the continuation of this project, this report has been written with a summary of all the technical work done so far.

In practical matters, one of the main drawbacks of Quantum Field Theory (QFT) is the inherent difficulty of its calculations. Feynman diagrams are not easy to solve and specially when high orders are involved. Consequently, the usage of algorithms and computer simulations have played an important role in the prediction of numerical results thanks to the great calculation power of modern computers. Several programs have been written with this purpose and today there exists a machinery which combines QFT, statistical models and Monte Carlo methods to reproduce High Energy Physics experiments.

In this project, three of those programs were used: MadGraph 5.2 (MadEvent) [1], Pythia 8.2 [2] [3] and Delphes 3.2 [4] with the aim of simulating proton - proton collision events. The description of those programs and their

particular purposes in the project are described in chapter 2. In addition, chapter 2 includes the explanation of the codes and the scripts that were developed both to integrate those programs, and to run the simulations under specific conditions.

In despite of the fact that those simulations demanded a huge amount of computational time, they just served as inputs of the algorithms written throughout the project, which contain the main proposed analysis and ideas. Altogether, four algorithms were elaborated. Each of them are explained in chapter 3, where their documentation and an overall description are presented.

Finally, chapter four includes a brief description of some software tools that were introduced to the project. Specifically, this project used C++ codes which included root libraries instead of root macros. This transition reduced the execution time of the algorithms six times. Additionally, the development environment *Eclipse* was also introduced, which made easier the programming process. Overall, these tools dramatically improved the technical work of the project.

Chapter 2

Simulation chain

“Divide et impera”,
“Divide and conquer”

Philip II of Macedon

At first glance, it is not clear why it is necessary to use three programs at the simulation stage instead of just one. The answer is quite simple: each one of those programs has been developed to run a specific task in the simulation process, and therefore, each one has been optimized to do so as accurate and fast as possible. While MadGraph and Pythia are responsible for the simulation of high energy collision’s Physics, Delphes takes the final state particles produced by the former programs, and determines what would be the corresponding response of a detector. This scheme is useful as it maintains the detector apart from the main calculations of the simulation. Additionally, it makes the change of experiment parameters as simple as modifying Delphes execution specifications.

As presented before, MadGraph and Pythia handle the Physics of the collision. Again, there is more than a single program for this task, and now the reason to use two programs lies on the limits of the theoretical models. At the very first moment of the collision when the Energy Density of the System is high enough, perturbative Quantum Chromo-Dynamics (pQCD), Quantum Electro-Dynamics (QED) and ElectroWeak Theory are the most

accurate models known so far. MadGraph, and specifically MadEvent, use them to calculate the transverse sections of a particular channel defined by the user. From this calculation and the Monte Carlo models, it randomly establishes the kinematic variables of the resulting particles of the collision.

Once the energy density of the collision has been reduced significantly, the models used by MadGraph are not valid, and then Pythia appears in the scene. The particles resulting from MadGraph are taken by Pythia, which makes the evolution to a multi-hadronic final state [2]. The task run by Pythia involves the usage of Monte Carlo techniques to simulate hadronization, decays and showers. Finally, the particles obtained at the end of the Pythia simulation are the inputs for the Delphes simulation.

Although the usage of several programs for the simulation means better results, it also implies the challenge of connecting them. This task has already been done inside the MadGraph package, which connects MadEvent + Pythia 6 + Delphes / PGS¹. However, the version of Pythia included there (v.6) is old and does not offer the possibility of controlling ISR emissions as the last one (v.8) does. As ISR emissions were the main focus of the project, it was convenient to use Pythia 8 instead of Pythia 6 and therefore to develop the integration of MadGraph 5.2 with Pythia 8.2 and Delphes 3.2.

Throughout this chapter, the codes and scripts written to achieve the simulation will be explained. One section is devoted to each program and another one presents the script that connects the three programs. Finally, the last section of this chapter presents a simulation example where such script is used.

2.1 Usage of MadGraph 5.2

The most basic procedure to simulate collision events using MadGraph is by means of its executable program. Follow the next steps to run a set of simulations of the channel $p p \rightarrow t \bar{t}$. It is important that MadGraph has

¹*Pretty Good Simulation*, PGS, is another program for detector simulation

been correctly installed ².

1. In the folder where MadGraph has been installed, type:
`./bin/mg5_aMC`
2. Once MadGraph has been initialized, import the Standard Model parameters:
`import model sm`
3. Generate the event $p p \rightarrow t \bar{t}$:
`generate p p > t t~`
4. Create an output folder where all the simulation files will be saved, in this case `test_t_tbar`:
`output test_t_tbar`
5. Launch the Feynman diagrams production:
`launch -m`
 and select the number of cores you want to use for the simulation
6. Turn off Pythia and other programs³. You can switch off and on by typing the number before the program (type 1 to toggle pythia, for instance). Then, press enter.
7. Modify the `run_card.dat` file by typing 2. Write `:32` and press enter to go to line 32, then type `i` and press enter to modify the file. Change the number of events from 10000 to 1000. Press `Esc` and write `:wq` to write and quit.
8. Press enter to run the simulation

Although simple, the latter approach is not the best as it requires the user interaction several times to configure the simulation, which is not desirable when more than a single simulation will be performed. In such situations,

²A full set of instructions to install MadGraph and other High Energy Physics programs can be found at <http://goo.gl/vigBdj>

³This project uses the last version of Pythia (8.2) instead of the sixth version that uses MadGraph

all the configuration parameters can be defined through an input file. For the previous example, the input file would be:

```
import model sm
generate p p > t t~
output test_t_tbar -f
launch -m
2
pythia=OFF
Template/L0/Cards/run_card.dat
models/sm.v4/param_card.dat
```

where 2 corresponds to the number of cores used in the simulation, `run_card.dat` is the default file of MadGraph and `param_card.dat` contains the Standard Model parameters and values. Here, these two files correspond to the default ones that MadGraph provide. In order to use another set of configuration parameters, the files should be copied to another location and modified according to desired simulation conditions.

The input file may be saved as `mg5_input.mg5` and the simulation can be executed as:

```
./bin/mg5_aMC -f mg5_input.mg5 4
```

As a result of the simulation by MadGraph, the output folder contains several folders with all the information related to the simulation. The folder `Cards` for instance, contains some parameter cards used in the simulation, while the folder `HTML`, and specially the file `info.html` present the Feynman diagrams created by MadGraph. The events resulting from the simulation are found in the folder `Events/run_01` in the form of two files: a root file called `unweighted_events.root` and a compressed Les Houches Event file with name `unweighted_events.lhe`.

⁴Observe that it is supposed that `mg5_input.mg5` is located at the MadGraph folder and that the command is run from the same directory. If not, the execution instruction and the input file should contain the full path accordingly.

2.2 Usage of Pythia 8.2

The simulation carried out by MadGraph is now passed to Pythia, which takes the file `unweighted_events.lhe` as input. Pythia uses the information contained in such file to develop the hadronization, and produces another file with the kinematic variables of the resulting particles. The task performed by Pythia can be summarized in the Black Box of Fig. 2.1, where in addition to the file produced by MadGraph, a plain text file with extension `.cmd` is passed by parameter to configure the simulation.

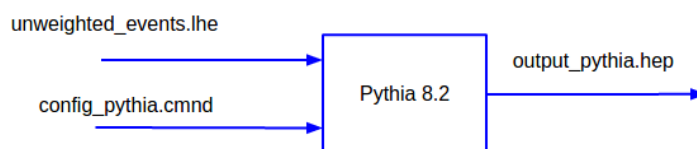


Figure 2.1

The functionality of the black box of 2.1 is done by a program written in C++, which is based on the examples provided by Pythia developers [3]. The code is called `hadronization02.cc`, was written in C++ and can be found at Appendix A.1. It performs specific requirements for this project that will be mentioned soon. Before presenting the operations performed by the program, it is convenient to describe how this code should be compiled and used.

2.2.1 Code Usage

To use `hadronization02.cc`, it is necessary to have installed Pythia⁵ and StdHep⁶ [5]. Once installed, go to the `examples` folder located at the Pythia directory⁷. Inside such folder, copy the code `hadronization02.cc` and

⁵Again, information to install Pythia 8.2 and HepMC can be found at <http://goo.gl/vigBdj>

⁶StdHep can be downloaded from <http://cepa.fnal.gov/psm/stdhep/getStdHep.shtml>. It is enough to type `make` to install it

⁷If `examples` is not exactly there, it may be in `share/Pythia8`

then modify the `Makefile` in order to compile it. It is enough to insert the following lines at the beginning of the `Makefile`:

```

1 # Include STDHEP libraries. The following 5 lines were
   sent by Mrenna.
2 STDHEP_DIR = <STDHEP Directory>
3 MCFIO_DIR = $(STDHEP_DIR)
4 SINC=$(STDHEP_DIR)/src/inc
5 INCS = -I$(SINC) -I$(STDHEP_DIR)/mcfio/src
6 LOCAL = -L$(STDHEP_DIR)/lib -lstdhepC -lFmcfio -lstdhep
   -lm

```

changing `<STDHEP Directory>` in line 2 by the local installation directory of `StdHep`. Furthermore, these other lines should be included at the end of the `Makefile`:

```

1 # Hadronization. (To compile files that read .lhe files
   and produce stdhep files)
2 # No further modifications are needed to compile the
   class UserHooks
3 hadronization% : hadronization%.cc $(PREFIX_LIB)/
   libpythia8.a
4      $(CXX) $^ -o $@ $(CXX_COMMON) $(INCS) $(LOCAL) -
      L$(PREFIX_LIB) -Wl,-rpath $(PREFIX_LIB) -
      lpythia8

```

After doing so, the code is compiled by typing on terminal:

```
make hadronization02
```

As a result, the executable file `hadronization02` is created in the current folder. It may be copied and used in other directory. The instruction to run this program is:

```
./hadronization02 input.cmnd [output.hep]
```

where `input.cmnd` is the full name (with the path) of the configuration file, and `output.hep` is an optional parameter that corresponds to the name of the output file.

Continuing with the $t\bar{t}$ production example of the previous section, the following file may be saved as `input.cmd` and used as input of the Pythia simulation:

```

1 ! Hadronization from a .lhe file
2 ! This file contains commands to be read on a Pythia8
   run.
3 ! Lines not beginning with a letter or digit are
   comments.
4
5 // Specify statistics parameters.
6 Main:numberOfEvents      = 1000 ! number of events
   generated (It needs to be <= Number of events
   generated in MG)
7 Init:showChangedParticleData = off ! not useful info
8 Next:numberShowInfo       = 1  ! 1 to show info, 0 to not
9 Next:numberShowEvent      = 0  ! Especificy the number of
   events that will be listed as output
10
11 // Read .lhe file
12 Beams:frameType = 4 ! To take a MG file as input
13 Beams:LHEF = unweighted_events.lhe ! MG .lhe file
14
15 ! Hadronization:
16 PartonLevel:FSR = off ! switch final state radiation
17 PartonLevel:ISR = on ! switch initial state radiation
18 PartonLevel:MPI = off ! switch off multiparton
   interactions
19 Random:setSeed = on ! For random seed
20 Random:seed = 1 ! any number between 1 and 900,000,000

```

Each line of this file is a different command, each of which is described after the exclamation mark character '!'. As it can be seen, 1000 events are hadronized, the file `unweighted_events.lhe` from MadGraph is read, and only ISR emissions are allowed.

2.2.2 The code

Having explained the procedure to compile and use the hadronization program, this subsection presents the code and what it does. As stated before, the code can be found in the Appendix A.1 and also, in the repository of the project: https://github.com/andresfgarcia150/ISR_tagging_project, at the folder `/Codes/Simulation/Pythia.Codes/`, where the modified Makefile is also included.

Overall, the code can be described in terms of two procedures: the configuration and the execution of the simulation. The first of them, that corresponds to lines 76 - 106 in Appendix A.1, establishes all the parameters needed for the simulation. It starts with the definition of some Strings to be used by the StdHep methods, and an object of class `Pythia` in line 82. Then, in lines 84-93, the names of the input file (`.cmd` file) and the output file are read from the execution instruction by means of `**argv`. Next, lines 95-98 define some variables to control the hadronization: `nEvent` corresponds to the number of events to be hadronized, while `nAbort` and `iAbort` are the maximum and current numbers of allowed events that present an error. Finally, the simulation configuration ends with some necessary functions to handle StdHep files (lines 100-102) and with the definition of an object of the class `MyUserHooks`.

The latter definition is extremely important for this project as it contains the restriction on the ISR emission. The object defined in line 105 belongs to the class `MyUserHooks`, which is written at the beginning of the code (lines 37-67). This class, in turn, inherits from `UserHooks` and just two of its methods are re-written: `canVetoISREmission()` and `doVetoISREmission()`. Each time an ISR emission is produced during the simulation of an event, the first of those methods stops the simulation and executes the second one, which counts the number of ISR partons produced so far and veto all the emissions in case that already exists one. This way, only one (or zero) ISR parton is produced in each event.

With the definition of the pointer `myUserHooks` and its inclusion in the object `pythia`, the configuration stage finishes. Then, the execution starts by initializing the simulation at line 109. Basically, the simulation consists of the *for* loop of lines 111-125, where each iteration corresponds to the generation

of a new event through the call of method `pythia.next()`. Observe that if the latter method returns `false`, either pythia has reached the end of the input file (from MadGraph), or an error has happened and the execution should stop if the maximum number of errors is reached. Once this has been verified, each cycle ends by writing the event in the output `.hep` file.

After the simulation has been completed, the StdHep file is closed in line 127, some statistics of the simulation are published (line 128) and the pointer `MyUserHooks` is deleted. These lines conclude the code that develops the hadronization process.

2.2.3 Pythia ntuple generation

Although the file produced by the latter code is passed directly to Delphes, it cannot be read by ROOT. Therefore, it is necessary to develop a conversion from `.hep` to `.root`, which is performed by `ExRootAnalysis`. After having it properly installed, go to the installation directory and run the executable file `ExRootSTDHEPConverter` by typing:

```
./ExRootSTDHEPConverter output_pythia.hep output_pythia.root
```

where `output_pythia.hep` is the full path name of the file produced by the hadronization code and `output_pythia.root` is the output `ntuple`. This procedure makes possible the reading of the pythia simulation when executing C++ codes with Root libraries.

To summarize, it has been shown how to carry out simulations with MadGraph and Pythia 8.2. As a result of the simulation of MadGraph, the file `unweighted_events.lhe` is produced. Pythia receives that file as parameter and creates the file `output_pythia.hep`. To complete the simulation process, the next section will introduce Delphes, that takes the file generated by Pythia and performs the detector simulation.

2.3 Usage of Delphes 3.2

Because High Energy Experiments such as the Compact Muon Solenoid (CMS) and A Toroidal LHC ApparatuS (ATLAS) are already created and there is not much we can do to modify them, the simulation of those detectors is a simple task. To use Delphes, for instance, it is enough to have it installed and use the existent cards.

For the CMS simulation of the $t\bar{t}$ production example that has been used throughout this chapter, go to the Delphes installation directory and use the execution file `DelphesSTDHEP`. To do so, type on the terminal:

```
./DelphesSTDHEP cards/delphes_card_CMS.tcl output_delphes.root
output_pythia.root
```

taking care that each one of the parameters should be replaced by the full path name of each file. With this instruction, `delphes_output.root` is generated and the files: `output_pythia.root` from the Pythia simulation, and `delphes_card_CMS.tcl` with CMS experiment specs are taken as inputs.

Delphes is the last link of the simulation chain and at the end, there are three ntuples to be used by the analysis algorithms:

1. `unweighted_events.root`: The ntuple produced by MadGraph. It contains the kinematic variables of the hard partons resulting from Feynman diagram calculations.
2. `output_pythia.root`: The ntuple generated by Pythia. It contains the information of all particles after hadronization and showering. In addition to final state particles, this file also stores a copy of all intermediate particles created during the hadronization process. It should be convenient to check the documentation about the particles' status [3] for more information.

3. `output_delphes.root`: The ntuple created by Delphes. It presents the simulation information as a detector should report, i.e. in terms of jets, photons, electrons, etc.

These three files are the final result of the simulation and as it will be presented later, the latter two will be used in this project. The procedure to obtain them has been presented and despite being straightforward, it is cumbersome as it requires several times the user intervention. Simulating would be a tedious task when several runs need to be executed such as the situation that this project deals with. Therefore, it was necessary to create an script that involved the three steps of the simulation. This script, originally written by Diego A. Sanz⁸ to run MadGraph alone, was modified to include Pythia 8.2 and Delphes 3.2, and it is the topic of the next section.

2.4 Integration of MadGraph 5.2 + Pythia 8.2 + Delphes 3.2

To integrate MadGraph 5.2 with Pythia 8.2 and Delphes 3.2 two scripts were written, which can be found in the Appendix A.2 and in the repository of the project⁹ at the folder `Codes/Simulation/MG_pythia8_delphes_parallel`. Those scripts allow parallel simulations taking advantage of the computing capabilities of the machine where the user is working.

Basically, the first script sets all the parameters needed for the simulation, which is executed by the second script. Thus, the user needs to modify all the variables in `config_Integration.ini` according to the local installation directories and the folders where the run and param cards are located. After doing so, it is sufficient to execute `script_Integration.sh` in order to run the simulation:

```
./script_Integration.sh
```

This way, there is not risk of accidentally changing the execution script.

⁸d-sanz@uniandes.edu.co

⁹https://github.com/andresfgarcia150/ISR_tagging_project

Although both scripts are well documented, it is worth mentioning some words about them:

- Because the scripts execute parallel simulations, it is necessary to specify two folders where they will be saved: `EVENTSFOLDER` is the name of the head directory where all simulations will be saved, and `NAMESUBFOLDER` is the generic name of the folders that contain each simulation and that are located at `EVENTSFOLDER`. Thus, simulation #3 is saved in `EVENTSFOLDER/NAMESUBFOLDER3`.
- In total, each execution of `script_Integration.sh` run simulations from `INIRUN` to `ENDRUN`. Each of them consists of `NUMEVENTSRUN` events and its seed is the simulation number.
- Because MadGraph can develop some parallel calculations, `CORESNUMBER` is the number of cores devoted to each MadGraph run. Be aware that the total number of parallel runs times `CORESNUMBER` needs to be less or equal than the number of cores of your machine. Once MadGraph has been executed, only one core of `CORESNUMBER` is used to run Pythia and Delphes, because they only manage one thread.
- There are two sequences inside `script_Integration.sh`. The first one copies and modifies the run and param cards according to each simulation (it changes the seed, for instance). At the end of this sequence, those copies are located at the folders `/RunCards/` and `/ParamCards/` inside `EVENTSFOLDER`. When configuring `config_Integration.ini`, it is extremely important to use the templates of the files:

```

- run_card.dat
- mgFile.mg5
- input_pythia.cmd

```

provided at the folder `Codes/Simulation/MG_pythia8_delphes_parallel/RunCard_Template` of the repository, as the script looks for certain variables defined in such templates and replace them with the specific parameters of each simulation.

- The second sequence inside `script_Integration.sh` runs the simulations. As it can be verified in Appendix A.2.2, it:

1. Runs Madgraph
2. Uncompresses the .lhe.gz file produced by MadGraph
3. Executes Pythia
4. Executes Delphes
5. Makes the conversion `output_pythia.hep -> output_pythia.root`
6. Remove unnecessary files.

Contrary to the first sequence, this second one is run in parallel using the program `Parallel` [6].

2.5 Example of the integration scripts

The example that was presented when each one of the programs was explained will now be repeated with the scripts introduced in above. Follow the next instructions to simulate 100000 events of the channel $p p \rightarrow t \bar{t}$, where additionally one W boson resulting from the tops' decays is required to decay hadronically while the other is forced to a leptonic decay:

1. Install the three programs and compile the code `hadronization02` of Pythia.
2. Download the folder `MG_pythia8_delphes_parallel` from the repository of the project.
3. Open the file `config_Integration.ini` and write all the installation folders in front of the corresponding variables. Use the path of the downloaded folder `RunCard_Template` as the directory of `RUNCARDFOLDER`, `MADGRAPHFILEFOLDER`, `PYTHIAPARAMFOLDER` and `DELPHECARDFOLDER`. For the variable `PARAMCARDFOLDER` use the directory where MadGraph is installed, followed by the folder `/models/sm_v4`.
4. In the file `config_Integration.ini`, modify the variables:
 - `CORESNUMBER=2` (To execute each run with 2 cores)
 - `NUMEVENTSRUN=10000` (To simulate 10000 events per run)

- INIRUN=1 (The first simulation goes with seed = 1)
- ENDRUN=10 (The last simulation goes with seed = 10)

5. Take a look of each one of the input files:

- (a) Open `/RunCard_Template/mgFile.mg5` and check the details of the MadGraph simulation. Observe, for instance, line 4 where the channel is specified.
- (b) Open `run_card.dat` and verify that the energy per beam is 6500GeV in lines 41 and 42.
- (c) In the file `input_pythia.cmd`, observe the same parameters presented in subsection 2.2.1. Additionally, the file includes some necessary settings to perform the *matching* procedure between MadGraph and Pythia. More information about it can be found at [7].

6. Execute the script by typing¹⁰:

```
./script_Integration.sh
```

¹⁰Possibly, you might want to run the simulation in background. In such case, type `screen`, then execute the simulation instruction and once it has started, type `Ctrl + a + d` to leave it in the background. If you want to return to the simulation, type on the terminal: `screen -r`.

Chapter 3

Analysis programs

“To be, or not to be, that is the question”

Hamlet
William Shakespeare

The simulations presented before are very important for this project as they serve to prove the ideas proposed to identify ISR jets. Now its time to present those ideas and the codes that were written to develop them. Before showing the programs, an introduction about joint aspects between them will be presented. Then, the tagging algorithm and some auxiliary programs will be explained. Finally, this chapter ends with general instructions about the process of using all the codes altogether.

3.1 Preparation of the codes

All the codes that will be presented in this chapter are included in Appendix B and in the repository of the project, at the folder `Codes/Codes_analysis`. Each of them is stored inside a different folder with other files that contain

functions used by the corresponding code. In order to compile each program, follow the next instructions:

1. Download the corresponding folder from the repository of the project.
2. Inside each folder, modify the **Makefile** according to your local c++ compiler and program installation folders. Change lines 23 to 49 of each **Makefile** to do so.
3. To compile each code, it is enough to type:
`make_compile_ROOT_Delphes`

Because the programs need a lot of parameters that the user may want to change from one simulation to another, the programs have been designed to include those parameters by means of a configuration file. Therefore, the user does not need to compile each program to execute different analysis. Inside each file, there is a list of definitions, which follow the structure:

`Variable_name=Variable.value.`

Every comment starts with the symbol ‘!’, even blank lines. It is important not to change the name of the variables and to follow the syntax rules in order to avoid problems at the execution of the codes.

Observe that the configuration file makes the program flexible as the user can easily define the name of the folders and the files. In despite of this, I preferred to follow a strict convention to name the files, which is illustrated in Table 3.1. When checking the files, take into account this table and the following rules. Recall, however, that it is my convention and you can easily modify it.

1. Each ‘s’ before the word Tops should be either a ‘s’ if the channel under analysis is stop pair production, or a ‘_’ if the studied channel is top pair production.
2. ‘WI’ corresponds to the case when there is an ISR jet in the simulated events. It changes to ‘SI’ if there are not ISR jets.

Item	Description/Contents	Name structure
Simulation head folder	Simulations' run folders of the same channel	sTops_Events_WI_Matching
Simulation run folder	Simulations' files of a particular run	sTops_MG_1K_AG_WI-004
Matching folder	All the matching head folders	matching_Results
Matching head folder	Matching result files of a particular simulation	sTops_matches_WI_Matching
Matching file	Matching information of a specific run	ISR_jetssTops_WI-005.bn
Histograms' folder	All histograms' head folders	histo_folder
Histograms' head folder	Histograms' files of a particular simulation (channel)	sTops_histos_WI_Matching
Histograms' files	Information of the N-dimensional histograms. Each histogram consists of 4 files: A binary and a plain text file for both ISR and Non ISR jets.	array_histo_ISR_0.1.2.bn array_histo_Non_ISR_0.1.2.bn info_histo_ISR_0.1.2.txt info_histo_Non_ISR_0.1.2.txt
Tagging folder	All tagging head folders	resultsTagging
Tagging head folder	Tagging result files of a particular simulation	sTops_result_WI_Matching
Tagging result files	Efficiency of the tagging algorithm for a particular channel and a specific selection of analysis variables.	sTops_WI-Overall_0.1.2.txt sTops_WI_hpt-050_0.1.2.txt sTops_WI_MET_pt_050_k_2.0_0.1.2.png

Table 3.1: Naming convention of folders and files

3. ‘*_Matching*’ appears if the matching procedure between MadGraph and Pythia has been done. If not, it does not appear in the name.
4. The sequence of numbers ‘*_0_1_2*’ corresponds to the set of variables used for the analysis (Those variables will be explained later on).

Other details to manage each program will be explained in the following sections. However, keep in mind this section when studying the next pages.

3.2 The ISR jet tagging method

The ISR jet tagging algorithm is the most important program of this project. It seeks to find the ISR jet in a event, in case it exists. Because of its importance, a complete explanation is presented bellow.

3.2.1 The method

Let’s suppose that there exists a kinematic variable y that distinguishes between ISR jets and Non ISR jets. The information of such variable is known by means of the distribution functions for each type of jet (f^{ISR} , $f^{Non\ ISR}$). Therefore, if a measurement of the variable y for a particular jet is y_0 , then $f^{ISR}(y_0)$ and $f^{Non\ ISR}(y_0)$ are known, as it is presented in Fig. 3.1.

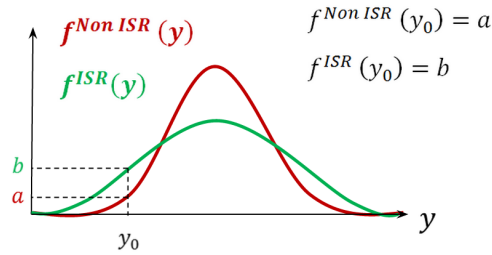


Figure 3.1: Probability distributions of a variable that distinguishes between ISR and Non ISR jets

The difference between both distributions could be used to write the probability of such jet being ISR or not. In fact, the probability of being ISR should be proportional to the ISR distribution function at the measurement. Likewise, the probability of being non ISR should be proportional to the Non ISR distribution function:

$$P^{ISR}(y_0) \propto f^{ISR}(y_0), \quad (3.1)$$

$$P^{Non\ ISR}(y_0) \propto f^{Non\ ISR}(y_0). \quad (3.2)$$

In addition to the information offered by the density functions, another important consideration to take into account is the *a priori* probability of being ISR. If just one jet of the N_{jets} in the event is ISR, the *a priori* probability of any jet being ISR is:

$$P_{apriori}^{ISR}(y_0) = \frac{1}{N_{jets}}, \quad (3.3)$$

and similarly, the *a priori* probability of any jet being Non ISR is:

$$P_{apriori}^{Non\ ISR}(y_0) = \frac{N_{jets} - 1}{N_{jets}}. \quad (3.4)$$

Combining both assumptions, the probabilities of being ISR and Non ISR could be written as:

$$P^{ISR}(y_0) = \alpha f^{ISR}(y_0) \frac{1}{N_{jets}}, \quad (3.5)$$

$$P^{Non\ ISR}(y_0) = \alpha f^{Non\ ISR}(y_0) \frac{N_{jets} - 1}{N_{jets}}, \quad (3.6)$$

where α is a constant that results from the normalization of the probabilities:

$$1 = P^{ISR}(y_0) + P^{Non\ ISR}(y_0), \quad (3.7)$$

$$\alpha = \frac{N_{jets}}{f^{ISR}(y_0) + (N_{jets} - 1)f^{Non\ ISR}(y_0)}. \quad (3.8)$$

If there are more than a single variable which differentiate between ISR and Non ISR jets, the previous analysis can be extended easily. In fact, it is enough to replace the single variable probability density functions by multidimensional probability densities. The formulas would take the same form as the probability density distributions are scalar functions, regardless they depend on a single variable y or on a vector \vec{y} . Therefore, in a multidimensional case, the formulas would be:

$$P^{ISR}(\vec{y}_0) = \alpha f^{ISR}(\vec{y}_0) \frac{1}{N_{jets}}, \quad (3.9)$$

$$P^{Non\ ISR}(\vec{y}_0) = \alpha f^{Non\ ISR}(\vec{y}_0) \frac{N_{jets} - 1}{N_{jets}}, \quad (3.10)$$

3.2.2 From probability density functions to normalized histograms

As the latter formulas show, the probabilities of each jet depend on the probability density distributions. In practical matters, these functions are replaced by normalized histograms whose entries are collected from simulations where the ISR jet is known.

However, the replacement is just an approximation because a bin of the histogram does not correspond exactly to the value of the probability density function. Instead, the histogram results from an integration of the probability distribution:

$$H(y_i) = \int_{\Omega_i} f(y) dy, \quad (3.11)$$

where Ω_i is the range of the bin, as it is presented in Fig. 3.2.

If the size of the bin is small enough, the expression 3.11 can be approximated by:

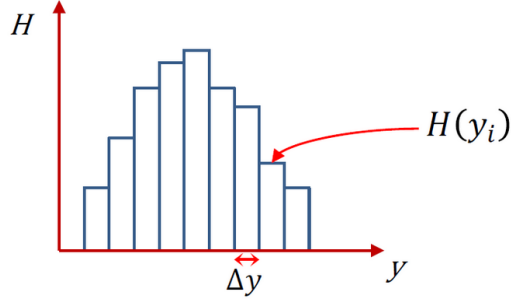


Figure 3.2: Shape of a histogram which does not exactly correspond with the probability density function

$$H(y_i) \approx f(y_i)\Delta y, \quad (3.12)$$

Using this approximation, the practical expressions of the probabilities of being ISR or Non ISR are:

$$P^{ISR}(\vec{y}_0) = \alpha H^{ISR}(\vec{y}_0) \frac{1}{N_{jets}}, \quad (3.13)$$

$$P^{Non\ ISR}(\vec{y}_0) = \alpha H^{Non\ ISR}(\vec{y}_0) \frac{N_{jets} - 1}{N_{jets}}. \quad (3.14)$$

To sum up, the usage of these formulas implies the necessity of running simulations of several events (with the scheme of chapter 2), identifying theoretically the ISR jet in each event, and filling a N-dimensional histogram for each type of jet (Non ISR and ISR).

3.2.3 The Algorithm

Once the method has been prepared by selecting the distinguishing variables and by filling the histograms, the algorithm of Fig. 3.3 is applied for each event. First, each jet in the event is studied and its probabilities of being ISR and Non ISR are determined from its kinematical variables and expressions 3.9 and 3.10.

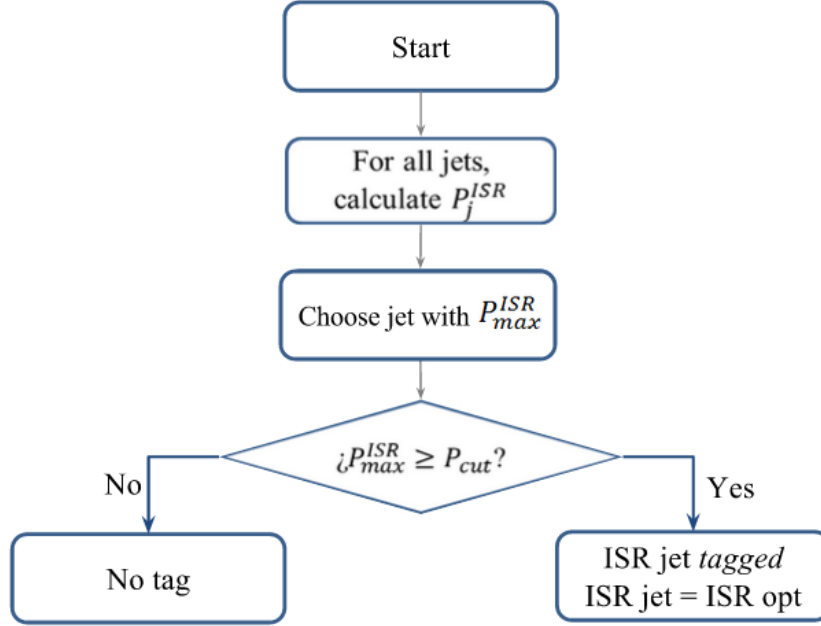


Figure 3.3: ISR jet tagging algorithm

Then, the jet with greatest probability of being ISR P_{max}^{ISR} is selected as ISR candidate. Finally, P_{max}^{ISR} is compared to a certain cut P_{cut} , in order to ensure that the algorithm is conclusive. For example, if $P_{max}^{ISR} < 1/N_{jets}$, the probability of the ISR candidate is fewer than the *a priori* probability, and therefore no tag should be imposed. The cut is written in terms of a variable k that corresponds to the minimum factor that the probability of the ISR candidate should be greater than the *a priori* probability:

$$P_{cut} = \frac{k}{N_{jets}} \quad (3.15)$$

This way, the ISR jet is tagged in each event based exclusively on preliminary histograms and simple probability considerations.

3.2.4 The code

The tagging code is presented in Appendix B.1 and in the repository of the project, at the folder `Codes/Codes_analysis/ISR_tagging_FV`. To compile it, follow the instructions of section 3.1. After compilation, the code can be executed by typing the instruction:

```
./ISR_tagging config_file.txt [N1 N2 [N3]
```

where the first parameter is mandatory and corresponds to the configuration file. Inside the folder of the program, there is an example of such file. Modify the values of the variables defined in such file. After the definition of the file names and folders, there are two important variables at the end of the configuration file (`k_cut` and `pt_cut`), which are used to perform an analysis of the tagging results. If those variables are uncommented, the tagging algorithm is executed with a probability cut `k_cut` and then, a selection of the tagged ISR jets is done by choosing those jets whose PT is larger than `pt_cut`. The performance of the algorithm is measured for this selection and plots of Missing Transverse Energy are generated.

On the other hand, the last three parameters are optional. Because the method uses three kinematic variables to distinguish ISR jets from Non ISR jets, the last three parameters correspond to the number of the variables the user wants for the analysis. There are eight possible variables defined in the program, that can be checked in the documentation at the beginning of the code. Although optional, the user cannot specify just one or two of them; it is important to execute the code by typing the three numbers or none of them. If no variables are written as inputs, the code takes by default the variables 0, 1 and 2.

Finally, it is important to mention that the tagging algorithm is executed for several runs. The *for* loop of line 308 defines the simulations (their seeds) to which the tagging algorithm will be applied. Other technical details of the tagging program can be found in the comments of the code.

In order to execute the *tagging* algorithm, it is important to prepare it.

That is, it is necessary to fill first the N-dimensional histograms. Therefore, in addition to the code corresponding to the *tagging* algorithm, other three codes were written to prepare the *tagging: Matching algorithm, ISR jet analysis* and *Histograms' creation*. In the next sections, these codes and their functionalities will be presented.

3.3 Matching algorithm

Some pages above, it was said that the success of the *tagging* algorithm is based on the information contained by the N-dimensional histograms. Naturally, those histograms need to be filled with events where the ISR jet is known. Because Delphes reports the results as the experiment does, the kinematic variables of the histograms should be taken from jets reported by Delphes, which implies the necessity of knowing the ISR jet at the Delphes simulation stage.

However, the ISR emission is done by Pythia, which introduces ISR partons and hadronizes them. Only the final particles that result from the hadronization are taken by Delphes in order to simulate the detector and thus, it is impossible to know the ‘theoretical’ ISR jet with the Delphes simulation exclusively. Therefore, it is necessary to *match*¹ the ISR parton from Pythia with one of the jets from Delphes. Observe that this is a computational procedure that cannot be done with real data; it is only useful to identify the ISR jet in Delphes and then to fill the N-dimensional histograms.

The *matching* algorithm is presented in Fig. 3.4. In practical matters, after knowing the ISR parton in Pythia, it looks for the closest jet using the cone-algorithm. It not only considers the jets reported by Delphes, but also combinations between them (i.e. up to three of them). This considers the case when a parton results in more than a single jet because of the detector interpretation. After choosing the closest jet (or combination) to the ISR parton, the algorithm ensures that the optimum jet is inside a reasonable region around the ISR parton. If the matched jet is too far from the ISR parton or if it is a combination of several jets, the method does not report

¹We have called this procedure *matching*. Please do not confuse it with the algorithm carried out between MadGraph and Pythia, that has been mentioned in chapter 2 [7].

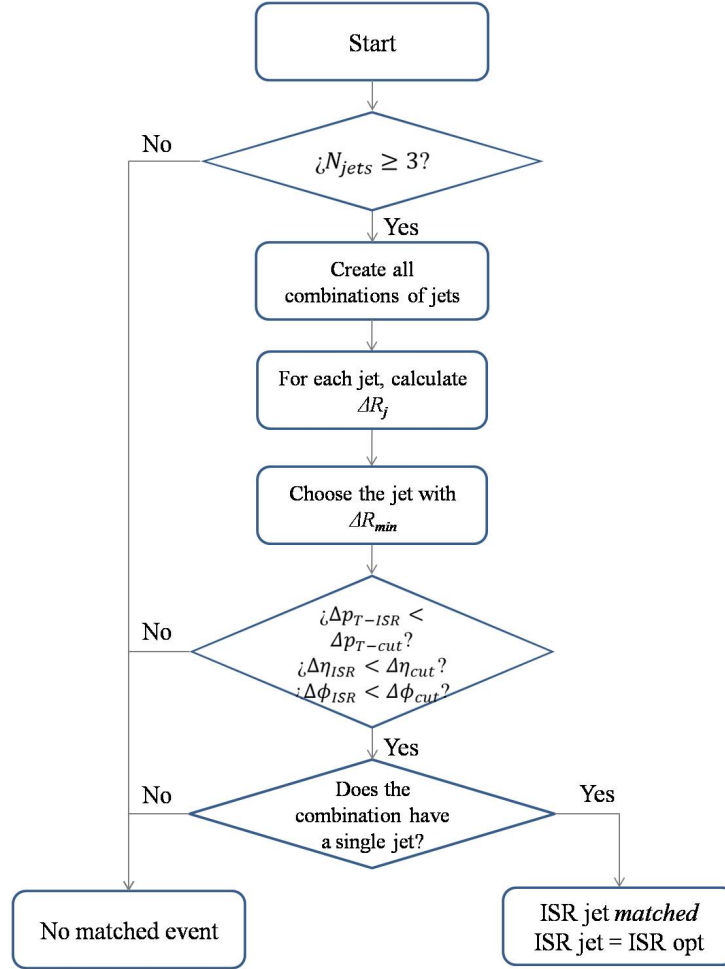


Figure 3.4: Matching algorithm between MadGraph and Pythia

any match as it is shown in the last two boxes of scheme 3.4.

As in the case of the tagging algorithm, follow the instructions of section 3.1 to compile and modify the global variables of the code, which can be found in Appendix B.2 and in the repository of the project. Once the code has been compiled, it can be executed by typing the instruction:

```
./ISR_matching config_file.txt [000]
```

where the first parameter is the mandatory configuration file. The last

three digits required as parameters are optional and correspond to the number of the simulation (its seed) to which the user wants to execute the matching. If no parameter is written, the simulation for analysis has seed 003.

Observe that in contrast with the tagging code, the matching code does not execute the algorithm for several runs but only one. In consequence, a script has been written in order to perform several matching procedures. This script, called `script_several_matchings.sh`, is available in the repository (in the same folder of the matching code). In order to use it, modify line 8 according to the simulations to which you want to perform the matching and then, type the instruction:

```
./script_several_matchings.sh 2
```

As a result of executing the matching algorithm, a binary file containing a list with the ISR partons is generated. For those events without matching, the entry of the list is -1 . The file, with name `ISR_jetssTops_WI_005.bn` ³, is used as input by the other codes to know which is in ‘theory’ the ISR jet.

Finally, more documentation can be found in the comments of the code.

3.4 ISR jet analysis code

Several times throughout the project, it was necessary to compare ISR jets and Non ISR jets. The comparison between both kind of jets allowed the subsequent selection of suitable variables for the execution of the tagging algorithm. Due to this importance, a separate code was written in order to develop such comparison. Again, the code can be found in Appendix B.3 and in the repository of the project, at the folder `Codes/Codes_analysis/ISR_jet_analysis_FV`.

The program takes a group of simulations and their corresponding matching results as inputs. Then, it creates histograms⁴ of kinematic variables

²Possibly, it is necessary to change the permissions of this script to execute it. Type `chmod a+x script_several_matchings.sh` to do so.

³Check the structure of the name in section 3.1

⁴Root TH1 histograms

and compares the distributions of ISR and Non ISR jets. To do so, several functions that plot graphics were written. They can be found in the files `graphs_Funcs.cpp` and `graphs_Funcs.h`, which are located at the same directory where `ISR_jet_analysis_FV` is. All these codes are fully documented and the compilation can be done by following the instructions of section 3.1.

3.5 Code to create N-dimensional histograms

So far, the codes of the tagging algorithm, the matching procedure and a program for analysis have been presented. Now it is time to introduce the code that creates the N-dimensional histograms that are used by the tagging algorithm to differentiate between both kind of jets. Once again, check the repository of the project (folder `Codes/Codes_analysis/Creating_histos_FV`) and the Appendix B.4 to read the code.

Inside the same folder where the code is available, the files `histoN.cpp` and `histoN.h` are also located. These files contain the definition of the class `histoN` which handles N-dimensional histograms. Objects of this class are declared in `Creating_histo` in order to collect the histograms' information. Afterwards, similar objects are used in the tagging program to develop the algorithm of Fig.3.3.

The procedure that `Creating_histo` executes is illustrated in Fig. 3.5. After declaring the objects, a loop over the events of the performed simulations is executed. Inside such loop, the histograms are filled using kinematic variables of the already matched ISR and Non ISR jets. Finally, the accumulated information is stored in the files of the eight row of Table 3.1. Each histogram corresponds to both a binary file with the entries of the bins, and a plain text file with the parameters that define the histogram.

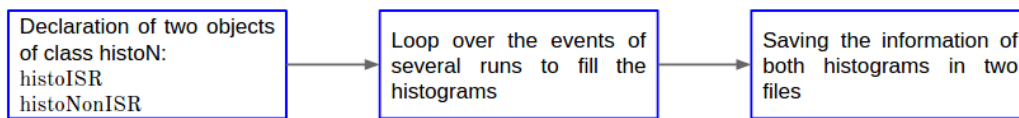


Figure 3.5: Procedure to create N-dimensional histograms

The compilation of the code is achieved by following the instructions of

section 3.1. To execute the code, type:

```
./ Creating_histo config_file.txt [N1 N2 N3]
```

where the first parameter is again the configuration file and the others at the end are optional and correspond to the variables with which the histograms will be filled.

The explanation of the most important codes written in this project finishes here. An example that involves the execution of all codes will be presented in the following section.

3.6 Example of the usage of the codes

Follow the next instructions to execute the tagging algorithm to a sample of ISR jets in top pair production events:

1. Simulate 25 million events of the top pair production channel following the steps of section 2.5.
2. Apply the matching algorithm to this simulation
3. Check the difference between ISR and Non ISR jets by executing the `ISR_jet_analysis` program.
4. Fill histograms of Non ISR and ISR jets by running `Creating_histo`. Use variables 0, 3 and 4 for analysis (for instance).
5. Simulate another one million events of the same channel.
6. Run the matching procedure to these events.
7. Apply the tagging algorithm to the latter simulation taking as parameters the histograms resulting from the fourth instruction. The matching results of step six are also an input of the program as they serve to compare the tagging results. Remember to execute the tagging using the same variables with which the histograms were filled.

Chapter 4

Software tools

“Man is a tool-making animal”

Benjamin Franklin

In addition to the ideas purposed to execute the ISR tagging method, this project meant the introduction of several Software tools that simplified the development of the project. This section contains two of those Software tools, which could be used in any project which requires the usage of ROOT libraries and C++ programs.

Note: It is extremely important to have correctly installed the programs you may use. ¹. Additionally, the directories of the corresponding libraries should be included in the `LD_LIBRARY_PATH` of the `bashrc`. To do so:

1. Open the `bashrc` by typing:

```
vi ~/.bashrc
```

2. At the end of the file, write for each library directory:

```
export LD_LIBRARY_PATH=/NEW_PATH/:LD_LIBRARY_PATH
```

¹Check the instructions to install MadGraph and other High Energy Physics programs at <http://goo.gl/vigBdj>

where `NEW_PATH` is the full path of the new library to be included (without the library name).

3. After saving and closing the file, charge the changes by typing:
`source ~/.bashrc`

4.1 Compilation of C++ programs including ROOT libraries

Usually, the first approach to ROOT is by means of ROOT macros, which are executed after initializing ROOT. This technique is simple but inefficient and when the program becomes large, the execution time increases significantly. In order to avoid those situations, compiled versions of the code might be created. These programs are optimized and run much faster than Macros do. In addition, the transition to C++ compiled programs is not difficult because Macros are already written in C++ and is sufficient to compile them correctly.

Aiming to make the compilation easy, a Makefile template was written. Using this file, you can create your own C++ codes including ROOT libraries. The procedure to configure it is quite simple and it is presented in the following instructions:

1. Download the compressed version of the Makefile from the repository of the project by typing:
`wget https://github.com/andresfgarcia150/ISR_tagging_project/raw/master/Codes/Makefile_template.tar.gz`
All the files of this folder are also located at the directory `Codes/Makefile_template` of the repository.
2. Uncompress the folder:
`tar -zxvf Makefile_template.tar.gz`
3. Inside the folder, modify lines 23 - 48 according to your local C++ compiler and installation directories. If any of the programs is missing in your PC, do not type anything.

4. Check the `HelloWorld.cc` program and include the `.h` files that you may utilize. Inside the folder, there are three files you could use:
 - `HepMCFunctions.h` with HepMC files,
 - `ROOTFunctions.h` with ROOT files ²,
 - `DelphesFunctions.h` with Delphes files.
5. To compile the `HelloWorld` program, type one of the next commands:
 - `make compile`, to compile without Root nor HepMC
 - `make compile_ROOT`, to compile with Root and without HepMC
 - `make compile_HepMC`, to compile without Root and with HepMC
 - `make compile_ROOT_HepMC`, to compile with Root and HepMC
 - `make compile_ROOT_Delphes`, to compile with Root and Delphes

4.2 Using Eclipse

By using the template of the previous section, the execution time of ROOT programs will reduce significantly. However, written codes is still a tedious task as it is generally done with plain text editors where syntax errors may easily appear. Those problems have been already solved with programming environments, which in addition, handle other issues that make programming an easy task. This section presents the configuration of Eclipse, a standard open-source environment. This introduction, however, will be focused only on the important points related to ROOT - C++ projects.

Follow the next instructions:

1. Before using eclipse, type the following command on a terminal:

```
root-config --cflags
```

It shows some miscellaneous flags and the ROOT include directory (that goes after `-I`). In the case of my PC, I obtain Fig. 4.1, where four

²Uncomment line 41 in case you do not use Delphes

flags are shown (`-pthread -std=c++11 -Wno-deprecated-declarations -m64`) followed by the directory of the ROOT include.

```
[af.garcia1214@dzero ISR_tagging_FV]$ root-config --cflags
-pthread -std=c++11 -Wno-deprecated-declarations -m64 -I/usr/local/root/include/root
```

Figure 4.1: Result of `root-config --cflags`

Then, type the following command:

```
root-config --libs
```

In my case, I obtain the sequence of Fig. 4.2. This command lists the directory of ROOT libraries (after `-L`) and their names (after `-l`).

```
[af.garcia1214@dzero ISR_tagging_FV]$ root-config --libs
-L/usr/local/root/lib/root -lCore -lRIO -lNet -lHist -lGraf -lGraf3d -lGpad -lTree
-lRint -lPostscript -lMatrix -lPhysics -lMathCore -lThread -pthread -lWl,-rpath,/usr
/local/root/lib/root -lm -ldl -ldynamic
```

Figure 4.2: Result of `root-config --libs`

Keep at hand the results given by your pc of the previous commands because they will be used in the subsequent instructions.

2. Download the Eclipse Luna IDE for C/C++ developers at <http://www.eclipse.org/downloads/packages/release/Luna/SR2>.

Choose your version according to your OS. If you are in a 64 bit Linux machine, it would be sufficient to type:

```
wget http://eclipse.c3sl.ufpr.br/technology/epp/downloads/release/
luna/SR2/eclipse-cpp-luna-SR2-linux-gtk-x86_64.tar.gz.
```

3. Uncompress the downloaded file:


```
tar -xzf eclipse-cpp-luna-SR2-linux-gtk-x86_64.tar.gz
```
4. Inside the new `eclipse` folder, open Eclipse. Type `./eclipse`
5. When asked to select a workspace, choose a directory where you want to save your future codes.
6. On the new window, select the option *Workbench*.
7. Create a new C++ project. Go to `File > New > C++ Project`

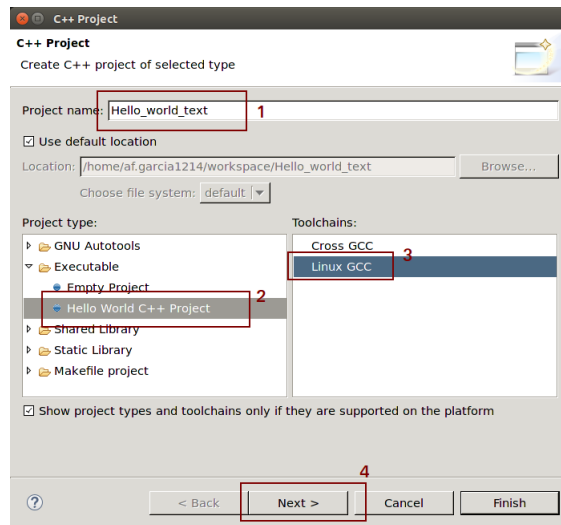


Figure 4.3: Configuration of a Hello_world program in Eclipse

8. In the new window, change the project name, the project type, the Toolchains as illustrated in Fig.4.3. Then click on **Next**.
9. In the New Window, modify the author and any other comment as Fig. 4.4, for instance. Then click on **Next**.

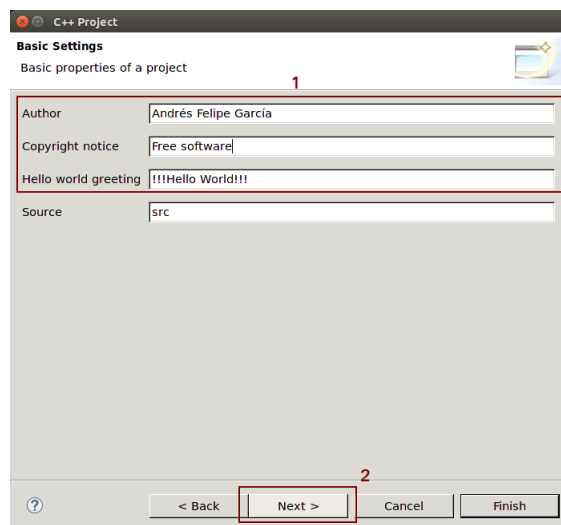


Figure 4.4: Setting the author in a eclipse project

10. In the last window (Fig. 4.5), select **Debug** and **Release**. Then click on finish.

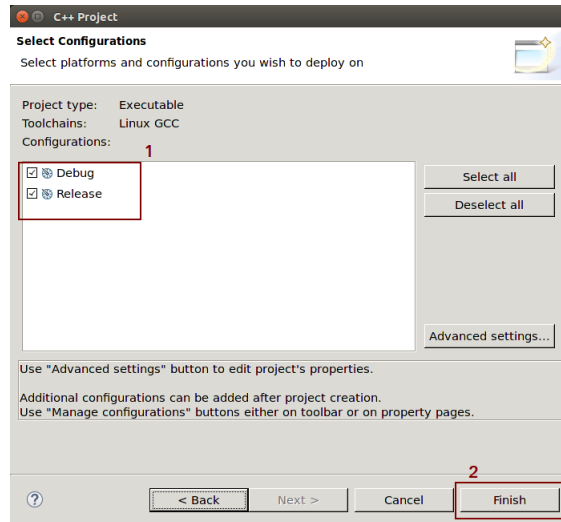


Figure 4.5: Setting *release* and *debug* in eclipse

11. After these settings, a basic Hello world template is created. On the left menu, right-click on the project folder and choose the option **Properties**. It also can be done by typing **Alt + enter**.
12. On the pop-up window, click on **C/C++ Build > Settings** and then on **GCC C++ Compiler > Includes** according to Fig. 4.6.
13. On the right panel include the full path of the directories where the `.h` files are located. To include the libraries of the following programs³:
- **ROOT**: Use the the directory reported in the command `root-config --cflags` of Fig. 4.1.
 - **ExRootAnalysis**: Search at the MadGraph directory and use the folder `MG5_aMC_v2.2.2/ExRootAnalysis/ExRootAnalysis`
 - **Delphes**: Because Delphes has several `.h` files in different folders, just include the path of the Delphes head folder. Then, at time of including a specific file in the code, write its relative location. Take a look, for instance, at the file `DelphesFunctions.h` of section 4.1.

³ROOT is necessary to use the other two

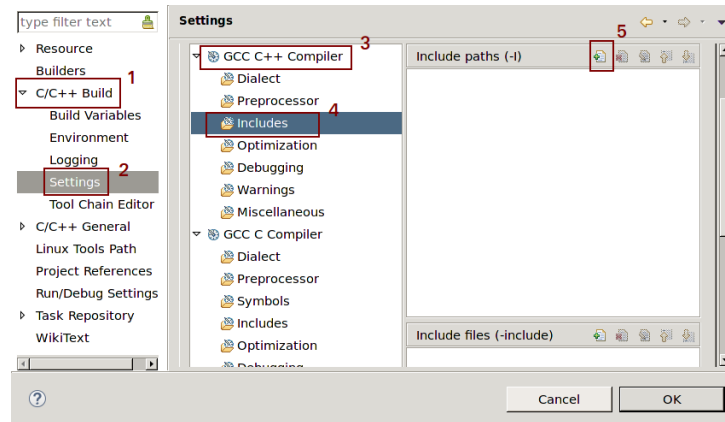


Figure 4.6: Steps to include paths in an eclipse project

14. In the same menu of **GCC C++ Compiler**, go to **Miscellaneous** as illustrated in Fig 4.7. Include the flags reported by the command of Fig. 4.1 after the already existing flags in the label of the right.

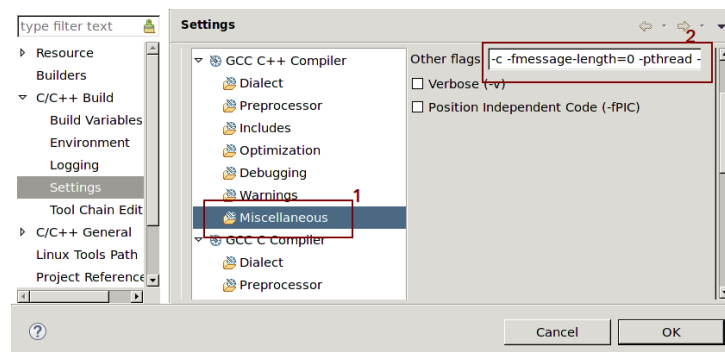


Figure 4.7: Setting miscellaneous flags on eclipse

15. Now, links to other libraries will be set. On the settings menu, go to **GCC C++ Linker > Libraries** as illustrated in steps 1 and 2 of Fig. 4.8.
16. Add the libraries reported by the command of Fig. 4.2 to the top right panel. In my case, these are the libraries I have to include:

Core
RIO

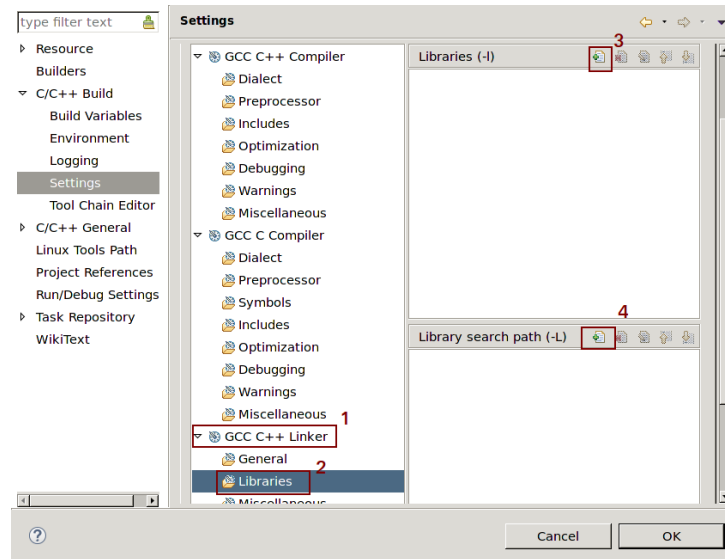


Figure 4.8: Configuration of libraries in an eclipse project

Net
 Hist
 Graf
 Graf3d
 Gpad
 Tree
 Rint
 Postscript
 Matrix
 Physics
 MathCore
 Thread
 ExRootAnalysis
 Delphes

Note: The last two libraries are from ExRootAnalysis and Delphes. The others are from ROOT and possibly, they are not the libraries installed on your PC, so type `root-config --libs` on terminal in order to check your ROOT variables.

17. Then, at the panel of the bottom right corner, write the directories where these libraries (.so files) are stored. In case you use one of the next programs:
 - ROOT: Use the directory displayed by the second command (Fig. 4.2).
 - ExRootAnalysis: Look at the MadGraph directory and use the folder MG5_aMC_v2.2.2/ExRootAnalysis/⁴.
 - Delphes: Use the Delphes head directory.
18. In the same menu of **GCC C++ Linker**, go to **Miscellaneous** and include the remaining flags of the command of Fig. 4.2 in the blank space next to **Linker flags**. (Fig. 4.9)

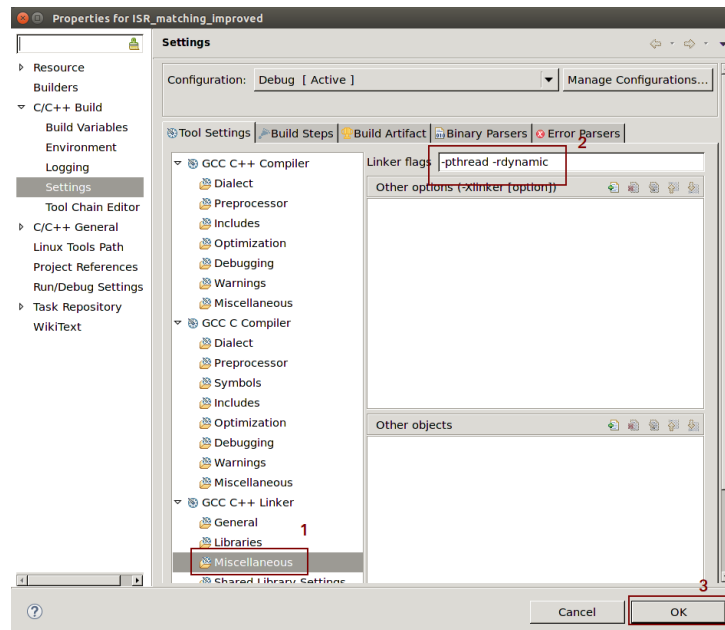


Figure 4.9: Setting miscellaneous linker flags in eclipse

19. Finally, click on **OK** at the bottom right corner.

⁴It is extremely important to include first the ROOT directory and then the others two.

20. To check that everything was set up correctly, go to **Project > Build Project**. Then execute the code by either clicking on **Run > Run** or from terminal, going to the folder **Debug** and typing `./Hello_world_text`.

These instructions have been taken from [8]. They have been modified to handle specific aspects of this project. However, do not hesitate to check such document to get a broader idea of setting eclipse. Specially, check slide 18 to find an easier way of setting up ROOT. (ExRootAnalysis and Delphes set up is not explained there).

Appendices

“Talk is cheap, show me the
code”

Linus Torvalds, creator of Linux
Kernel

Appendix A

Simulation codes and scripts

A.1 Pythia code: hadronization02.cc

```
1 // Copyright (C) 2015 Torbjorn Sjostrand.
2 // PYTHIA is licenced under the GNU GPL version 2, see
  // COPYING for details.
3 // Please respect the MCnet Guidelines, see GUIDELINES
  // for details.
4
5 /*
6 -----      Universidad de los Andes      -----
7 -----      Departamento de Fisica        -----
8 -----      Proyecto Joven Investigador    -----
9 -----      Andres Felipe Garcia Albarracin -----
10 -----      Juan Carlos Sanabria Arenas    -----
11
12 This code develops pythia hadronization. Takes as
13 parameter a .cmd file, where a .lhe file from MadGraph
14 and other parameters are specified. Then the code
15 produces .hep files after making the hadronization
16
17 Obs: The class MyUserHooks is written in order to
18 veto all the ISR emissions produced after the
19 first ISR parton. It is an extension of the code
```

```

20 hadronization01
21
22 run as ./hadronization02 input.cmnd [output.hep]
23
24 The MakeFile has been also modified to compile
25 this file
26 */
27
28 #include "Pythia8/Pythia.h"
29 #include "stdhep.h"
30 #include "stdcnt.h"
31 #include "stdhep_mcfio.h"
32 #include <string.h>
33
34 using namespace Pythia8;
35 void fill_stdhep(int i, Event &e);
36
37 // Write own derived UserHooks class.
38
39 class MyUserHooks : public UserHooks {
40
41 public:
42
43     // Constructor.
44     MyUserHooks() { }
45
46     // Destructor.
47     ~MyUserHooks() { }
48
49     // Allow a veto of ISR emissions
50     virtual bool canVetoISREmission(){
51         return true;    // Interrupts the initial shower
52                         // emission after each emission
53                         // and allow the emission to be vetoed by
54                         // the next method.
55     }
56
57     // Analyze each emission and asks for the number of
58     // the ISR emissions so far, in order
59     // to allow just 1 ISR parton per event

```



```

57     virtual bool doVetoISREmission(int sizeOld, const
      Event& event, int iSys){
58         // counts the number of ISR partons (i.e. the
          number of particles with status 43)
59         int ISR_part = 0;
60         for( int i = 0; i < event.size(); i++){
61             if (event[i].status() == 43 || event[i].status
              () == -43)
62                 ISR_part ++;
63         }
64         if (ISR_part > 1)
65             return true;
66         else
67             return false;
68     }
69 };
70
71 //=====
72
73
74 int main(int argc, char** argv) {
75
76     // Interface for conversion from Pythia8::Event to
      HepMC event.
77     char fileout[500], title[100];
78     strcpy(title, "output_pythia8\0");
79
80     // Set up generation.
81     // Declare Pythia object
82     Pythia pythia;
83
84     // Set simulation configurations. Read the file
      as parameter. If none, it reads hadro_input.
      cmdnd
85     if (argc > 1 ) pythia.readFile(argv[1]);
86     else {
87         cout << "ERROR: \n No parameters file has passed
          as parameter. Abort " << endl;
88         return 1;
89     }

```

```

90
91 // Specify the name of the output file
92 if (argc > 2 ) strcpy(fileout,argv[2]);
93 else strcpy(fileout,"output_pythia8.hep\0");
94
95 // Especificy the number of events
96     int nEvent = pythia.mode("Main:numberOfEvents");
97     // For reading only
98 int nAbort = 10; // Maximum number of failures
99     accepted
100 int iAbort = 0; // Abortions counter
101
102 // Necessary stdhep functions
103 int istr(0);
104 int ierr = StdHepXdrWriteOpen(fileout, title, nEvent,
105     istr);
106
107 // Set up to do a user veto and send it in.
108 MyUserHooks* myUserHooks = new MyUserHooks();
109 pythia.setUserHooksPtr( myUserHooks);
110
111 // Initialize simulation
112 pythia.init();
113
114 // Begin event loop; generate until none left in
115 // input file.
116 for (int iEvent = 0; iEvent < nEvent ; ++iEvent) {
117     // Generate events, and check whether generation
118     // failed.
119     if (!pythia.next()) {
120         // If failure because reached end of file then
121         // exit event loop.
122         if (pythia.info.atEndOfFile()) break;
123         // First few failures write off as "acceptable"
124         // errors, then quit.
125         if (++iAbort < nAbort) continue;
126         break;
127     }
128
129     // Fill stdhep file

```

```

123     fill_stdhep(iEvent+1,pythia.event);
124     ierr = StdHepXdrWrite(1,istr);
125 }
126
127 StdHepXdrEnd(istr);
128 pythia.stat();
129 cout << ierr;
130 delete myUserHooks;
131 return 0;
132
133 }
134
135 // This functions writes in stdhep format. It was
    written by Steve Mrenna
136 void fill_stdhep(int i, Event &e)
137 {
138     int num = e.size();
139     hepevt_.nevhep = i;
140     hepevt_.nhep = num;
141     for (int j = 0; j < num; j++) {
142         hepevt_.idhep[j] = e[j].id();
143         hepevt_.isthep[j] = e[j].statusHepMC();
144         hepevt_.jmohep[j][0] = (e[j].mother1()>0) ? e[j].
            mother1()+1 : 0;
145         hepevt_.jmohep[j][1] = (e[j].mother2()>0) ? e[j].
            mother2()+1 : 0;
146         hepevt_.jdahep[j][0] = (e[j].daughter1()>0) ? e[j]
            ].daughter1()+1 : 0;
147         hepevt_.jdahep[j][1] = (e[j].daughter2()>0) ? e[j]
            ].daughter2()+1 : 0;
148         hepevt_.phep[j][0] = e[j].px();
149         hepevt_.phep[j][1] = e[j].py();
150         hepevt_.phep[j][2] = e[j].pz();
151         hepevt_.phep[j][3] = e[j].e();
152         hepevt_.phep[j][4] = e[j].m();
153         hepevt_.vhhep[j][0] = e[j].xProd();
154         hepevt_.vhhep[j][1] = e[j].yProd();
155         hepevt_.vhhep[j][2] = e[j].zProd();
156         hepevt_.vhhep[j][3] = e[j].tProd();
157     }

```

158 }



A.2 Integration scripts: MadGraph + Pythia + Delphes

A.2.1 Configuration script: config_Integration.ini

```

1 # -----
2 # -----      Universidad de los Andes      -----
3 # -----      Departamento de Fisica        -----
4 # -----      Joven Investigador            -----
5 # -----      Andres Felipe Garcia Albarracin -----
6 # -----      Diego Alejandro Sanz Becerra   -----
7 # -----      Juan Carlos Sanabria Arenas    -----
8 # -----
9 # This file configures the inputs for MadGraph execution
10 # Based on Diego Sanz's configuration file:
11     configMGParallel.ini
12 ## EVENTSFOLDER IS THE NAME OF THE FOLDER WHERE ALL RUNS
13     WILL BE SAVED
14 EVENTSFOLDER="current_dir/_Channel_Events"
15 ## NAMESUBFOLDER IS THE NAME-STEM OF ALL THE RUNS. THE
16     SUBFOLDERS INSIDE EVENTSFOLDER WILL START WITH THIS
17 NAMESUBFOLDER="_Channel_Sim_"
18 ## MADGRAPHFOLDER IS THE LOCATION WHERE MADGRAPH IS
19     INSTALLED. USER SHOULD CHANGE THIS TO HIS MADGRAPH
20     INSTALLATION FOLDER
21 MADGRAPHFOLDER=
22 ## RUNCARDFOLDER IS THE LOCATION WHERE THE RUN_CARD
23     FRAME USED FOR ALL THE RUNS IS
24 RUNCARDFOLDER=
25 ## PARAMCARDFOLDER IS THE LOCATION WHERE THE PARAM_CARD
26     FOR ALL THE RUNS IS (check at the Madgraph folder: /
27     models/sm_v4, for instance)
28 PARAMCARDFOLDER=
29 ## MADGRAPHFILEFOLDER IS THE LOCATION WHERE THE MADGRAPH
30     -SCRIPT FRAME IS
31 MADGRAPHFILEFOLDER=
32 ## RUNCARDFILE IS THE NAME OF THE RUN_CARD FRAME USED

```

```

    FOR ALL THE RUNS
25 RUNCARDFILE="run_card.dat"
26 ## PARAMCARDFILE IS THE NAME OF THE PARAM_CARD USED FOR
    ALL THE RUNS
27 PARAMCARDFILE="param_card.dat"
28 ## MADGRAPHFILE IS THE NAME OF THE MADGRAPH-SCRIPT FRAME
    USED FOR ALL THE RUNS
29 MADGRAPHFILE="mgFile.mg5"
30 ## CORESNUMBER IS THE NUMER OF CORES USED FOR EACH RUN
31 CORESNUMBER=2
32 ## NUMEVENTSRUN IS THE NUMBER OF EVENTS FOR EACH OF THE
    RUNS
33 NUMEVENTSRUN=100000
34 ## INIRUN IS THE INITIAL SEED USED FOR THE PARALLEL RUNS
35 INIRUN=20
36 ## ENDRUN IS THE FINAL SEED USED FOR THE PARALLEL RUNS
37 ENDRUN=20
38
39 ## *** Pythia 8
40 ## DIRECTORY OF PYTHIA 8 EXECUTABLE (WHERE
    hadronization02 IS LOCATED)
41 PYTHIA8FOLDER=
42 ## PYTHIA 8 .EXE
43 PYTHIA8EXE="hadronization02"
44 ## PYTHIAPARAMFOLDER IS THE NAME OF THE FOLDER WHERE THE
    PYTHIA PARAMETER FILE IS LOCATED
45 PYTHIAPARAMFOLDER=
46 ## PYTHIAPARAM IS THE NAME OF THE .cmnd FILE THAT SERVES
    AS PARAMETER TO PYTHIA
47 PYTHIAPARAM="input_pythia.cmnd"
48
49 ## *** Delphes
50 ## DIRECTORY OF DELPHES EXEUTUABLE
51 DELPHESFOLDER=
52 ## DELPHES .EXE
53 DELPHESEXE="DelphesSTDHEP"
54 ## DELPHESCARDFOLDER IS THE NAME OF THE FOLDER WHERE THE
    DELPHES CARD IS LOCATED (check at the Delphes folder
    : /cards/)
55 DELPHESCARDFOLDER=

```

```

56 ## DELPHESCARD IS THE NAME OF THE .lct FILE THAT SERVES
    AS PARAMETER TO DELPHES
57 DELPHESCARD="delphes_card_CMS.tcl"
58
59 ## EXROOTANALYSIS
60 ## DIRECTORY OF EXROOTANALYSIS
61 EXROOTFOLDER=
62 ## EXROOT .EXE (STDHEP ---> .ROOT)
63 EXROOTEXE="ExRootSTDHEPConverter"

```

A.2.2 Execution script: script_Integration.sh

```

1  #!/bin/bash
2  # -----
3  # ----- Universidad de los Andes -----
4  # ----- Departamento de Fisica -----
5  # ----- Joven Investigador -----
6  # ----- Andres Felipe Garcia Albarracin -----
7  # ----- Diego Alejandro Sanz Becerra -----
8  # ----- Juan Carlos Sanabria Arenas -----
9  # -----
10 # This file executes parallel simulations with the
    programs: MadGraph 5.2 + Pythia 8.2 + Delphes 3.2
11 # Based on Diego Sanz's execution file:
    scriptMGParallelV2.sh
12
13 # Load the parameter file
14 source config_Integration.ini
15 ## make the RunCards Folder in the EVENTSFOLDER
16 mkdir ${EVENTSFOLDER}/RunCards
17 ## make the ParamCard Folder in the EVENTSFOLDER
18 mkdir ${EVENTSFOLDER}/ParamCard
19 ## copy the param card supplied to the EVENTSFOLDER/
    ParamCard and name it param_card.dat
20 cp ${PARAMCARDFOLDER}/${PARAMCARDFILE} ${EVENTSFOLDER}/
    ParamCard/param_card.dat
21
22 ## first sequence for each run, where the madgraph files
    and the run cards are created
23 sequ () {

```

```

24  ## copy the run card frame to the RunCards directory
    and append the seed (counter $i)
25  cp ${RUNCARDFOLDER}/${RUNCARDFILE} ${EVENTSFOLDER}/
    RunCards/run_card_${i}.dat
26  ## copy the MadGraph file to the RunCards directory
    as mgParallelFile_${i}
27  cp ${MADGRAPHFILEFOLDER}/${MADGRAPHFILE} ${
    EVENTSFOLDER}/RunCards/mgFile_${i}.mg5
28  ## copy the parameter pythia file to the RunCards
    directory
29  cp ${PYTHIAPARAMFOLDER}/${PYTHIAPARAM} ${EVENTSFOLDER
    }/RunCards/input_pythia_${i}.cmnd
30  ## copy the delphes card to the RunCards directory
    *** Delphes card is the same for all runs
31  cp ${DELPHESCARDFOLDER}/${DELPHESCARD} ${EVENTSFOLDER
    }/RunCards/${DELPHESCARD}
32  ## change all the instances of SEED to the counter $i
    on the file run_card_${i}.dat
33  sed -i "s/SEED/${i}/g" ${EVENTSFOLDER}/RunCards/
    run_card_${i}.dat
34  ## change all the instances of SEED to the counter $i
    on the file mgParallelFile_${i}.mg5
35  sed -i "s/SEED/${i}/g" ${EVENTSFOLDER}/RunCards/
    mgFile_${i}.mg5
36  ## change all the instances of SEED to the counter $i
    on the file input_pythia_${i}.cmnd
37  sed -i "s/SEED/${i}/g" ${EVENTSFOLDER}/RunCards/
    input_pythia_${i}.cmnd
38  ## change all the instances of RUNEVENTSNUM to
    $NUMEVENTSRUN on the file run_card_${i}.dat
39  sed -i "s/RUNEVENTSNUM/$NUMEVENTSRUN/g" ${
    EVENTSFOLDER}/RunCards/run_card_${i}.dat
40  ## change all the instances of FOLDEREVENTS to
    $EVENTSFOLDER on the file mgParallelFile.mg5
41  sed -i "s|FOLDEREVENTS|${EVENTSFOLDER}|g" ${
    EVENTSFOLDER}/RunCards/mgFile_${i}.mg5
42  ## change all the instances of NUMBERCORES to
    $CORESNUMBER on the file mgParallelFile.mg5
43  sed -i "s|NUMBERCORES|${CORESNUMBER}|g" ${EVENTSFOLDER
    }/RunCards/mgFile_${i}.mg5

```



```

44     ## change all the instances of SUBFOLDERNAME to
      $NAMESUBFOLDER on the file mgParallelFile_${i}.mg5
45     sed -i "s|SUBFOLDERNAME|$NAMESUBFOLDER|g" ${
      EVENTSFOLDER}/RunCards/mgFile_${i}.mg5
46     ## change all the instances of RESULTSFolder to the
      name of the folder where the results are located
47     sed -i "s|RESULTSFolder|${EVENTSFOLDER}/${
      NAMESUBFOLDER}_${i}/Events/run_01|g" ${EVENTSFOLDER
      }/RunCards/input_pythia_${i}.cmnd
48     ## change all the instances of RUNEVENTSNUM to
      $NUMEVENTSRUN on the file parameter pythia file
49     sed -i "s/RUNEVENTSNUM/$NUMEVENTSRUN/g" ${
      EVENTSFOLDER}/RunCards/input_pythia_${i}.cmnd
50 }
51
52 ## second sequence for each run, where the madgraph is
      called for each of the madgraph files (
      mgParallelFile_i.mg5). Pythia8 and Delphes are also
      executed
53 sequ2 () {
54     source config_Integration.ini
55     ## run madgraph with the corresponding madgraph
      file .mg5. all the messages are thrown to /
      dev/null
56     ## Madgraph execution
57     $1/bin/mg5_aMC -f $2/RunCards/mgFile_${4}.mg5 # &> /dev
      /null
58     ## sleep for 1s. Important, for the wait order
      to work
59     sleep 1s
60     ## wait for previous subprocesses to finish
61     wait
62     # Uncompress .lhe.gz file
63     gzip -d $2/$3_${4}/Events/run_01/unweighted_events.lhe.
      gz
64
65     ## Pythia 8 execution
66     ${PYTHIA8FOLDER}/${PYTHIA8EXE} $2/RunCards/
      input_pythia_${4}.cmnd $2/$3_${4}/Events/run_01/
      output_pythia8.hep # &> /dev/null

```

```

67
68     ## Delphes execution
69     ${DELPHESFOLDER}/${DELPHESEXE} $2/RunCards/${
        DELPHESCARD} $2/$3_$4/Events/run_01/output_delphes
        .root $2/$3_$4/Events/run_01/output_pythia8.hep
70
71     ## ExRootAnalysis execution
72     ${EXROOTFOLDER}/${EXROOTEXE} $2/$3_$4/Events/run_01/
        output_pythia8.hep $2/$3_$4/Events/run_01/
        output_pythia8.root
73
74     ## Remove unnecessary files
75     rm $2/$3_$4/Events/run_01/output_pythia8.hep
76
77 }
78
79 export -f sequ
80 export -f sequ2
81 ## start PARAMETERS variable
82 PARAMETERS=""
83 ## loop to execute sequence "sequ" for all the values
    from $INIRUN to $ENDRUN
84 for i in `seq ${INIRUN} ${ENDRUN}`; do # {21,28}; do ##
    `seq ${INIRUN} ${ENDRUN}`; do
85     ## execute sequ
86     sequ
87     ## concatenate the variable PARAMETERS with the
        current value of $i
88     PARAMETERS="$PARAMETERS ${i}"
89 done
90
91 ## execute gnuparallel. Use %% as the replacement string
    instead of {}.
92 parallel -0 -I %% --gnu "sequ2 ${MADGRAPHFOLDER} ${
    EVENTSFOLDER} ${NAMESUBFOLDER} %" ::: $PARAMETERS

```

Appendix B

Analysis codes

B.1 Tagging algorithm

```
1  /*
2  -----
3  -----      Universidad de los Andes      -----
4  -----      Departamento de Fisica        -----
5  -----      Joven Investigador            -----
6  -----      Andres Felipe Garcia Albarracin -----
7  -----      Juan Carlos Sanabria Arenas    -----
8  -----
9
10 This algorithm tags ISR jet in a certain sample.
11 It takes 2 N-dimensional histograms which contain
12 information about ISR and Non ISR Jets as input
13 and developes the ISR tagging in another sample.
14
15 The user can choose 3 of 8 variables for
16 developing the algorithm
17 1. PT
18 2. Abs(Eta) // Eta is a pair function
19 3. Delta Phi_MET
20 4. PT_ratio
21 5. Delta Eta_aver
```

```

22 6. Delta Phi_MET_others
23 7. Delta PT_others
24 8. Delta Eta_others
25
26 In order to choose them, the code should be run as:
27
28 ./ISR tagging config_file.txt [N1 N2 N3]
29
30 where [config_file.txt] is a configuration file with
31 all parameters needed for the simulation.
32
33 N1 N2 and N3 are the index of the 3 variables.
34 If no parameter is passed as parameter, N1 N2 and N3
35 will be 0,1 and 2 by default.
36
37 */
38
39
40 #include "ROOTFunctions.h"
41 #include "graphs_Funcs.h"
42 #include "functions.h"
43 #include "histoN.h"
44 #include "DelphesFunctions.h"
45
46 // Global Variables
47 const Double_t PI = TMath::Pi();
48
49 int main(int argc, char **argv){
50     std::cout.precision(4);
51     // Counting time
52     Double_t initialTime = clock();
53
54     // Folder variables
55     string head_folder = "/home/af.garcia1214/
        PhenoMCsamples/Simulations/
        MG_pythia8_delphes_parallel/
        _Tops_Events_WI_Matching/";
56     string current_folder = "_Tops_MG_1K_AG_WI_003/";
57
58     string head_folder_binary = "/home/af.garcia1214/

```

```

        PhenoMCsamples/Results_Improved_Codes/
        matching_Results/_Tops_matches_WI_Matching/";
59   string matching_name = "ISR_jets_Tops_WI_003.bn";
60
61   string head_folder_histos = "/home/af.garcia1214/
        PhenoMCsamples/Results_Improved_Codes/histo_folder
        /_Tops_histos_WI_Matching/";
62   string head_folder_results = "/home/af.garcia1214/
        PhenoMCsamples/Results_Improved_Codes/
        resultsTagging/_Tops_histos_WI_Matching/";
63
64   Bool_t ISR_OR_NOT = true;
65
66   // Variables for analysis
67   Double_t pt_cut = 0.0; // ISR jet pt cut
68   Double_t Jet_cut = 2;  // Ptobability cut 'K'
69   Bool_t do_pt_cut = false;
70   Bool_t do_jet_cut = false;
71
72   // Checking input parameters
73   string config_file_name = "Debug/config_file.txt";
74   // Reading the file as first parameter
75   if (argc>1){
76       config_file_name = argv[1];
77   }
78   else{
79       cout << "It is necessary to type a configuration
           file as parameter. Execute as ./ISR tagging
           config_file.txt [N1 N2 N3]" << endl;
80       return 1;
81   }
82   cout << "Reading input parameters" << endl;
83   cout << "\tUsing as parameters' file: " <<
        config_file_name << endl;
84
85   ifstream config_file (config_file_name);
86   if (config_file.is_open()){
87       cout << "\tReading file" << endl;
88       string line;
89       int number_line = 1;

```

```

90     while (getline(config_file,line)){
91         // Skipping commented lines
92         if (line[0] == '!')
93             continue;
94
95         // Finding the position of the equal sign
96         int pos_equal = -1;
97         pos_equal = line.find('=');
98
99         if (pos_equal == -1){
100             cout << "\tLine " << number_line << " is
                incorrect" << endl;
101             continue;
102         }
103
104         // Splitting the line according to the position
                of equal sign
105         string var_name = line.substr(0,pos_equal);
106         string var_value = line.substr(pos_equal+1);
107
108         // Reading head folder
109         if(var_name.compare("head_folder") == 0){
110             head_folder = var_value;
111             cout << "\tVariable head folder set as: " <<
                head_folder << endl;
112         }
113         // Reading current folder
114         else if (var_name.compare("current_folder") ==
                0){
115             current_folder = var_value;
116             cout << "\tVariable current folder set as: "
                << current_folder <<endl;
117         }
118         // Reading head folder binary
119         else if (var_name.compare("head_folder_binary")
                == 0){
120             head_folder_binary = var_value;
121             cout << "\tVariable head folder binary set
                as: " << head_folder_binary << endl;
122         }

```

```

123         // Reading matching name
124         else if (var_name.compare("matching_name") ==
125                 0){
126             matching_name = var_value;
127             cout << "\tVariable matching_name set as: "
128                 << matching_name << endl;
129         }
130         // Reading head folder histos
131         else if (var_name.compare("head_folder_histos")
132                 == 0){
133             head_folder_histos = var_value;
134             cout << "\tVariable head folder histos set
135                 as: " << head_folder_histos << endl;
136         }
137         // Reading head folder results
138         else if (var_name.compare("head_folder_results"
139                 ) == 0){
140             head_folder_results = var_value;
141             cout << "\tVariable head folder results set
142                 as: " << head_folder_results << endl;
143         }
144         // Reading pt_cut
145         else if (var_name.compare("pt_cut") == 0){
146             pt_cut = atof((Char_t *) var_value.c_str());
147             do_pt_cut = true;
148         }
149         // Reading jet_cut
150         else if (var_name.compare("Jet_cut") == 0){
151             Jet_cut = atof((Char_t *) var_value.c_str())
152                 ;
153             do_jet_cut = true;
154         }
155         // Reading ISR_OR_NOT
156         else if (var_name.compare("ISR_OR_NOT") == 0){
157             if (var_value.compare("1") == 0)
158                 ISR_OR_NOT = true;
159             else
160                 ISR_OR_NOT = false;
161         }

```

```

156         number_line ++;
157     }
158 }
159 else
160 {
161     cout << "ERROR: File " << config_file_name << "
162           does not exist. Terminating program" << endl;
163     return 0;
164 }
165 cout << "\n *** Running the tagging Algorithm *** \n"
166       << endl;
167
168 // Variables for initializing histograms
169 Int_t dims = 3;
170
171 /*
172  * Read inputs and set variables for analysis
173  */
174 Int_t var_index[3] = {0,1,2}; // Index of the 3
175                               // variables for analysis. By default 0, 1 and 2
176 string variables[8] = {"PT","Abs(Eta)","Delta Phi_MET",
177                       "PT_ratio","Delta Eta_aver","Delta
178                       Phi_MET_others","Delta PT_leading","Delta
179                       Eta_leading"};
180 Double_t var_values[8] =
181     {0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0}; // Vector with
182     the values of the 8 variables
183
184 if (argc == 2) {
185     cout << "\tRunning the algorithm with the default
186           variables:" << endl;
187 }
188 else if (argc >= 5){
189     cout << "\tRunning the algorithm with the
190           variables:" << endl;
191     for (Int_t ind = 0; ind < 3; ind++){
192         var_index[ind] = atoi(argv[ind+2]);
193     }

```



```

186     }
187     else {
188         cout << "\tError at calling this algorithm. Use as
            : " << endl;
189         cout << "\t ./ISR_tagging config_file.txt,  ./
            ISR_tagging config_file.txt N1 N2 N3 or just ./
            ISR_tagging" << endl;
190         cout << "\tRead the documentation at the beginning
            of the code for further information\n" << endl
            ;
191         return 1;
192     }
193
194     cout << "\tVar \t\t min_Value \t max_Value" << endl;
195     for (Int_t ind = 0; ind < 3; ind ++){
196
197         cout << "\t" << var_index[ind] << ". " <<
            variables[var_index[ind]] << endl;
198     }
199     cout << endl;
200
201     cout << "\tTransverse momentum of the ISR: " <<
        pt_cut << endl;
202
203     /*
204     * Initializing the 3-dimensional histogram
205     */
206     // Defining the names of the files
207     string combination = "_____"; // Combination of
        variables
208     for (Int_t ind = 0; ind < dims; ind ++){
209         combination[(ind*2)+1] = (Char_t) (0x30 +
            var_index[ind]); // Int to char
210     }
211
212     string info_ISR_name_str = head_folder_histos + "
        info_histo_ISR" + combination + ".txt";
213     Char_t *info_ISR_name = (Char_t *) info_ISR_name_str.
        c_str();
214

```

```

215     string array_ISR_name_str = head_folder_histos + "
        array_histo_ISR" + combination + ".bn";
216     Char_t *array_ISR_name = (Char_t *)
        array_ISR_name_str.c_str();
217
218     string info_Non_ISR_name_str = head_folder_histos + "
        info_histo_Non_ISR" + combination + ".txt";
219     Char_t *info_Non_ISR_name = (Char_t *)
        info_Non_ISR_name_str.c_str();
220
221     string array_Non_ISR_name_str = head_folder_histos +
        "array_histo_Non_ISR" + combination + ".bn";
222     Char_t *array_Non_ISR_name = (Char_t *)
        array_Non_ISR_name_str.c_str();
223
224     histoN* histoISR = new histoN(info_ISR_name,
        array_ISR_name);
225     histoN* histoNonISR = new histoN(info_Non_ISR_name,
        array_Non_ISR_name);
226
227     cout << "\tEntradas ISR: " << histoISR->getEntries()
        << endl;
228     cout << "\tEntradas FSR: " << histoNonISR->getEntries
        () << endl;
229
230     // Input variables of each histogram
231     Double_t values[3] = {0.0,0.0,0.0};
232
233     /*
234     * MET histograms
235     */
236     TH1 *h_MET = new TH1F("Missing ET","All events"
        ,300,0,1000);
237     Char_t *name_histo_MET;
238     name_histo_MET = (Char_t*) malloc(sizeof(char)*512);
239     strcpy(name_histo_MET,"ISR jet PT > ");
240     Char_t pt_str[] = " ";
241     pt_str[0] = 0x30 + int(pt_cut/100)%10;
242     pt_str[1] = 0x30 + int(pt_cut/10)%10;
243     pt_str[2] = 0x30 + int(pt_cut)%10;

```

```

244   strcat(name_histo_MET,pt_str);
245   strcat(name_histo_MET,"-k = ");
246   Char_t k_str[] = "   ";
247   k_str[0] = 0x30 + int(Jet_cut)%10;
248   k_str[1] = '.';
249   k_str[2] = 0x30 + int(Jet_cut*10)%10;
250   strcat(name_histo_MET,k_str);
251   TH1 *h_MET_hpt1 = new TH1F(name_histo_MET,"Missing ET
      high_ISR_pt-1",300,0.0,1000);

252
253   if (do_jet_cut && do_pt_cut)
254       cout << "\tThe algorithm will evaluate the MET for
          a sample with PT > " << pt_str << " at k = "
          << k_str << endl;

255   /*
256    * Tagging variables
257    */
258
259   cout << "\tJet cut, k = " << Jet_cut << endl;
260
261   // Arrays with the number of tags, Misstags and
       events rejected
262   // Probability cut
263   Double_t Prob_cut = 0;
264   Double_t k_min = 1.2; // Minimum probability cut =
       k_min/num_jets
265   Double_t k_max = 3.0; // Maximum probability cut =
       k_max/num_jets
266   Int_t k_bins = 100; // Number of values of k between
       k_min and k_max
267   Double_t k_step = (Double_t) (k_max-k_min)/k_bins;
268   Double_t k_values[k_bins];
269   for(Int_t ind = 0; ind < k_bins; ind++){
270       k_values[ind] = k_min + k_step*ind;
271   }
272
273   // Tagging results
274   Int_t Num_Tags = 0;
275   Int_t Num_MisTags = 0;
276   Int_t Num_Rejected = 0;

```

```

277
278     Double_t Num_Tags_array[k_bins];
279     Double_t Num_MissTags_array[k_bins];
280     Double_t Num_Rejected_array[k_bins];
281     Double_t Num_Total_Jets[k_bins];
282
283     Double_t Num_Tags_array_hpt[k_bins];
284     Double_t Num_MissTags_array_hpt[k_bins];
285     // Double_t Num_Rejected_array_hpt[k_bins];
286     Double_t Num_Total_Jets_hpt[k_bins];
287
288
289     for (Int_t ind = 0; ind < k_bins; ind++){
290         Num_Tags_array[ind] = 0;
291         Num_MissTags_array[ind] = 0;
292         Num_Rejected_array[ind] = 0;
293         Num_Total_Jets[ind] = 0;
294         Num_Tags_array_hpt[ind] = 0;
295         Num_MissTags_array_hpt[ind] = 0;
296         // Num_Rejected_array_hpt[ind] = 0;
297         Num_Total_Jets_hpt[ind] = 0;
298     }
299
300     // Variables of the ISR tagging algorithm
301     Double_t H_ISR, H_Non_ISR, alpha;
302     Double_t prob_max = 0;
303     Double_t probISR = 0;
304     Double_t k_ISR = 0;
305     Double_t k_ISR_pos = 0; // Position of the ISR in the
                             // vector
306     Int_t ISR_tag_index = -1;
307
308     // Cycle over several runs . iRun corresponds to the
                             // seed of the current run
309     for(int iRun = 261; iRun < 270; iRun++){
310         // Create chains of root trees
311         TChain chain_Delphes("Delphes");
312
313         Char_t unidad = 0x30 + iRun%10;
314         Char_t decena = 0x30 + int(iRun/10)%10;

```

```

315     Char_t centena = 0x30 + int(iRun/100)%10;
316
317     current_folder[current_folder.size()-4] = centena;
318     current_folder[current_folder.size()-3] = decena;
319     current_folder[current_folder.size()-2] = unidad;
320     matching_name[matching_name.size()-6] = centena;
321     matching_name[matching_name.size()-5] = decena;
322     matching_name[matching_name.size()-4] = unidad;
323
324     string file_delphes_str = head_folder +
        current_folder + "Events/run_01/output_delphes.
        root";
325
326     Char_t *file_delphes = (Char_t *) file_delphes_str
        .c_str();
327
328     cout << "\n\tStudying run: " << centena << decena <<
        unidad << endl;
329     cout << "\tReading the file: \n\tDelphes: " <<
        file_delphes << endl;
330
331     chain_Delphes.Add(file_delphes);
332     // Objects of class ExRootTreeReader for reading
        the information
333     ExRootTreeReader *treeReader_Delphes = new
        ExRootTreeReader(&chain_Delphes);
334
335     Long64_t numberOfEntries = treeReader_Delphes->
        GetEntries();
336
337     // Get pointers to branches used in this analysis
338     TClonesArray *branchJet = treeReader_Delphes->
        UseBranch("Jet");
339     TClonesArray *branchMissingET = treeReader_Delphes
        ->UseBranch("MissingET");
340
341     cout << "\tNumber of Entries Delphes = " <<
        numberOfEntries << endl;
342     cout << endl;
343

```

```

344 // particles, jets and vectors
345 MissingET *METpointer;
346 TLorentzVector *vect_currentJet = new
    TLorentzVector;
347 TLorentzVector *vect_auxJet = new TLorentzVector;
348 TLorentzVector *vect_leading = new TLorentzVector;
349 Jet *currentJet = new Jet;
350 Jet *auxJet = new Jet;
351
352 // Temporary variables
353 Double_t MET = 0.0; // Missing transverse energy
354 Double_t delta_phi = 0.0; // difference between
    the phi angle of MET and the jet
355 Double_t transverse_mass = 0.0; // Transverse mass
356 Double_t delta_PT_jet = 0.0; // |PT-<PT>|
357 Double_t PT_sum = 0.0; // sum(PT)
358 Double_t PT_aver = 0.0; // <PT>
359 Double_t Delta_eta_aver = 0.0; // sum_i|eta-eta_i
    |/(Nj-1)
360 Double_t Delta_phi_sum = 0.0; // sum delta_phi
361 Double_t Delta_phi_other_jets = 0.0; // Average of
    delta phi of other jets
362 Double_t PT_ratio = 0.0; // PT/PT_others
363 Double_t Delta_PT_leading = 0.0; // PT -
    PT_leading
364 Double_t Delta_Eta_leading = 0.0; // |Eta -
    Eta_leading|
365
366 // Jet with greatest PT
367 Double_t PT_max = 0;
368 Int_t posLeadingPT = -1;
369 Int_t ISR_greatest_PT = 0;
370 Double_t MT_leading_jet = 0.0; // Transverse mass
371
372 /*
373  * Some variables used through the code
374  */
375 Int_t ISR_jets[numberOfEntries];
376 Int_t NumJets = 0;
377

```

```

378     string fileName_str = head_folder_binary +
        matching_name;
379
380     Char_t * fileName = (Char_t *) fileName_str.c_str
        ();
381
382     if (ISR_OR_NOT == true){
383         ifstream ifs(fileName, ios::in | ios::binary);
384
385         for (Int_t j = 0; j<numberOfEntries; j++){
386             ifs.read((Char_t *) (ISR_jets+j), sizeof(
                Int_t));
387         }
388         ifs.close();
389     }
390     else{
391         for (Int_t j = 0; j<numberOfEntries; j++){
392             ISR_jets[j] = -2; // There is not ISR jet
                but also there is not matching
393         }
394     }
395
396     /*
397     * Main cycle of the program
398     */
399     numberOfEntries = 100000;
400     for (Int_t entry = 0; entry < numberOfEntries; ++
        entry){
401         // Progress
402         if(numberOfEntries>10 && (entry%((int)
            numberOfEntries/10))==0.0){
403             cout<<"\tprogress = "<<(entry*100/
                numberOfEntries)<<"%\t";
404             cout<< "Time : "<< (clock()-initialTime)/
                double_t(CLOCKS_PER_SEC)<<"s"<<endl;
405         }
406
407         // Load selected branches with data from
            specified event
408         treeReader_Delphes->ReadEntry(entry);

```

```
409
410     // MET
411     METpointer = (MissingET*) branchMissingET->At
        (0);
412     MET = METpointer->MET;
413
414     NumJets=branchJet->GetEntries();
415
416     // checking the ISR
417     if (NumJets < 3 || ISR_jets[entry] == -1)
418         continue;
419
420     h_MET->Fill(MET);
421
422     if (ISR_jets[entry] >= NumJets){
423         cout << "Error en el matching" << endl;
424         return 1;
425     }
426
427     // 3 PT ratio
428     PT_aver = 0.0;
429     PT_sum = 0.0;
430     PT_ratio = 0.0;
431
432     // 4 Delta Eta aver
433     Delta_eta_aver = 0.0;
434
435     // 5 Delta Phi others
436     Delta_phi_sum = 0.0;
437     Delta_phi_other_jets = 0.0;
438
439     // 6 Delta PT leading
440     PT_max = 0.0;
441     Delta_PT_leading = 0.0;
442     delta_PT_jet = 0.0; // If needed
443
444     // 7 Delta Eta leading
445     Delta_Eta_leading = 0.0;
446
447     // Reset Var_values (Not necessary)
```



```

448     for(Int_t ind = 0; ind < 8; ind++){
449         var_values[ind] = 0.0;
450         if (ind < dims) values[ind] = 0.0;
451     }
452
453     // Preliminary for. It is used to calculate
454     // PT_aver and Delta_phi_sum
455     for (Int_t iJet = 0; iJet<NumJets; iJet++){
456         currentJet = (Jet*) branchJet->At(iJet);
457         vect_currentJet->SetPtEtaPhiM(currentJet->PT
458             ,currentJet->Eta,currentJet->Phi,
459             currentJet->Mass);
460         PT_sum += vect_currentJet->Pt();
461         delta_phi = deltaAng(vect_currentJet->Phi(),
462             METpointer->Phi());
463         Delta_phi_sum += delta_phi;
464         // PT Leading jet
465         if(PT_max < vect_currentJet->Pt()){
466             PT_max = vect_currentJet->Pt();
467             posLeadingPT = iJet;
468         }
469     }
470
471     //PT_aver
472     PT_aver = PT_sum/NumJets;
473
474     // Leading PT
475     currentJet = (Jet*) branchJet->At(posLeadingPT)
476         ;
477     vect_leading->SetPtEtaPhiM(currentJet->PT,
478         currentJet->Eta,currentJet->Phi,currentJet->
479         Mass);
480
481     // The best ISR candidate
482     TLorentzVector *vect_optimum = new
483         TLorentzVector;
484
485     // Reset variables
486     probISR = 0.0;
487     k_ISR = 0.0;

```

```

480     prob_max = 0;
481     ISR_tag_index = -1;
482
483     for (Int_t iJet = 0; iJet<NumJets; iJet++){
484         currentJet = (Jet*) branchJet->At(iJet);
485         vect_currentJet->SetPtEtaPhiM(currentJet->PT
            ,currentJet->Eta,currentJet->Phi,
            currentJet->Mass);
486
487         // 2 Delta Phi MET
488         delta_phi = deltaAng(vect_currentJet->Phi(),
            METpointer->Phi);
489
490         // PT ratio
491         PT_ratio = vect_currentJet->Pt()*(NumJets-1)
            /(PT_sum-vect_currentJet->Pt());
492
493         // 4 Delta Eta Aver
494         Delta_eta_aver = 0.0;
495         // For cycle used to calculate
            Delta_eta_aver
496         for(Int_t iJet2 = 0; iJet2<NumJets; iJet2++)
            {
497             auxJet = (Jet*) branchJet->At(iJet2);
498             vect_auxJet->SetPtEtaPhiM(auxJet->PT,
                auxJet->Eta,auxJet->Phi,auxJet->Mass);
499             if (iJet2 != iJet) Delta_eta_aver +=
                TMath::Abs(vect_auxJet->Eta() -
                vect_currentJet->Eta());
500         }
501         Delta_eta_aver = Delta_eta_aver/(NumJets-1);
502
503         // 5 Delta Phi MET Others
504         Delta_phi_other_jets = (Delta_phi_sum -
            delta_phi)/(NumJets-1);
505
506         // 6 Delta PT leading
507         Delta_PT_leading = vect_leading->Pt() -
            vect_currentJet->Pt();
508

```

```

509      // 7 Delta Eta leading
510      Delta_Eta_leading = TMath::Abs(
          vect_currentJet->Eta()-vect_leading->Eta
          ());
511
512      // Other variables
513      delta_PT_jet = TMath::Abs(vect_currentJet->
          Pt()-PT_aver);
514      transverse_mass = sqrt(2*vect_currentJet->Pt
          ()*MET*(1-cos(delta_phi)));
515
516      // Filling the array with the variables'
          values
517      var_values[0] = vect_currentJet->Pt();
518      var_values[1] = TMath::Abs(vect_currentJet->
          Eta());
519      var_values[2] = delta_phi;
520      var_values[3] = PT_ratio;
521      var_values[4] = Delta_eta_aver;
522      var_values[5] = Delta_phi_other_jets;
523      var_values[6] = Delta_PT_leading;
524      var_values[7] = Delta_Eta_leading;
525
526      for (Int_t ind = 0; ind < dims; ind++){
527          int pos = *(var_index+ind);
528          values[ind] = *(var_values+pos);
529      }
530
531      // Comparing with histos
532      H_ISR = histoISR->getProbVal(values);
533      H_Non_ISR = histoNonISR->getProbVal(values);
534
535      if (H_ISR > 3e-7 || H_Non_ISR > 3e-7){
536          alpha = NumJets/(H_Non_ISR*(NumJets-1)+
          H_ISR);
537          probISR = alpha*H_ISR/NumJets;
538
539          if(probISR > (1.0 + 1.0e-10)){
540              cout << setprecision(20) << "\n\t ***
          ERROR: La probabilidad no puede ser

```

```

541         mayor a 1 ***" << endl;
542         return 1;
543     }
544     if (probISR >= prob_max){
545         prob_max = probISR;
546         vect_optimum->SetPtEtaPhiM(
            vect_currentJet->Pt(),
            vect_currentJet->Eta(),
            vect_currentJet->Phi(),
            vect_currentJet->M());
547         ISR_tag_index = iJet;
548     }
549 }
550 }
551
552 k_ISR = prob_max*NumJets;
553
554 // Check the tagging results
555 k_ISR_pos = findPosition(k_min,k_max,k_bins,
    k_ISR);
556
557 if(k_ISR == 0.0) k_ISR_pos = -1;
558
559 if (ISR_jets[entry] != -1 && ISR_OR_NOT == true
    ){
560     // A comparison can be handled
561     for (Int_t ind = 0; ind < k_ISR_pos + 1; ind
        ++){
562         if (ISR_tag_index == ISR_jets[entry])
563             Num_Tags_array[ind]++;
564         else
565             Num_MissTags_array[ind]++;
566     }
567     for (Int_t ind = k_ISR_pos+1; ind < k_bins;
        ind++){
568         Num_Rejected_array[ind]++;
569     }
570 }
571 else if (ISR_jets[entry] == -2 && ISR_OR_NOT ==

```

```

572         false){
573         for (Int_t ind = 0; ind < k_ISR_pos + 1; ind
574             ++){
575             Num_MissTags_array[ind]++;
576         }
577         for (Int_t ind = k_ISR_pos+1; ind < k_bins;
578             ind++){
579             Num_Rejected_array[ind]++;
580         }
581     }
582
583     if (ISR_tag_index != -1 && vect_optimum->Pt() >
584         pt_cut && ISR_OR_NOT == true){ // != S
585         means bb or WI
586         for (Int_t ind = 0; ind < k_ISR_pos + 1; ind
587             ++){
588             if (ISR_tag_index == ISR_jets[entry])
589                 Num_Tags_array_hpt[ind]++;
590             else
591                 Num_MissTags_array_hpt[ind]++;
592         }
593         for (Int_t ind = k_ISR_pos+1; ind < k_bins;
594             ind++){
595             Num_Rejected_array_hpt[ind]++; //
596             // Under a certain k_cut, there are not rejected events
597         }
598     }
599
600     Prob_cut = Jet_cut/NumJets;
601     if(prob_max >= Prob_cut){
602         if (ISR_tag_index == ISR_jets[entry] &&
603             ISR_OR_NOT == true) // != S means bb or
604             WI
605             Num_Tags++;
606         else
607             Num_MissTags++;
608
609         // Cheching MET boosting
610         if(vect_optimum->Pt() > pt_cut){
611             h_MET_hpt1->Fill(MET);

```

```

602         }
603     }
604     else
605         Num_Rejected++;
606
607 }
608
609 cout<<"\tprogress = 100%\t";
610 cout<<"Time : "<< (clock()-initialTime)/double_t(
        CLOCKS_PER_SEC)<<"s"<<endl;
611
612 } // End run's for cicle
613
614 /*
615  * Tagging results
616  */
617
618 Int_t Num_Studied = Num_Tags + Num_MissTags +
        Num_Rejected;
619 cout << "\nOverall tagging results" << endl;
620 cout << "\tNumber of compared events (between the
        matching and tagging algorithms) : " <<
        Num_Studied << endl;
621 cout << "\tPer. Tags: \t" << ((Double_t)Num_Tags/
        Num_Studied)*100 << "%" << endl;
622 cout << "\tPer. MissTags: \t" << ((Double_t)
        Num_MissTags/Num_Studied)*100 << "%" << endl;
623 cout << "\tPer. Rejected: \t" << ((Double_t)
        Num_Rejected/Num_Studied)*100 << "%" << endl;
624
625 // Calculating percentages
626 for (Int_t ind=0; ind < k_bins; ind++){
627     Num_Total_Jets[ind] = Num_Tags_array[ind] +
        Num_MissTags_array[ind] + Num_Rejected_array[
        ind];
628     Num_Tags_array[ind] = Num_Tags_array[ind]/
        Num_Total_Jets[ind];
629     Num_MissTags_array[ind] = Num_MissTags_array[ind]/
        Num_Total_Jets[ind];
630     Num_Rejected_array[ind] = Num_Rejected_array[ind]/

```

```

        Num_Total_Jets[ind];
631     Num_Total_Jets_hpt[ind] = Num_Tags_array_hpt[ind]
        + Num_MissTags_array_hpt[ind]; // +
        Num_Rejected_array_hpt[ind];
632     Num_Tags_array_hpt[ind] = Num_Tags_array_hpt[ind]/
        Num_Total_Jets_hpt[ind];
633     Num_MissTags_array_hpt[ind] =
        Num_MissTags_array_hpt[ind]/Num_Total_Jets_hpt[
        ind];
634 //     Num_Rejected_array_hpt[ind] =
        Num_Rejected_array_hpt[ind]/Num_Total_Jets_hpt[ind];
635 }
636
637 /*
638  * Writing results
639  */
640 Bool_t archivoExiste = false;
641
642 Char_t outNamept[] = "Percenta_hpt-100";
643 outNamept[13] = 0x30 + int(pt_cut/100)%10;
644 outNamept[14] = 0x30 + int(pt_cut/10)%10;
645 outNamept[15] = 0x30 + int(pt_cut)%10;
646
647 string outFileTotal_str = head_folder_results + "
        Overall_percs" + combination + ".txt";
648 Char_t *outFileTotal = (Char_t *) outFileTotal_str.
        c_str();
649
650 string outFileTotalpt_str = head_folder_results +
        outNamept + combination + ".txt";
651 Char_t *outFileTotalpt = (Char_t *)
        outFileTotalpt_str.c_str();
652
653 ifstream my_file(outFileTotal);
654 if(my_file.good()){
655     archivoExiste = true;
656 }
657 my_file.close();
658
659 ofstream ofs_over(outFileTotal,ios::out);

```

```

660     if(!archivoExiste){
661         // If file already exists
662     }
663
664     ofs_over << "# Number of Tags, Misstags and Rejected
        as a function of k" << endl;
665     ofs_over << "# Number of Events " << Num_Total_Jets
        [0] << endl;
666     ofs_over << "# k_cut \t Tags \t MissTags \t Rejected
        \t Total_Events " << endl;
667
668
669     for (Int_t ind = 0; ind < k_bins; ind++){
670         ofs_over << setiosflags(ios::fixed) <<
            setprecision(6) << setw(6) << k_values[ind]
671             << "\t" << Num_Tags_array[ind] << "\t" <<
            Num_MissTags_array[ind] << "\t" <<
            Num_Rejected_array[ind]
672             << "\t" << setprecision(0) << Num_Total_Jets
            [ind] << endl;
673     }
674
675     if (do_pt_cut){
676         ofstream ofs_pt(outFileTotalpt,ios::out);
677         ofs_pt << "# Number of Tags, Misstags and Rejected
            as a function of k. The ISR has pt > " <<
            pt_cut << endl;
678         ofs_pt << "# Number of Events " <<
            Num_Total_Jets_hpt[0] << endl;
679         ofs_pt << "# k_cut \t Tags \t MissTags \t
            Total_Events " << endl;
680         for (Int_t ind = 0; ind < k_bins; ind++){
681             ofs_pt << setiosflags(ios::fixed) <<
                setprecision(6) << setw(6) << k_values[ind]
682                 << "\t" << Num_Tags_array_hpt[ind] << "\t"
                << Num_MissTags_array_hpt[ind] // <<
                << "\t" << Num_Rejected_array_hpt[ind]
683                 << "\t" << setprecision(0) <<
                Num_Total_Jets_hpt[ind] << endl;
684         }

```



```

685     ofs_pt.close();
686 }
687
688 if (do_jet_cut){
689     Char_t outNameMET[] = "AbsValue_MET_pt_000_k_2.0";
690     outNameMET[16] = pt_str[0];
691     outNameMET[17] = pt_str[1];
692     outNameMET[18] = pt_str[2];
693     outNameMET[22] = k_str[0];
694     outNameMET[23] = k_str[1];
695     outNameMET[24] = k_str[2];
696
697     string outFileMET_str = head_folder_results +
        outNameMET + combination;
698     Char_t *outFileMET = (Char_t *) outFileMET_str.
        c_str();
699
700     Char_t *outFilehist;
701     outFilehist = (Char_t*) malloc(sizeof(char)*512);
702     strcpy(outFilehist,outFileMET);
703     strcat(outFilehist,".root");
704
705     TFile* hfile = new TFile("histos.root", "RECREATE"
        );
706     TCanvas *C = new TCanvas(outFileMET,"MET in a
        sample with high PT ISR jets",1280,720);
707     Present(h_MET,h_MET_hpt1,C,2,"MET [GeV]","Num.
        Jets / Total");
708     C->Write();
709     C->Close();
710     hfile->Close();
711
712 }
713
714 ofs_over.close();
715
716 cout<<"\nEND :)"<<endl;
717
718 return 0;
719 }

```

B.2 Matching algorithm

```

1  /*
2  -----
3  -----      Universidad de los Andes      -----
4  -----      Departamento de Fisica      -----
5  -----      Joven Investigador          -----
6  -----      Andres Felipe Garcia Albarracin -----
7  -----      Juan Carlos Sanabria Arenas    -----
8  -----
9
10 This algorithm looks for the ISR parton into the
11 pythia8 simulation file and then finds the
12 corresponding ISR jet
13
14 It also stores in a binary file the matching
15 results
16
17 To run, type
18
19 ./ISR_matching_improved [config.txt] [000]
20
21 where [config.txt] is the configuration file and
22 [000] is the seed of the simulation under analysis
23 */
24
25 #include <iostream>
26 #include "ROOTFunctions.h"
27 #include "graphs_Funcs.h"
28 #include "functions.h"
29 #include "DelphesFunctions.h"
30
31 using namespace std;
32 // Global Variables
33 const Double_t PI = TMath::Pi();
34
35 int main(int argc, char **argv){
36
37     std::cout.precision(4);
38     // Counting time

```

```

39     Double_t initialTime = clock();
40
41     // Folder variables
42     string head_folder = "/home/af.garcia1214/
        PhenoMCsamples/Simulations/
        MG_pythia8_delphes_parallel/
        _Tops_Events_WI_Matching/";
43     string current_folder = "_Tops_MG_1K_AG_WI_003/";
44
45     string head_folder_results = "/home/af.garcia1214/
        PhenoMCsamples/Results_Improved_Codes/
        matching_Results/_Tops_matches_WI_Matching/";
46     string matching_name = "ISR_jets_Tops_WI_003.bn";
47
48     // Checking input parameters
49     string config_file_name = "Debug/config_file.txt";
50     // Reading the file as first parameter
51     if (argc>1){
52         config_file_name = argv[1];
53     }
54     else{
55         cout << "It is necessary to type a configuration
            file as parameter. Execute as ./ISR_matching
            config_file.txt [000]" << endl;
56         return 1;
57     }
58     cout << "Reading input parameters" << endl;
59     cout << "\tUsing as parameters' file: " <<
        config_file_name << endl;
60
61     ifstream config_file (config_file_name);
62     if (config_file.is_open()){
63         cout << "\tReading file" << endl;
64         string line;
65         int number_line = 1;
66         while (getline(config_file,line)){
67             // Skipping commented lines
68             if (line[0] == '!')
69                 continue;
70

```

```

71 // Finding the position of the equal sign
72 int pos_equal = -1;
73 pos_equal = line.find('=');
74
75 if (pos_equal == -1){
76     cout << "\tLine " << number_line << " is
77         incorrect" << endl;
78     continue;
79 }
80 // Splitting the line according to the position
81 // of equal sign
82 string var_name = line.substr(0,pos_equal);
83 string var_value = line.substr(pos_equal+1);
84
85 // Reading head folder
86 if(var_name.compare("head_folder") == 0){
87     head_folder = var_value;
88     cout << "\tVariable head folder set as: " <<
89         head_folder << endl;
90 }
91 // Reading current folder
92 else if (var_name.compare("current_folder") ==
93     0){
94     current_folder = var_value;
95     cout << "\tVariable current folder set as: "
96         << current_folder <<endl;
97 }
98 // Reading head folder results
99 else if (var_name.compare("head_folder_results"
100     ) == 0){
101     head_folder_results = var_value;
102     cout << "\tVariable head folder results set
103         as: " << head_folder_results << endl;
104 }
105 // Reading matching name
106 else if (var_name.compare("matching_name") ==
107     0){
108     matching_name = var_value;
109     cout << "\tVariable matching_name set as: "

```

```

103         << matching_name << endl;
104     }
105     number_line ++;
106 }
107 }
108 else
109 {
110     cout << "ERROR: File " << config_file_name << "
111         does not exist. Terminating program" << endl;
112     return 0;
113 }
114 // Reading the seed of the simulation. This parameter
115 // is optional and is the second of argv
116 Char_t unidad = '3'; Char_t decena = '0'; Char_t
117     centena = '0';
118 if (argc > 2){
119     cout << "\tRemember: The number of the simulation
120         should consist of 3 digits" << endl;
121     centena = argv[2][0];
122     decena = argv[2][1];
123     unidad = argv[2][2];
124     current_folder[current_folder.size()-4] = centena;
125     current_folder[current_folder.size()-3] = decena;
126     current_folder[current_folder.size()-2] = unidad;
127     matching_name[matching_name.size()-6] = centena;
128     matching_name[matching_name.size()-5] = decena;
129     matching_name[matching_name.size()-4] = unidad;
130 }
131
132 cout << "\tThe seed of the simulation is: " <<
133     centena << decena << unidad << endl;
134
135 // Full path name of pythia and Delphes simulations
136 string file_pythia_str = head_folder + current_folder
137     + "Events/run_01/output_pythia8.root";
138 Char_t *file_pythia = (Char_t *) file_pythia_str.
139     c_str(); //Pass string to char_t *

```

```

135
136     string file_delphes_str = head_folder +
        current_folder + "Events/run_01/output_delphes.
        root";
137     Char_t *file_delphes = (Char_t *) file_delphes_str.
        c_str();
138
139     if (argc > 2){
140         cout << "\n\tReading the files: \n\tPythia8: " <<
            file_pythia << "\n\tDelphes: " << file_delphes
            << endl;
141     }
142     else
143         cout << "\n\tReading the default files: \n\
            tPythia8: " << file_pythia << "\n\tDelphes: "
            << file_delphes << endl;
144
145
146     // Loading simulations of Pythia and Delphes
147     cout << "\nLoading simulations of Pythia and Delphes"
        << endl;
148     // Create chains of root trees
149     TChain chain_Pythia("STDHEP");
150     TChain chain_Delphes("Delphes");
151
152     chain_Pythia.Add(file_pythia);
153     chain_Delphes.Add(file_delphes);
154
155     // Objects of class ExRootTreeReader for reading the
        information
156     ExRootTreeReader *treeReader_Pythia = new
        ExRootTreeReader(&chain_Pythia);
157     ExRootTreeReader *treeReader_Delphes = new
        ExRootTreeReader(&chain_Delphes);
158
159     Long64_t numberOfEntries = treeReader_Pythia->
        GetEntries();
160     Long64_t numberOfEntries_Delphes = treeReader_Delphes
        ->GetEntries();
161

```

```

162 // Get pointers to branches used in this analysis
163 TClonesArray *branchParticlePythia =
    treeReader_Pythia->UseBranch("GenParticle");
164 TClonesArray *branchJet = treeReader_Delphes->
    UseBranch("Jet");
165 TClonesArray *branchMissingET = treeReader_Delphes->
    UseBranch("MissingET");
166
167 cout << endl;
168 cout << "\tNumber of Entries Pythia = " <<
    numberOfEntries << endl;
169 cout << "\tNumber of Entries Delphes = " <<
    numberOfEntries_Delphes << endl;
170 cout << endl;
171
172 // particles, jets and vectors
173 TRootGenParticle *particle_pythia;
174 TRootGenParticle *ISR_particle;
175 MissingET *METpointer;
176 TLorentzVector *vect_ISR_particle = new
    TLorentzVector;
177
178 // Temporary variables
179 Bool_t ISR_parton_found = false; // true if the
    initial ISR_parton (with status 43) was found
180 Int_t pos_ISR = -1; // position of the ISR_parton
    into the branchParticlePythia array
181 Double_t MET = 0.0; // Missing transverse energy
182
183 /*
184 * Some variables used through the code
185 */
186 Int_t NumEvents1ISRJet = 0; // Number of events
    where the number of ISR jets is 1
187 Int_t NumMatches = 0; // Number of matches
188 Int_t NumJets = 0;
189 Int_t ISR_match_index = -1;
190 Double_t Cut_matching_DPT = 50.0;
191 Double_t Cut_matching_DEta = 0.4;
192 Double_t Cut_matching_DPhi = 0.4;

```

```

193 Double_t Cut_matching_Dy = 0.4;
194 Int_t ISR_jets[numberOfEntries];
195
196 /*
197  * Main cycle of the program
198  */
199 cout << "Running the matching algorithm" << endl;
200 numberOfEntries = 100000;
201 for (Int_t entry = 0; entry < numberOfEntries; ++
    entry){
202     // Progress
203     if(numberOfEntries>10 && (entry%((int)
        numberOfEntries/10))==0.0){
204         cout<<"\tprogress = "<<(entry*100/
            numberOfEntries)<<"%\t";
205         cout<< "Time : "<< (clock()-initialTime)/
            double_t(CLOCKS_PER_SEC)<<"s"<<endl;
206     }
207
208     // Load selected branches with data from specified
        event
209     treeReader_Pythia->ReadEntry(entry);
210     treeReader_Delphes->ReadEntry(entry);
211
212     // By default, the ISR jet was not matched
213     ISR_jets[entry] = -1;
214
215     // MET
216     METpointer = (MissingET*) branchMissingET->At(0);
217     MET = METpointer->MET;
218
219     // Finding the ISR parton
220     ISR_parton_found = false;
221     pos_ISR = -1;
222     for(Int_t iPart = 0; iPart < branchParticlePythia
        ->GetEntries(); iPart++){
223         particle_pythia = (TRootGenParticle*)
            branchParticlePythia->At(iPart);
224         if( abs(particle_pythia->Status) == 43){
225             pos_ISR = iPart;

```



```

226         ISR_particle = (TRootGenParticle*)
                branchParticlePythia->At(pos_ISR);
227         ISR_parton_found = true;
228 //         cout << pos_ISR << "\t\t" << ISR_particle->
Status << "\t\t" << ISR_particle->PID
229 //         << "\t\t" << ISR_particle->M1 << "\t\t"
<< ISR_particle->M2
230 //         << "\t\t" << ISR_particle->D1 << "\t\t"
<< ISR_particle->D2 << endl;
231     }
232 }
233
234 // If there is not ISR parton, pass to the next
event
235 if (ISR_parton_found == false){
236     continue;
237 }
238
239 // Finding the last copy of the ISR_parton
240 ISR_parton_found = false;
241 while (!ISR_parton_found){
242     if (ISR_particle->D1 != ISR_particle->D2)
243         ISR_parton_found = true;
244     else{
245         pos_ISR = ISR_particle->D1;
246         if(pos_ISR != -1) // To avoid an incoherent
event
247             ISR_particle = (TRootGenParticle*)
                branchParticlePythia->At(pos_ISR);
248         else
249             ISR_parton_found = true; // To end up the
while loop
250     }
251 }
252
253 if (pos_ISR == -1) // End the incoherent events
254     continue;
255
256 // Matching algorithm
257 // Matching between the ISR parton and a jet

```

```

258 // Auxiliary variables
259 Double_t R_min = 2.0;
260 Double_t r; // Current deltaR
261 ISR_match_index = -1;
262 Int_t mixJets = 0;
263 TLorentzVector *vect_Jet1 = new TLorentzVector();
    // Four-momentum of the jet of the 1st
    for
264 TLorentzVector *vect_Jetc = new TLorentzVector();
    // Four-momentum of the jet of the 2nd, 3
    rd ... for
265 TLorentzVector *vect_Jets = new TLorentzVector();
    // Four-momentum of the sum of jets
266 TLorentzVector *vect_Jeto = new TLorentzVector();
    // Four-momentum of the optimal
    combination
267 Jet *jet = new Jet();
268 Jet *jet2 = new Jet();
269
270 NumJets = branchJet->GetEntries();
271 vect_ISR_particle->SetPtEtaPhiE(ISR_particle->PT,
    ISR_particle->Eta, ISR_particle->Phi,
    ISR_particle->E);
272
273 if (NumJets < 3) // Minimum 3 jets per event
274     continue;
275
276 // Finding the jet with the minimum R to the ISR
    parton
277 for ( Int_t j = 0; j < NumJets; j++ ) { //
    Loop over jets finding the one with the minimum
    R
278     jet = (Jet*) branchJet->At(j);
279     vect_Jet1->SetPtEtaPhiM(jet->PT, jet->Eta,
        jet->Phi, jet->Mass);
280     r = vect_ISR_particle->DeltaR(*vect_Jet1);
281     if ( r < R_min ) {
282         R_min = r;
283         ISR_match_index = j;
284         mixJets = 1;

```

```

285         *vect_Jeto = *vect_Jet1;
286     }
287     // Checking if there are two jets mixed
288     for ( Int_t k = j+1; k<NumJets; k++){
289         jet2 = (Jet*) branchJet->At(k);
290         vect_Jetc->SetPtEtaPhiM(jet2->PT, jet2
            ->Eta, jet2->Phi, jet2->Mass);
291         *vect_Jets = *vect_Jet1 + *vect_Jetc;
292         r = vect_ISR_particle->DeltaR(*
            vect_Jets);
293         if ( r < R_min ) {
294             R_min = r;
295             ISR_match_index = j;
296             mixJets = 2;
297             *vect_Jeto = *vect_Jets;
298         }
299         // Checking if there are three
            jets mixed
300     for (Int_t m = k+1; m<NumJets; m++){
301         jet2 = (Jet*) branchJet->At(m);
302         vect_Jetc->SetPtEtaPhiM(
            jet2->PT, jet2->Eta,
            jet2->Phi, jet2->Mass
            );
303         *vect_Jets = *vect_Jets
            + *vect_Jetc;
304         r = vect_ISR_particle->
            DeltaR(*vect_Jets);
305         if ( r < R_min ) {
306             R_min = r;
307             ISR_match_index
                = j;
308             mixJets = 3;
309             *vect_Jeto = *
                vect_Jets;
310         }
311         // Checking if there are
            four jets mixed
312     for (Int_t n = m+1; n<
        NumJets; n++){

```

```

313         jet2 = (Jet*)
           branchJet->At
           (n);
314         vect_Jetc->
           SetPtEtaPhiM(
           jet2->PT,
           jet2->Eta,
           jet2->Phi,
           jet2->Mass);
315         *vect_Jets = *
           vect_Jets + *
           vect_Jetc;
316         r =
           vect_ISR_particle
           ->DeltaR(*
           vect_Jets);
317         if ( r < R_min )
           {
318             R_min =
                 r;
319             ISR_match_index
                 = j;
320             mixJets
                 = 4;
321             *
                 vect_Jeto
                 = *
                 vect_Jets
                 ;
322         }
323     }
324 }
325 }
326 } // Loop over jets finding the one with the
    minimum R
327
328 if( (mixJets == 1) && (ISR_match_index >= 0) &&
    (ISR_match_index < NumJets) ) {
329     NumEvents1ISRJet++;
330     Double_t Delta_PT = TMath::Abs(vect_Jeto

```

```

331         ->Pt() - vect_ISR_particle->Pt());
        Double_t Delta_Eta = TMath::Abs(
            vect_Jeto->Eta() - vect_ISR_particle
            ->Eta());
332         Double_t Delta_Phi = vect_Jeto->DeltaPhi
            (*vect_ISR_particle);
333         Double_t Delta_y = TMath::Abs(vect_Jeto
            ->Rapidity() - vect_ISR_particle->
            Rapidity());
334
335         if ( (Delta_PT > Cut_matching_DPT) || (
            Delta_Eta > Cut_matching_DEta) || (
            Delta_Phi > Cut_matching_DPhi ) || (
            Delta_y > Cut_matching_Dy) ) {
336             ISR_jets[entry] = -1;
337         }
338         else {
339             NumMatches++;
340             ISR_jets[entry] =
                ISR_match_index;
341         }
342     }
343
344     if (ISR_jets[entry] >= NumJets){
345         cout << "Error en el matching. Terminating
            program" << endl;
346         return 1;
347     }
348 }
349
350 cout<<"\tprogress = 100%\t";
351 cout<< "Time :"<< (clock()-initialTime)/double_t(
    CLOCKS_PER_SEC)<<"s"<<endl;
352
353 /*
354  * Writing results
355  */
356 cout << "\nWriting files" << endl;
357 string fileName_str = head_folder_results +
    matching_name;

```

```

358
359     Char_t * fileName = (Char_t *) fileName_str.c_str();
360
361     if (argc > 2)
362         cout << "\t Writing the binary file..." <<
            fileName << endl;
363     else
364         cout<<"\t Writing the default binary file..." <<
            fileName << endl;
365
366     ofstream ofs(fileName,ios::out|ios::binary);
367     if (!ofs){
368         cout << "Problemas al escribir el archivo" << endl
            ;
369     }
370     else{
371         for(Int_t j = 0; j<numberOfEntries; j++){
372             ofs.write((Char_t *) (ISR_jets+j),sizeof(Int_t)
                );
373         }
374     }
375     ofs.close();
376
377     cout << "\nSome overall results: " << endl;
378     cout << "\tNumber of events with a single ISR jet = "
        << NumEvents1ISRJet <<endl;
379     cout << "\tNumber of matches = " << NumMatches <<
        endl;
380     cout << endl;
381
382     return 0;
383 }

```

B.3 ISR jet analysis

```

1  /*
2  -----
3  -----      Universidad de los Andes      -----
4  -----      Departamento de Fisica      -----

```

```

5  -----      Joven Investigador      -----
6  -----      Andres Felipe Garcia Albarracin      -----
7  -----      Juan Carlos Sanabria Arenas      -----
8  -----
9
10 This algorithm studies the kinematic properties
11 of the ISR jets. It reads the results of the
12 matching algorithm
13
14 To execute, type:
15
16 ./ISR_jet_analysis config_file.txt
17
18 where config_file.txt is the mandatory configuration
19 file
20 */
21
22
23 #include "ROOTFunctions.h"
24 #include "graphs_Funcs.h"
25 #include "functions.h"
26 #include "Rtypes.h"
27 #include "DelphesFunctions.h"
28
29 // Global Variables
30 const Double_t PI = TMath::Pi();
31
32 int main(int argc, char **argv){
33     std::cout.precision(4);
34     // Counting time
35     Double_t initialTime = clock();
36
37     // Folder variables
38     string head_folder = "/home/af.garcia1214/
        PhenoMCsamples/Simulations/
        MG_pythia8_delphes_parallel/
        _Tops_Events_WI_Matching/";
39     string current_folder = "_Tops_MG_1K_AG_WI_003/";
40
41     string head_folder_binary = "/home/af.garcia1214/

```

```

        PhenoMCsamples/Results_Improved_Codes/
        matching_Results/_Tops_matches_WI_Matching/";
42   string matching_name = "ISR_jets_Tops_WI_003.bn";
43
44   string head_folder_results = "/home/af.garcia1214/
        PhenoMCsamples/Results_Improved_Codes/histo_folder
        /_Tops_histos_WI_Matching/";
45
46   // Checking input parameters
47   string config_file_name = "Debug/config_file.txt";
48   // Reading the file as first parameter
49   if (argc>1){
50       config_file_name = argv[1];
51   }
52   else{
53       cout << "It is necessary to type a configuration
           file as parameter. Execute as ./
           ISR_jet_analysis config_file.txt" << endl;
54       return 1;
55   }
56   cout << "Reading input parameters" << endl;
57   cout << "\tUsing as parameters' file: " <<
        config_file_name << endl;
58
59   ifstream config_file (config_file_name);
60   if (config_file.is_open()){
61       cout << "\tReading file" << endl;
62       string line;
63       int number_line = 1;
64       while (getline(config_file,line)){
65           // Skipping commented lines
66           if (line[0] == '!')
67               continue;
68
69           // Finding the position of the equal sign
70           int pos_equal = -1;
71           pos_equal = line.find('=');
72
73           if (pos_equal == -1){
74               cout << "\tLine " << number_line << " is

```



```

75         incorrect" << endl;
76         continue;
77     }
78     // Splitting the line according to the position
       of equal sign
79     string var_name = line.substr(0,pos_equal);
80     string var_value = line.substr(pos_equal+1);
81
82     // Reading head folder
83     if(var_name.compare("head_folder") == 0){
84         head_folder = var_value;
85         cout << "\tVariable head folder set as: " <<
           head_folder << endl;
86     }
87     // Reading current folder
88     else if (var_name.compare("current_folder") ==
           0){
89         current_folder = var_value;
90         cout << "\tVariable current folder set as: "
           << current_folder <<endl;
91     }
92     // Reading head folder binary
93     else if (var_name.compare("head_folder_binary")
           == 0){
94         head_folder_binary = var_value;
95         cout << "\tVariable head folder binary set
           as: " << head_folder_binary << endl;
96     }
97     // Reading matching name
98     else if (var_name.compare("matching_name") ==
           0){
99         matching_name = var_value;
100        cout << "\tVariable matching_name set as: "
           << matching_name << endl;
101    }
102    // Reading head folder results
103    else if (var_name.compare("head_folder_results"
           ) == 0){
104        head_folder_results = var_value;

```

```

105         cout << "\tVariable head folder results set
           as: " << head_folder_results << endl;
106     }
107
108     number_line ++;
109 }
110 }
111 else
112 {
113     cout << "ERROR: File " << config_file_name << "
           does not exist. Terminating program" << endl;
114     return 0;
115 }
116
117
118 cout << "\n *** ISR jet analysis *** \n" << endl;
119
120 /*
121  * Histograms
122  */
123 // All jets
124 TH1 *h_numberJet = new TH1F("Number Jets","Number
           Jets",11,-0.5,10.5);
125
126 // Non Isr jets
127 TH1 *h_jet_PT = new TH1F("Jet PT","Jet PT",
           201,0.0,600.0);
128 TH1 *h_jet_Eta = new TH1F("Jet Eta","Jet Eta",
           171,-5.0,5.0);
129 TH1 *h_jet_Phi = new TH1F("Jet Phi","Jet Phi",
           375,-3.5,3.5);
130 TH1 *h_jet_DPhi_MET = new TH1F("Jet - MET Delta_Phi",
           "Jet - MET Delta_Phi",300,0.0,4.0);
131 TH1 *h_jet_DPhi_MET_hpt = new TH1F("Jet - MET
           Delta_Phi_hpt","Jet - MET Delta_Phi_hpt"
           ,300,0.0,4.0);
132 TH1 *h_jet_MT = new TH1F("Jet Transverse mass","Jet
           Transverse Mass",201,0.0,600.0);
133 TH1 *h_jet_Delta_PT = new TH1F("Jet Delta-PT","Non
           ISR Delta-PT", 201,0.0,300.0);

```

```

134 TH1 *h_jet_PT_HT = new TH1F("Jet PT-HT ratio","Jet PT
    -HT ratio",201,-0.0025,1.0025);
135 TH1 *h_jet_PT_over_PT_others = new TH1F("Jet PT/
    PT_others","Jet PT/PT_others",401,-0.0025,2.0025);
136 TH1 *h_jet_Eta_over_Eta_others = new TH1F("Jet Eta/
    Eta_others","Jet Eta/Eta_others"
    ,401,-0.0025,2.0025);
137 TH1 *h_jet_DPhi_over_Phi_others = new TH1F("Jet Phi/
    Phi_others","Jet Phi/Phi_others"
    ,401,-0.0025,2.0025);
138 TH1 *h_jet_Delta_Eta = new TH1F("Jet Delta-Eta","Jet
    Delta-Eta", 171,0.0,5.0);
139 TH1 *h_jet_DPhi_MET_other = new TH1F("Jet - MET
    Delta_Phi other","Jet - MET Delta_Phi other"
    ,300,0.0,4.0);
140 TH1 *h_jet_multiplicity = new TH1F("Jet -
    Multiplicity","Jet - Multiplicity",101,-0.5,100.5)
    ;
141 TH1 *h_jet_DeltaR = new TH1F ("Jet - Delta_R","Jet -
    Delta_R",201,-0.0025,0.8025);
142 TH1 *h_jet_Delta_PT_leading = new TH1F("Delta PT:
    leading - Jet","Delta PT: leading - Jet",
    201,0.0,600.0);
143 TH1 *h_jet_Delta_Eta_leading = new TH1F("Delta Eta:
    Jet - leading","Delta Eta: Jet - leading",
    171,0.0,8.0);
144
145 TH2 *h2_jet_PTEta=new TH2F("Non_ISR_Jet_PT_Eta","Non
    ISR Jet PT Vs. Eta"
    ,201,-1.25,501.25,201,-4.02,4.02);
146
147 // ISR jets
148 TH1 *h_ISR_PT = new TH1F("ISR PT","ISR PT",
    201,0.0,600.0);
149 TH1 *h_ISR_Eta = new TH1F("ISR Eta","ISR Eta",
    171,-5.0,5.0);
150 TH1 *h_ISR_Phi = new TH1F("ISR Phi","ISR Phi",
    375,-3.5,3.5);
151 TH1 *h_ISR_DPhi_MET = new TH1F("ISR - MET Delta_Phi",
    "ISR - MET Delta_Phi",300,0.0,4.0);

```

```

152 TH1 *h_ISR_DPhi_MET_hpt = new TH1F("ISR - MET
    Delta_Phi_hpt","ISR - MET Delta_Phi_hpt"
    ,300,0.0,4.0);
153 TH1 *h_ISR_MT = new TH1F("ISR Transverse mass","ISR
    Transverse Mass",201,0.0,600.0);
154 TH1 *h_ISR_Delta_PT = new TH1F("ISR Delta-PT","ISR
    Delta-PT", 201,0.0,300.0);
155 TH1 *h_ISR_PT_HT = new TH1F("ISR PT-HT ratio","ISR PT
    -HT ratio",201,-0.0025,1.0025);
156 TH1 *h_ISR_PT_over_PT_others = new TH1F("ISR PT/
    PT_others","ISR PT/PT_others",401,-0.0025,2.0025);
157 TH1 *h_ISR_Eta_over_Eta_others = new TH1F("ISR Eta/
    Eta_others","ISR Eta/Eta_others"
    ,401,-0.0025,2.0025);
158 TH1 *h_ISR_DPhi_over_Phi_others = new TH1F("ISR Phi/
    Phi_others","ISR Phi/Phi_others"
    ,401,-0.0025,2.0025);
159 TH1 *h_ISR_Delta_Eta = new TH1F("ISR Delta-Eta","ISR
    Delta-Eta", 171,0.0,5.0);
160 TH1 *h_ISR_DPhi_MET_other = new TH1F("ISR - MET
    Delta_Phi other","ISR - MET Delta_Phi other"
    ,300,0.0,4.0);
161 TH1 *h_ISR_multiplicity = new TH1F("ISR -
    Multiplicity","ISR - Multiplicity",101,-0.5,100.5)
    ;
162 TH1 *h_ISR_DeltaR = new TH1F ("ISR - Delta_R","ISR -
    Delta_R",201,-0.0025,0.8025);
163 TH1 *h_ISR_Delta_PT_leading = new TH1F("Delta PT:
    leading - ISR","Delta PT: leading - ISR",
    201,0.0,600.0);
164 TH1 *h_ISR_Delta_Eta_leading = new TH1F("Delta Eta:
    ISR - leading","Delta Eta: ISR - leading",
    171,0.0,8.0);
165
166 TH2 *h2_ISR_PTEta=new TH2F("ISR_Jet_PT_Eta","ISR Jet
    PT Vs. Eta",201,-1.25,501.25,201,-4.02,4.02);
167
168 // MET
169 TH1 *h_MET = new TH1F("Missing ET","Missing ET"
    ,200,0,600);

```

```

170 TH1 *h_MET_hpt1 = new TH1F("Missing ET high_ISR_pt-1"
    , "Missing ET high_ISR_pt-1", 200, 0.0, 600.0);
171 TH1 *h_MET_hpt2 = new TH1F("Missing ET high_ISR_pt-2"
    , "Missing ET high_ISR_pt-2", 200, 0.0, 600.0);
172 TH1 *h_MET_hpt3 = new TH1F("Missing ET high_ISR_pt-3"
    , "Missing ET high_ISR_pt-3", 200, 0.0, 600.0);
173 TH1 *h_MET_hpt4 = new TH1F("Missing ET high_ISR_pt-4"
    , "Missing ET high_ISR_pt-4", 200, 0.0, 600.0);
174
175 TH2 *h2_dif_PTEta=new TH2F("
    FSR_ISR_Jet_PT_Eta_Difference", "Difference between
    FSR and ISR Jet PT Vs. Eta distributions"
    , 201, -1.25, 501.25, 201, -4.02, 4.02);
176 TH2 *h2_dif_lead_PTEta=new TH2F("
    Lead_ISR_Jet_PT_Eta_Difference", "Difference
    between Lead and ISR Jet PT Vs. Eta distributions"
    , 201, -1.25, 501.25, 201, -4.02, 4.02);
177
178 // Leading PT
179 TH1 *h_leading_PT = new TH1F("Leading PT", "Leading PT
    ", 201, 0.0, 600.0);
180 TH1 *h_leading_MT = new TH1F("Leading Transverse mass
    ", "Leading Transverse Mass", 201, 0.0, 600.0);
181 TH1 *h_leading_Eta = new TH1F("Leading Eta", "Leading
    Eta", 171, -5.0, 5.0);
182 TH1 *h_leading_DPhi_MET = new TH1F("Leading - MET
    Delta_Phi", "Leading - MET Delta_Phi", 300, 0.0, 4.0);
183
184 TH2 *h2_leading_PTEta=new TH2F("Leading_Jet_PT_Eta", "
    Leading Jet PT Vs. Eta"
    , 201, -1.25, 501.25, 201, -4.02, 4.02);
185
186 // Other variables
187 TH1 *h_HT = new TH1F("HT", "HT", 201, 0.0, 600.0);
188 TH1 *h_HT_R1 = new TH1F("HT_R1", "HT_R1"
    , 51, -0.01, 1.01);
189 TH1 *h_HT_R2 = new TH1F("HT_R2", "HT_R2"
    , 51, -0.01, 1.01);
190
191 // B tagging

```

```

192 TH1 *h_BTag = new TH1F("BTag","BTag",5,-0.5,4.5);
193 TH1 *h_BTag_PT = new TH1F("BTag PT","BTag PT",
    201,0.0,600.0);
194 TH1 *h_BTag_Eta = new TH1F("BTag Eta","BTag Eta",
    171,-5.0,5.0);
195 TH1 *h_BTag_DPhi_MET = new TH1F("BTag - MET Delta_Phi
    ", "BTag - MET Delta_Phi", 300,0.0,4.0);
196 TH1 *h_BTags_per_Event = new TH1F("BTags per event","
    BTags per event", 5, -0.5, 4.5);
197
198 // Further analysis
199 TH1 *h_ISR_PT_comp = new TH1F("ISR PT for comparison"
    , "ISR PT for comparison with histo", 20, 0.0, 800.0)
    ;
200 TH1 *h_ISR_Eta_comp = new TH1F("ISR Eta for
    comparison", "ISR Eta for comparison with histo",
    20, -4.2, 4.2);
201 TH1 *h_ISR_DPhi_MET_comp = new TH1F("ISR Phi for
    comparison", "ISR Phi for comparison with histo",
    20, 0, PI);
202
203 // To check the histograms' creation
204 TH1 *hist_ISR_PT = new TH1F("ISR PT comp", "ISR PT
    comp", 20, 0.0, 800.0);
205 TH1 *hist_ISR_Abs_Eta = new TH1F("ISR Abs Eta comp", "
    ISR Abs Eta comp", 20, 0.0, 5.2);
206 TH1 *hist_ISR_DPhi_MET = new TH1F("ISR Delta Phi comp
    ", "ISR Delta Phi comp", 20, 0.0, PI);
207 TH1 *hist_ISR_PT_ratio = new TH1F("ISR PT/PT_others
    comp", "ISR PT/PT_others comp", 20, 0.0, 8.0);
208 TH1 *hist_ISR_Delta_Eta = new TH1F("ISR Delta-Eta
    comp", "ISR Delta-Eta comp", 20, 0.0, 7.0);
209 TH1 *hist_ISR_DPhi_MET_other = new TH1F("ISR - MET
    Delta_Phi other comp", "ISR - MET Delta_Phi other
    comp", 20, 0.0, PI);
210 TH1 *hist_ISR_Delta_PT_leading = new TH1F("Delta PT:
    leading - ISR comp", "Delta PT: leading - ISR comp"
    , 20, 0.0, 500.0);
211 TH1 *hist_ISR_Delta_Eta_leading = new TH1F("Delta Eta
    : ISR - leading comp", "Delta Eta: ISR - leading

```

```

    comp", 20,0.0,6.5);
212 TH1 *hist_jet_PT = new TH1F("Jet PT comp","Jet PT
    comp", 20,0.0,800.0);
213 TH1 *hist_jet_Abs_Eta = new TH1F("Jet Abs Eta comp","
    Jet Abs Eta comp", 20,0.0,5.2);
214 TH1 *hist_jet_DPhi_MET = new TH1F("Jet Delta Phi comp
    ","Jet Delta Phi comp", 20,0.0,PI);
215 TH1 *hist_jet_PT_ratio = new TH1F("Jet PT/PT_others
    comp","Jet PT/PT_others comp",20,0.0,7.0);
216 TH1 *hist_jet_Delta_Eta = new TH1F("Jet Delta-Eta
    comp","Jet Delta-Eta comp", 20,0.0,8.0);
217 TH1 *hist_jet_DPhi_MET_other = new TH1F("Jet - MET
    Delta_Phi other comp","Jet - MET Delta_Phi other
    comp",20,0.0,PI);
218 TH1 *hist_jet_Delta_PT_leading = new TH1F("Delta PT:
    leading - Jet comp","Delta PT: leading - Jet comp"
    , 20,0.0,500.0);
219 TH1 *hist_jet_Delta_Eta_leading = new TH1F("Delta Eta
    : Jet - leading comp","Delta Eta: Jet - leading
    comp", 20,0.0,6.5);

220
221 // Cycle over several runs . iRun corresponds to the
    seed of the current run
222 for(int iRun = 1; iRun < 11; iRun ++){
223     // Create chains of root trees
224     TChain chain_Delphes("Delphes");
225
226     // Loading simulations from Delphes
227     Char_t unidad = 0x30 + iRun%10;
228     Char_t decena = 0x30 + int(iRun/10)%10;
229     Char_t centena = 0x30 + int(iRun/100)%10;
230
231     current_folder[current_folder.size()-4] = centena;
232     current_folder[current_folder.size()-3] = decena;
233     current_folder[current_folder.size()-2] = unidad;
234     matching_name[matching_name.size()-6] = centena;
235     matching_name[matching_name.size()-5] = decena;
236     matching_name[matching_name.size()-4] = unidad;
237
238     string file_pythia_str = head_folder +

```

```

    current_folder + "Events/run_01/output_pythia8.
    root";
239
240    string file_delphes_str = head_folder +
        current_folder + "Events/run_01/output_delphes.
        root";
241    Char_t *file_delphes = (Char_t *) file_delphes_str
        .c_str();
242
243    cout << "\nReading the file: \nDelphes: " <<
        file_delphes << endl;
244
245    chain_Delphes.Add(file_delphes);
246    // Objects of class ExRootTreeReader for reading
        the information
247    ExRootTreeReader *treeReader_Delphes = new
        ExRootTreeReader(&chain_Delphes);
248
249    Long64_t numberOfEntries = treeReader_Delphes->
        GetEntries();
250
251    // Get pointers to branches used in this analysis
252    TClonesArray *branchJet = treeReader_Delphes->
        UseBranch("Jet");
253    TClonesArray *branchMissingET = treeReader_Delphes
        ->UseBranch("MissingET");
254
255    cout << endl;
256    cout << " Number of Entries Delphes = " <<
        numberOfEntries << endl;
257    cout << endl;
258
259    // particles, jets and vectors
260    MissingET *METpointer;
261    TLorentzVector *vect_currentJet = new
        TLorentzVector;
262    TLorentzVector *vect_auxJet = new TLorentzVector;
263    TLorentzVector *vect_leading = new TLorentzVector;
264    Jet *currentJet = new Jet;
265    Jet *auxJet = new Jet;

```



```

266     TRefArray array_temp;
267
268     // Temporary variables
269     Double_t MET = 0.0; // Missing transverse energy
270     Double_t delta_phi = 0.0; // difference between
        the phi angle of MET and the jet
271     Double_t transverse_mass = 0.0; // Transverse mass
272     Double_t HT = 0.0; // Sum of jets' PT
273     Double_t HT_R1 = 0.0; // Sum of jets' PT which are
        in the same hemisphere of the ISR jet
        hemisphere
274     Double_t HT_R2 = 0.0; // Sum of jets' PT which are
        in the opposite hemisphere of the ISR jet
        hemisphere
275     Double_t ISR_Eta = 0.0; // Pseudorapidity of the
        ISR jet
276     Int_t number_Btags = 0; // Number of B jets per
        event
277     Int_t ISR_Btags = 0; // Number of BTags which are
        also ISR jets
278     Double_t delta_PT_jet = 0.0; // |PT-<PT>|
279     Double_t PT_sum = 0.0; // sum(PT)
280     Double_t PT_aver = 0.0; // <PT>
281     Double_t Delta_eta_aver = 0.0; // sum_i|eta-eta_i
        |/(Nj-1)
282     Double_t Delta_phi_sum = 0.0; // sum delta_phi
283     Double_t Delta_phi_other_jets = 0.0; // Average of
        delta phi of other jets
284     Double_t PT_ratio = 0.0; // PT/PT_others
285     Double_t Eta_ratio = 0.0; // Eta/Eta_others
286     Double_t Eta_sum = 0.0; // sum(Eta)
287     Double_t Delta_R = 0.0; // Size of the jet
288     Double_t Delta_phi_ratio = 0.0; // Delta_phi/
        Delta_phi_others
289     Double_t Delta_PT_leading = 0.0; // PT -
        PT_leading
290     Double_t Delta_Eta_leading = 0.0; // |Eta -
        Eta_leading|
291
292     /*

```

```

293     * Some variables used through the code
294     */
295     Int_t ISR_jets[numberOfEntries];
296     Int_t NumJets = 0;
297
298     string fileName_str = head_folder_binary +
299         matching_name;
300
301     Char_t * fileName = (Char_t *) fileName_str.c_str
302         ();
303
304     ifstream ifs(fileName, ios::in | ios::binary);
305
306     for (Int_t j = 0; j<numberOfEntries; j++){
307         ifs.read((Char_t *) (ISR_jets+j), sizeof(
308             Int_t));
309     }
310     ifs.close();
311
312     // Jet with greatest PT
313     Double_t PT_max = 0;
314     Int_t posLeadingPT = -1;
315     Int_t ISR_greatest_PT = 0;
316     Double_t MT_leading_jet = 0.0; // Transverse mass
317
318     /*
319     * Main cycle of the program
320     */
321     numberOfEntries = 100000;
322     for (Int_t entry = 0; entry < numberOfEntries; ++
323         entry){
324         // Progress
325         if(numberOfEntries>10 && (entry%((int)
326             numberOfEntries/10))==0.0){
327             cout<<"progress = "<<(entry*100/
328                 numberOfEntries)<<"%\t";
329             cout<< "Time : "<< (clock()-initialTime)/
330                 double_t(CLOCKS_PER_SEC)<<"s"<<endl;
331         }
332     }

```

```
326         // Load selected branches with data from
327         // specified event
328         treeReader_Delphes->ReadEntry(entry);
329
330         // MET
331         METpointer = (MissingET*) branchMissingET->At
332         (0);
333         MET = METpointer->MET;
334         h_MET->Fill(MET);
335
336         NumJets=branchJet->GetEntries();
337         h_numberJet->Fill(NumJets);
338
339         // checking the ISR
340         if (ISR_jets[entry] == -1 || NumJets < 3)
341             continue;
342
343         PT_max = 0;
344         posLeadingPT = -1;
345         HT = 0;
346         HT_R1 = 0;
347         HT_R2 = 0;
348         number_Btags = 0;
349
350         delta_PT_jet = 0.0;
351         PT_aver = 0.0;
352         PT_sum = 0.0;
353         Delta_eta_aver = 0.0;
354         Delta_phi_sum = 0.0;
355         Delta_phi_other_jets = 0.0;
356         Delta_phi_ratio = 0.0;
357         Delta_PT_leading = 0.0;
358         Delta_Eta_leading = 0.0;
359
360         PT_ratio = 0.0;
361         Eta_ratio = 0.0;
362         Eta_sum = 0.0;
363
364         Delta_R = 0.0;
```

```

364         if (ISR_jets[entry] >= NumJets){
365             cout << "Error en el matching" << endl;
366             return 1;
367         }
368
369         // Preliminary for. It is used to calculate
370         // PT_aver and Delta_phi_sum
371         for (Int_t iJet = 0; iJet<NumJets; iJet++){
372             currentJet = (Jet*) branchJet->At(iJet);
373             vect_currentJet->SetPtEtaPhiM(currentJet->PT
374                 ,currentJet->Eta,currentJet->Phi,
375                 currentJet->Mass);
376             delta_phi = deltaAng(vect_currentJet->Phi(),
377                 METpointer->Phi);
378             PT_sum += vect_currentJet->Pt();
379             Eta_sum += vect_currentJet->Eta();
380             Delta_phi_sum += delta_phi;
381             // HT
382             HT += vect_currentJet->Pt();
383             // HT ratios
384             if((vect_currentJet->Eta()*ISR_Eta) > 0)
385                 HT_R1 += vect_currentJet->Pt();
386             else
387                 HT_R2 += vect_currentJet->Pt();
388             // PT Leading jet
389             if(PT_max < vect_currentJet->Pt()){
390                 PT_max = vect_currentJet->Pt();
391                 posLeadingPT = iJet;
392             }
393         }
394
395         //PT_aver
396         PT_aver = PT_sum/NumJets;
397
398         // Leading PT
399         currentJet = (Jet*) branchJet->At(posLeadingPT)
400             ;
401         vect_leading->SetPtEtaPhiM(currentJet->PT,
402             currentJet->Eta,currentJet->Phi,currentJet->
403             Mass);

```

```

397
398 // ISR jet
399 currentJet = (Jet*) branchJet->At(ISR_jets[
    entry]);
400 vect_currentJet->SetPtEtaPhiM(currentJet->PT,
    currentJet->Eta, currentJet->Phi, currentJet->
    Mass);
401 ISR_Eta = vect_currentJet->Eta();
402
403 for (Int_t iJet = 0; iJet<NumJets; iJet++){
404     currentJet = (Jet*) branchJet->At(iJet);
405     vect_currentJet->SetPtEtaPhiM(currentJet->PT
        ,currentJet->Eta, currentJet->Phi,
        currentJet->Mass);
406     delta_phi = deltaAng(vect_currentJet->Phi(),
        METpointer->Phi());
407     transverse_mass = sqrt(2*vect_currentJet->Pt
        ()*MET*(1-cos(delta_phi)));
408
409 // Correlated variables
410 delta_PT_jet = TMath::Abs(vect_currentJet->
    Pt()-PT_aver);
411 Delta_phi_other_jets = (Delta_phi_sum-
    delta_phi)/(NumJets-1);
412 PT_ratio = vect_currentJet->Pt()*(NumJets-1)
    /(PT_sum-vect_currentJet->Pt());
413 Eta_ratio = vect_currentJet->Eta()*(NumJets
    -1)/(Eta_sum-vect_currentJet->Eta());
414 Delta_phi_ratio = delta_phi*(NumJets-1)/(
    Delta_phi_sum-delta_phi);
415
416 Delta_Eta_leading = TMath::Abs(
    vect_currentJet->Eta()-vect_leading->Eta
    ());
417 Delta_PT_leading = vect_leading->Pt()-
    vect_currentJet->Pt();
418
419 Delta_eta_aver = 0.0;
420 // For cycle used to calculate
    Delta_eta_aver

```

```

421     for(Int_t iJet2 = 0; iJet2<NumJets; iJet2++)
422     {
423         auxJet = (Jet*) branchJet->At(iJet2);
424         vect_auxJet->SetPtEtaPhiM(auxJet->PT,
425                                   auxJet->Eta,auxJet->Phi,auxJet->Mass);
426         if (iJet2 != iJet) Delta_eta_aver +=
427             TMath::Abs(vect_auxJet->Eta()-
428                       vect_currentJet->Eta());
429     }
430     Delta_eta_aver = Delta_eta_aver/(NumJets-1);
431     Delta_R = sqrt(pow(currentJet->DeltaEta,2)+
432                   pow(currentJet->DeltaPhi,2));
433
434     // Multiplicity
435     array_temp = (TRefArray) currentJet->
436         Constituents;
437
438     if (iJet != ISR_jets[entry]){ // Non ISR
439         h_jet_PT->Fill(vect_currentJet->Pt());
440         h_jet_Eta->Fill(vect_currentJet->Eta());
441         h_jet_Phi->Fill(vect_currentJet->Phi());
442         h_jet_DPhi_MET->Fill(delta_phi);
443         h_jet_MT->Fill(transverse_mass);
444         h_jet_Delta_PT->Fill(delta_PT_jet);
445         h_jet_Delta_Eta->Fill(Delta_eta_aver);
446         h_jet_DPhi_MET_other->Fill(
447             Delta_phi_other_jets);
448         h_jet_PT_HT->Fill(vect_currentJet->Pt()/
449                           HT);
450         h_jet_multiplicity->Fill(array_temp.
451                                   GetEntries());
452         h_jet_PT_over_PT_others->Fill(PT_ratio);
453         h_jet_Eta_over_Eta_others->Fill(Eta_ratio
454                                         );
455         h_jet_DeltaR->Fill(Delta_R);
456         h_jet_DPhi_over_Phi_others->Fill(
457             Delta_phi_ratio);
458         h_jet_Delta_PT_leading->Fill(
459             Delta_PT_leading);
460         h_jet_Delta_Eta_leading->Fill(

```

```

Delta_Eta_leading);
449   if (vect_currentJet->Pt()>240)
450       h_jet_DPhi_MET_hpt->Fill(delta_phi);
451   h2_jet_PTEta->Fill(vect_currentJet->Pt(),
                      vect_currentJet->Eta());
452
453   // For testing creating histo
454   hist_jet_PT->Fill(vect_currentJet->Pt());
455   hist_jet_Abs_Eta->Fill(TMath::Abs(
                      vect_currentJet->Eta()));
456   hist_jet_DPhi_MET->Fill(delta_phi);
457   hist_jet_PT_ratio->Fill(PT_ratio);
458   hist_jet_Delta_Eta->Fill(Delta_eta_aver);
459   hist_jet_DPhi_MET_other->Fill(
                      Delta_phi_other_jets);
460   hist_jet_Delta_PT_leading->Fill(
                      Delta_PT_leading);
461   hist_jet_Delta_Eta_leading->Fill(
                      Delta_Eta_leading);
462   }
463   else{ //ISR
464       h_ISR_PT->Fill(vect_currentJet->Pt());
465       h_ISR_Eta->Fill(vect_currentJet->Eta());
466       h_ISR_Phi->Fill(vect_currentJet->Phi());
467       h_ISR_DPhi_MET->Fill(delta_phi);
468       h_ISR_Eta_comp->Fill(vect_currentJet->Eta
                          ());
469       h_ISR_PT_comp->Fill(vect_currentJet->Pt()
                          );
470       h_ISR_DPhi_MET_comp->Fill(delta_phi);
471       h_ISR_Delta_PT->Fill(delta_PT_jet);
472       h_ISR_Delta_Eta->Fill(Delta_eta_aver);
473       h_ISR_DPhi_MET_other->Fill(
                      Delta_phi_other_jets);
474       h_ISR_PT_HT->Fill(vect_currentJet->Pt()/
                          HT);
475       h_ISR_multiplicity->Fill(array_temp.
                          GetEntries());
476       h_ISR_PT_over_PT_others->Fill(PT_ratio);
477       h_ISR_Eta_over_Eta_others->Fill(Eta_ratio

```

```

    );
478     h_ISR_DeltaR->Fill(Delta_R);
479     h_ISR_DPhi_over_Phi_others->Fill(
        Delta_phi_ratio);
480     h_ISR_Delta_PT_leading->Fill(
        Delta_PT_leading);
481     h_ISR_Delta_Eta_leading->Fill(
        Delta_Eta_leading);
482     if (vect_currentJet->Pt()>120)
483         h_MET_hpt1->Fill(MET);
484     if (vect_currentJet->Pt()>200)
485         h_MET_hpt2->Fill(MET);
486     if (vect_currentJet->Pt()>240){
487         h_MET_hpt3->Fill(MET);
488         h_ISR_DPhi_MET_hpt->Fill(delta_phi);
489     }
490     if (vect_currentJet->Pt()>300)
491         h_MET_hpt4->Fill(MET);
492     h2_ISR_PTEta->Fill(vect_currentJet->Pt(),
        vect_currentJet->Eta());
493     // Transverse mass
494     h_ISR_MT->Fill(transverse_mass);
495
496     // For testing creating histo
497     hist_ISR_PT->Fill(vect_currentJet->Pt());
498     hist_ISR_Abs_Eta->Fill(TMath::Abs(
        vect_currentJet->Eta()));
499     hist_ISR_DPhi_MET->Fill(delta_phi);
500     hist_ISR_PT_ratio->Fill(PT_ratio);
501     hist_ISR_Delta_Eta->Fill(Delta_eta_aver);
502     hist_ISR_DPhi_MET_other->Fill(
        Delta_phi_other_jets);
503     hist_ISR_Delta_PT_leading->Fill(
        Delta_PT_leading);
504     hist_ISR_Delta_Eta_leading->Fill(
        Delta_Eta_leading);
505 }
506
507 // BTag
508 h_BTag->Fill(currentJet->BTag);

```



```

509         if (currentJet->BTag == 1){ // The current
510             jet is B Tagged
511             h_BTag_PT->Fill(vect_currentJet->Pt());
512             h_BTag_Eta->Fill(vect_currentJet->Eta());
513             h_BTag_DPhi_MET->Fill(delta_phi);
514             number_Btags++;
515
516             if (iJet == ISR_jets[entry]){ // If the
517                 ISR jet is also a B jet
518                 ISR_Btags++;
519             }
520         }
521
522         // Jet with greatest PT
523         if (posLeadingPT != -1){
524             h_leading_PT->Fill(PT_max);
525             if(posLeadingPT == ISR_jets[entry])
526                 ISR_greatest_PT++;
527
528             currentJet = (Jet*) branchJet->At(
529                 posLeadingPT);
530             vect_currentJet->SetPtEtaPhiM(currentJet->PT
531                 ,currentJet->Eta,currentJet->Phi,
532                 currentJet->Mass);
533             delta_phi = deltaAng(vect_currentJet->Phi(),
534                 METpointer->Phi);
535             MT_leading_jet = sqrt(2*vect_currentJet->Pt
536                 ()*MET*(1-cos(delta_phi)));
537             h_leading_MT->Fill(MT_leading_jet);
538
539             h_leading_Eta->Fill(vect_currentJet->Eta());
540             h_leading_DPhi_MET->Fill(delta_phi);
541
542             h2_leading_PTEta->Fill(vect_currentJet->Pt()
543                 ,vect_currentJet->Eta());
544         }
545
546         // HT
547         if (1 < HT_R1/HT || 1 < HT_R2/HT){

```

```

540         cout << "Error en el evento: " << entry <<
           endl;
541         cout << "HT: " << HT << "\tHT_R1: " << HT_R1
           << "\tHT_R2: " << HT_R2 << endl;
542         return 1;
543     }
544
545     h_HT->Fill(HT);
546     h_HT_R1->Fill(HT_R1/HT);
547     h_HT_R2->Fill(HT_R2/HT);
548     h_BTags_per_Event->Fill(number_Btags);
549
550 }
551
552 cout<<"progress = 100%\t";
553 cout<< "Time : "<< (clock()-initialTime)/double_t(
    CLOCKS_PER_SEC)<<"s"<<endl;
554 cout<< "Percentage of events where the ISR jet is
    the jet with greatest PT: " << (Double_t) (
    ISR_greatest_PT*100)/numberOfEntries << "%\n";
555 cout<< "Percentage of events where the ISR jet is
    tagged as Bjet: " << (Double_t) (ISR_Btags*100)
    /numberOfEntries << "%\n";
556
557 } // End run's for cicle
558
559 string hfile_name_str = head_folder_results + "histos
    .root";
560
561 Char_t *hfile_name = (Char_t *) hfile_name_str.c_str
    ();
562
563 TFile* hfile = new TFile(hfile_name, "RECREATE");
564 h_jet_DPhi_MET->Write();
565 h_jet_Eta->Write();
566 h_jet_PT->Write();
567 h_jet_Phi->Write();
568 h_jet_MT->Write();
569 h_jet_Delta_PT->Write();
570 h_jet_Delta_Eta->Write();

```

```
571     h_jet_DPhi_MET_other->Write();
572     h_jet_PT_HT->Write();
573     h_jet_multiplicity->Write();
574     h_jet_PT_over_PT_others->Write();
575     h_jet_Eta_over_Eta_others->Write();
576     h_jet_DeltaR->Write();
577     h_jet_DPhi_over_Phi_others->Write();
578     h_jet_Delta_Eta_leading->Write();
579     h_jet_Delta_PT_leading->Write();
580
581     h_ISR_DPhi_MET->Write();
582     h_ISR_Eta->Write();
583     h_ISR_PT->Write();
584     h_ISR_Phi->Write();
585     h_ISR_MT->Write();
586     h_ISR_Delta_PT->Write();
587     h_ISR_Delta_Eta->Write();
588     h_ISR_DPhi_MET_other->Write();
589     h_ISR_PT_HT->Write();
590     h_ISR_multiplicity->Write();
591     h_ISR_PT_over_PT_others->Write();
592     h_ISR_Eta_over_Eta_others->Write();
593     h_ISR_DeltaR->Write();
594     h_ISR_DPhi_over_Phi_others->Write();
595     h_ISR_Delta_Eta_leading->Write();
596     h_ISR_Delta_PT_leading->Write();
597
598     h_MET->Write();
599     h_MET_hpt1->Write();
600     h_MET_hpt2->Write();
601     h_MET_hpt3->Write();
602
603     h_leading_MT->Write();
604     h_leading_PT->Write();
605     h_leading_Eta->Write();
606     h_leading_DPhi_MET->Write();
607
608     h_HT->Write();
609     h_HT_R1->Write();
610     h_HT_R2->Write();
```

```

611
612     h_numberJet->Write();
613
614     h_BTag->Write();
615     h_BTag_PT->Write();
616     h_BTag_Eta->Write();
617     h_BTag_DPhi_MET->Write();
618     h_BTags_per_Event->Write();
619
620     h2_ISR_PTEta->Write();
621     h2_jet_PTEta->Write();
622     h2_dif_PTEta->Add(h2_ISR_PTEta,h2_jet_PTEta,1,-1);
623     h2_dif_PTEta->Write();
624
625     h2_dif_lead_PTEta->Add(h2_ISR_PTEta,h2_leading_PTEta
        ,1,-1);
626     h2_dif_lead_PTEta->Write();
627
628     {
629         string salida_str = head_folder_results + "Eta";
630         Char_t *salida = (Char_t *) salida_str.c_str();
631         TCanvas *C = new TCanvas(salida,"Pseudorapidity"
            ,1280,720);
632         Present(h_ISR_Eta,h_jet_Eta,C,1,"h","Num. Jets /
            Total",122);
633         C->Write();
634         C->Close();
635
636         salida_str = head_folder_results + "Eta ISR vs
            BTag";
637         salida = (Char_t *) salida_str.c_str();
638         C = new TCanvas(salida,"Pseudorapidity ISR vs BTag
            ",1280,720);
639         Present(h_ISR_Eta,h_BTag_Eta,C,1,"h","Num. Jets /
            Total",122);
640         C->Write();
641         C->Close();
642
643         salida_str = head_folder_results + "Eta ISR vs
            Leading";

```

```

644     salida = (Char_t *) salida_str.c_str();
645     C = new TCanvas(salida, "Pseudorapidity ISR vs
        Leading", 1280, 720);
646     Present(h_ISR_Eta, h_leading_Eta, C, 1, "h", "Num. Jets
        / Total", 122);
647     C->Write();
648     C->Close();
649
650     salida_str = head_folder_results + "Transverse
        momentum";
651     salida = (Char_t *) salida_str.c_str();
652     C = new TCanvas(salida, "Transverse momentum"
        , 1280, 720);
653     Present(h_ISR_PT, h_jet_PT, C, 2, "PT [GeV]", "Num.
        Jets / Total");
654     C->Write();
655     C->Close();
656
657     salida_str = head_folder_results + "Transverse
        momentum ISR vs Leading";
658     salida = (Char_t *) salida_str.c_str();
659     C = new TCanvas(salida, "Transverse momentum ISR vs
        Leading", 1280, 720);
660     Present(h_ISR_PT, h_leading_PT, C, 2, "PT [GeV]", "Num.
        Jets / Total");
661     C->Write();
662     C->Close();
663
664     salida_str = head_folder_results + "Transverse
        momentum ISR vs B_Tag";
665     salida = (Char_t *) salida_str.c_str();
666     C = new TCanvas(salida, "Transverse momentum ISR vs
        B_Tag", 1280, 720);
667     Present(h_ISR_PT, h_BTag_PT, C, 2, "PT [GeV]", "Num.
        Jets / Total");
668     C->Write();
669     C->Close();
670
671     salida_str = head_folder_results + "Transverse
        momentum ISR, B_Tag, Leading";

```

```

672     salida = (Char_t *) salida_str.c_str();
673     C = new TCanvas(salida,"Transverse momentum ISR,
        B_Tag, Leading",1280,720);
674     Present_3(h_ISR_PT,h_BTag_PT,h_leading_PT,C,2,"PT
        [GeV]","Num. Jets / Total");
675     C->Write();
676     C->Close();
677
678     salida_str = head_folder_results + "Transverse
        momentum ISR, B_Tag, Leading LOG";
679     salida = (Char_t *) salida_str.c_str();
680     C = new TCanvas(salida,"Transverse momentum ISR,
        B_Tag, Leading LOG",1280,720);
681     Present_3(h_ISR_PT,h_BTag_PT,h_leading_PT,C,2,"PT
        [GeV]","Num. Jets / Total",12,12,true);
682     C->Write();
683     C->Close();
684
685     salida_str = head_folder_results + "Transverse
        mass Leading vs ISR Jet";
686     salida = (Char_t *) salida_str.c_str();
687     C = new TCanvas(salida,"Transverse mass Leading vs
        ISR Jet",1280,720);
688     Present(h_ISR_MT,h_leading_MT,C,2,"MT [GeV]","Num.
        Jets / Total");
689     C->Write();
690     C->Close();
691
692     salida_str = head_folder_results + "Transverse
        mass ISR vs Jet";
693     salida = (Char_t *) salida_str.c_str();
694     C = new TCanvas(salida,"Transverse mass ISR vs Jet
        ",1280,720);
695     Present(h_ISR_MT,h_jet_MT,C,2,"MT [GeV]","Num.
        Jets / Total");
696     C->Write();
697     C->Close();
698
699     salida_str = head_folder_results + "Phi";
700     salida = (Char_t *) salida_str.c_str();

```

```

701 C = new TCanvas(salida,"Phi",1280,720);
702 Present(h_ISR_Phi,h_jet_Phi,C,3,"f","Num. Jets /
    Total",122);
703 C->Write();
704 C->Close();
705
706 salida_str = head_folder_results + "Delta Phi -
    Jet - MET";
707 salida = (Char_t *) salida_str.c_str();
708 C = new TCanvas(salida,"Delta Phi - Jet - MET"
    ,1280,720);
709 Present(h_ISR_DPhi_MET,h_jet_DPhi_MET,C,3,"Df","
    Num. Jets / Total",122);
710 C->Write();
711 C->Close();
712
713 salida_str = head_folder_results + "Delta Phi -
    Jet - MET - Btag";
714 salida = (Char_t *) salida_str.c_str();
715 C = new TCanvas(salida,"Delta Phi - Jet - MET -
    Btag",1280,720);
716 Present(h_ISR_DPhi_MET,h_BTag_DPhi_MET,C,3,"Df","
    Num. Jets / Total",122);
717 C->Write();
718 C->Close();
719
720 salida_str = head_folder_results + "Delta Phi -
    Jet - MET - leading";
721 salida = (Char_t *) salida_str.c_str();
722 C = new TCanvas(salida,"Delta Phi - Jet - MET -
    leading",1280,720);
723 Present(h_ISR_DPhi_MET,h_leading_DPhi_MET,C,1,"Df"
    ,"Num. Jets / Total",122);
724 C->Write();
725 C->Close();
726
727 salida_str = head_folder_results + "MET > 120";
728 salida = (Char_t *) salida_str.c_str();
729 C = new TCanvas(salida,"MET > 120",1280,720);
730 Present(h_MET,h_MET_hpt1,C,2,"MET","Num. Jets /

```

```

731         Total");
732     C->Write();
733     C->Close();
734
735     salida_str = head_folder_results + "MET > 200";
736     salida = (Char_t *) salida_str.c_str();
737     C = new TCanvas(salida,"MET > 200",1280,720);
738     Present(h_MET,h_MET_hpt2,C,2,"MET","Num. Jets /
739           Total");
740     C->Write();
741     C->Close();
742
743     salida_str = head_folder_results + "MET > 240";
744     salida = (Char_t *) salida_str.c_str();
745     C = new TCanvas(salida,"MET > 240",1280,720);
746     Present(h_MET,h_MET_hpt3,C,2,"MET","Num. Jets /
747           Total");
748     C->Write();
749     C->Close();
750
751     salida_str = head_folder_results + "HT ratio
752           comparison";
753     salida = (Char_t *) salida_str.c_str();
754     C = new TCanvas(salida,"HT ratio comparison",
755           1280,720);
756     Present(h_HT_R1,h_HT_R2,C,2,"HT","Num. Jets /
757           Total");
758     C->Write();
759     C->Close();
760
761     salida_str = head_folder_results + "PT vs ETA -
762           ISR";
763     salida = (Char_t *) salida_str.c_str();
764     C = new TCanvas(salida,"PT vs ETA - ISR",1280,720)
765           ;
766     Plot_Single_2D(h2_ISR_PTEta,C,2, "PT [GeV]", "h",
767           12, 122);
768     C->Write();
769     C->Close();
770

```



```

762     salida_str = head_folder_results + "PT vs ETA -
       Jet";
763     salida = (Char_t *) salida_str.c_str();
764     C = new TCanvas(salida,"PT vs ETA - Jet",1280,720)
       ;
765     Plot_Single_2D(h2_jet_PTEta,C,2, "PT [GeV]", "h",
       12, 122);
766     C->Write();
767     C->Close();
768
769     salida_str = head_folder_results + "PT vs ETA -
       Diff with any jet";
770     salida = (Char_t *) salida_str.c_str();
771     C = new TCanvas(salida,"PT vs ETA - Diff with any
       jet",1280,720);
772     Plot_Single_2D(h2_dif_PTEta,C,2, "PT [GeV]", "h",
       12, 122);
773     C->Write();
774     C->Close();
775
776     salida_str = head_folder_results + "PT vs ETA -
       leading";
777     salida = (Char_t *) salida_str.c_str();
778     C = new TCanvas(salida,"PT vs ETA - leading"
       ,1280,720);
779     Plot_Single_2D(h2_leading_PTEta,C,2, "PT [GeV]", "
       h", 12, 122);
780     C->Write();
781     C->Close();
782
783     salida_str = head_folder_results + "PT vs ETA -
       Diff with leading";
784     salida = (Char_t *) salida_str.c_str();
785     C = new TCanvas(salida,"PT vs ETA - Diff with
       leading",1280,720);
786     Plot_Single_2D(h2_dif_lead_PTEta,C,2, "PT [GeV]",
       "h", 12, 122);
787     C->Write();
788     C->Close();
789

```

```

790     salida_str = head_folder_results + "HT";
791     salida = (Char_t *) salida_str.c_str();
792     C = new TCanvas(salida,"HT",1280,720);
793     Plot_Single(h_HT,C,2, "HT [GeV]", "Num. Jets /
        Total", 12, 12);
794     C->Write();
795     C->Close();
796
797     salida_str = head_folder_results + "
        Number_of_B_Tags";
798     salida = (Char_t *) salida_str.c_str();
799     C = new TCanvas(salida,"Number of B Tags"
        ,1280,720);
800     Plot_Single(h_BTags_per_Event,C,2, "B Tags / event
        ", "Num. Jets / Total", 12, 12);
801     C->Write();
802     C->Close();
803
804     salida_str = head_folder_results + "
        Jet_multiplicity";
805     salida = (Char_t *) salida_str.c_str();
806     C = new TCanvas(salida,"Jet multiplicity"
        ,1280,720);
807     Present(h_ISR_multiplicity,h_jet_multiplicity,C,2,
        "Tracks","Num. Jets / Total");
808     C->Write();
809     C->Close();
810
811     salida_str = head_folder_results + "Delta_R -
        _Jet_size";
812     salida = (Char_t *) salida_str.c_str();
813     C = new TCanvas(salida,"Delta R - Jet Size"
        ,1280,720);
814     Present(h_ISR_DeltaR,h_jet_DeltaR,C,1,"Delta_R",
        "Num. Jets / Total");
815     C->Write();
816     C->Close();
817
818     // Correlated variables
819     salida_str = head_folder_results + "

```

```

Cor_Delta_PT_Jet";
820 salida = (Char_t *) salida_str.c_str();
821 C = new TCanvas(salida,"Delta PT jet",1280,720);
822 Present(h_ISR_Delta_PT,h_jet_Delta_PT,C,2,"PT [GeV
    ]","Num. Jets / Total");
823 C->Write();
824 C->Close();
825
826 salida_str = head_folder_results + "
    Cor_PT_proportion";
827 salida = (Char_t *) salida_str.c_str();
828 C = new TCanvas(salida,"PT proportion",1280,720);
829 Present(h_ISR_PT_HT,h_jet_PT_HT,C,2,"PT/HT","Num.
    Jets / Total");
830 C->Write();
831 C->Close();
832
833 salida_str = head_folder_results + "
    Cor_Delta_Eta_Average";
834 salida = (Char_t *) salida_str.c_str();
835 C = new TCanvas(salida,"Delta Eta Average"
    ,1280,720);
836 Present(h_ISR_Delta_Eta,h_jet_Delta_Eta,C,2,"Dh","
    Num. Jets / Total",122);
837 C->Write();
838 C->Close();
839
840 salida_str = head_folder_results + "
    Cor_Delta_Phi_Jet_MET_other_jets";
841 salida = (Char_t *) salida_str.c_str();
842 C = new TCanvas(salida,"Delta Phi - Jet MET -
    other jets",1280,720);
843 Present(h_ISR_DPhi_MET_other,h_jet_DPhi_MET_other,
    C,2,"Df","Num. Jets / Total",122);
844 C->Write();
845 C->Close();
846
847 salida_str = head_folder_results + "Cor_PT_over_<
    PT_other>";
848 salida = (Char_t *) salida_str.c_str();

```

```

849     C = new TCanvas(salida,"PT/<PT_other>",1280,720);
850     Present(h_ISR_PT_over_PT_others,
            h_jet_PT_over_PT_others,C,2,"PT/<PT>","Num.
            Jets / Total");
851     C->Write();
852     C->Close();
853
854     salida_str = head_folder_results + "Cor_Eta_over_<
            Eta_other>";
855     salida = (Char_t *) salida_str.c_str();
856     C = new TCanvas(salida,"Eta/<Eta_other>",1280,720)
            ;
857     Present(h_ISR_Eta_over_Eta_others,
            h_jet_Eta_over_Eta_others,C,3,"h/<h>","Num.
            Jets / Total",122);
858     C->Write();
859     C->Close();
860
861     salida_str = head_folder_results + "
            Cor_Delta_Phi_over_<Delta_Phi_other>";
862     salida = (Char_t *) salida_str.c_str();
863     C = new TCanvas(salida,"Delta_Phi/<Delta_Phi_other
            >",1280,720);
864     Present(h_ISR_DPhi_over_Phi_others,
            h_jet_DPhi_over_Phi_others,C,3,"Df/<Df>","Num.
            Jets / Total",122);
865     C->Write();
866     C->Close();
867
868     // Comparison with the leading Jet
869     salida_str = head_folder_results + "
            Leading_Delta_PT";
870     salida = (Char_t *) salida_str.c_str();
871     C = new TCanvas(salida,"Delta PT: PT_leading-PT"
            ,1280,720);
872     Present(h_ISR_Delta_PT_leading,
            h_jet_Delta_PT_leading,C,2,"(PT_leading - PT)",
            "Num. Jets / Total");
873     C->Write();
874     C->Close();

```

```

875
876     salida_str = head_folder_results + "
        Leading_Delta_Eta";
877     salida = (Char_t *) salida_str.c_str();
878     C = new TCanvas(salida,"Delta Eta: |Eta-
        Eta_leading|",1280,720);
879     Present(h_ISR_Delta_Eta_leading,
        h_jet_Delta_Eta_leading,C,2,"|Eta - Eta_leading
        |","Num. Jets / Total");
880     C->Write();
881     C->Close();
882
883 }
884
885 hfile->Close();
886
887 hfile_name_str = head_folder_results + "histos2.root"
    ;
888
889 hfile_name = (Char_t *) hfile_name_str.c_str();
890
891 TFile* hfile2 = new TFile(hfile_name, "RECREATE");
892 h_ISR_PT_comp->Write();
893 h_ISR_Eta_comp->Write();
894 h_ISR_DPhi_MET_comp->Write();
895
896 hist_ISR_PT->Write();
897 hist_ISR_Abs_Eta->Write();
898 hist_ISR_DPhi_MET->Write();
899 hist_ISR_PT_ratio->Write();
900 hist_ISR_Delta_Eta->Write();
901 hist_ISR_DPhi_MET_other->Write();
902 hist_ISR_Delta_PT_leading->Write();
903 hist_ISR_Delta_Eta_leading->Write();
904
905 hist_jet_PT->Write();
906 hist_jet_Abs_Eta->Write();
907 hist_jet_DPhi_MET->Write();
908 hist_jet_PT_ratio->Write();
909 hist_jet_Delta_Eta->Write();

```

```

910     hist_jet_DPhi_MET_other->Write();
911     hist_jet_Delta_PT_leading->Write();
912     hist_jet_Delta_Eta_leading->Write();
913
914     hfile2->Close();
915
916     return 0;
917 }

```

B.4 Creating histograms

```

1  /*
2  -----
3  -----      Universidad de los Andes      -----
4  -----      Departamento de Fisica        -----
5  -----      Joven Investigador            -----
6  -----      Andres Felipe Garcia Albarracin -----
7  -----      Juan Carlos Sanabria Arenas    -----
8  -----
9
10 This algorithm fills 2 N-dimensional histograms.
11 The histograms contain kinematic variables of ISR
12 jets and non ISR jets.
13
14 The user can choose 3 of 8 variables for filling
15 the histograms:
16 1. PT
17 2. Abs(Eta) // Eta is a pair function
18 3. Delta Phi_MET
19 4. PT_ratio
20 5. Delta Eta_aver
21 6. Delta Phi_MET_others
22 7. Delta PT_others
23 8. Delta Eta_others
24
25 In order to choose them, the code should be run as
26 ./Creating_histo config_file.txt [N1 N2 N3]
27
28 where [config_file.txt] is a configuration file with

```

```

29 all parameters needed for the simulation.
30
31 N1 N2 and N3 are the index of the 3 variables.
32 If no parameter is passed as parameter, N1 N2 and N3
33 will be 0,1 and 2 by default.
34 */
35
36
37 #include "ROOTFunctions.h"
38 #include "graphs_Funcs.h"
39 #include "functions.h"
40 #include "histoN.h"
41 #include "DelphesFunctions.h"
42
43 // Global Variables
44 const Double_t PI = TMath::Pi();
45
46 int main(int argc, char **argv){
47     std::cout.precision(4);
48     // Counting time
49     Double_t initialTime = clock();
50
51     // Folder variables
52     string head_folder = "/home/af.garcia1214/
        PhenoMCsamples/Simulations/
        MG_pythia8_delphes_parallel/
        _Tops_Events_WI_Matching/";
53     string current_folder = "_Tops_MG_1K_AG_WI_003/";
54
55     string head_folder_binary = "/home/af.garcia1214/
        PhenoMCsamples/Results_Improved_Codes/
        matching_Results/_Tops_matches_WI_Matching/";
56     string matching_name = "ISR_jets_Tops_WI_003.bn";
57
58     string head_folder_results = "/home/af.garcia1214/
        PhenoMCsamples/Results_Improved_Codes/histo_folder
        /_Tops_histos_WI_Matching/";
59
60     // Checking input parameters
61     string config_file_name = "Debug/config_file.txt";

```

```

62 // Reading the file as first parameter
63 if (argc>1){
64     config_file_name = argv[1];
65 }
66 else{
67     cout << "It is necessary to type a configuration
        file as parameter. Execute as ./Creating_histo
        config_file.txt [N1 N2 N3]" << endl;
68     return 1;
69 }
70 cout << "Reading input parameters" << endl;
71 cout << "\tUsing as parameters' file: " <<
    config_file_name << endl;
72
73 ifstream config_file (config_file_name);
74 if (config_file.is_open()){
75     cout << "\tReading file" << endl;
76     string line;
77     int number_line = 1;
78     while (getline(config_file,line)){
79         // Skipping commented lines
80         if (line[0] == '!')
81             continue;
82
83         // Finding the position of the equal sign
84         int pos_equal = -1;
85         pos_equal = line.find('=');
86
87         if (pos_equal == -1){
88             cout << "\tLine " << number_line << " is
                incorrect" << endl;
89             continue;
90         }
91
92         // Splitting the line according to the position
            of equal sign
93         string var_name = line.substr(0,pos_equal);
94         string var_value = line.substr(pos_equal+1);
95
96         // Reading head folder

```



```

97         if(var_name.compare("head_folder") == 0){
98             head_folder = var_value;
99             cout << "\tVariable head folder set as: " <<
              head_folder << endl;
100         }
101         // Reading current folder
102         else if (var_name.compare("current_folder") ==
              0){
103             current_folder = var_value;
104             cout << "\tVariable current folder set as: "
              << current_folder <<endl;
105         }
106         // Reading head folder binary
107         else if (var_name.compare("head_folder_binary")
              == 0){
108             head_folder_binary = var_value;
109             cout << "\tVariable head folder binary set
              as: " << head_folder_binary << endl;
110         }
111         // Reading matching name
112         else if (var_name.compare("matching_name") ==
              0){
113             matching_name = var_value;
114             cout << "\tVariable matching_name set as: "
              << matching_name << endl;
115         }
116         // Reading head folder results
117         else if (var_name.compare("head_folder_results"
              ) == 0){
118             head_folder_results = var_value;
119             cout << "\tVariable head folder results set
              as: " << head_folder_results << endl;
120         }
121
122         number_line ++;
123     }
124 }
125 else
126 {
127     cout << "ERROR: File " << config_file_name << "

```

```

128         does not exist. Terminating program" << endl;
129         return 0;
130     }
131
132     cout << "\n *** Creating histograms *** \n" << endl;
133
134     // Variables for initializing histograms
135     Int_t dims = 3;
136     Double_t min_Values[3] = {0,-5.2,0};
137     Double_t max_Values[3] = {800,5.2,PI};
138
139     /*
140      * Read inputs and set variables for analysis
141      */
142     Int_t var_index[3] = {0,1,2}; // Index of the 3
143     string variables[8] = {"PT","Abs(Eta)","Delta Phi_MET",
144     "","PT_ratio","Delta Eta_aver","Delta
145     Phi_MET_others","Delta PT_leading","Delta
146     Eta_leading"};
147     Double_t var_values[8] =
148     {0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0}; // Vector with
149     the values of the 8 variables
150     // Min and maximum values of the eight variables
151     Double_t var_min_values[8] =
152     {0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0};
153     Double_t var_max_values[8] = {800,5.2,PI,8.0,7.0,PI
154     ,500,6.5};
155
156     if (argc == 5){
157         cout << "Filling histograms with the variables:"
158         << endl;
159         for (Int_t ind = 0; ind < 3; ind++){
160             var_index[ind] = atoi(argv[ind+2]);
161         }
162         cout << endl;
163     }
164     else if (argc == 1 || argc == 2) {
165         cout << "Filling histograms with the default

```

```

        variables:" << endl;
158     }
159     else {
160         cout << "Error at calling this algorithm. Use as:"
            << endl;
161         cout << "\t ./Creating_histo N1 N2 N3 or ./
            Creating_histo" << endl;
162         cout << "Read the documentation at the beginning
            of the code for further information\n" << endl;
163         return 1;
164     }
165
166     cout << "Var \t\t min_Value \t max_Value" << endl;
167     for (Int_t ind = 0; ind < 3; ind++){
168         min_Values[ind] = var_min_values[var_index[ind]];
169         max_Values[ind] = var_max_values[var_index[ind]];
170         cout << var_index[ind] << ". " << variables[
            var_index[ind]] <<
171             "\t" << min_Values[ind] << "\t" <<
            max_Values[ind] << endl;
172     }
173     cout << endl;
174
175
176     /*
177     * Initializing the 3-dimensional histogram
178     */
179     Int_t bins[3] = {20,20,20};
180     histoN* histoISR = new histoN(dims,min_Values,
        max_Values,bins);
181     histoN* histoNonISR = new histoN(dims,min_Values,
        max_Values,bins);
182     // Input variables of each histogram
183     Double_t values[3] = {0.0,0.0,0.0};
184
185     // For loop over several simulations. iRun is the
        seed of the current simulation
186     for(int iRun = 11; iRun < 261; iRun++){
187         // Create chains of root trees
188         TChain chain_Delphes("Delphes");

```

```

189
190     Char_t unidad = 0x30 + iRun%10;
191     Char_t decena = 0x30 + int(iRun/10)%10;
192     Char_t centena = 0x30 + int(iRun/100)%10;
193
194     current_folder[current_folder.size()-4] = centena;
195     current_folder[current_folder.size()-3] = decena;
196     current_folder[current_folder.size()-2] = unidad;
197     matching_name[matching_name.size()-6] = centena;
198     matching_name[matching_name.size()-5] = decena;
199     matching_name[matching_name.size()-4] = unidad;
200
201     string file_delphes_str = head_folder +
        current_folder + "Events/run_01/output_delphes.
        root";
202
203     Char_t *file_delphes = (Char_t *) file_delphes_str
        .c_str();
204
205     cout << "\nWriting run: " << centena << decena << unidad
        << endl;
206     cout << "\tReading the file: \n\tDelphes: " <<
        file_delphes << endl;
207
208     chain_Delphes.Add(file_delphes);
209     // Objects of class ExRootTreeReader for reading
        the information
210     ExRootTreeReader *treeReader_Delphes = new
        ExRootTreeReader(&chain_Delphes);
211
212     Long64_t numberOfEntries = treeReader_Delphes->
        GetEntries();
213
214     // Get pointers to branches used in this analysis
215     TClonesArray *branchJet = treeReader_Delphes->
        UseBranch("Jet");
216     TClonesArray *branchMissingET = treeReader_Delphes
        ->UseBranch("MissingET");
217
218     cout << "\tNumber of Entries Delphes = " <<

```

```

        numberOfEntries << endl;
219     cout << endl;
220
221     // particles, jets and vectors
222     MissingET *METpointer;
223     TLorentzVector *vect_currentJet = new
        TLorentzVector;
224     TLorentzVector *vect_auxJet = new TLorentzVector;
225     TLorentzVector *vect_leading = new TLorentzVector;
226     Jet *currentJet = new Jet;
227     Jet *auxJet = new Jet;
228
229     // Temporary variables
230     Double_t MET = 0.0; // Missing transverse energy
231     Double_t delta_phi = 0.0; // difference between
        the phi angle of MET and the jet
232     Double_t transverse_mass = 0.0; // Transverse mass
233     Int_t numMatches = 0; // Number of matched jets
234     Double_t delta_PT_jet = 0.0; // |PT-<PT>|
235     Double_t PT_sum = 0.0; // sum(PT)
236     Double_t PT_aver = 0.0; // <PT>
237     Double_t Delta_eta_aver = 0.0; // sum_i|eta-eta_i
        |/(Nj-1)
238     Double_t Delta_phi_sum = 0.0; // sum delta_phi
239     Double_t Delta_phi_other_jets = 0.0; // Average of
        delta phi of other jets
240     Double_t PT_ratio = 0.0; // PT/PT_others
241     Double_t Delta_PT_leading = 0.0; // PT -
        PT_leading
242     Double_t Delta_Eta_leading = 0.0; // |Eta -
        Eta_leading|
243
244     // Jet with greatest PT
245     Double_t PT_max = 0;
246     Int_t posLeadingPT = -1;
247     Int_t ISR_greatest_PT = 0;
248     Double_t MT_leading_jet = 0.0; // Transverse mass
249
250     /*
251     * Some variables used through the code

```

```

252     */
253     Int_t ISR_jets[numberOfEntries];
254     Int_t NumJets = 0;
255
256     string fileName_str = head_folder_binary +
257         matching_name;
258
259     Char_t * fileName = (Char_t *) fileName_str.c_str
260         ();
261
262     ifstream ifs(fileName, ios::in | ios::binary);
263
264     for (Int_t j = 0; j<numberOfEntries; j++){
265         ifs.read((Char_t *) (ISR_jets+j), sizeof(
266             Int_t));
267     }
268     ifs.close();
269
270     /*
271     * Main cycle of the program
272     */
273     numberOfEntries = 100000;
274     for (Int_t entry = 0; entry < numberOfEntries; ++
275         entry){
276         // Progress
277         if(numberOfEntries>10 && (entry%((int)
278             numberOfEntries/10))==0.0){
279             cout<<"\tprogress = "<<(entry*100/
280                 numberOfEntries)<<"%\t";
281             cout<< "Time : "<< (clock()-initialTime)/
282                 double_t(CLOCKS_PER_SEC)<<"s"<<endl;
283         }
284
285         // Load selected branches with data from
286         // specified event
287         treeReader_Delphes->ReadEntry(entry);
288
289         // MET
290         METpointer = (MissingET*) branchMissingET->At
291             (0);

```

```

283     MET = METpointer->MET;
284
285     NumJets=branchJet->GetEntries();
286
287     // checking the ISR
288     if (ISR_jets[entry] == -1 || NumJets < 3)
289         continue;
290
291     if (ISR_jets[entry] >= NumJets){
292         cout << "Error en el matching" << endl;
293         return 1;
294     }
295
296     // 3 PT ratio
297     PT_aver = 0.0;
298     PT_sum = 0.0;
299     PT_ratio = 0.0;
300
301     // 4 Delta Eta aver
302     Delta_eta_aver = 0.0;
303
304     // 5 Delta Phi others
305     Delta_phi_sum = 0.0;
306     Delta_phi_other_jets = 0.0;
307
308     // 6 Delta PT leading
309     PT_max = 0.0;
310     Delta_PT_leading = 0.0;
311     delta_PT_jet = 0.0; // If needed
312
313     // 7 Delta Eta leading
314     Delta_Eta_leading = 0.0;
315
316     // Reset Var_values (Not necessary)
317     for(Int_t ind = 0; ind < 8; ind++){
318         var_values[ind] = 0.0;
319         if (ind < dims) values[ind] = 0.0;
320     }
321
322     // Preliminary for. It is used to calculate

```

```

323         PT_aver and Delta_phi_sum
324         for (Int_t iJet = 0; iJet<NumJets; iJet++){
325             currentJet = (Jet*) branchJet->At(iJet);
326             vect_currentJet->SetPtEtaPhiM(currentJet->PT
327                 ,currentJet->Eta,currentJet->Phi,
328                 currentJet->Mass);
329             PT_sum += vect_currentJet->Pt();
330             delta_phi = deltaAng(vect_currentJet->Phi(),
331                 METpointer->Phi());
332             Delta_phi_sum += delta_phi;
333             // PT Leading jet
334             if(PT_max < vect_currentJet->Pt()){
335                 PT_max = vect_currentJet->Pt();
336                 posLeadingPT = iJet;
337             }
338         }
339
340         numMatches++;
341
342         //PT_aver
343         PT_aver = PT_sum/NumJets;
344
345         // Leading PT
346         currentJet = (Jet*) branchJet->At(posLeadingPT)
347             ;
348         vect_leading->SetPtEtaPhiM(currentJet->PT,
349             currentJet->Eta,currentJet->Phi,currentJet->
350             Mass);
351
352         for (Int_t iJet = 0; iJet<NumJets; iJet++){
353             currentJet = (Jet*) branchJet->At(iJet);
354             vect_currentJet->SetPtEtaPhiM(currentJet->PT
355                 ,currentJet->Eta,currentJet->Phi,
356                 currentJet->Mass);
357
358             // 2 Delta Phi MET
359             delta_phi = deltaAng(vect_currentJet->Phi(),
360                 METpointer->Phi());
361
362             // PT ratio

```



```

353     PT_ratio = vect_currentJet->Pt()*(NumJets-1)
              /(PT_sum-vect_currentJet->Pt());
354
355     // 4 Delta Eta Aver
356     Delta_eta_aver = 0.0;
357     // For cycle used to calculate
        Delta_eta_aver
358     for(Int_t iJet2 = 0; iJet2<NumJets; iJet2++)
        {
359         auxJet = (Jet*) branchJet->At(iJet2);
360         vect_auxJet->SetPtEtaPhiM(auxJet->PT,
            auxJet->Eta,auxJet->Phi,auxJet->Mass);
361         if (iJet2 != iJet) Delta_eta_aver +=
            TMath::Abs(vect_auxJet->Eta()-
                vect_currentJet->Eta());
362     }
363     Delta_eta_aver = Delta_eta_aver/(NumJets-1);
364
365     // 5 Delta Phi MET Others
366     Delta_phi_other_jets = (Delta_phi_sum-
        delta_phi)/(NumJets-1);
367
368     // 6 Delta PT leading
369     Delta_PT_leading = vect_leading->Pt()-
        vect_currentJet->Pt();
370
371     // 7 Delta Eta leading
372     Delta_Eta_leading = TMath::Abs(
        vect_currentJet->Eta()-vect_leading->Eta
        ());
373
374     // Other variables
375     delta_PT_jet = TMath::Abs(vect_currentJet->
        Pt()-PT_aver);
376     transverse_mass = sqrt(2*vect_currentJet->Pt
        ()*MET*(1-cos(delta_phi)));
377
378     // Filling the array with the variables'
        values
379     var_values[0] = vect_currentJet->Pt();

```

```

380         var_values[1] = TMath::Abs(vect_currentJet->
           Eta());
381         var_values[2] = delta_phi;
382         var_values[3] = PT_ratio;
383         var_values[4] = Delta_eta_aver;
384         var_values[5] = Delta_phi_other_jets;
385         var_values[6] = Delta_PT_leading;
386         var_values[7] = Delta_Eta_leading;
387
388         for (Int_t ind = 0; ind < dims; ind++){
389             int pos = *(var_index+ind);
390             values[ind] = *(var_values+pos);
391         }
392
393         if (iJet != ISR_jets[entry]){
394             // Non ISR jet
395             histoNonISR->fill(values);
396         }
397         else{
398             // ISR jet
399             histoISR->fill(values);
400         }
401     }
402 }
403 }
404
405     cout<<"\tprogress = 100%\t";
406     cout<<"Time : "<< (clock()-initialTime)/double_t(
         CLOCKS_PER_SEC)<<"s"<<endl;
407     cout<<"\n\tNumber of Written Events: "<<numMatches
         <<endl;
408 } // End run's for cicle
409
410 /*
411  * Writing the histogram
412  */
413 // Counting time
414 Double_t partialTime = clock();
415 cout<< "Time building the histogram:"<< (partialTime-
         initialTime)/double_t(CLOCKS_PER_SEC)<<"s"<<endl;

```

```

416
417 // Writing the histogram
418 cout<<"Min value 1: "<<min_Values[0]<<endl;
419 Int_t* freq;
420 for(Int_t j = 0; j<dims; j++){
421     cout<<"ISR Jets - Events of the dimension:\t"<<
422         j<<endl;
423     freq = histoISR->getHistDim(j);
424     for(Int_t i = 0; i<bins[j];i++){
425         cout<<"\t"<<freq[i];
426         if(i>0 && ((i+1)%10 == 0.0))
427             cout<<endl;
428     }
429     cout<<endl;
430 }
431 cout<<endl<<"\t\t ***"<<endl<<endl;
432 for(Int_t j = 0; j<dims; j++){
433     cout<<"Non ISR Jets - Events of the dimension:\t"
434         <<j<<endl;
435     freq = histoNonISR->getHistDim(j);
436     for(Int_t i = 0; i<bins[j];i++){
437         cout<<"\t"<<freq[i];
438         if(i>0 && ((i+1)%10 == 0.0))
439             cout<<endl;
440     }
441     cout<<endl;
442 }
443 cout<<"Entries: "<<histoISR->getEntries()<<endl;
444
445 /*
446  * Creating histograms
447  */
448 cout<<"\nWriting..."<<endl;
449
450 // Defining the names of the files
451 string combination = "_____"; // Combination of
452     variables
453 for (Int_t ind = 0; ind < dims; ind ++){

```

```

453     combination[(ind*2)+1] = (Char_t) (0x30 +
454         var_index[ind]); // Int to char
455 }
456 string info_ISR_name_str = head_folder_results + "
457     info_histo_ISR" + combination + ".txt";
458 Char_t *info_ISR_name = (Char_t *) info_ISR_name_str.
459     c_str();
460 string array_ISR_name_str = head_folder_results + "
461     array_histo_ISR" + combination + ".bn";
462 Char_t *array_ISR_name = (Char_t *)
463     array_ISR_name_str.c_str();
464 string info_Non_ISR_name_str = head_folder_results +
465     "info_histo_Non_ISR" + combination + ".txt";
466 Char_t *info_Non_ISR_name = (Char_t *)
467     info_Non_ISR_name_str.c_str();
468 string array_Non_ISR_name_str = head_folder_results +
469     "array_histo_Non_ISR" + combination + ".bn";
470 Char_t *array_Non_ISR_name = (Char_t *)
471     array_Non_ISR_name_str.c_str();
472 cout << "Output files:\n\t" << info_ISR_name << "\n\t"
473     << array_ISR_name << "\n\t" << info_Non_ISR_name
474     << "\n\t" << array_Non_ISR_name << endl;
475 histoISR->writeClass((Char_t*) info_ISR_name,(Char_t
476     *) array_ISR_name);
477 histoNonISR->writeClass((Char_t*) info_Non_ISR_name,(
478     Char_t*)array_Non_ISR_name);
479 cout<< "Time writing the file:"<< (clock()-
480     partialTime)/double_t(CLOCKS_PER_SEC)<<"s"<<endl;
481
482 cout<<"Fin :)"<<endl;
483
484 return 0;
485 }

```

Bibliography

- [1] J. Alwall, R. Frederix, S. Frixione, V. Hirschi, F. Maltoni, et al. The automated computation of tree-level and next-to-leading order differential cross sections, and their matching to parton shower simulations. *JHEP*, 1407:079, 2014.
- [2] Torbjörn Sjöstrand, Stefan Ask, Jesper R. Christiansen, Richard Corke, Nishita Desai, et al. An Introduction to PYTHIA 8.2. *Comput.Phys.Commun.*, 191:159–177, 2015.
- [3] Torbjorn Sjostrand, Stephen Mrenna, and Peter Skands. Pythia 6.4 physics and manual. *JHEP*, 05:026, 2006.
- [4] J. de Favereau et al. DELPHES 3, A modular framework for fast simulation of a generic collider experiment. *JHEP*, 1402:057, 2014.
- [5] Lynn Garren. *StdHep 5.06.01 Monte Carlo Standardization at FNAL Fortran and C Implementation*. Fermilab, 11 2006. Available at <http://cepa.fnal.gov/psm/stdhep/>.
- [6] O. Tange. Gnu parallel - the command-line power tool. *;login: The USENIX Magazine*, 36(1):42–47, Feb 2011.
- [7] Simon De Visscher and Johan Alwall. Introduction to jet-parton matching in mg/me, 03 2011. Available at <https://cp3.irmp.ucl.ac.be/projects/madgraph/wiki/IntroMatching>.
- [8] Karl Kaess. Configuring eclipse for root. Available at <http://www.tc.umn.edu/~kaes0001/randomstuff/EclipseRoot.pdf>.