Documentation of the project: ISR jet tagging

Author:
Andrés Felipe García Albarracín

Advisor:
Juan Carlos Sanabria, Ph.D.

July 14, 2015

Contents

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 (QFD) 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 QFD, 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 / PGS[1]. 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 $p\ p\ \rightarrow t\ \tilde{t}$. 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 $p\ p\ \rightarrow t\ \tilde{t}$:
   `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 programs[3]. 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,

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

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

all the configuration parameters can be defined trough 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/LO/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`.

[4] Observe that it is supposed that `mg5_input.mg5` is localed 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 `.cmnd` is passed by parameter to configure the simulation.

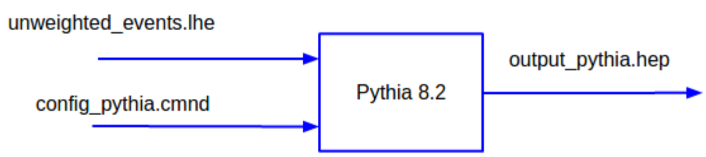


Figure 2.1

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 Pythia[5] and StdHep[6] [5]. Once installed, go to the `examples` folder located at the Pythia directory [7]. Inside such folder, copy the code `hadronization02.cc` and

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

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

[7] If `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`:

```
# Hadronization. (To compile files that read .lhe files and produce stdhep files)
# No further modifications are needed to compile the class UserHooks
hadronization% : hadronization%.cc $(PREFIX_LIB)/libpythia8.a
	$(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 $t\,\tilde{t}$ production example of the previous section, the following file may be saved as `input.cmnd` and used as input of the Pythia simulation:

```
! Hadronization from a .lhe file
! This file contains commands to be read on a Pythia8 run.
! Lines not beginning with a letter or digit are comments.

// Specify statistics parameters.
Main:numberOfEvents    = 1000  ! number of events generated (It needs to be <= Number of events generated in MG)
Init:showChangedParticleData = off ! not useful info
Next:numberShowInfo    = 1  ! 1 to show info, 0 to not
Next:numberShowEvent   = 0  ! Especify the number of events that will be listed as output

// Read .lhe file
Beams:frameType = 4 ! To take a MG file as input
Beams:LHEF = unweighted_events.lhe  ! MG .lhe file

! Hadronization:
PartonLevel:FSR = off ! switch final state radiation
PartonLevel:ISR = on ! switch initial state radiation
PartonLevel:MPI = off ! switch off multiparton interactions
Random:setSeed = on ! For random seed
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 (`.cmnd` 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 exits 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 $t\,\tilde{t}$ 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. Sanz[8] 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 project[9] 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.

[8] d-sanz@uniandes.edu.co
[9] https://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.cmnd`

  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 $p\ p \rightarrow t\ \tilde{t}$, 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 `DELPHESCARDFOLDER`. 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 $6500 GeV$ in lines 41 and 42.
   (c) In the file `input_pythia.cmnd`, 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 typing[10]:

```
./script_Integration.sh
```

[10] Possibly, 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_matchs_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.

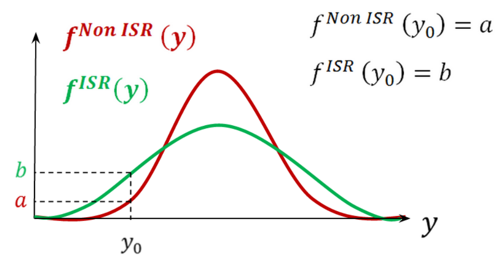


Figure 3.1: Probability distributions of a variable that distinguishes between ISR and Non ISR jets

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:

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

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

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

$$P^{ISR}_{apriori}(y_0) = \frac{1}{N_{jets}}, \tag{3.3}$$

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

$$P^{Non\ ISR}_{apriori}(y_0) = \frac{N_{jets} - 1}{N_{jets}}. \tag{3.4}$$

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

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

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

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

$$1 = P^{ISR}(y_0) + P^{FSR}(y_0), \tag{3.7}$$

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

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 de 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:

$$P^{ISR}(\vec{y_0}) = \alpha f^{ISR}(\vec{y_0}) \frac{1}{N_{jets}}, \tag{3.9}$$

$$P^{Non\ ISR}(\vec{y_0}) = \alpha f^{Non\ ISR}(\vec{y_0}) \frac{N_{jets} - 1}{N_{jets}}, \tag{3.10}$$

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:

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

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:

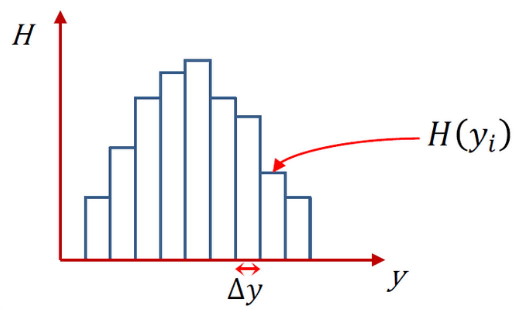


Figure 3.2: Shape of a histogram which does not exactly correspond with the probability density function

$$H(y_i) \approx f(y_i)\Delta y, \tag{3.12}$$

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

$$P^{ISR}(\vec{y_0}) = \alpha H^{ISR}(\vec{y_0})\frac{1}{N_{jets}}, \tag{3.13}$$

$$P^{Non\ ISR}(\vec{y_0}) = \alpha H^{Non\ ISR}(\vec{y_0})\frac{N_{jets}-1}{N_{jets}}. \tag{3.14}$$

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.

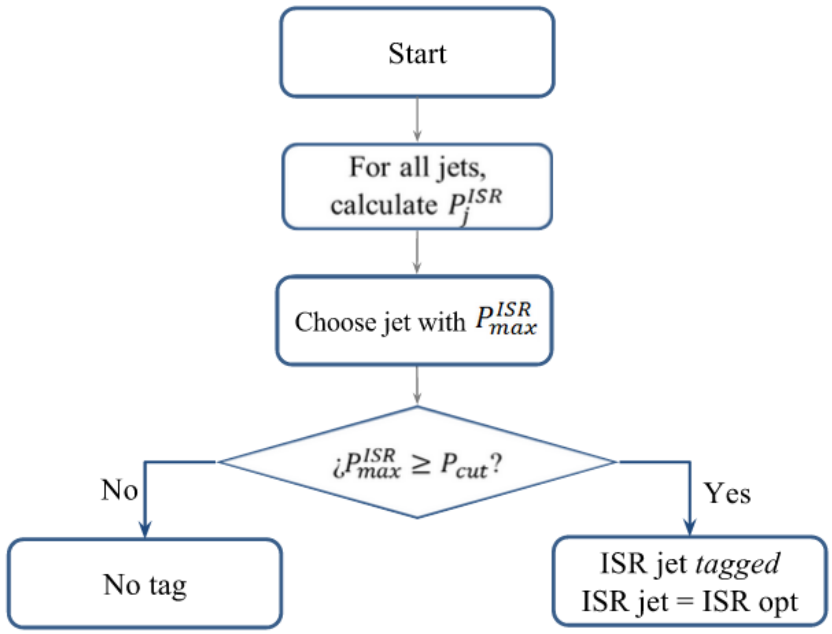


Figure 3.3: ISR jet tagging algorithm

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 *apriori* 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 *apriori* probability:

$$P_{cut} = \frac{k}{N_{jets}} \tag{3.15}$$

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

[1] We 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].

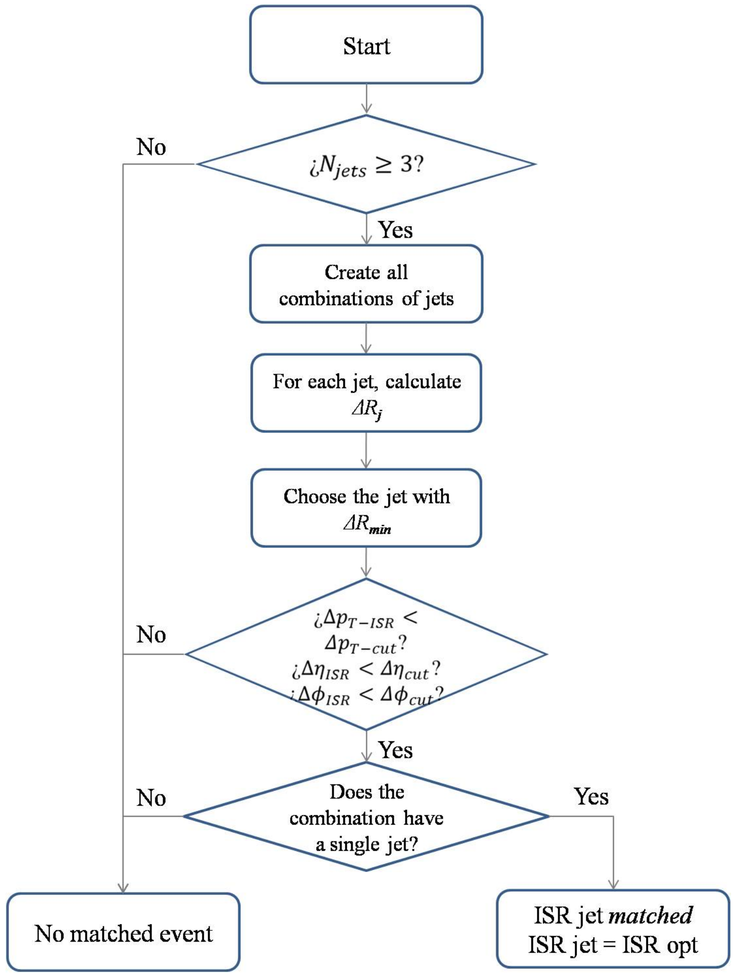


Figure 3.4: Matching algorithm between MadGraph and Pythia

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 histograms[4] of kinematic variables

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

[3] Check the structure of the name in section 3.1

[4] Root 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.

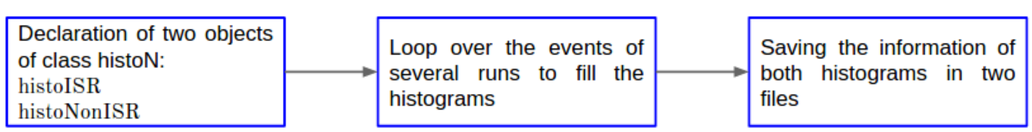


Figure 3.5: Procedure to create N-dimensional histograms

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`

[1] Check 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:

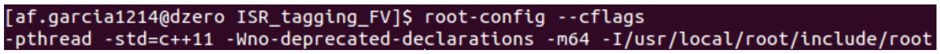
1. Before using eclipse, type the following command on a terminal:

    `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

[2] Uncomment 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.



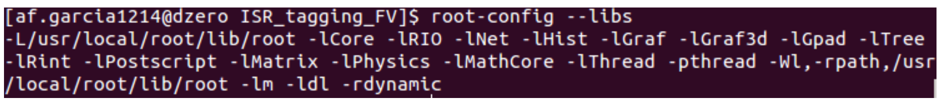
```
[af.garcia1214@dzero ISR_tagging_FV]$ root-config --cflags
-pthread -std=c++11 -Wno-deprecated-declarations -m64 -I/usr/local/root/include/root
```

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 -Wl,-rpath,/usr
/local/root/lib/root -lm -ldl -rdynamic
```

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 -xzvf 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`

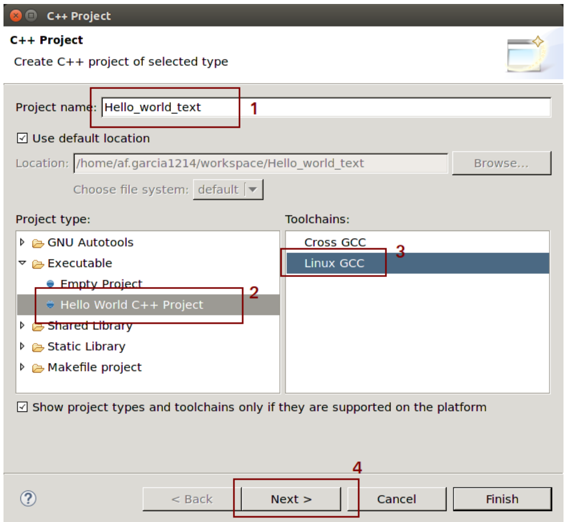


Figure 4.3: Configuration of a Hello_world program in Eclipse

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`.

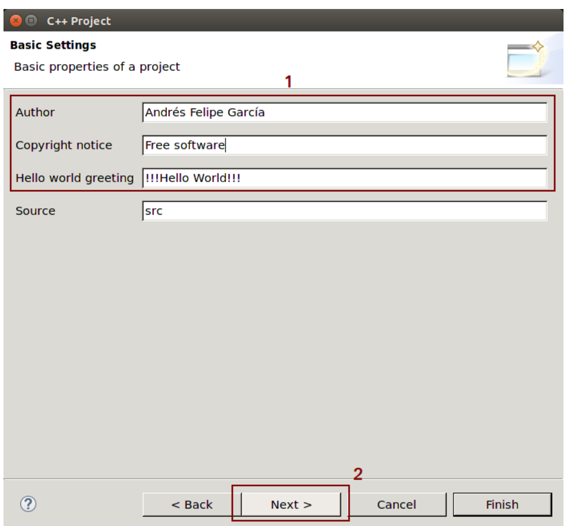


Figure 4.4: Setting the author in a eclipse project

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

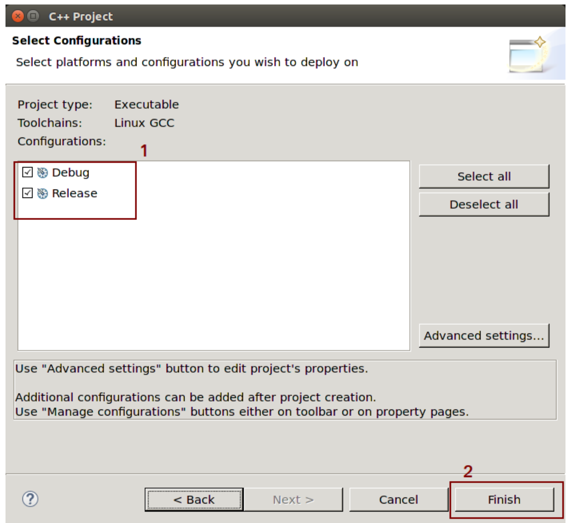


Figure 4.5: Setting *release* and *debug* in eclipse

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 programs[3]:

    - 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.

[3]ROOT is necessary to use the other two

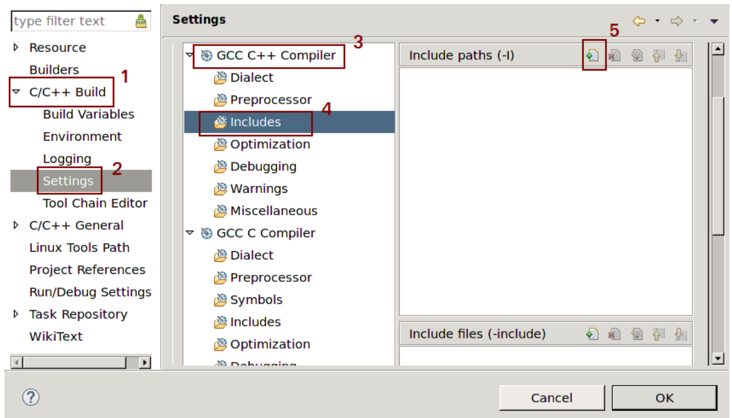


Figure 4.6: Steps to include paths in an eclipse project

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.

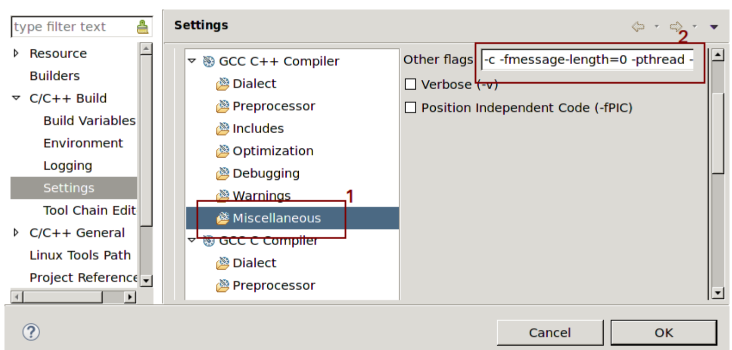


Figure 4.7: Setting miscellaneous flags on eclipse

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

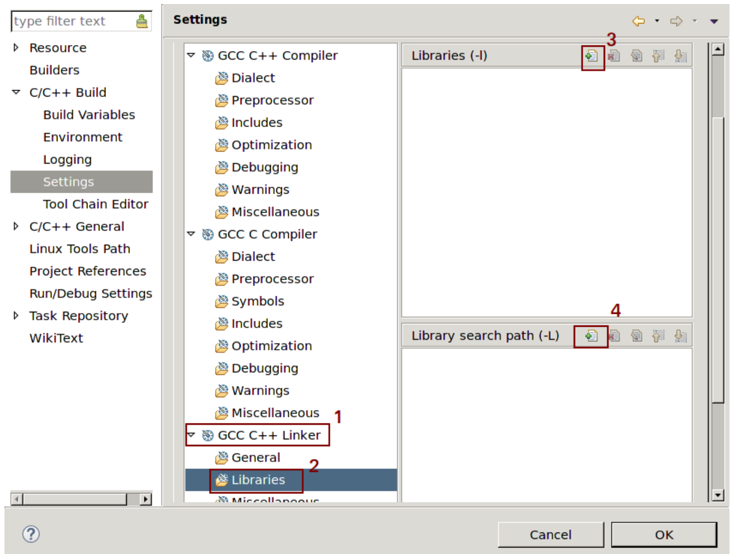


Figure 4.8: Configuration of libraries in an eclipse project

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)

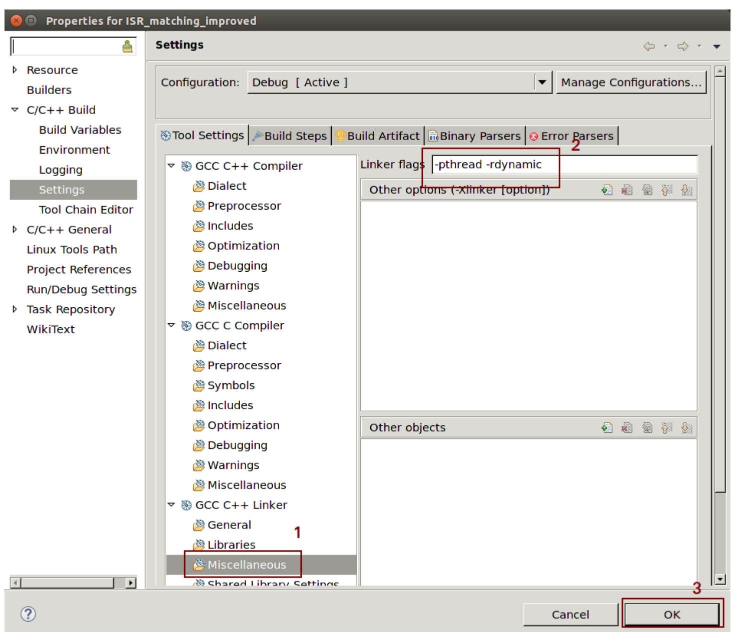


Figure 4.9: Setting miscellaneous linker flags in eclipse

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

[4]It 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

```
// Copyright (C) 2015 Torbjorn Sjostrand.
// PYTHIA is licenced under the GNU GPL version 2, see
   COPYING for details.
// Please respect the MCnet Guidelines, see GUIDELINES
   for details.

/*
-------     Universidad de los Andes      -------
-------      Departamento de Fisica       -------
-------   Proyecto Joven Investigador     -------
-------  Andres Felipe Garcia Albarracin  -------
-------    Juan Carlos Sanabria Arenas    -------

This code develops pythia hadronization. Takes as
parameter a .cmnd file, where a .lhe file from MadGraph
and other parameters are specified. Then the code
produces .hep files after making the hadronization

Obs: The class MyUserHooks is written in order to
veto all the ISR emissions produced after the
first ISR parton. It is an extension of the code
```

```
hadronization01

run as ./hadronization02 input.cmnd [output.hep]

The MakeFile has been also modified to compile
this file
*/

#include "Pythia8/Pythia.h"
#include "stdhep.h"
#include "stdcnt.h"
#include "stdhep_mcfio.h"
#include <string.h>

using namespace Pythia8;
void fill_stdhep(int i, Event &e);

// Write own derived UserHooks class.

class MyUserHooks : public UserHooks {

public:

   // Constructor.
   MyUserHooks() { }

   // Destructor.
   ~MyUserHooks() { }

   // Allow a veto of ISR emissions
   virtual bool canVetoISREmission(){
      return true;   // Interrupts the initial shower emission after each emission
            // and allow the emission to be vetoed by the next method.
   }

   // Analize each emissionand asks for the number of the ISR emissions so far, in order
   // to allow just 1 ISR parton per event
```

```
  virtual bool doVetoISREmission(int sizeOld, const
     Event& event, int iSys){
     // counts the number of ISR partons (i.e. the
        numer of particles with status 43)
     int ISR_part = 0;
     for( int i = 0; i < event.size(); i++){
        if (event[i].status() == 43 || event[i].status
           () == -43)
           ISR_part ++;
     }
     if (ISR_part > 1)
        return true;
     else
        return false;
  }
};

//==================================================


int main(int argc, char** argv) {

  // Interface for conversion from Pythia8::Event to
     HepMC event.
  char fileout[500], title[100];
  strcpy(title,"output_pythia8\0");

       // Set up generation.
  // Declare Pythia object
       Pythia pythia;

       // Set simulation configurations. Read the file
          as parameter. If none, it reads hadro_input.
          cmnd
       if (argc > 1 ) pythia.readFile(argv[1]);
       else {
     cout << "ERROR: \n No  parameters file has passed
        as parameter. Abort " << endl;
     return 1;
  }
```

```

  // Specify the name of the output file
  if (argc > 2 ) strcpy(fileout,argv[2]);
  else strcpy(fileout,"output_pythia8.hep\0");

  // Especify the number of events
       int nEvent = pythia.mode("Main:numberOfEvents"); // For reading only
  int nAbort = 10; // Maximum number of failures accepted
  int iAbort = 0; // Abortions counter

  // Necessary stdhep functions
  int istr(0);
  int ierr = StdHepXdrWriteOpen(fileout, title, nEvent, istr);

  // Set up to do a user veto and send it in.
  MyUserHooks* myUserHooks = new MyUserHooks();
  pythia.setUserHooksPtr( myUserHooks);

  // Initialize simulation
  pythia.init();

  // Begin event loop; generate until none left in input file.
  for (int iEvent = 0; iEvent < nEvent ; ++iEvent) {
     // Generate events, and check whether generation failed.
     if (!pythia.next()) {
        // If failure because reached end of file then exit event loop.
        if (pythia.info.atEndOfFile()) break;
        // First few failures write off as "acceptable" errors, then quit.
        if (++iAbort < nAbort) continue;
        break;
     }

     // Fill stdhep file
```

```
        fill_stdhep(iEvent+1,pythia.event);
        ierr = StdHepXdrWrite(1,istr);
    }

    StdHepXdrEnd(istr);
    pythia.stat();
    cout << ierr;
    delete myUserHooks;
    return 0;

}

// This functions writes in stdhep format. It was
   written by Steve Mrenna
void fill_stdhep(int i, Event &e)
{
    int num = e.size();
    hepevt_.nevhep = i;
    hepevt_.nhep = num;
    for (int j = 0; j < num; j++) {
        hepevt_.idhep[j] = e[j].id();
        hepevt_.isthep[j] = e[j].statusHepMC();
        hepevt_.jmohep[j][0] = (e[j].mother1()>0) ? e[j].
           mother1()+1 : 0;
        hepevt_.jmohep[j][1] = (e[j].mother2()>0) ? e[j].
           mother2()+1 : 0;
        hepevt_.jdahep[j][0] = (e[j].daughter1()>0) ? e[j
           ].daughter1()+1 : 0;
        hepevt_.jdahep[j][1] = (e[j].daughter2()>0) ? e[j
           ].daughter2()+1 : 0;
        hepevt_.phep[j][0] = e[j].px();
        hepevt_.phep[j][1] = e[j].py();
        hepevt_.phep[j][2] = e[j].pz();
        hepevt_.phep[j][3] = e[j].e();
        hepevt_.phep[j][4] = e[j].m();
        hepevt_.vhep[j][0] = e[j].xProd();
        hepevt_.vhep[j][1] = e[j].yProd();
        hepevt_.vhep[j][2] = e[j].zProd();
        hepevt_.vhep[j][3] = e[j].tProd();
    }
```

```
}
```

A.2 Integration scripts: MadGraph + Pythia + Delphes

A.2.1 Configuration script: config_Integration.ini

```
# ------------------------------------------------
# -------     Universidad de los Andes      -------
# -------      Departamento de Fisica       -------
# -------        Joven Investigador         -------
# -------  Andres Felipe Garcia Albarracin  -------
# -------    Diego Alejandro Sanz Becerra   -------
# -------    Juan Carlos Sanabria Arenas    -------
# ------------------------------------------------
# This file configures the inputs for MadGraph execution
# Based on Diego Sanz's configuration file:
   configMGParallel.ini

## EVENTSFOLDER IS THE NAME OF THE FOLDER WHERE ALL RUNS
    WILL BE SAVED
EVENTSFOLDER="current_dir/_Channel_Events"
## NAMESUBFOLDER IS THE NAME-STEM OF ALL THE RUNS. THE
   SUBFOLDERS INSIDE EVENTSFOLDER WILL START WITH THIS
NAMESUBFOLDER="_Channel_Sim_"
## MADGRAPHFOLDER IS THE LOCATION WHERE MADGRAPH IS
   INSTALLED. USER SHOULD CHANGE THIS TO HIS MADGRAPH
   INSTALLATION FOLDER
MADGRAPHFOLDER=
## RUNCARDFOLDER IS THE LOCATION WHERE THE RUN_CARD
   FRAME USED FOR ALL THE RUNS IS
RUNCARDFOLDER=
## PARAMCARDFOLDER IS THE LOCATION WHERE THE PARAM_CARD
   FOR ALL THE RUNS IS (check at the Madgraph folder: /
   models/sm_v4, for instance)
PARAMCARDFOLDER=
## MADGRAPHFILEFOLDER IS THE LOCATION WHERE THE MADGRAPH
   -SCRIPT FRAME IS
MADGRAPHFILEFOLDER=
## RUNCARDFILE IS THE NAME OF THE RUN_CARD FRAME USED
```

```
   FOR ALL THE RUNS
RUNCARDFILE="run_card.dat"
## PARAMCARDFILE IS THE NAME OF THE PARAM_CARD USED FOR
   ALL THE RUNS
PARAMCARDFILE="param_card.dat"
## MADGRAPHFILE IS THE NAME OF THE MADGRAPH-SCRIPT FRAME
    USED FOR ALL THE RUNS
MADGRAPHFILE="mgFile.mg5"
## CORESNUMBER IS THE NUMER OF CORES USED FOR EACH RUN
CORESNUMBER=2
## NUMEVENTSRUN IS THE NUMBER OF EVENTS FOR EACH OF THE
   RUNS
NUMEVENTSRUN=100000
## INIRUN IS THE INITIAL SEED USED FOR THE PARALLEL RUNS
INIRUN=20
## ENDRUN IS THE FINAL SEED USED FOR THE PARALLEL RUNS
ENDRUN=20

## *** Pythia 8
## DIRECTORY OF PYTHIA 8 EXECUTABLE (WHERE
   hadronization02 IS LOCATED)
PYTHIA8FOLDER=
## PYTHIA 8 .EXE
PYTHIA8EXE="hadronization02"
## PYTHIAPARAMFOLDER IS THE NAME OF THE FOLDER WHERE THE
    PYTHIA PARAMETER FILE IS LOCATED
PYTHIAPARAMFOLDER=
## PYTHIAPARAM IS THE NAME OF THE .cmnd FILE THAT SERVES
    AS PARAMETER TO PYTHIA
PYTHIAPARAM="input_pythia.cmnd"

## *** Delphes
## DIRECTORY OF DELPHES EXECTUABLE
DELPHESFOLDER=
## DELPHES .EXE
DELPHESEXE="DelphesSTDHEP"
## DELPHESCARDFOLDER IS THE NAME OF THE FOLDER WHERE THE
    DELPHES CARD IS LOCATED (check at the Delphes folder
   : /cards/)
DELPHESCARDFOLDER=
```

```
## DELPHESCARD IS THE NAME OF THE .lct FILE THAT SERVES AS PARAMETER TO DELPHES
DELPHESCARD="delphes_card_CMS.tcl"

## EXROOTANALYSIS
## DIRECTORY OF EXROOTANALYSIS
EXROOTFOLDER=
## EXROOT .EXE (STDHEP ---> .ROOT)
EXROOTEXE="ExRootSTDHEPConverter"
```

A.2.2 Execution script: script_Integration.sh

```
#!/bin/bash
# --------------------------------------------------
# -------      Universidad de los Andes       -------
# -------       Departamento de Fisica        -------
# -------         Joven Investigador          -------
# -------  Andres Felipe Garcia Albarracin    -------
# -------     Diego Alejandro Sanz Becerra    -------
# -------     Juan Carlos Sanabria Arenas     -------
# --------------------------------------------------
# This file executes parallel simulations with the programs: MadGraph 5.2 + Pythia 8.2 + Delphes 3.2
# Based on Diego Sanz's execution file: scriptMGParallelV2.sh

# Load the parameter file
source config_Integration.ini
## make the RunCards Folder in the EVENTSFOLDER
mkdir ${EVENTSFOLDER}/RunCards
## make the ParamCard Folder in the EVENTSFOLDER
mkdir ${EVENTSFOLDER}/ParamCard
## copy the param card supplied to the EVENTSFOLDER/ParamCard and name it param_card.dat
cp ${PARAMCARDFOLDER}/${PARAMCARDFILE} ${EVENTSFOLDER}/ParamCard/param_card.dat

## first sequence for each run, where the madgraph files and the run cards are created
sequ () {
```

```
    ## copy the run card frame to the RunCards directory and append the seed (counter $i)
    cp ${RUNCARDFOLDER}/${RUNCARDFILE} ${EVENTSFOLDER}/RunCards/run_card_$i.dat
    ## copy the MadGraph file to the RunCards directory as mgParallelFile_$i
    cp ${MADGRAPHFILEFOLDER}/${MADGRAPHFILE} ${EVENTSFOLDER}/RunCards/mgFile_$i.mg5
    ## copy the parameter pythia file to the RunCards directory
    cp ${PYTHIAPARAMFOLDER}/${PYTHIAPARAM} ${EVENTSFOLDER}/RunCards/input_pythia_$i.cmnd
    ## copy the delphes card to the RunCards directory *** Delphes card is the same for all runs
    cp ${DELPHESCARDFOLDER}/${DELPHESCARD} ${EVENTSFOLDER}/RunCards/${DELPHESCARD}
    ## change all the instances of SEED to the counter $i on the file run_card_$i.dat
    sed -i "s/SEED/$i/g" ${EVENTSFOLDER}/RunCards/run_card_$i.dat
    ## change all the instances of SEED to the counter $i on the file mgParallelFile_$i.mg5
    sed -i "s/SEED/$i/g" ${EVENTSFOLDER}/RunCards/mgFile_$i.mg5
    ## change all the instances of SEED to the counter $i on the file input_pythia_$i.cmnd
    sed -i "s/SEED/$i/g" ${EVENTSFOLDER}/RunCards/input_pythia_$i.cmnd
    ## change all the instances of RUNEVENTSNUM to $NUMEVENTSRUN on the file run_card_$i.dat
    sed -i "s/RUNEVENTSNUM/$NUMEVENTSRUN/g" ${EVENTSFOLDER}/RunCards/run_card_$i.dat
    ## change all the instances of FOLDEREVENTS to $EVENTSFOLDER on the file mgParallelFile.mg5
    sed -i "s|FOLDEREVENTS|$EVENTSFOLDER|g" ${EVENTSFOLDER}/RunCards/mgFile_$i.mg5
    ## change all the instances of NUMBERCORES to $CORESNUMBER on the file mgParallelFile.mg5
    sed -i "s|NUMBERCORES|$CORESNUMBER|g" ${EVENTSFOLDER}/RunCards/mgFile_$i.mg5
```

```
    ## change all the instances of SUBFOLDERNAME to $NAMESUBFOLDER on the file mgParallelFile_$i.mg5
    sed -i "s|SUBFOLDERNAME|$NAMESUBFOLDER|g" ${EVENTSFOLDER}/RunCards/mgFile_$i.mg5
    ## change all the instances of RESULTSFOLDER to the name of the folder where the results are located
    sed -i "s|RESULTSFOLDER|${EVENTSFOLDER}/${NAMESUBFOLDER}_$i/Events/run_01|g" ${EVENTSFOLDER}/RunCards/input_pythia_$i.cmnd
    ## change all the instances of RUNEVENTSNUM to $NUMEVENTSRUN on the file parameter pythia file
    sed -i "s/RUNEVENTSNUM/$NUMEVENTSRUN/g" ${EVENTSFOLDER}/RunCards/input_pythia_$i.cmnd
}

## 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
sequ2 () {
    source config_Integration.ini
        ## run madgraph with the corresponding madgraph file .mg5. all the messages are thrown to /dev/null
    ## Madgraph execution
    $1/bin/mg5_aMC -f $2/RunCards/mgFile_$4.mg5 # &> /dev/null
        ## sleep for 1s. Important, for the wait order to work
        sleep 1s
        ## wait for previous subprocesses to finish
        wait
    # Uncompress .lhe.gz file
    gzip -d $2/$3_$4/Events/run_01/unweighted_events.lhe.gz

    ## Pythia 8 execution
    ${PYTHIA8FOLDER}/${PYTHIA8EXE} $2/RunCards/input_pythia_$4.cmnd $2/$3_$4/Events/run_01/output_pythia8.hep # &> /dev/null
```

```

    ## Delphes execution
    ${DELPHESFOLDER}/${DELPHESEXE} $2/RunCards/${DELPHESCARD} $2/$3_$4/Events/run_01/output_delphes.root $2/$3_$4/Events/run_01/output_pythia8.hep

    ## ExRootAnalysis execution
    ${EXROOTFOLDER}/${EXROOTEXE} $2/$3_$4/Events/run_01/output_pythia8.hep $2/$3_$4/Events/run_01/output_pythia8.root

    ## Remove unnecessary files
    rm $2/$3_$4/Events/run_01/output_pythia8.hep

}

export -f sequ
export -f sequ2
## start PARAMETERS variable
PARAMETERS=""
## loop to execute sequence "sequ" for all the values from $INIRUN to $ENDRUN
for i in `seq ${INIRUN} ${ENDRUN}`; do  # {21,28}; do ## `seq ${INIRUN} ${ENDRUN}`; do
        ## execute sequ
        sequ
        ## concatenate the variable PARAMETERS with the current value of $i
        PARAMETERS="$PARAMETERS ${i}"
done

## execute gnuparallel. Use %% as the replacement string instead of {}.
parallel -0 -I %% --gnu "sequ2 ${MADGRAPHFOLDER} ${EVENTSFOLDER} ${NAMESUBFOLDER} %%" ::: $PARAMETERS
```

Appendix B

Analysis codes

B.1 Tagging algorithm

```
/*
------------------------------------------------
-------     Universidad de los Andes      -------
-------      Departamento de Fisica       -------
-------        Joven Investigador         -------
-------  Andres Felipe Garcia Albarracin  -------
-------    Juan Carlos Sanabria Arenas    -------
------------------------------------------------

This algorithm tags ISR jet in a certain sample.
It takes 2 N-dimensional histograms which contain
information about ISR and Non ISR Jets as input
and developes the ISR tagging in another sample.

The user can choose 3 of 8 variables for
developing the algorithm
1. PT
2. Abs(Eta) // Eta is a pair function
3. Delta Phi_MET
4. PT_ratio
5. Delta Eta_aver
```

```
6. Delta Phi_MET_others
7. Delta PT_others
8. Delta Eta_others

In order to choose them, the code should be run as:

./ISR tagging config_file.txt [N1 N2 N3]

where [config_file.txt] is a configuration file with
all parameters needed for the simulation.

N1 N2 and N3 are the index of the 3 variables.
If no parameter is passed as parameter, N1 N2 and N3
will be 0,1 and 2 by default.

*/


#include "ROOTFunctions.h"
#include "graphs_Funcs.h"
#include "functions.h"
#include "histoN.h"
#include "DelphesFunctions.h"

// Global Variables
const Double_t PI = TMath::Pi();

int main(int argc, char **argv){
   std::cout.precision(4);
   // Counting time
   Double_t initialTime = clock();

   // Folder variables
   string head_folder = "/home/af.garcia1214/
      PhenoMCsamples/Simulations/
      MG_pythia8_delphes_parallel/
      _Tops_Events_WI_Matching/";
   string current_folder = "_Tops_MG_1K_AG_WI_003/";

   string head_folder_binary = "/home/af.garcia1214/
```

```
      PhenoMCsamples/Results_Improved_Codes/
      matching_Results/_Tops_matchs_WI_Matching/";
   string matching_name = "ISR_jets_Tops_WI_003.bn";

   string head_folder_histos = "/home/af.garcia1214/
      PhenoMCsamples/Results_Improved_Codes/histo_folder
      /_Tops_histos_WI_Matching/";
   string head_folder_results = "/home/af.garcia1214/
      PhenoMCsamples/Results_Improved_Codes/
      resultsTagging/_Tops_histos_WI_Matching/";

   Bool_t ISR_OR_NOT = true;

   // Variables for analysis
   Double_t pt_cut = 0.0; // ISR jet pt cut
   Double_t Jet_cut = 2;  // Ptobability cut 'K'
   Bool_t do_pt_cut = false;
   Bool_t do_jet_cut = false;

   // Checking input parameters
   string config_file_name = "Debug/config_file.txt";
   // Reading the file as first parameter
   if (argc>1){
      config_file_name = argv[1];
   }
   else{
      cout << "It is necessary to type a configuration
         file as parameter. Execute as ./ISR tagging
         config_file.txt [N1 N2 N3]" << endl;
      return 1;
   }
   cout << "Reading input parameters" << endl;
   cout << "\tUsing as parameters' file: " <<
      config_file_name << endl;

   ifstream config_file (config_file_name);
   if (config_file.is_open()){
      cout << "\tReading file" << endl;
      string line;
      int number_line = 1;
```

```
    while (getline(config_file,line)){
        // Skipping commented lines
        if (line[0] == '!')
            continue;

        // Finding the position of the equal sign
        int pos_equal = -1;
        pos_equal = line.find('=');

        if (pos_equal == -1){
            cout << "\tLine " << number_line << " is incorrect" << endl;
            continue;
        }

        // Splitting the line according to the position of equal sign
        string var_name = line.substr(0,pos_equal);
        string var_value = line.substr(pos_equal+1);

        // Reading head folder
        if(var_name.compare("head_folder") == 0){
            head_folder = var_value;
            cout << "\tVariable head folder set as: " << head_folder << endl;
        }
        // Reading current folder
        else if (var_name.compare("current_folder") == 0){
            current_folder = var_value;
            cout << "\tVariable current folder set as: " << current_folder <<endl;
        }
        // Reading head folder binary
        else if (var_name.compare("head_folder_binary") == 0){
            head_folder_binary = var_value;
            cout << "\tVariable head folder binary set as: " << head_folder_binary << endl;
        }
```

```
      // Reading matching name
      else if (var_name.compare("matching_name") == 0){
         matching_name = var_value;
         cout << "\tVariable matching_name set as: " << matching_name << endl;
      }
      // Reading head folder histos
      else if (var_name.compare("head_folder_histos") == 0){
         head_folder_histos = var_value;
         cout << "\tVariable head folder histos set as: " << head_folder_histos << endl;
      }
      // Reading head folder results
      else if (var_name.compare("head_folder_results") == 0){
         head_folder_results = var_value;
         cout << "\tVariable head folder results set as: " << head_folder_results << endl;
      }
      // Reading pt_cut
      else if (var_name.compare("pt_cut") == 0){
         pt_cut = atof((Char_t *) var_value.c_str());
         do_pt_cut = true;
      }
      // Reading jet_cut
      else if (var_name.compare("Jet_cut") == 0){
         Jet_cut = atof((Char_t *) var_value.c_str());
         do_jet_cut = true;
      }
      // Reading ISR_OR_NOT
      else if (var_name.compare("ISR_OR_NOT") == 0){
         if (var_value.compare("1") == 0)
            ISR_OR_NOT = true;
         else
            ISR_OR_NOT = false;
      }

```

```
        number_line ++;
      }
    }
    else
    {
      cout << "ERROR: File " << config_file_name << "
        does not exist. Terminating program" << endl;
      return 0;
    }

    cout << "\n *** Running the tagging Algorithm *** \n"
       << endl;

    // Variables for initializing histograms
    Int_t dims = 3;

    /*
     * Read inputs and set variables for analysis
     */
    Int_t var_index[3] = {0,1,2}; // Index of the 3
      variables for analysis. By default 0, 1 and 2
    string variables[8] = {"PT","Abs(Eta)","Delta Phi_MET
      ","PT_ratio","Delta Eta_aver","Delta
      Phi_MET_others","Delta PT_leading","Delta
      Eta_leading"};
    Double_t var_values[8] =
      {0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0}; // Vector with
      the values of the 8 variables


    if (argc == 2) {
      cout << "\tRunning the algorithm with the default
        variables:" << endl;
    }
    else if (argc >= 5){
      cout << "\tRunning the algorithm with the
        variables:" << endl;
      for (Int_t ind = 0; ind < 3; ind ++){
        var_index[ind] = atoi(argv[ind+2]);
      }
```

```
    }
    else {
       cout << "\tError at calling this algorithm. Use as:" << endl;
       cout << "\t ./ISR_tagging config_file.txt,  ./ISR_tagging config_file.txt N1 N2 N3 or just ./ISR_tagging" << endl;
       cout << "\tRead the documentation at the beginning of the code for further information\n" << endl;
       return 1;
    }

    cout << "\tVar \t\t min_Value \t max_Value" << endl;
    for (Int_t ind = 0; ind < 3; ind ++){

       cout << "\t" << var_index[ind] << ". " << variables[var_index[ind]] << endl;
    }
    cout << endl;

    cout << "\tTransverse momentum of the ISR: " << pt_cut << endl;

    /*
     * Initializing the 3-dimensional histogram
     */
    // Defining the names of the files
    string combination = "______"; // Combination of variables
    for (Int_t ind = 0; ind < dims; ind ++){
       combination[(ind*2)+1] = (Char_t) (0x30 + var_index[ind]); // Int to char
    }

    string info_ISR_name_str = head_folder_histos + "info_histo_ISR" + combination + ".txt";
    Char_t *info_ISR_name = (Char_t *) info_ISR_name_str.c_str();

```

```
    string array_ISR_name_str = head_folder_histos + "array_histo_ISR" + combination + ".bn";
    Char_t *array_ISR_name = (Char_t *) array_ISR_name_str.c_str();

    string info_Non_ISR_name_str = head_folder_histos + "info_histo_Non_ISR" + combination + ".txt";
    Char_t *info_Non_ISR_name = (Char_t *) info_Non_ISR_name_str.c_str();

    string array_Non_ISR_name_str = head_folder_histos + "array_histo_Non_ISR" + combination + ".bn";
    Char_t *array_Non_ISR_name = (Char_t *) array_Non_ISR_name_str.c_str();

    histoN* histoISR = new histoN(info_ISR_name, array_ISR_name);
    histoN* histoNonISR = new histoN(info_Non_ISR_name, array_Non_ISR_name);

    cout << "\tEntradas ISR: " << histoISR->getEntries() << endl;
    cout << "\tEntradas FSR: " << histoNonISR->getEntries() << endl;

    // Input variables of each histogram
    Double_t values[3] = {0.0,0.0,0.0};

    /*
     * MET histograms
     */
    TH1 *h_MET = new TH1F("Missing ET","All events",300,0,1000);
    Char_t *name_histo_MET;
    name_histo_MET = (Char_t*) malloc(sizeof(char)*512);
    strcpy(name_histo_MET,"ISR jet PT > ");
    Char_t pt_str[] = "   ";
    pt_str[0] = 0x30 + int(pt_cut/100)%10;
    pt_str[1] = 0x30 + int(pt_cut/10)%10;
    pt_str[2] = 0x30 + int(pt_cut)%10;
```

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

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

    cout << "\tJet cut, k = " << Jet_cut << endl;

    // Arrays with the number of tags, Misstags and events rejected
    // Probability cut
    Double_t Prob_cut = 0;
    Double_t k_min = 1.2; // Minimum probability cut = k_min/num_jets
    Double_t k_max = 3.0; // Maximum probability cut = k_max/num_jets
    Int_t k_bins = 100; // Number of values of k between k_min and k_max
    Double_t k_step = (Double_t) (k_max-k_min)/k_bins;
    Double_t k_values[k_bins];
    for(Int_t ind = 0; ind < k_bins; ind ++){
       k_values[ind] = k_min + k_step*ind;
    }

    // Tagging results
    Int_t Num_Tags = 0;
    Int_t Num_MissTags = 0;
    Int_t Num_Rejected = 0;
```

```

   Double_t Num_Tags_array[k_bins];
   Double_t Num_MissTags_array[k_bins];
   Double_t Num_Rejected_array[k_bins];
   Double_t Num_Total_Jets[k_bins];

   Double_t Num_Tags_array_hpt[k_bins];
   Double_t Num_MissTags_array_hpt[k_bins];
// Double_t Num_Rejected_array_hpt[k_bins];
   Double_t Num_Total_Jets_hpt[k_bins];


   for (Int_t ind = 0; ind < k_bins; ind ++){
      Num_Tags_array[ind] = 0;
      Num_MissTags_array[ind] = 0;
      Num_Rejected_array[ind] = 0;
      Num_Total_Jets[ind] = 0;
      Num_Tags_array_hpt[ind] = 0;
      Num_MissTags_array_hpt[ind] = 0;
//    Num_Rejected_array_hpt[ind] = 0;
      Num_Total_Jets_hpt[ind] = 0;
   }

   // Variables of the ISR tagging algorithm
   Double_t H_ISR, H_Non_ISR, alpha;
   Double_t prob_max = 0;
   Double_t probISR = 0;
   Double_t k_ISR = 0;
   Double_t k_ISR_pos = 0; // Position of the ISR in the
       vector
   Int_t ISR_tag_index = -1;

   // Cycle over several runs . iRun corresponds to the
      seed of the current run
   for(int iRun = 261; iRun < 270; iRun ++){
      // Create chains of root trees
      TChain chain_Delphes("Delphes");

      Char_t unidad = 0x30 + iRun%10;
      Char_t decena = 0x30 + int(iRun/10)%10;
```

```
    Char_t centena = 0x30 + int(iRun/100)%10;

    current_folder[current_folder.size()-4] = centena;
    current_folder[current_folder.size()-3] = decena;
    current_folder[current_folder.size()-2] = unidad;
    matching_name[matching_name.size()-6] = centena;
    matching_name[matching_name.size()-5] = decena;
    matching_name[matching_name.size()-4] = unidad;

    string file_delphes_str = head_folder + current_folder + "Events/run_01/output_delphes.root";

    Char_t *file_delphes = (Char_t *) file_delphes_str.c_str();

    cout << "\n\tStudying run: "<<centena<<decena<<unidad<<endl;
    cout << "\tReading the file: \n\tDelphes: " << file_delphes << endl;

    chain_Delphes.Add(file_delphes);
    // Objects of class ExRootTreeReader for reading the information
    ExRootTreeReader *treeReader_Delphes = new ExRootTreeReader(&chain_Delphes);

    Long64_t numberOfEntries = treeReader_Delphes->GetEntries();

    // Get pointers to branches used in this analysis
    TClonesArray *branchJet = treeReader_Delphes->UseBranch("Jet");
    TClonesArray *branchMissingET = treeReader_Delphes->UseBranch("MissingET");

    cout << "\tNumber of Entries Delphes = " << numberOfEntries << endl;
    cout << endl;

```

```
    // particles, jets and vectors
    MissingET *METpointer;
    TLorentzVector *vect_currentJet = new TLorentzVector;
    TLorentzVector *vect_auxJet = new TLorentzVector;
    TLorentzVector *vect_leading = new TLorentzVector;
    Jet *currentJet = new Jet;
    Jet *auxJet = new Jet;

    // Temporary variables
    Double_t MET = 0.0; // Missing transverse energy
    Double_t delta_phi = 0.0; // difference between the phi angle of MET and the jet
    Double_t transverse_mass = 0.0; // Transverse mass
    Double_t delta_PT_jet = 0.0; // |PT-<PT>|
    Double_t PT_sum = 0.0; // sum(PT)
    Double_t PT_aver = 0.0; // <PT>
    Double_t Delta_eta_aver = 0.0; // sum_i|eta-eta_i|/(Nj-1)
    Double_t Delta_phi_sum = 0.0; // sum delta_phi
    Double_t Delta_phi_other_jets = 0.0; // Average of delta phi of other jets
    Double_t PT_ratio = 0.0; // PT/PT_others
    Double_t Delta_PT_leading = 0.0; // PT - PT_leading
    Double_t Delta_Eta_leading = 0.0; // |Eta - Eta_leading|

    // Jet with greatest PT
    Double_t PT_max = 0;
    Int_t posLeadingPT = -1;
    Int_t ISR_greatest_PT = 0;
    Double_t MT_leading_jet = 0.0; // Transverse mass

    /*
     * Some variables used through the code
     */
    Int_t ISR_jets[numberOfEntries];
    Int_t NumJets = 0;

```

```
      string fileName_str = head_folder_binary + matching_name;

      Char_t * fileName = (Char_t *) fileName_str.c_str();

      if (ISR_OR_NOT == true){
         ifstream ifs(fileName,ios::in | ios::binary);

         for (Int_t j = 0; j<numberOfEntries; j++){
            ifs.read((Char_t *) (ISR_jets+j),sizeof(Int_t));
         }
         ifs.close();
      }
      else{
         for (Int_t j = 0; j<numberOfEntries; j++){
            ISR_jets[j] = -2; // There is not ISR jet but also there is not matching
         }
      }

      /*
       * Main cycle of the program
       */
      numberOfEntries = 100000;
      for (Int_t entry = 0; entry < numberOfEntries; ++entry){
         // Progress
         if(numberOfEntries>10 && (entry%((int)numberOfEntries/10))==0.0){
            cout<<"\tprogress = "<<(entry*100/numberOfEntries)<<"%\t";
            cout<< "Time :"<< (clock()-initialTime)/double_t(CLOCKS_PER_SEC)<<"s"<<endl;
         }

         // Load selected branches with data from specified event
         treeReader_Delphes->ReadEntry(entry);
```

```

        // MET
        METpointer = (MissingET*) branchMissingET->At
           (0);
        MET = METpointer->MET;

        NumJets=branchJet->GetEntries();

        // checking the ISR
        if (NumJets < 3 || ISR_jets[entry] == -1)
           continue;

        h_MET->Fill(MET);

        if (ISR_jets[entry] >= NumJets){
           cout << "Error en el matching" << endl;
           return 1;
        }

        // 3 PT ratio
        PT_aver = 0.0;
        PT_sum = 0.0;
        PT_ratio = 0.0;

        // 4 Delta Eta aver
        Delta_eta_aver = 0.0;

        // 5 Delta Phi others
        Delta_phi_sum = 0.0;
        Delta_phi_other_jets = 0.0;

        // 6 Delta PT leading
        PT_max = 0.0;
        Delta_PT_leading = 0.0;
        delta_PT_jet = 0.0; // If needed

        // 7 Delta Eta leading
        Delta_Eta_leading = 0.0;

        // Reset Var_values (Not necessary)
```

```
        for(Int_t ind = 0; ind < 8; ind++){
           var_values[ind] = 0.0;
           if (ind < dims) values[ind] = 0.0;
        }

        // Preliminary for. It is used to calculate
           PT_aver and Delta_phi_sum
        for (Int_t iJet = 0; iJet<NumJets; iJet++){
           currentJet = (Jet*) branchJet->At(iJet);
           vect_currentJet->SetPtEtaPhiM(currentJet->PT
              ,currentJet->Eta,currentJet->Phi,
              currentJet->Mass);
           PT_sum += vect_currentJet->Pt();
           delta_phi = deltaAng(vect_currentJet->Phi(),
               METpointer->Phi);
           Delta_phi_sum += delta_phi;
           // PT Leading jet
           if(PT_max < vect_currentJet->Pt()){
              PT_max = vect_currentJet->Pt();
              posLeadingPT = iJet;
           }
        }

        //PT_aver
        PT_aver = PT_sum/NumJets;

        // Leading PT
        currentJet = (Jet*) branchJet->At(posLeadingPT)
           ;
        vect_leading->SetPtEtaPhiM(currentJet->PT,
           currentJet->Eta,currentJet->Phi,currentJet->
           Mass);

        // The best ISR candidate
        TLorentzVector *vect_optimum = new
           TLorentzVector;

        // Reset variables
        probISR = 0.0;
        k_ISR = 0.0;
```

```
      prob_max = 0;
      ISR_tag_index = -1;

      for (Int_t iJet = 0; iJet<NumJets; iJet++){
        currentJet = (Jet*) branchJet->At(iJet);
        vect_currentJet->SetPtEtaPhiM(currentJet->PT,currentJet->Eta,currentJet->Phi,currentJet->Mass);

        // 2 Delta Phi MET
        delta_phi = deltaAng(vect_currentJet->Phi(), METpointer->Phi);

        // PT ratio
        PT_ratio = vect_currentJet->Pt()*(NumJets-1)/(PT_sum-vect_currentJet->Pt());

        // 4 Delta Eta Aver
        Delta_eta_aver = 0.0;
        // For cycle used to calculate Delta_eta_aver
        for(Int_t iJet2 = 0; iJet2<NumJets; iJet2++){
          auxJet = (Jet*) branchJet->At(iJet2);
          vect_auxJet->SetPtEtaPhiM(auxJet->PT,auxJet->Eta,auxJet->Phi,auxJet->Mass);
          if (iJet2 != iJet) Delta_eta_aver += TMath::Abs(vect_auxJet->Eta()-vect_currentJet->Eta());
        }
        Delta_eta_aver = Delta_eta_aver/(NumJets-1);

        // 5 Delta Phi MET Others
        Delta_phi_other_jets = (Delta_phi_sum-delta_phi)/(NumJets-1);

        // 6 Delta PT leading
        Delta_PT_leading = vect_leading->Pt()-vect_currentJet->Pt();

```

```
            // 7 Delta Eta leading
            Delta_Eta_leading = TMath::Abs(vect_currentJet->Eta()-vect_leading->Eta());

            // Other variables
            delta_PT_jet = TMath::Abs(vect_currentJet->Pt()-PT_aver);
            transverse_mass = sqrt(2*vect_currentJet->Pt()*MET*(1-cos(delta_phi)));

            // Filling the array with the variables' values
            var_values[0] = vect_currentJet->Pt();
            var_values[1] = TMath::Abs(vect_currentJet->Eta());
            var_values[2] = delta_phi;
            var_values[3] = PT_ratio;
            var_values[4] = Delta_eta_aver;
            var_values[5] = Delta_phi_other_jets;
            var_values[6] = Delta_PT_leading;
            var_values[7] = Delta_Eta_leading;

            for (Int_t ind = 0; ind < dims; ind++){
               int pos = *(var_index+ind);
               values[ind] = *(var_values+pos);
            }

            // Comparing with histos
            H_ISR = histoISR->getProbVal(values);
            H_Non_ISR = histoNonISR->getProbVal(values);

            if (H_ISR >3e-7 || H_Non_ISR>3e-7){
               alpha = NumJets/(H_Non_ISR*(NumJets-1)+H_ISR);
               probISR = alpha*H_ISR/NumJets;

               if(probISR > (1.0 + 1.0e-10)){
                  cout << setprecision(20) << "\n\t *** ERROR: La probabilidad no puede ser
```

```
          mayor a 1 ***" << endl;
            return 1;
          }

          if (probISR >= prob_max){
            prob_max = probISR;
            vect_optimum->SetPtEtaPhiM(
              vect_currentJet->Pt(),
              vect_currentJet->Eta(),
              vect_currentJet->Phi(),
              vect_currentJet->M());
            ISR_tag_index = iJet;
          }
        }
      }

      k_ISR = prob_max*NumJets;

      // Check the tagging results
      k_ISR_pos = findPosition(k_min,k_max,k_bins,
        k_ISR);

      if(k_ISR == 0.0) k_ISR_pos = -1;

      if (ISR_jets[entry] != -1 && ISR_OR_NOT == true
        ){
        // A comparison can be handled
        for (Int_t ind = 0; ind < k_ISR_pos + 1; ind
          ++){
          if (ISR_tag_index == ISR_jets[entry])
            Num_Tags_array[ind]++;
          else
            Num_MissTags_array[ind]++;
        }
        for (Int_t ind = k_ISR_pos+1; ind < k_bins;
          ind++){
          Num_Rejected_array[ind]++;
        }
      }
      else if (ISR_jets[entry] == -2 && ISR_OR_NOT ==
```

```
                false){
            for (Int_t ind = 0; ind < k_ISR_pos + 1; ind++){
               Num_MissTags_array[ind]++;
            }
            for (Int_t ind = k_ISR_pos+1; ind < k_bins; ind++){
               Num_Rejected_array[ind]++;
            }
         }

         if (ISR_tag_index != -1 && vect_optimum->Pt()>pt_cut && ISR_OR_NOT == true){  // != S means bb or WI
            for (Int_t ind = 0; ind < k_ISR_pos + 1; ind++){
               if (ISR_tag_index == ISR_jets[entry])
                  Num_Tags_array_hpt[ind]++;
               else
                  Num_MissTags_array_hpt[ind]++;
            }
            for (Int_t ind = k_ISR_pos+1; ind < k_bins; ind++){
//             Num_Rejected_array_hpt[ind]++;      // Under a certain k_cut, there are not rejected events
            }
         }

         Prob_cut = Jet_cut/NumJets;
         if(prob_max >= Prob_cut){
            if (ISR_tag_index == ISR_jets[entry] && ISR_OR_NOT == true)  // != S means bb or WI
               Num_Tags++;
            else
               Num_MissTags++;

            // Cheching MET boosting
            if(vect_optimum->Pt()>pt_cut){
               h_MET_hpt1->Fill(MET);
```

```
            }
         }
         else
            Num_Rejected++;

      }

      cout<<"\tprogress = 100%\t";
      cout<<"Time :"<< (clock()-initialTime)/double_t(
         CLOCKS_PER_SEC)<<"s"<<endl;

   } // End run's for cicle

   /*
    * Tagging results
    */

   Int_t Num_Studied = Num_Tags + Num_MissTags +
      Num_Rejected;
   cout << "\nOverall tagging results" << endl;
   cout << "\tNumber of compared events (between the
      matching and tagging algorithms) : " <<
      Num_Studied << endl;
   cout << "\tPer. Tags: \t" << ((Double_t)Num_Tags/
      Num_Studied)*100 << "%" << endl;
   cout << "\tPer. MissTags: \t" << ((Double_t)
      Num_MissTags/Num_Studied)*100 << "%" << endl;
   cout << "\tPer. Rejected: \t" << ((Double_t)
      Num_Rejected/Num_Studied)*100 << "%" << endl;

   // Calculating percentages
   for (Int_t ind=0; ind < k_bins; ind++){
      Num_Total_Jets[ind] = Num_Tags_array[ind] +
         Num_MissTags_array[ind] + Num_Rejected_array[
         ind];
      Num_Tags_array[ind] = Num_Tags_array[ind]/
         Num_Total_Jets[ind];
      Num_MissTags_array[ind] = Num_MissTags_array[ind]/
         Num_Total_Jets[ind];
      Num_Rejected_array[ind] = Num_Rejected_array[ind]/
```

```
            Num_Total_Jets[ind];
        Num_Total_Jets_hpt[ind] = Num_Tags_array_hpt[ind] + Num_MissTags_array_hpt[ind]; // + Num_Rejected_array_hpt[ind];
        Num_Tags_array_hpt[ind] = Num_Tags_array_hpt[ind]/Num_Total_Jets_hpt[ind];
        Num_MissTags_array_hpt[ind] = Num_MissTags_array_hpt[ind]/Num_Total_Jets_hpt[ind];
//      Num_Rejected_array_hpt[ind] = Num_Rejected_array_hpt[ind]/Num_Total_Jets_hpt[ind];
    }

    /*
     * Writing results
     */
    Bool_t archivoExiste = false;

    Char_t outNamept[] = "Percenta_hpt-100";
    outNamept[13] = 0x30 + int(pt_cut/100)%10;
    outNamept[14] = 0x30 + int(pt_cut/10)%10;
    outNamept[15] = 0x30 + int(pt_cut)%10;

    string outFileTotal_str = head_folder_results + "Overall_percs" + combination + ".txt";
    Char_t *outFileTotal = (Char_t *) outFileTotal_str.c_str();

    string outFileTotalpt_str = head_folder_results + outNamept + combination + ".txt";
    Char_t *outFileTotalpt = (Char_t *) outFileTotalpt_str.c_str();

    ifstream my_file(outFileTotal);
    if(my_file.good()){
        archivoExiste = true;
    }
    my_file.close();

    ofstream ofs_over(outFileTotal,ios::out);
```

```
    if(!archivoExiste){
       // If file already exists
    }

    ofs_over << "# Number of Tags, Misstags and Rejected
       as a function of k" << endl;
    ofs_over << "# Number of Events " << Num_Total_Jets
       [0] << endl;
    ofs_over << "# k_cut \t Tags \t MissTags \t Rejected
       \t Total_Events " << endl;


    for (Int_t ind = 0; ind < k_bins; ind ++){
       ofs_over << setiosflags(ios::fixed) <<
          setprecision(6) << setw(6) << k_values[ind]
             << "\t" << Num_Tags_array[ind] << "\t" <<
                Num_MissTags_array[ind] << "\t" <<
                Num_Rejected_array[ind]
             << "\t" << setprecision(0) << Num_Total_Jets
                [ind] << endl;
    }

    if (do_pt_cut){
        ofstream ofs_pt(outFileTotalpt,ios::out);
       ofs_pt << "# Number of Tags, Misstags and Rejected
           as a function of k. The ISR has pt > " <<
          pt_cut << endl;
       ofs_pt << "# Number of Events " <<
          Num_Total_Jets_hpt[0] << endl;
       ofs_pt << "# k_cut \t Tags \t MissTags \t
          Total_Events " << endl;
       for (Int_t ind = 0; ind < k_bins; ind ++){
          ofs_pt << setiosflags(ios::fixed) <<
             setprecision(6) << setw(6) << k_values[ind]
                << "\t" << Num_Tags_array_hpt[ind] << "\t
                   " << Num_MissTags_array_hpt[ind] // <<
                    "\t" << Num_Rejected_array_hpt[ind]
                << "\t" << setprecision(0) <<
                   Num_Total_Jets_hpt[ind] << endl;
        }
```

```
        ofs_pt.close();
    }

    if (do_jet_cut){
        Char_t outNameMET[] = "AbsValue_MET_pt_000_k_2.0";
        outNameMET[16] = pt_str[0];
        outNameMET[17] = pt_str[1];
        outNameMET[18] = pt_str[2];
        outNameMET[22] = k_str[0];
        outNameMET[23] = k_str[1];
        outNameMET[24] = k_str[2];

        string outFileMET_str = head_folder_results +
            outNameMET + combination;
        Char_t *outFileMET = (Char_t *) outFileMET_str.
            c_str();

        Char_t *outFilehist;
        outFilehist = (Char_t*) malloc(sizeof(char)*512);
        strcpy(outFilehist,outFileMET);
        strcat(outFilehist,".root");

        TFile* hfile = new TFile("histos.root", "RECREATE"
            );
        TCanvas *C = new TCanvas(outFileMET,"MET in a
            sample with high PT ISR jets",1280,720);
        Present(h_MET,h_MET_hpt1,C,2,"MET [GeV]","Num.
            Jets / Total");
        C->Write();
        C->Close();
        hfile->Close();

    }

    ofs_over.close();

    cout<<"\nEND :)"<<endl;

    return 0;
}
```

B.2 Matching algorithm

```
/*
---------------------------------------------------
-------     Universidad de los Andes      -------
-------      Departamento de Fisica       -------
-------        Joven Investigador         -------
-------  Andres Felipe Garcia Albarracin  -------
-------    Juan Carlos Sanabria Arenas    -------
---------------------------------------------------

This algorithm looks for the ISR parton into the
pythia8 simulation file and then finds the
corresponding ISR jet

It also stores in a binary file the matching
results

To run, type

./ISR_matching_improved [config.txt] [000]

where [config.txt] is the configuration file and
[000] is the seed of the simulation under analysis
*/

#include <iostream>
#include "ROOTFunctions.h"
#include "graphs_Funcs.h"
#include "functions.h"
#include "DelphesFunctions.h"

using namespace std;
// Global Variables
const Double_t PI = TMath::Pi();

int main(int argc, char **argv){

   std::cout.precision(4);
   // Counting time
```

```
    Double_t initialTime = clock();

    // Folder variables
    string head_folder = "/home/af.garcia1214/
       PhenoMCsamples/Simulations/
       MG_pythia8_delphes_parallel/
       _Tops_Events_WI_Matching/";
    string current_folder = "_Tops_MG_1K_AG_WI_003/";

    string head_folder_results = "/home/af.garcia1214/
       PhenoMCsamples/Results_Improved_Codes/
       matching_Results/_Tops_matchs_WI_Matching/";
    string matching_name = "ISR_jets_Tops_WI_003.bn";

    // Checking input parameters
    string config_file_name = "Debug/config_file.txt";
    // Reading the file as first parameter
    if (argc>1){
       config_file_name = argv[1];
    }
    else{
       cout << "It is necessary to type a configuration
          file as parameter. Execute as ./ISR_matching
          config_file.txt [000]" << endl;
       return 1;
    }
    cout << "Reading input parameters" << endl;
    cout << "\tUsing as parameters' file: " <<
       config_file_name << endl;

    ifstream config_file (config_file_name);
    if (config_file.is_open()){
       cout << "\tReading file" << endl;
       string line;
       int number_line = 1;
       while (getline(config_file,line)){
          // Skipping commented lines
          if (line[0] == '!')
             continue;

```

```
      // Finding the position of the equal sign
      int pos_equal = -1;
      pos_equal = line.find('=');

      if (pos_equal == -1){
         cout << "\tLine " << number_line << " is incorrect" << endl;
         continue;
      }

      // Splitting the line according to the position of equal sign
      string var_name = line.substr(0,pos_equal);
      string var_value = line.substr(pos_equal+1);

      // Reading head folder
      if(var_name.compare("head_folder") == 0){
         head_folder = var_value;
         cout << "\tVariable head folder set as: " << head_folder << endl;
      }
      // Reading current folder
      else if (var_name.compare("current_folder") == 0){
         current_folder = var_value;
         cout << "\tVariable current folder set as: " << current_folder <<endl;
      }
      // Reading head folder results
      else if (var_name.compare("head_folder_results") == 0){
         head_folder_results = var_value;
         cout << "\tVariable head folder results set as: " << head_folder_results << endl;
      }
      // Reading matching name
      else if (var_name.compare("matching_name") == 0){
         matching_name = var_value;
         cout << "\tVariable matching_name set as: "
```

```
                << matching_name << endl;
         }

         number_line ++;
      }
   }
   else
   {
      cout << "ERROR: File " << config_file_name << " does not exist. Terminating program" << endl;
      return 0;
   }

   // Reading the seed of the simulation. This parameter is optional and is the second of argv

   Char_t unidad = '3'; Char_t decena = '0'; Char_t centena = '0';
   if (argc > 2){
      cout << "\tRemember: The number of the simulation should consist of 3 digits" << endl;
      centena = argv[2][0];
      decena = argv[2][1];
      unidad = argv[2][2];
      current_folder[current_folder.size()-4] = centena;
      current_folder[current_folder.size()-3] = decena;
      current_folder[current_folder.size()-2] = unidad;
      matching_name[matching_name.size()-6] = centena;
      matching_name[matching_name.size()-5] = decena;
      matching_name[matching_name.size()-4] = unidad;
   }

   cout << "\tThe seed of the simulation is: " << centena << decena << unidad << endl;

   // Full path name of pythia and Delphes simulations
   string file_pythia_str = head_folder + current_folder + "Events/run_01/output_pythia8.root";
   Char_t *file_pythia = (Char_t *) file_pythia_str.c_str(); //Pass string to char_t *
```

```
    string file_delphes_str = head_folder +
        current_folder + "Events/run_01/output_delphes.
        root";
    Char_t *file_delphes = (Char_t *) file_delphes_str.
        c_str();

    if (argc > 2){
        cout << "\n\tReading the files: \n\tPythia8: " <<
            file_pythia << "\n\tDelphes: " << file_delphes
            << endl;
    }
    else
        cout << "\n\tReading the default files: \n\
            tPythia8: " << file_pythia << "\n\tDelphes: "
            << file_delphes << endl;


    // Loading simulations of Pythia and Delphes
    cout << "\nLoading simulations of Pythia and Delphes"
         << endl;
    // Create chains of root trees
    TChain chain_Pythia("STDHEP");
    TChain chain_Delphes("Delphes");

    chain_Pythia.Add(file_pythia);
    chain_Delphes.Add(file_delphes);

    // Objects of class ExRootTreeReader for reading the
        information
    ExRootTreeReader *treeReader_Pythia = new
        ExRootTreeReader(&chain_Pythia);
    ExRootTreeReader *treeReader_Delphes = new
        ExRootTreeReader(&chain_Delphes);

    Long64_t numberOfEntries = treeReader_Pythia->
        GetEntries();
    Long64_t numberOfEntries_Delphes = treeReader_Delphes
        ->GetEntries();

```

```
    // Get pointers to branches used in this analysis
    TClonesArray *branchParticlePythia =
        treeReader_Pythia->UseBranch("GenParticle");
    TClonesArray *branchJet = treeReader_Delphes->
        UseBranch("Jet");
    TClonesArray *branchMissingET = treeReader_Delphes->
        UseBranch("MissingET");

    cout << endl;
    cout << "\tNumber of Entries Pythia = " <<
        numberOfEntries << endl;
    cout << "\tNumber of Entries Delphes = " <<
        numberOfEntries_Delphes << endl;
    cout << endl;

    // particles, jets and vectors
    TRootGenParticle *particle_pythia;
    TRootGenParticle *ISR_particle;
    MissingET *METpointer;
    TLorentzVector *vect_ISR_particle = new
        TLorentzVector;

    // Temporary variables
    Bool_t ISR_parton_found = false; // true if the
        initial ISR_parton (with status 43) was found
    Int_t pos_ISR = -1; // position of the ISR_parton
        into the branchParticlePythia array
    Double_t MET = 0.0; // Missing transverse energy

    /*
    * Some variables used through the code
    */
    Int_t NumEvents1ISRJet = 0;      // Number of events
        where the number of ISR jets is 1
    Int_t NumMatches = 0;            // Number of matches
    Int_t NumJets = 0;
    Int_t ISR_match_index = -1;
    Double_t Cut_matching_DPT = 50.0;
    Double_t Cut_matching_DEta = 0.4;
    Double_t Cut_matching_DPhi = 0.4;
```

```
   Double_t Cut_matching_Dy = 0.4;
   Int_t ISR_jets[numberOfEntries];

   /*
    * Main cycle of the program
    */
   cout << "Running the matching algorithm" << endl;
   numberOfEntries = 100000;
   for (Int_t entry = 0; entry < numberOfEntries; ++entry){
      // Progress
      if(numberOfEntries>10 && (entry%((int)numberOfEntries/10))==0.0){
         cout<<"\tprogress = "<<(entry*100/numberOfEntries)<<"%\t";
         cout<< "Time :"<< (clock()-initialTime)/double_t(CLOCKS_PER_SEC)<<"s"<<endl;
      }

      // Load selected branches with data from specified event
      treeReader_Pythia->ReadEntry(entry);
      treeReader_Delphes->ReadEntry(entry);

      // By default, the ISR jet was not matched
      ISR_jets[entry] = -1;

      // MET
      METpointer = (MissingET*) branchMissingET->At(0);
      MET = METpointer->MET;

      // Finding the ISR parton
      ISR_parton_found = false;
      pos_ISR = -1;
      for(Int_t iPart = 0; iPart < branchParticlePythia->GetEntries(); iPart++){
         particle_pythia = (TRootGenParticle*) branchParticlePythia->At(iPart);
         if( abs(particle_pythia->Status) == 43){
            pos_ISR = iPart;
```

```
            ISR_particle = (TRootGenParticle*) branchParticlePythia->At(pos_ISR);
            ISR_parton_found = true;
//          cout << pos_ISR << "\t\t" << ISR_particle->Status << "\t\t" << ISR_particle->PID
//             << "\t\t" << ISR_particle->M1 << "\t\t" << ISR_particle->M2
//             << "\t\t" << ISR_particle->D1 << "\t\t" << ISR_particle->D2 << endl;
         }
      }

      // If there is not ISR parton, pass to the next event
      if (ISR_parton_found == false){
         continue;
      }

      // Finding the last copy of the ISR_parton
      ISR_parton_found = false;
      while (!ISR_parton_found){
         if (ISR_particle->D1 != ISR_particle->D2)
            ISR_parton_found = true;
         else{
            pos_ISR = ISR_particle->D1;
            if(pos_ISR != -1) // To avoid an incoherent event
               ISR_particle = (TRootGenParticle*) branchParticlePythia->At(pos_ISR);
            else
               ISR_parton_found = true; // To end up the while loop
         }
      }

      if (pos_ISR == -1) // End the incoherent events
         continue;

      // Matching algorithm
      // Matching between the ISR parton and a jet
```

```
    // Auxiliary variables
    Double_t R_min = 2.0;
    Double_t r; // Current deltaR
    ISR_match_index = -1;
    Int_t mixJets = 0;
    TLorentzVector *vect_Jet1 = new TLorentzVector();         // Four-momentum of the jet of the 1st for
    TLorentzVector *vect_Jetc = new TLorentzVector();         // Four-momentum of the jet of the 2nd, 3rd ... for
    TLorentzVector *vect_Jets = new TLorentzVector();         // Four-momentum of the sum of jets
    TLorentzVector *vect_Jeto = new TLorentzVector();         // Four-momentum of the optimal combination
    Jet *jet = new Jet();
    Jet *jet2 = new Jet();

    NumJets = branchJet->GetEntries();
    vect_ISR_particle->SetPtEtaPhiE(ISR_particle->PT, ISR_particle->Eta,ISR_particle->Phi, ISR_particle->E);

    if (NumJets < 3) // Minimun 3 jets per event
       continue;

    // Finding the jet with the minimum R to the ISR parton
    for ( Int_t j = 0; j < NumJets; j++ ) {      // Loop over jets finding the one with the minimum R
       jet = (Jet*) branchJet->At(j);
       vect_Jet1->SetPtEtaPhiM(jet->PT, jet->Eta, jet->Phi, jet->Mass);
       r = vect_ISR_particle->DeltaR(*vect_Jet1);
       if ( r < R_min ) {
            R_min = r;
            ISR_match_index = j;
            mixJets = 1;
```

```
                *vect_Jeto = *vect_Jet1;
            }
            // Checking if there are two jets mixed
            for ( Int_t k = j+1; k<NumJets; k++){
                jet2 = (Jet*) branchJet->At(k);
                vect_Jetc->SetPtEtaPhiM(jet2->PT, jet2->Eta, jet2->Phi, jet2->Mass);
                *vect_Jets = *vect_Jet1 + *vect_Jetc;
                r = vect_ISR_particle->DeltaR(*vect_Jets);
                if ( r < R_min ) {
                    R_min = r;
                    ISR_match_index = j;
                    mixJets = 2;
                    *vect_Jeto = *vect_Jets;
                }
                    // Checking if there are three jets mixed
        for (Int_t m = k+1; m<NumJets; m++){
            jet2 = (Jet*) branchJet->At(m);
                            vect_Jetc->SetPtEtaPhiM(jet2->PT, jet2->Eta, jet2->Phi, jet2->Mass);
                            *vect_Jets = *vect_Jets + *vect_Jetc;
                            r = vect_ISR_particle->DeltaR(*vect_Jets);
                            if ( r < R_min ) {
                                R_min = r;
                                ISR_match_index = j;
                                mixJets = 3;
                                *vect_Jeto = *vect_Jets;
                    }
                            // Checking if there are four jets mixed
                            for (Int_t n = m+1; n<NumJets; n++){
```

```
                                        jet2 = (Jet*) branchJet->At(n);
                                        vect_Jetc->SetPtEtaPhiM(jet2->PT, jet2->Eta, jet2->Phi, jet2->Mass);
                                        *vect_Jets = *vect_Jets + *vect_Jetc;
                                        r = vect_ISR_particle->DeltaR(*vect_Jets);
                                        if ( r < R_min ) {
                                                R_min = r;
                                                ISR_match_index = j;
                                                mixJets = 4;
                                                *vect_Jeto = *vect_Jets;
                                        }
                                }
                        }
                }
        }       // Loop over jets finding the one with the minimum R

        if( (mixJets == 1) && (ISR_match_index >= 0) && (ISR_match_index < NumJets) ) {
                NumEvents1ISRJet++;
                Double_t Delta_PT = TMath::Abs(vect_Jeto
```

```
                        ->Pt() - vect_ISR_particle->Pt());
                    Double_t Delta_Eta = TMath::Abs(
                        vect_Jeto->Eta() - vect_ISR_particle
                        ->Eta());
                    Double_t Delta_Phi = vect_Jeto->DeltaPhi
                        (*vect_ISR_particle);
                    Double_t Delta_y = TMath::Abs(vect_Jeto
                        ->Rapidity() - vect_ISR_particle->
                        Rapidity());

                    if ( (Delta_PT > Cut_matching_DPT) || (
                        Delta_Eta > Cut_matching_DEta) || (
                        Delta_Phi > Cut_matching_DPhi ) || (
                        Delta_y > Cut_matching_Dy) ) {
                            ISR_jets[entry] = -1;
                    }
                    else {
                            NumMatches++;
                            ISR_jets[entry] =
                                ISR_match_index;
                    }
            }

            if (ISR_jets[entry] >= NumJets){
             cout << "Error en el matching. Terminating
                program" << endl;
             return 1;
            }
    }

    cout<<"\tprogress = 100%\t";
    cout<< "Time :"<< (clock()-initialTime)/double_t(
        CLOCKS_PER_SEC)<<"s"<<endl;

    /*
     * Writing results
     */
    cout << "\nWriting files" << endl;
    string fileName_str = head_folder_results +
        matching_name;
```

```

   Char_t * fileName = (Char_t *) fileName_str.c_str();

   if (argc > 2)
      cout << "\t Writing the binary file...:" << fileName << endl;
   else
      cout<<"\t Writing the default binary file...:" << fileName << endl;

   ofstream ofs(fileName,ios::out|ios::binary);
   if (!ofs){
      cout << "Problemas al escribir el archivo" << endl;
       }
   else{
      for(Int_t j = 0; j<numberOfEntries; j++){
         ofs.write((Char_t *) (ISR_jets+j),sizeof(Int_t));
      }
   }
   ofs.close();

   cout << "\nSome overal results: " << endl;
   cout << "\tNumber of events with a single ISR jet = " << NumEvents1ISRJet <<endl;
   cout << "\tNumber of matches = " << NumMatches << endl;
   cout << endl;

   return 0;
}
```

B.3 ISR jet analysis

```
/*
------------------------------------------------
-------     Universidad de los Andes      -------
-------      Departamento de Fisica       -------
```

```
-------          Joven Investigador          -------
-------  Andres Felipe Garcia Albarracin  -------
-------     Juan Carlos Sanabria Arenas     -------
-------------------------------------------------

This algorithm studies the kinematic properties
of the ISR jets. It reads the results of the
matching algorithm

To execute, type:

./ISR_jet_analysis config_file.txt

where config_file.txt is the mandatory configuration
file
*/


#include "ROOTFunctions.h"
#include "graphs_Funcs.h"
#include "functions.h"
#include "Rtypes.h"
#include "DelphesFunctions.h"

// Global Variables
const Double_t PI = TMath::Pi();

int main(int argc, char **argv){
   std::cout.precision(4);
   // Counting time
   Double_t initialTime = clock();

   // Folder variables
   string head_folder = "/home/af.garcia1214/
      PhenoMCsamples/Simulations/
      MG_pythia8_delphes_parallel/
      _Tops_Events_WI_Matching/";
   string current_folder = "_Tops_MG_1K_AG_WI_003/";

   string head_folder_binary = "/home/af.garcia1214/
```

```
        PhenoMCsamples/Results_Improved_Codes/
        matching_Results/_Tops_matchs_WI_Matching/";
    string matching_name = "ISR_jets_Tops_WI_003.bn";

    string head_folder_results = "/home/af.garcia1214/
        PhenoMCsamples/Results_Improved_Codes/histo_folder
        /_Tops_histos_WI_Matching/";

    // Checking input parameters
    string config_file_name = "Debug/config_file.txt";
    // Reading the file as first parameter
    if (argc>1){
        config_file_name = argv[1];
    }
    else{
        cout << "It is necessary to type a configuration
            file as parameter. Execute as ./
            ISR_jet_analysis config_file.txt" << endl;
        return 1;
    }
    cout << "Reading input parameters" << endl;
    cout << "\tUsing as parameters' file: " <<
        config_file_name << endl;

    ifstream config_file (config_file_name);
    if (config_file.is_open()){
        cout << "\tReading file" << endl;
        string line;
        int number_line = 1;
        while (getline(config_file,line)){
            // Skipping commented lines
            if (line[0] == '!')
                continue;

            // Finding the position of the equal sign
            int pos_equal = -1;
            pos_equal = line.find('=');

            if (pos_equal == -1){
                cout << "\tLine " << number_line << " is
```

```
                incorrect" << endl;
            continue;
        }

        // Splitting the line according to the position
             of equal sign
        string var_name = line.substr(0,pos_equal);
        string var_value = line.substr(pos_equal+1);

        // Reading head folder
        if(var_name.compare("head_folder") == 0){
            head_folder = var_value;
            cout << "\tVariable head folder set as: " <<
                  head_folder << endl;
        }
        // Reading current folder
        else if (var_name.compare("current_folder") ==
            0){
            current_folder = var_value;
            cout << "\tVariable current folder set as: "
                  << current_folder <<endl;
        }
        // Reading head folder binary
        else if (var_name.compare("head_folder_binary")
             == 0){
            head_folder_binary = var_value;
            cout << "\tVariable head folder binary set
                as: " << head_folder_binary << endl;
        }
        // Reading matching name
        else if (var_name.compare("matching_name") ==
            0){
            matching_name = var_value;
            cout << "\tVariable matching_name set as: "
                << matching_name << endl;
        }
        // Reading head folder results
        else if (var_name.compare("head_folder_results"
            ) == 0){
            head_folder_results = var_value;
```

```
            cout << "\tVariable head folder results set
               as: " << head_folder_results << endl;
         }

         number_line ++;
      }
   }
   else
   {
      cout << "ERROR: File " << config_file_name << "
         does not exist. Terminating program" << endl;
      return 0;
   }


   cout << "\n *** ISR jet analysis *** \n" << endl;

   /*
    * Histograms
    */
   // All jets
   TH1 *h_numberJet = new TH1F("Number Jets","Number
      Jets",11,-0.5,10.5);

   // Non Isr jets
   TH1 *h_jet_PT = new TH1F("Jet PT","Jet PT",
      201,0.0,600.0);
   TH1 *h_jet_Eta = new TH1F("Jet Eta","Jet Eta",
      171,-5.0,5.0);
   TH1 *h_jet_Phi = new TH1F("Jet Phi","Jet Phi",
      375,-3.5,3.5);
   TH1 *h_jet_DPhi_MET = new TH1F("Jet - MET Delta_Phi",
      "Jet - MET Delta_Phi",300,0.0,4.0);
   TH1 *h_jet_DPhi_MET_hpt = new TH1F("Jet - MET
      Delta_Phi_hpt","Jet - MET Delta_Phi_hpt"
      ,300,0.0,4.0);
   TH1 *h_jet_MT = new TH1F("Jet Transverse mass","Jet
      Transverse Mass",201,0.0,600.0);
   TH1 *h_jet_Delta_PT = new TH1F("Jet Delta-PT","Non
      ISR Delta-PT", 201,0.0,300.0);
```

```
    TH1 *h_jet_PT_HT = new TH1F("Jet PT-HT ratio","Jet PT-HT ratio",201,-0.0025,1.0025);
    TH1 *h_jet_PT_over_PT_others = new TH1F("Jet PT/PT_others","Jet PT/PT_others",401,-0.0025,2.0025);
    TH1 *h_jet_Eta_over_Eta_others = new TH1F("Jet Eta/Eta_others","Jet Eta/Eta_others",401,-0.0025,2.0025);
    TH1 *h_jet_DPhi_over_Phi_others = new TH1F("Jet Phi/Phi_others","Jet Phi/Phi_others",401,-0.0025,2.0025);
    TH1 *h_jet_Delta_Eta = new TH1F("Jet Delta-Eta","Jet Delta-Eta", 171,0.0,5.0);
    TH1 *h_jet_DPhi_MET_other = new TH1F("Jet - MET Delta_Phi other","Jet - MET Delta_Phi other",300,0.0,4.0);
    TH1 *h_jet_multiplicity = new TH1F("Jet - Multiplicity","Jet - Multiplicity",101,-0.5,100.5);
    TH1 *h_jet_DeltaR = new TH1F ("Jet - Delta_R","Jet - Delta_R",201,-0.0025,0.8025);
    TH1 *h_jet_Delta_PT_leading = new TH1F("Delta PT: leading - Jet","Delta PT: leading - Jet",201,0.0,600.0);
    TH1 *h_jet_Delta_Eta_leading = new TH1F("Delta Eta: Jet - leading","Delta Eta: Jet - leading",171,0.0,8.0);

    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);

    // ISR jets
    TH1 *h_ISR_PT = new TH1F("ISR PT","ISR PT",201,0.0,600.0);
    TH1 *h_ISR_Eta = new TH1F("ISR Eta","ISR Eta",171,-5.0,5.0);
    TH1 *h_ISR_Phi = new TH1F("ISR Phi","ISR Phi",375,-3.5,3.5);
    TH1 *h_ISR_DPhi_MET = new TH1F("ISR - MET Delta_Phi","ISR - MET Delta_Phi",300,0.0,4.0);
```

```
  TH1 *h_ISR_DPhi_MET_hpt = new TH1F("ISR - MET Delta_Phi_hpt","ISR - MET Delta_Phi_hpt",300,0.0,4.0);
  TH1 *h_ISR_MT = new TH1F("ISR Transverse mass","ISR Transverse Mass",201,0.0,600.0);
  TH1 *h_ISR_Delta_PT = new TH1F("ISR Delta-PT","ISR Delta-PT", 201,0.0,300.0);
  TH1 *h_ISR_PT_HT = new TH1F("ISR PT-HT ratio","ISR PT-HT ratio",201,-0.0025,1.0025);
  TH1 *h_ISR_PT_over_PT_others = new TH1F("ISR PT/PT_others","ISR PT/PT_others",401,-0.0025,2.0025);
  TH1 *h_ISR_Eta_over_Eta_others = new TH1F("ISR Eta/Eta_others","ISR Eta/Eta_others",401,-0.0025,2.0025);
  TH1 *h_ISR_DPhi_over_Phi_others = new TH1F("ISR Phi/Phi_others","ISR Phi/Phi_others",401,-0.0025,2.0025);
  TH1 *h_ISR_Delta_Eta = new TH1F("ISR Delta-Eta","ISR Delta-Eta", 171,0.0,5.0);
  TH1 *h_ISR_DPhi_MET_other = new TH1F("ISR - MET Delta_Phi other","ISR - MET Delta_Phi other",300,0.0,4.0);
  TH1 *h_ISR_multiplicity = new TH1F("ISR - Multiplicity","ISR - Multiplicity",101,-0.5,100.5);
  TH1 *h_ISR_DeltaR = new TH1F ("ISR - Delta_R","ISR - Delta_R",201,-0.0025,0.8025);
  TH1 *h_ISR_Delta_PT_leading = new TH1F("Delta PT: leading - ISR","Delta PT: leading - ISR",201,0.0,600.0);
  TH1 *h_ISR_Delta_Eta_leading = new TH1F("Delta Eta: ISR - leading","Delta Eta: ISR - leading",171,0.0,8.0);

  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);

  // MET
  TH1 *h_MET = new TH1F("Missing ET","Missing ET",200,0,600);
```

```
    TH1 *h_MET_hpt1 = new TH1F("Missing ET high_ISR_pt-1","Missing ET high_ISR_pt-1",200,0.0,600.0);
    TH1 *h_MET_hpt2 = new TH1F("Missing ET high_ISR_pt-2","Missing ET high_ISR_pt-2",200,0.0,600.0);
    TH1 *h_MET_hpt3 = new TH1F("Missing ET high_ISR_pt-3","Missing ET high_ISR_pt-3",200,0.0,600.0);
    TH1 *h_MET_hpt4 = new TH1F("Missing ET high_ISR_pt-4","Missing ET high_ISR_pt-4",200,0.0,600.0);

    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);
    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);

    // Leading PT
    TH1 *h_leading_PT = new TH1F("Leading PT","Leading PT", 201,0.0,600.0);
    TH1 *h_leading_MT = new TH1F("Leading Transverse mass","Leading Transverse Mass",201,0.0,600.0);
    TH1 *h_leading_Eta = new TH1F("Leading Eta","Leading Eta", 171,-5.0,5.0);
    TH1 *h_leading_DPhi_MET = new TH1F("Leading - MET Delta_Phi","Leading - MET Delta_Phi",300,0.0,4.0);

    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);

    // Other variables
    TH1 *h_HT = new TH1F("HT","HT",201,0.0,600.0);
    TH1 *h_HT_R1 = new TH1F("HT_R1","HT_R1",51,-0.01,1.01);
    TH1 *h_HT_R2 = new TH1F("HT_R2","HT_R2",51,-0.01,1.01);

    // B tagging
```

```
    TH1 *h_BTag = new TH1F("BTag","BTag",5,-0.5,4.5);
    TH1 *h_BTag_PT = new TH1F("BTag PT","BTag PT",201,0.0,600.0);
    TH1 *h_BTag_Eta = new TH1F("BTag Eta","BTag Eta",171,-5.0,5.0);
    TH1 *h_BTag_DPhi_MET = new TH1F("BTag - MET Delta_Phi","BTag - MET Delta_Phi",300,0.0,4.0);
    TH1 *h_BTags_per_Event = new TH1F("BTags per event","BTags per event",5,-0.5,4.5);

    // Further analysis
    TH1 *h_ISR_PT_comp = new TH1F("ISR PT for comparison","ISR PT for comparison with histo", 20,0.0,800.0);
    TH1 *h_ISR_Eta_comp = new TH1F("ISR Eta for comparison","ISR Eta for comparison with histo",20,-4.2,4.2);
    TH1 *h_ISR_DPhi_MET_comp = new TH1F("ISR Phi for comparison","ISR Phi for comparison with histo",20,0,PI);

    // To check the histograms' creation
    TH1 *hist_ISR_PT = new TH1F("ISR PT comp","ISR PT comp", 20,0.0,800.0);
    TH1 *hist_ISR_Abs_Eta = new TH1F("ISR Abs Eta comp","ISR Abs Eta comp", 20,0.0,5.2);
    TH1 *hist_ISR_DPhi_MET = new TH1F("ISR Delta Phi comp","ISR Delta Phi comp", 20,0.0,PI);
    TH1 *hist_ISR_PT_ratio = new TH1F("ISR PT/PT_others comp","ISR PT/PT_others comp",20,0.0,8.0);
    TH1 *hist_ISR_Delta_Eta = new TH1F("ISR Delta-Eta comp","ISR Delta-Eta comp", 20,0.0,7.0);
    TH1 *hist_ISR_DPhi_MET_other = new TH1F("ISR - MET Delta_Phi other comp","ISR - MET Delta_Phi other comp",20,0.0,PI);
    TH1 *hist_ISR_Delta_PT_leading = new TH1F("Delta PT: leading - ISR comp","Delta PT: leading - ISR comp", 20,0.0,500.0);
    TH1 *hist_ISR_Delta_Eta_leading = new TH1F("Delta Eta: ISR - leading comp","Delta Eta: ISR - leading
```

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

    // Cycle over several runs . iRun corresponds to the seed of the current run
    for(int iRun = 1; iRun < 11; iRun ++){
        // Create chains of root trees
        TChain chain_Delphes("Delphes");

        // Loading simulations from Delphes
        Char_t unidad = 0x30 + iRun%10;
        Char_t decena = 0x30 + int(iRun/10)%10;
        Char_t centena = 0x30 + int(iRun/100)%10;

        current_folder[current_folder.size()-4] = centena;
        current_folder[current_folder.size()-3] = decena;
        current_folder[current_folder.size()-2] = unidad;
        matching_name[matching_name.size()-6] = centena;
        matching_name[matching_name.size()-5] = decena;
        matching_name[matching_name.size()-4] = unidad;

        string file_pythia_str = head_folder +
```

```
current_folder + "Events/run_01/output_pythia8.root";

    string file_delphes_str = head_folder + current_folder + "Events/run_01/output_delphes.root";
    Char_t *file_delphes = (Char_t *) file_delphes_str.c_str();

    cout << "\nReading the file: \nDelphes: " << file_delphes << endl;

    chain_Delphes.Add(file_delphes);
    // Objects of class ExRootTreeReader for reading the information
    ExRootTreeReader *treeReader_Delphes = new ExRootTreeReader(&chain_Delphes);

    Long64_t numberOfEntries = treeReader_Delphes->GetEntries();

    // Get pointers to branches used in this analysis
    TClonesArray *branchJet = treeReader_Delphes->UseBranch("Jet");
    TClonesArray *branchMissingET = treeReader_Delphes->UseBranch("MissingET");

    cout << endl;
    cout << " Number of Entries Delphes = " << numberOfEntries << endl;
    cout << endl;

    // particles, jets and vectors
    MissingET *METpointer;
    TLorentzVector *vect_currentJet = new TLorentzVector;
    TLorentzVector *vect_auxJet = new TLorentzVector;
    TLorentzVector *vect_leading = new TLorentzVector;
    Jet *currentJet = new Jet;
    Jet *auxJet = new Jet;
```

```
    TRefArray array_temp;

    // Temporary variables
    Double_t MET = 0.0; // Missing transverse energy
    Double_t delta_phi = 0.0; // difference between the phi angle of MET and the jet
    Double_t transverse_mass = 0.0; // Transverse mass
    Double_t HT = 0.0; // Sum of jets' PT
    Double_t HT_R1 = 0.0; // Sum of jets' PT which are in the same hemisphere of the ISR jet hemisphere
    Double_t HT_R2 = 0.0; // Sum of jets' PT which are in the opposite hemisphere of the ISR jet hemisphere
    Double_t ISR_Eta = 0.0; // Pseudorapidity of the ISR jet
    Int_t number_Btags = 0; // Number of B jets per event
    Int_t ISR_Btags = 0; // Number of BTags which are also ISR jets
    Double_t delta_PT_jet = 0.0; // |PT-<PT>|
    Double_t PT_sum = 0.0; // sum(PT)
    Double_t PT_aver = 0.0; // <PT>
    Double_t Delta_eta_aver = 0.0; // sum_i|eta-eta_i|/(Nj-1)
    Double_t Delta_phi_sum = 0.0; // sum delta_phi
    Double_t Delta_phi_other_jets = 0.0; // Average of delta phi of other jets
    Double_t PT_ratio = 0.0; // PT/PT_others
    Double_t Eta_ratio = 0.0; // Eta/Eta_others
    Double_t Eta_sum = 0.0; // sum(Eta)
    Double_t Delta_R = 0.0; // Size of the jet
    Double_t Delta_phi_ratio = 0.0; // Delta_phi/Delta_phi_others
    Double_t Delta_PT_leading = 0.0; // PT - PT_leading
    Double_t Delta_Eta_leading = 0.0; // |Eta - Eta_leading|

    /*
```

```
     * Some variables used through the code
     */
    Int_t ISR_jets[numberOfEntries];
    Int_t NumJets = 0;

    string fileName_str = head_folder_binary +
       matching_name;

    Char_t * fileName = (Char_t *) fileName_str.c_str
       ();

    ifstream ifs(fileName,ios::in | ios::binary);

    for (Int_t j = 0; j<numberOfEntries; j++){
          ifs.read((Char_t *) (ISR_jets+j),sizeof(
             Int_t));
    }
    ifs.close();

    // Jet with greatest PT
    Double_t PT_max = 0;
    Int_t posLeadingPT = -1;
    Int_t ISR_greatest_PT = 0;
    Double_t MT_leading_jet = 0.0; // Transverse mass

    /*
     * Main cycle of the program
     */
    numberOfEntries = 100000;
    for (Int_t entry = 0; entry < numberOfEntries; ++
       entry){
       // Progress
       if(numberOfEntries>10 && (entry%((int)
          numberOfEntries/10))==0.0){
          cout<<"progress = "<<(entry*100/
             numberOfEntries)<<"%\t";
          cout<< "Time :"<< (clock()-initialTime)/
             double_t(CLOCKS_PER_SEC)<<"s"<<endl;
       }

```

```
        // Load selected branches with data from
           specified event
        treeReader_Delphes->ReadEntry(entry);

        // MET
        METpointer = (MissingET*) branchMissingET->At
           (0);
        MET = METpointer->MET;
        h_MET->Fill(MET);

        NumJets=branchJet->GetEntries();
        h_numberJet->Fill(NumJets);

        // checking the ISR
        if (ISR_jets[entry] == -1 || NumJets < 3)
           continue;

        PT_max = 0;
        posLeadingPT = -1;
        HT = 0;
        HT_R1 = 0;
        HT_R2 = 0;
        number_Btags = 0;

        delta_PT_jet = 0.0;
        PT_aver = 0.0;
        PT_sum = 0.0;
        Delta_eta_aver = 0.0;
        Delta_phi_sum = 0.0;
        Delta_phi_other_jets = 0.0;
        Delta_phi_ratio = 0.0;
        Delta_PT_leading = 0.0;
        Delta_Eta_leading = 0.0;

        PT_ratio = 0.0;
        Eta_ratio = 0.0;
        Eta_sum = 0.0;

        Delta_R = 0.0;

```

```
        if (ISR_jets[entry] >= NumJets){
           cout << "Error en el matching" << endl;
           return 1;
        }

        // Preliminary for. It is used to calculate
           PT_aver and Delta_phi_sum
        for (Int_t iJet = 0; iJet<NumJets; iJet++){
           currentJet = (Jet*) branchJet->At(iJet);
           vect_currentJet->SetPtEtaPhiM(currentJet->PT
              ,currentJet->Eta,currentJet->Phi,
              currentJet->Mass);
           delta_phi = deltaAng(vect_currentJet->Phi(),
               METpointer->Phi);
           PT_sum += vect_currentJet->Pt();
           Eta_sum += vect_currentJet->Eta();
           Delta_phi_sum += delta_phi;
           // HT
           HT += vect_currentJet->Pt();
           // HT ratios
           if((vect_currentJet->Eta()*ISR_Eta) > 0)
              HT_R1 += vect_currentJet->Pt();
           else
              HT_R2 += vect_currentJet->Pt();
           // PT Leading jet
           if(PT_max < vect_currentJet->Pt()){
              PT_max = vect_currentJet->Pt();
              posLeadingPT = iJet;
           }
        }

        //PT_aver
        PT_aver = PT_sum/NumJets;

        // Leading PT
        currentJet = (Jet*) branchJet->At(posLeadingPT)
           ;
        vect_leading->SetPtEtaPhiM(currentJet->PT,
           currentJet->Eta,currentJet->Phi,currentJet->
           Mass);
```

```

        // ISR jet
        currentJet = (Jet*) branchJet->At(ISR_jets[entry]);
        vect_currentJet->SetPtEtaPhiM(currentJet->PT, currentJet->Eta,currentJet->Phi,currentJet->Mass);
        ISR_Eta = vect_currentJet->Eta();

        for (Int_t iJet = 0; iJet<NumJets; iJet++){
            currentJet = (Jet*) branchJet->At(iJet);
            vect_currentJet->SetPtEtaPhiM(currentJet->PT,currentJet->Eta,currentJet->Phi,currentJet->Mass);
            delta_phi = deltaAng(vect_currentJet->Phi(), METpointer->Phi);
            transverse_mass = sqrt(2*vect_currentJet->Pt()*MET*(1-cos(delta_phi)));

            // Correlated variables
            delta_PT_jet = TMath::Abs(vect_currentJet->Pt()-PT_aver);
            Delta_phi_other_jets = (Delta_phi_sum-delta_phi)/(NumJets-1);
            PT_ratio = vect_currentJet->Pt()*(NumJets-1)/(PT_sum-vect_currentJet->Pt());
            Eta_ratio = vect_currentJet->Eta()*(NumJets-1)/(Eta_sum-vect_currentJet->Eta());
            Delta_phi_ratio = delta_phi*(NumJets-1)/(Delta_phi_sum-delta_phi);

            Delta_Eta_leading = TMath::Abs(vect_currentJet->Eta()-vect_leading->Eta());
            Delta_PT_leading = vect_leading->Pt()-vect_currentJet->Pt();

            Delta_eta_aver = 0.0;
            // For cycle used to calculate Delta_eta_aver
```

```
            for(Int_t iJet2 = 0; iJet2<NumJets; iJet2++){
               auxJet = (Jet*) branchJet->At(iJet2);
               vect_auxJet->SetPtEtaPhiM(auxJet->PT,auxJet->Eta,auxJet->Phi,auxJet->Mass);
               if (iJet2 != iJet) Delta_eta_aver += TMath::Abs(vect_auxJet->Eta()-vect_currentJet->Eta());
            }
            Delta_eta_aver = Delta_eta_aver/(NumJets-1);
            Delta_R = sqrt(pow(currentJet->DeltaEta,2)+pow(currentJet->DeltaPhi,2));

            // Multiplicity
            array_temp = (TRefArray) currentJet->Constituents;

            if (iJet != ISR_jets[entry]){ // Non ISR
               h_jet_PT->Fill(vect_currentJet->Pt());
               h_jet_Eta->Fill(vect_currentJet->Eta());
               h_jet_Phi->Fill(vect_currentJet->Phi());
               h_jet_DPhi_MET->Fill(delta_phi);
               h_jet_MT->Fill(transverse_mass);
               h_jet_Delta_PT->Fill(delta_PT_jet);
               h_jet_Delta_Eta->Fill(Delta_eta_aver);
               h_jet_DPhi_MET_other->Fill(Delta_phi_other_jets);
               h_jet_PT_HT->Fill(vect_currentJet->Pt()/HT);
               h_jet_multiplicity->Fill(array_temp.GetEntries());
               h_jet_PT_over_PT_others->Fill(PT_ratio);
               h_jet_Eta_over_Eta_others->Fill(Eta_ratio);
               h_jet_DeltaR->Fill(Delta_R);
               h_jet_DPhi_over_Phi_others->Fill(Delta_phi_ratio);
               h_jet_Delta_PT_leading->Fill(Delta_PT_leading);
               h_jet_Delta_Eta_leading->Fill(
```

```
                    Delta_Eta_leading);
                if (vect_currentJet->Pt()>240)
                    h_jet_DPhi_MET_hpt->Fill(delta_phi);
                h2_jet_PTEta->Fill(vect_currentJet->Pt(),
                    vect_currentJet->Eta());

                // For testing creating histo
                hist_jet_PT->Fill(vect_currentJet->Pt());
                hist_jet_Abs_Eta->Fill(TMath::Abs(
                    vect_currentJet->Eta()));
                hist_jet_DPhi_MET->Fill(delta_phi);
                hist_jet_PT_ratio->Fill(PT_ratio);
                hist_jet_Delta_Eta->Fill(Delta_eta_aver);
                hist_jet_DPhi_MET_other->Fill(
                    Delta_phi_other_jets);
                hist_jet_Delta_PT_leading->Fill(
                    Delta_PT_leading);
                hist_jet_Delta_Eta_leading->Fill(
                    Delta_Eta_leading);
            }
            else{ //ISR
                h_ISR_PT->Fill(vect_currentJet->Pt());
                h_ISR_Eta->Fill(vect_currentJet->Eta());
                h_ISR_Phi->Fill(vect_currentJet->Phi());
                h_ISR_DPhi_MET->Fill(delta_phi);
                h_ISR_Eta_comp->Fill(vect_currentJet->Eta
                    ());
                h_ISR_PT_comp->Fill(vect_currentJet->Pt()
                    );
                h_ISR_DPhi_MET_comp->Fill(delta_phi);
                h_ISR_Delta_PT->Fill(delta_PT_jet);
                h_ISR_Delta_Eta->Fill(Delta_eta_aver);
                h_ISR_DPhi_MET_other->Fill(
                    Delta_phi_other_jets);
                h_ISR_PT_HT->Fill(vect_currentJet->Pt()/
                    HT);
                h_ISR_multiplicity->Fill(array_temp.
                    GetEntries());
                h_ISR_PT_over_PT_others->Fill(PT_ratio);
                h_ISR_Eta_over_Eta_others->Fill(Eta_ratio
```

```
                    );
                 h_ISR_DeltaR->Fill(Delta_R);
                 h_ISR_DPhi_over_Phi_others->Fill(
                    Delta_phi_ratio);
                 h_ISR_Delta_PT_leading->Fill(
                    Delta_PT_leading);
                 h_ISR_Delta_Eta_leading->Fill(
                    Delta_Eta_leading);
                 if (vect_currentJet->Pt()>120)
                    h_MET_hpt1->Fill(MET);
                 if (vect_currentJet->Pt()>200)
                    h_MET_hpt2->Fill(MET);
                 if (vect_currentJet->Pt()>240){
                    h_MET_hpt3->Fill(MET);
                    h_ISR_DPhi_MET_hpt->Fill(delta_phi);
                 }
                 if (vect_currentJet->Pt()>300)
                    h_MET_hpt4->Fill(MET);
                 h2_ISR_PTEta->Fill(vect_currentJet->Pt(),
                    vect_currentJet->Eta());
                 // Transverse mass
                 h_ISR_MT->Fill(transverse_mass);

                 // For testing creating histo
                 hist_ISR_PT->Fill(vect_currentJet->Pt());
                 hist_ISR_Abs_Eta->Fill(TMath::Abs(
                    vect_currentJet->Eta()));
                 hist_ISR_DPhi_MET->Fill(delta_phi);
                 hist_ISR_PT_ratio->Fill(PT_ratio);
                 hist_ISR_Delta_Eta->Fill(Delta_eta_aver);
                 hist_ISR_DPhi_MET_other->Fill(
                    Delta_phi_other_jets);
                 hist_ISR_Delta_PT_leading->Fill(
                    Delta_PT_leading);
                 hist_ISR_Delta_Eta_leading->Fill(
                    Delta_Eta_leading);
              }

              // BTag
              h_BTag->Fill(currentJet->BTag);
```

```
            if (currentJet->BTag == 1){ // The current jet is B Tagged
               h_BTag_PT->Fill(vect_currentJet->Pt());
               h_BTag_Eta->Fill(vect_currentJet->Eta());
               h_BTag_DPhi_MET->Fill(delta_phi);
               number_Btags++;

               if (iJet == ISR_jets[entry]){ // If the ISR jet is also a B jet
                  ISR_Btags++;
               }
            }
         }

         // Jet with greatest PT
         if (posLeadingPT != -1){
            h_leading_PT->Fill(PT_max);
            if(posLeadingPT == ISR_jets[entry]) ISR_greatest_PT++;

            currentJet = (Jet*) branchJet->At(posLeadingPT);
            vect_currentJet->SetPtEtaPhiM(currentJet->PT,currentJet->Eta,currentJet->Phi,currentJet->Mass);
            delta_phi = deltaAng(vect_currentJet->Phi(), METpointer->Phi);
            MT_leading_jet = sqrt(2*vect_currentJet->Pt()*MET*(1-cos(delta_phi)));
            h_leading_MT->Fill(MT_leading_jet);

            h_leading_Eta->Fill(vect_currentJet->Eta());
            h_leading_DPhi_MET->Fill(delta_phi);

            h2_leading_PTEta->Fill(vect_currentJet->Pt(),vect_currentJet->Eta());
         }

         // HT
         if (1 < HT_R1/HT || 1 < HT_R2/HT){
```

```
                cout << "Error en el evento: " << entry << endl;
                cout << "HT: " << HT << "\tHT_R1: " << HT_R1 << "\tHT_R2: " << HT_R2 << endl;
                return 1;
            }

            h_HT->Fill(HT);
            h_HT_R1->Fill(HT_R1/HT);
            h_HT_R2->Fill(HT_R2/HT);
            h_BTags_per_Event->Fill(number_Btags);

        }

        cout<<"progress = 100%\t";
        cout<< "Time :"<< (clock()-initialTime)/double_t(CLOCKS_PER_SEC)<<"s"<<endl;
        cout<< "Percentage of events where the ISR jet is the jet with greatest PT: " << (Double_t) (ISR_greatest_PT*100)/numberOfEntries << "%\n";
        cout<< "Percentage of events where the ISR jet is tagged as Bjet: " << (Double_t) (ISR_Btags*100)/numberOfEntries << "%\n";

    } // End run's for cicle

    string hfile_name_str = head_folder_results + "histos.root";

    Char_t *hfile_name = (Char_t *) hfile_name_str.c_str();

    TFile* hfile = new TFile(hfile_name, "RECREATE");
    h_jet_DPhi_MET->Write();
    h_jet_Eta->Write();
    h_jet_PT->Write();
    h_jet_Phi->Write();
    h_jet_MT->Write();
    h_jet_Delta_PT->Write();
    h_jet_Delta_Eta->Write();
```

```
    h_jet_DPhi_MET_other->Write();
    h_jet_PT_HT->Write();
    h_jet_multiplicity->Write();
    h_jet_PT_over_PT_others->Write();
    h_jet_Eta_over_Eta_others->Write();
    h_jet_DeltaR->Write();
    h_jet_DPhi_over_Phi_others->Write();
    h_jet_Delta_Eta_leading->Write();
    h_jet_Delta_PT_leading->Write();

    h_ISR_DPhi_MET->Write();
    h_ISR_Eta->Write();
    h_ISR_PT->Write();
    h_ISR_Phi->Write();
    h_ISR_MT->Write();
    h_ISR_Delta_PT->Write();
    h_ISR_Delta_Eta->Write();
    h_ISR_DPhi_MET_other->Write();
    h_ISR_PT_HT->Write();
    h_ISR_multiplicity->Write();
    h_ISR_PT_over_PT_others->Write();
    h_ISR_Eta_over_Eta_others->Write();
    h_ISR_DeltaR->Write();
    h_ISR_DPhi_over_Phi_others->Write();
    h_ISR_Delta_Eta_leading->Write();
    h_ISR_Delta_PT_leading->Write();

    h_MET->Write();
    h_MET_hpt1->Write();
    h_MET_hpt2->Write();
    h_MET_hpt3->Write();

    h_leading_MT->Write();
    h_leading_PT->Write();
    h_leading_Eta->Write();
    h_leading_DPhi_MET->Write();

    h_HT->Write();
    h_HT_R1->Write();
    h_HT_R2->Write();
```

```

  h_numberJet->Write();

  h_BTag->Write();
  h_BTag_PT->Write();
  h_BTag_Eta->Write();
  h_BTag_DPhi_MET->Write();
  h_BTags_per_Event->Write();

  h2_ISR_PTEta->Write();
  h2_jet_PTEta->Write();
  h2_dif_PTEta->Add(h2_ISR_PTEta,h2_jet_PTEta,1,-1);
  h2_dif_PTEta->Write();

  h2_dif_lead_PTEta->Add(h2_ISR_PTEta,h2_leading_PTEta,1,-1);
  h2_dif_lead_PTEta->Write();

  {
    string salida_str = head_folder_results + "Eta";
    Char_t *salida = (Char_t *) salida_str.c_str();
    TCanvas *C = new TCanvas(salida,"Pseudorapidity",1280,720);
    Present(h_ISR_Eta,h_jet_Eta,C,1,"h","Num. Jets / Total",122);
    C->Write();
    C->Close();

    salida_str = head_folder_results + "Eta ISR vs BTag";
    salida = (Char_t *) salida_str.c_str();
    C = new TCanvas(salida,"Pseudorapidity ISR vs BTag",1280,720);
    Present(h_ISR_Eta,h_BTag_Eta,C,1,"h","Num. Jets / Total",122);
    C->Write();
    C->Close();

    salida_str = head_folder_results + "Eta ISR vs Leading";
```

```
    salida = (Char_t *) salida_str.c_str();
      C = new TCanvas(salida,"Pseudorapidity ISR vs Leading",1280,720);
    Present(h_ISR_Eta,h_leading_Eta,C,1,"h","Num. Jets / Total",122);
    C->Write();
    C->Close();

    salida_str = head_folder_results + "Transverse momentum";
    salida = (Char_t *) salida_str.c_str();
    C = new TCanvas(salida,"Transverse momentum",1280,720);
    Present(h_ISR_PT,h_jet_PT,C,2,"PT [GeV]","Num. Jets / Total");
    C->Write();
    C->Close();

    salida_str = head_folder_results + "Transverse momentum ISR vs Leading";
    salida = (Char_t *) salida_str.c_str();
    C = new TCanvas(salida,"Transverse momentum ISR vs Leading",1280,720);
    Present(h_ISR_PT,h_leading_PT,C,2,"PT [GeV]","Num. Jets / Total");
    C->Write();
    C->Close();

    salida_str = head_folder_results + "Transverse momentum ISR vs B_Tag";
    salida = (Char_t *) salida_str.c_str();
    C = new TCanvas(salida,"Transverse momentum ISR vs B_Tag",1280,720);
    Present(h_ISR_PT,h_BTag_PT,C,2,"PT [GeV]","Num. Jets / Total");
    C->Write();
    C->Close();

    salida_str = head_folder_results + "Transverse momentum ISR, B_Tag, Leading";
```

```
    salida = (Char_t *) salida_str.c_str();
    C = new TCanvas(salida,"Transverse momentum ISR, B_Tag, Leading",1280,720);
    Present_3(h_ISR_PT,h_BTag_PT,h_leading_PT,C,2,"PT [GeV]","Num. Jets / Total");
    C->Write();
    C->Close();

    salida_str = head_folder_results + "Transverse momentum ISR, B_Tag, Leading LOG";
    salida = (Char_t *) salida_str.c_str();
    C = new TCanvas(salida,"Transverse momentum ISR, B_Tag, Leading LOG",1280,720);
    Present_3(h_ISR_PT,h_BTag_PT,h_leading_PT,C,2,"PT [GeV]","Num. Jets / Total",12,12,true);
    C->Write();
    C->Close();

    salida_str = head_folder_results + "Transverse mass Leading vs ISR Jet";
    salida = (Char_t *) salida_str.c_str();
    C = new TCanvas(salida,"Transverse mass Leading vs ISR Jet",1280,720);
    Present(h_ISR_MT,h_leading_MT,C,2,"MT [GeV]","Num. Jets / Total");
    C->Write();
    C->Close();

    salida_str = head_folder_results + "Transverse mass ISR vs Jet";
    salida = (Char_t *) salida_str.c_str();
    C = new TCanvas(salida,"Transverse mass ISR vs Jet",1280,720);
    Present(h_ISR_MT,h_jet_MT,C,2,"MT [GeV]","Num. Jets / Total");
    C->Write();
    C->Close();

    salida_str = head_folder_results + "Phi";
    salida = (Char_t *) salida_str.c_str();
```

```
    C = new TCanvas(salida,"Phi",1280,720);
    Present(h_ISR_Phi,h_jet_Phi,C,3,"f","Num. Jets / Total",122);
    C->Write();
    C->Close();

    salida_str = head_folder_results + "Delta Phi - Jet - MET";
    salida = (Char_t *) salida_str.c_str();
    C = new TCanvas(salida,"Delta Phi - Jet - MET",1280,720);
    Present(h_ISR_DPhi_MET,h_jet_DPhi_MET,C,3,"Df","Num. Jets / Total",122);
    C->Write();
    C->Close();

    salida_str = head_folder_results + "Delta Phi - Jet - MET - Btag";
    salida = (Char_t *) salida_str.c_str();
    C = new TCanvas(salida,"Delta Phi - Jet - MET - Btag",1280,720);
    Present(h_ISR_DPhi_MET,h_BTag_DPhi_MET,C,3,"Df","Num. Jets / Total",122);
    C->Write();
    C->Close();

    salida_str = head_folder_results + "Delta Phi - Jet - MET - leading";
    salida = (Char_t *) salida_str.c_str();
    C = new TCanvas(salida,"Delta Phi - Jet - MET - leading",1280,720);
    Present(h_ISR_DPhi_MET,h_leading_DPhi_MET,C,1,"Df","Num. Jets / Total",122);
    C->Write();
    C->Close();

    salida_str = head_folder_results + "MET > 120";
    salida = (Char_t *) salida_str.c_str();
    C = new TCanvas(salida,"MET > 120",1280,720);
    Present(h_MET,h_MET_hpt1,C,2,"MET","Num. Jets /
```

```
Total");
        C->Write();
        C->Close();

        salida_str = head_folder_results + "MET > 200";
        salida = (Char_t *) salida_str.c_str();
        C = new TCanvas(salida,"MET > 200",1280,720);
        Present(h_MET,h_MET_hpt2,C,2,"MET","Num. Jets / Total");
        C->Write();
        C->Close();

        salida_str = head_folder_results + "MET > 240";
        salida = (Char_t *) salida_str.c_str();
        C = new TCanvas(salida,"MET > 240",1280,720);
        Present(h_MET,h_MET_hpt3,C,2,"MET","Num. Jets / Total");
        C->Write();
        C->Close();

        salida_str = head_folder_results + "HT ratio comparison";
        salida = (Char_t *) salida_str.c_str();
        C = new TCanvas(salida,"HT ratio comparison",1280,720);
        Present(h_HT_R1,h_HT_R2,C,2,"HT","Num. Jets / Total");
        C->Write();
        C->Close();

        salida_str = head_folder_results + "PT vs ETA - ISR";
        salida = (Char_t *) salida_str.c_str();
        C = new TCanvas(salida,"PT vs ETA - ISR",1280,720);
        Plot_Single_2D(h2_ISR_PTEta,C,2, "PT [GeV]", "h", 12, 122);
        C->Write();
        C->Close();

```

```
      salida_str = head_folder_results + "PT vs ETA - Jet";
      salida = (Char_t *) salida_str.c_str();
      C = new TCanvas(salida,"PT vs ETA - Jet",1280,720);
      Plot_Single_2D(h2_jet_PTEta,C,2, "PT [GeV]", "h", 12, 122);
      C->Write();
      C->Close();

      salida_str = head_folder_results + "PT vs ETA - Diff with any jet";
      salida = (Char_t *) salida_str.c_str();
      C = new TCanvas(salida,"PT vs ETA - Diff with any jet",1280,720);
      Plot_Single_2D(h2_dif_PTEta,C,2, "PT [GeV]", "h", 12, 122);
      C->Write();
      C->Close();

      salida_str = head_folder_results + "PT vs ETA - leading";
      salida = (Char_t *) salida_str.c_str();
      C = new TCanvas(salida,"PT vs ETA - leading",1280,720);
      Plot_Single_2D(h2_leading_PTEta,C,2, "PT [GeV]", "h", 12, 122);
      C->Write();
      C->Close();

      salida_str = head_folder_results + "PT vs ETA - Diff with leading";
      salida = (Char_t *) salida_str.c_str();
      C = new TCanvas(salida,"PT vs ETA - Diff with leading",1280,720);
      Plot_Single_2D(h2_dif_lead_PTEta,C,2, "PT [GeV]", "h", 12, 122);
      C->Write();
      C->Close();

```

```
        salida_str = head_folder_results + "HT";
        salida = (Char_t *) salida_str.c_str();
        C = new TCanvas(salida,"HT",1280,720);
        Plot_Single(h_HT,C,2, "HT [GeV]", "Num. Jets / Total", 12, 12);
        C->Write();
        C->Close();

        salida_str = head_folder_results + "Number_of_B_Tags";
        salida = (Char_t *) salida_str.c_str();
        C = new TCanvas(salida,"Number of B Tags",1280,720);
        Plot_Single(h_BTags_per_Event,C,2, "B Tags / event", "Num. Jets / Total", 12, 12);
        C->Write();
        C->Close();

        salida_str = head_folder_results + "Jet_multiplitcity";
        salida = (Char_t *) salida_str.c_str();
        C = new TCanvas(salida,"Jet multiplicity",1280,720);
        Present(h_ISR_multiplicity,h_jet_multiplicity,C,2,"Tracks","Num. Jets / Total");
        C->Write();
        C->Close();

        salida_str = head_folder_results + "Delta_R_-_Jet_size";
        salida = (Char_t *) salida_str.c_str();
        C = new TCanvas(salida,"Delta R - Jet Size",1280,720);
        Present(h_ISR_DeltaR,h_jet_DeltaR,C,1,"Delta_R","Num. Jets / Total");
        C->Write();
        C->Close();

          // Correlated variables
        salida_str = head_folder_results + "
```

```
        Cor_Delta_PT_Jet";
    salida = (Char_t *) salida_str.c_str();
    C = new TCanvas(salida,"Delta PT jet",1280,720);
    Present(h_ISR_Delta_PT,h_jet_Delta_PT,C,2,"PT [GeV
        ]","Num. Jets / Total");
    C->Write();
    C->Close();

    salida_str = head_folder_results + "
        Cor_PT_proportion";
    salida = (Char_t *) salida_str.c_str();
    C = new TCanvas(salida,"PT proportion",1280,720);
    Present(h_ISR_PT_HT,h_jet_PT_HT,C,2,"PT/HT","Num.
        Jets / Total");
    C->Write();
    C->Close();

    salida_str = head_folder_results + "
        Cor_Delta_Eta_Average";
    salida = (Char_t *) salida_str.c_str();
    C = new TCanvas(salida,"Delta Eta Average"
        ,1280,720);
    Present(h_ISR_Delta_Eta,h_jet_Delta_Eta,C,2,"Dh","
        Num. Jets / Total",122);
    C->Write();
    C->Close();

    salida_str = head_folder_results + "
        Cor_Delta_Phi_Jet_MET_other_jets";
    salida = (Char_t *) salida_str.c_str();
    C = new TCanvas(salida,"Delta Phi - Jet MET -
        other jets",1280,720);
    Present(h_ISR_DPhi_MET_other,h_jet_DPhi_MET_other,
        C,2,"Df","Num. Jets / Total",122);
    C->Write();
    C->Close();

    salida_str = head_folder_results + "Cor_PT_over_<
        PT_other>";
    salida = (Char_t *) salida_str.c_str();
```

```
        C = new TCanvas(salida,"PT/<PT_other>",1280,720);
        Present(h_ISR_PT_over_PT_others, h_jet_PT_over_PT_others,C,2,"PT/<PT>","Num. Jets / Total");
        C->Write();
        C->Close();

        salida_str = head_folder_results + "Cor_Eta_over_<Eta_other>";
        salida = (Char_t *) salida_str.c_str();
        C = new TCanvas(salida,"Eta/<Eta_other>",1280,720);
        Present(h_ISR_Eta_over_Eta_others, h_jet_Eta_over_Eta_others,C,3,"h/<h>","Num. Jets / Total",122);
        C->Write();
        C->Close();

        salida_str = head_folder_results + "Cor_Delta_Phi_over_<Delta_Phi_other>";
        salida = (Char_t *) salida_str.c_str();
        C = new TCanvas(salida,"Delta_Phi/<Delta_Phi_other>",1280,720);
        Present(h_ISR_DPhi_over_Phi_others, h_jet_DPhi_over_Phi_others,C,3,"Df/<Df>","Num. Jets / Total",122);
        C->Write();
        C->Close();

        // Comparison with the leading Jet
        salida_str = head_folder_results + "Leading_Delta_PT";
        salida = (Char_t *) salida_str.c_str();
        C = new TCanvas(salida,"Delta PT: PT_leading-PT",1280,720);
        Present(h_ISR_Delta_PT_leading, h_jet_Delta_PT_leading,C,2,"(PT_leading - PT)","Num. Jets / Total");
        C->Write();
        C->Close();
```

```

        salida_str = head_folder_results + "Leading_Delta_Eta";
        salida = (Char_t *) salida_str.c_str();
        C = new TCanvas(salida,"Delta Eta: |Eta-Eta_leading|",1280,720);
        Present(h_ISR_Delta_Eta_leading,h_jet_Delta_Eta_leading,C,2,"|Eta - Eta_leading|","Num. Jets / Total");
        C->Write();
        C->Close();

    }

    hfile->Close();

    hfile_name_str = head_folder_results + "histos2.root";

    hfile_name = (Char_t *) hfile_name_str.c_str();

    TFile* hfile2 = new TFile(hfile_name, "RECREATE");
    h_ISR_PT_comp->Write();
    h_ISR_Eta_comp->Write();
    h_ISR_DPhi_MET_comp->Write();

    hist_ISR_PT->Write();
    hist_ISR_Abs_Eta->Write();
    hist_ISR_DPhi_MET->Write();
    hist_ISR_PT_ratio->Write();
    hist_ISR_Delta_Eta->Write();
    hist_ISR_DPhi_MET_other->Write();
    hist_ISR_Delta_PT_leading->Write();
    hist_ISR_Delta_Eta_leading->Write();

    hist_jet_PT->Write();
    hist_jet_Abs_Eta->Write();
    hist_jet_DPhi_MET->Write();
    hist_jet_PT_ratio->Write();
    hist_jet_Delta_Eta->Write();
```

```
    hist_jet_DPhi_MET_other->Write();
    hist_jet_Delta_PT_leading->Write();
    hist_jet_Delta_Eta_leading->Write();

    hfile2->Close();

    return 0;
}
```

B.4 Creating histograms

```
/*
--------------------------------------------------
-------     Universidad de los Andes      -------
-------      Departamento de Fisica       -------
-------        Joven Investigador         -------
-------  Andres Felipe Garcia Albarracin  -------
-------    Juan Carlos Sanabria Arenas    -------
--------------------------------------------------

This algorithm fills 2 N-dimensional histograms.
The histograms contain kinematic variables of ISR
jets and non ISR jets.

The user can choose 3 of 8 variables for filling
 the histograms:
1. PT
2. Abs(Eta) // Eta is a pair function
3. Delta Phi_MET
4. PT_ratio
5. Delta Eta_aver
6. Delta Phi_MET_others
7. Delta PT_others
8. Delta Eta_others

In order to choose them, the code should be run as
./Creating_histo config_file.txt [N1 N2 N3]

where [config_file.txt] is a configuration file with
```

```
all parameters needed for the simulation.

N1 N2 and N3 are the index of the 3 variables.
If no parameter is passed as parameter, N1 N2 and N3
will be 0,1 and 2 by default.
*/


#include "ROOTFunctions.h"
#include "graphs_Funcs.h"
#include "functions.h"
#include "histoN.h"
#include "DelphesFunctions.h"

// Global Variables
const Double_t PI = TMath::Pi();

int main(int argc, char **argv){
    std::cout.precision(4);
    // Counting time
    Double_t initialTime = clock();

    // Folder variables
    string head_folder = "/home/af.garcia1214/PhenoMCsamples/Simulations/MG_pythia8_delphes_parallel/_Tops_Events_WI_Matching/";
    string current_folder = "_Tops_MG_1K_AG_WI_003/";

    string head_folder_binary = "/home/af.garcia1214/PhenoMCsamples/Results_Improved_Codes/matching_Results/_Tops_matchs_WI_Matching/";
    string matching_name = "ISR_jets_Tops_WI_003.bn";

    string head_folder_results = "/home/af.garcia1214/PhenoMCsamples/Results_Improved_Codes/histo_folder/_Tops_histos_WI_Matching/";

    // Checking input parameters
    string config_file_name = "Debug/config_file.txt";
```

```
  // Reading the file as first parameter
  if (argc>1){
     config_file_name = argv[1];
  }
  else{
     cout << "It is necessary to type a configuration
        file as parameter. Execute as ./Creating_histo
        config_file.txt [N1 N2 N3]" << endl;
     return 1;
  }
  cout << "Reading input parameters" << endl;
  cout << "\tUsing as parameters' file: " <<
     config_file_name << endl;

  ifstream config_file (config_file_name);
  if (config_file.is_open()){
     cout << "\tReading file" << endl;
     string line;
     int number_line = 1;
     while (getline(config_file,line)){
        // Skipping commented lines
        if (line[0] == '!')
           continue;

        // Finding the position of the equal sign
        int pos_equal = -1;
        pos_equal = line.find('=');

        if (pos_equal == -1){
           cout << "\tLine " << number_line << " is
              incorrect" << endl;
           continue;
        }

        // Splitting the line according to the position
            of equal sign
        string var_name = line.substr(0,pos_equal);
        string var_value = line.substr(pos_equal+1);

        // Reading head folder
```

```
            if(var_name.compare("head_folder") == 0){
                head_folder = var_value;
                cout << "\tVariable head folder set as: " << head_folder << endl;
            }
            // Reading current folder
            else if (var_name.compare("current_folder") == 0){
                current_folder = var_value;
                cout << "\tVariable current folder set as: " << current_folder <<endl;
            }
            // Reading head folder binary
            else if (var_name.compare("head_folder_binary") == 0){
                head_folder_binary = var_value;
                cout << "\tVariable head folder binary set as: " << head_folder_binary << endl;
            }
            // Reading matching name
            else if (var_name.compare("matching_name") == 0){
                matching_name = var_value;
                cout << "\tVariable matching_name set as: " << matching_name << endl;
            }
            // Reading head folder results
            else if (var_name.compare("head_folder_results") == 0){
                head_folder_results = var_value;
                cout << "\tVariable head folder results set as: " << head_folder_results << endl;
            }

            number_line ++;
        }
    }
    else
    {
        cout << "ERROR: File " << config_file_name << "
```

```
        does not exist. Terminating program" << endl;
      return 0;
   }


   cout << "\n *** Creating histograms *** \n" << endl;

   // Variables for initializing histograms
   Int_t dims = 3;
   Double_t min_Values[3] = {0,-5.2,0};
   Double_t max_Values[3] = {800,5.2,PI};

   /*
    * Read inputs and set variables for analysis
    */
   Int_t var_index[3] = {0,1,2}; // Index of the 3 variables for analysis. By default 0, 1 and 2
   string variables[8] = {"PT","Abs(Eta)","Delta Phi_MET","PT_ratio","Delta Eta_aver","Delta Phi_MET_others","Delta PT_leading","Delta Eta_leading"};
   Double_t var_values[8] = {0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0}; // Vector with the values of the 8 variables
   // Min and maximun values of the eight variables
   Double_t var_min_values[8] = {0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0};
   Double_t var_max_values[8] = {800,5.2,PI,8.0,7.0,PI,500,6.5};

   if (argc == 5){
      cout << "Filling histograms with the variables:" << endl;
      for (Int_t ind = 0; ind < 3; ind ++){
         var_index[ind] = atoi(argv[ind+2]);
      }
      cout << endl;
   }
   else if (argc == 1 || argc == 2) {
      cout << "Filling histograms with the default
```

```
            variables:" << endl;
    }
    else {
        cout << "Error at calling this algorithm. Use as:"
             << endl;
        cout << "\t ./Creating_histo N1 N2 N3 or ./
            Creating_histo" << endl;
        cout << "Read the documentation at the beginning
            of the code for further information\n" << endl;
        return 1;
    }

    cout << "Var \t\t min_Value \t max_Value" << endl;
    for (Int_t ind = 0; ind < 3; ind ++){
        min_Values[ind] = var_min_values[var_index[ind]];
        max_Values[ind] = var_max_values[var_index[ind]];
        cout << var_index[ind] << ". " << variables[
            var_index[ind]] <<
                "\t" << min_Values[ind] << "\t" <<
                    max_Values[ind] << endl;
    }
    cout << endl;


    /*
     * Initializing the 3-dimensional histogram
     */
    Int_t bins[3] = {20,20,20};
    histoN* histoISR = new histoN(dims,min_Values,
        max_Values,bins);
    histoN* histoNonISR = new histoN(dims,min_Values,
        max_Values,bins);
    // Input variables of each histogram
    Double_t values[3] = {0.0,0.0,0.0};

    // For loop over several simulations. iRun is the
        seed of the current simulation
    for(int iRun = 11; iRun < 261; iRun ++){
        // Create chains of root trees
        TChain chain_Delphes("Delphes");
```

```

        Char_t unidad = 0x30 + iRun%10;
        Char_t decena = 0x30 + int(iRun/10)%10;
        Char_t centena = 0x30 + int(iRun/100)%10;

        current_folder[current_folder.size()-4] = centena;
        current_folder[current_folder.size()-3] = decena;
        current_folder[current_folder.size()-2] = unidad;
        matching_name[matching_name.size()-6] = centena;
        matching_name[matching_name.size()-5] = decena;
        matching_name[matching_name.size()-4] = unidad;

        string file_delphes_str = head_folder +
           current_folder + "Events/run_01/output_delphes.
           root";

        Char_t *file_delphes = (Char_t *) file_delphes_str
           .c_str();

        cout << "\nWriting run: "<<centena<<decena<<unidad
           <<endl;
        cout << "\tReading the file: \n\tDelphes: " <<
           file_delphes << endl;

        chain_Delphes.Add(file_delphes);
        // Objects of class ExRootTreeReader for reading
           the information
        ExRootTreeReader *treeReader_Delphes = new
           ExRootTreeReader(&chain_Delphes);

        Long64_t numberOfEntries = treeReader_Delphes->
           GetEntries();

        // Get pointers to branches used in this analysis
        TClonesArray *branchJet = treeReader_Delphes->
           UseBranch("Jet");
        TClonesArray *branchMissingET = treeReader_Delphes
           ->UseBranch("MissingET");

        cout << "\tNumber of Entries Delphes = " <<
```

```
        numberOfEntries << endl;
    cout << endl;

    // particles, jets and vectors
    MissingET *METpointer;
    TLorentzVector *vect_currentJet = new
        TLorentzVector;
    TLorentzVector *vect_auxJet = new TLorentzVector;
    TLorentzVector *vect_leading = new TLorentzVector;
    Jet *currentJet = new Jet;
    Jet *auxJet = new Jet;

    // Temporary variables
    Double_t MET = 0.0; // Missing transverse energy
    Double_t delta_phi = 0.0; // difference between
        the phi angle of MET and the jet
    Double_t transverse_mass = 0.0; // Transverse mass
    Int_t numMatches = 0; // Number of matched jets
    Double_t delta_PT_jet = 0.0; // |PT-<PT>|
    Double_t PT_sum = 0.0; // sum(PT)
    Double_t PT_aver = 0.0; // <PT>
    Double_t Delta_eta_aver = 0.0; // sum_i|eta-eta_i
        |/(Nj-1)
    Double_t Delta_phi_sum = 0.0; // sum delta_phi
    Double_t Delta_phi_other_jets = 0.0; // Average of
         delta phi of other jets
    Double_t PT_ratio = 0.0; // PT/PT_others
    Double_t Delta_PT_leading = 0.0; // PT -
        PT_leading
    Double_t Delta_Eta_leading = 0.0; // |Eta -
        Eta_leading|

    // Jet with greatest PT
    Double_t PT_max = 0;
    Int_t posLeadingPT = -1;
    Int_t ISR_greatest_PT = 0;
    Double_t MT_leading_jet = 0.0; // Transverse mass

    /*
     * Some variables used through the code
```

```
         */
        Int_t ISR_jets[numberOfEntries];
        Int_t NumJets = 0;

        string fileName_str = head_folder_binary +
           matching_name;

        Char_t * fileName = (Char_t *) fileName_str.c_str
           ();

        ifstream ifs(fileName,ios::in | ios::binary);

        for (Int_t j = 0; j<numberOfEntries; j++){
              ifs.read((Char_t *) (ISR_jets+j),sizeof(
                 Int_t));
        }
        ifs.close();

        /*
         * Main cycle of the program
         */
        numberOfEntries = 100000;
        for (Int_t entry = 0; entry < numberOfEntries; ++
           entry){
           // Progress
           if(numberOfEntries>10 && (entry%((int)
              numberOfEntries/10))==0.0){
              cout<<"\tprogress = "<<(entry*100/
                 numberOfEntries)<<"%\t";
              cout<< "Time :"<< (clock()-initialTime)/
                 double_t(CLOCKS_PER_SEC)<<"s"<<endl;
           }

           // Load selected branches with data from
              specified event
           treeReader_Delphes->ReadEntry(entry);

           // MET
           METpointer = (MissingET*) branchMissingET->At
              (0);
```

```
        MET = METpointer->MET;

        NumJets=branchJet->GetEntries();

        // checking the ISR
        if (ISR_jets[entry] == -1 || NumJets < 3)
           continue;

        if (ISR_jets[entry] >= NumJets){
           cout << "Error en el matching" << endl;
           return 1;
        }

        // 3 PT ratio
        PT_aver = 0.0;
        PT_sum = 0.0;
        PT_ratio = 0.0;

        // 4 Delta Eta aver
        Delta_eta_aver = 0.0;

        // 5 Delta Phi others
        Delta_phi_sum = 0.0;
        Delta_phi_other_jets = 0.0;

        // 6 Delta PT leading
        PT_max = 0.0;
        Delta_PT_leading = 0.0;
        delta_PT_jet = 0.0; // If needed

        // 7 Delta Eta leading
        Delta_Eta_leading = 0.0;

        // Reset Var_values (Not necessary)
        for(Int_t ind = 0; ind < 8; ind++){
           var_values[ind] = 0.0;
           if (ind < dims) values[ind] = 0.0;
        }

        // Preliminary for. It is used to calculate
```

```
            PT_aver and Delta_phi_sum
        for (Int_t iJet = 0; iJet<NumJets; iJet++){
            currentJet = (Jet*) branchJet->At(iJet);
            vect_currentJet->SetPtEtaPhiM(currentJet->PT,currentJet->Eta,currentJet->Phi,currentJet->Mass);
            PT_sum += vect_currentJet->Pt();
            delta_phi = deltaAng(vect_currentJet->Phi(), METpointer->Phi);
            Delta_phi_sum += delta_phi;
            // PT Leading jet
            if(PT_max < vect_currentJet->Pt()){
                PT_max = vect_currentJet->Pt();
                posLeadingPT = iJet;
            }
        }

        numMatches++;

        //PT_aver
        PT_aver = PT_sum/NumJets;

        // Leading PT
        currentJet = (Jet*) branchJet->At(posLeadingPT);
        vect_leading->SetPtEtaPhiM(currentJet->PT,currentJet->Eta,currentJet->Phi,currentJet->Mass);

        for (Int_t iJet = 0; iJet<NumJets; iJet++){
            currentJet = (Jet*) branchJet->At(iJet);
            vect_currentJet->SetPtEtaPhiM(currentJet->PT,currentJet->Eta,currentJet->Phi,currentJet->Mass);

            // 2 Delta Phi MET
            delta_phi = deltaAng(vect_currentJet->Phi(), METpointer->Phi);

            // PT ratio
```

```
            PT_ratio = vect_currentJet->Pt()*(NumJets-1)
               /(PT_sum-vect_currentJet->Pt());

            // 4 Delta Eta Aver
            Delta_eta_aver = 0.0;
            // For cycle used to calculate
               Delta_eta_aver
            for(Int_t iJet2 = 0; iJet2<NumJets; iJet2++)
               {
               auxJet = (Jet*) branchJet->At(iJet2);
               vect_auxJet->SetPtEtaPhiM(auxJet->PT,
                  auxJet->Eta,auxJet->Phi,auxJet->Mass);
               if (iJet2 != iJet) Delta_eta_aver +=
                  TMath::Abs(vect_auxJet->Eta()-
                  vect_currentJet->Eta());
            }
            Delta_eta_aver = Delta_eta_aver/(NumJets-1);

            // 5 Delta Phi MET Others
            Delta_phi_other_jets = (Delta_phi_sum-
               delta_phi)/(NumJets-1);

            // 6 Delta PT leading
            Delta_PT_leading = vect_leading->Pt()-
               vect_currentJet->Pt();

            // 7 Delta Eta leading
            Delta_Eta_leading = TMath::Abs(
               vect_currentJet->Eta()-vect_leading->Eta
               ());

            // Other variables
            delta_PT_jet = TMath::Abs(vect_currentJet->
               Pt()-PT_aver);
            transverse_mass = sqrt(2*vect_currentJet->Pt
               ()*MET*(1-cos(delta_phi)));

            // Filling the array with the variables'
               values
            var_values[0] = vect_currentJet->Pt();
```

```
            var_values[1] = TMath::Abs(vect_currentJet->
               Eta());
            var_values[2] = delta_phi;
            var_values[3] = PT_ratio;
            var_values[4] = Delta_eta_aver;
            var_values[5] = Delta_phi_other_jets;
            var_values[6] = Delta_PT_leading;
            var_values[7] = Delta_Eta_leading;

            for (Int_t ind = 0; ind < dims; ind++){
               int pos = *(var_index+ind);
               values[ind] = *(var_values+pos);
            }

            if (iJet != ISR_jets[entry]){
               // Non ISR jet
               histoNonISR->fill(values);
            }
            else{
               // ISR jet
               histoISR->fill(values);
            }

         }
      }

      cout<<"\tprogress = 100%\t";
      cout<<"Time :"<< (clock()-initialTime)/double_t(
         CLOCKS_PER_SEC)<<"s"<<endl;
      cout<<"\n\tNumber of Written Events: "<<numMatches
         <<endl;
   } // End run's for cicle

   /*
    * Writing the histogram
    */
   // Counting time
   Double_t partialTime = clock();
   cout<< "Time building the histogram:"<< (partialTime-
      initialTime)/double_t(CLOCKS_PER_SEC)<<"s"<<endl;
```

```

    // Writing the histogram
    cout<<"Min value 1: "<<min_Values[0]<<endl;
    Int_t* freq;
    for(Int_t j = 0; j<dims; j++){
          cout<<"ISR Jets - Events of the dimension:\t"<<
             j<<endl;
          freq = histoISR->getHistDim(j);
          for(Int_t i = 0; i<bins[j];i++){
                cout<<"\t"<<freq[i];
                if(i>0 && ((i+1)%10 == 0.0))
                      cout<<endl;
          }
          cout<<endl;
    }

    cout<<endl<<"\t\t ***"<<endl<<endl;
    for(Int_t j = 0; j<dims; j++){
          cout<<"Non ISR Jets - Events of the dimension:\
             t"<<j<<endl;
          freq = histoNonISR->getHistDim(j);
          for(Int_t i = 0; i<bins[j];i++){
                cout<<"\t"<<freq[i];
                if(i>0 && ((i+1)%10 == 0.0))
                      cout<<endl;
          }
          cout<<endl;
    }

    cout<<"Entries: "<<histoISR->getEntries()<<endl;

    /*
     * Creating histograms
     */
    cout<<"\nWriting..."<<endl;

    // Defining the names of the files
    string combination = "______"; // Combination of
       variables
    for (Int_t ind = 0; ind < dims; ind ++){
```

```
        combination[(ind*2)+1] = (Char_t) (0x30 + var_index[ind]); // Int to char
    }

    string info_ISR_name_str = head_folder_results + "info_histo_ISR" + combination + ".txt";
    Char_t *info_ISR_name = (Char_t *) info_ISR_name_str.c_str();

    string array_ISR_name_str = head_folder_results + "array_histo_ISR" + combination + ".bn";
    Char_t *array_ISR_name = (Char_t *) array_ISR_name_str.c_str();

    string info_Non_ISR_name_str = head_folder_results + "info_histo_Non_ISR" + combination + ".txt";
    Char_t *info_Non_ISR_name = (Char_t *) info_Non_ISR_name_str.c_str();

    string array_Non_ISR_name_str = head_folder_results + "array_histo_Non_ISR" + combination + ".bn";
    Char_t *array_Non_ISR_name = (Char_t *) array_Non_ISR_name_str.c_str();

    cout << "Output files:\n\t" << info_ISR_name << "\n\t" << array_ISR_name << "\n\t" << info_Non_ISR_name << "\n\t" << array_Non_ISR_name << endl;

    histoISR->writeClass((Char_t*) info_ISR_name,(Char_t*) array_ISR_name);
    histoNonISR->writeClass((Char_t*) info_Non_ISR_name,(Char_t*)array_Non_ISR_name);
    cout<< "Time writing the file:"<< (clock()-partialTime)/double_t(CLOCKS_PER_SEC)<<"s"<<endl;

    cout<<"Fin :)"<<endl;

    return 0;
}
```

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.