

Documentation of the project: ISR jet tagging

Author:

Andrés Felipe García Albarracín

Advisor:

Juan Carlos Sanabria, Ph.D.

July 6, 2015

Contents

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

3.2.2	From probability density functions to normalized histograms	22
3.2.3	The Algorithm	23
3.2.4	The code	25
3.3	Matching algorithm	26
3.4	ISR jet analysis code	28
3.5	Histograms code	29
3.6	Example of the usage of the codes	30
4	Software tools	31
4.1	C++ and ROOT Makefile	31
4.2	Using Eclipse	33
	Appendices	36
A	Simulation codes	37
A.1	Pythia code	37
A.2	Integration scripts	43
A.2.1	Configuration script: <code>config_Integration.ini</code>	43
A.2.2	Execution script: <code>script_Integration.sh</code>	45
B	Analysis codes	49
B.1	Tagging algorithm	49
B.2	Matching algorithm	73

<i>CONTENTS</i>	iii
B.3 ISR jet analysis	84
B.4 Creating histograms	112
Bibliography	127

Chapter 1

Introduction

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

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

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

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

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

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

Chapter 2

Simulation chain

“Divide et impera”,
“Divide and conquer”

Philip II of Macedon

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

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

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

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

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

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

2.1 Usage of MadGraph 5.2

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

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

been correctly installed ².

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

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

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

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

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

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

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

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

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

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

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

2.2 Usage of Pythia 8.2

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

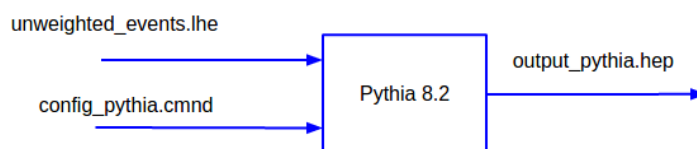


Figure 2.1

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

2.2.1 Code Usage

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

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

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

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

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

```

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

```

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

```

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

```

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

```
make hadronization02
```

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

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

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

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

```

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

```

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

2.2.2 The code

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

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

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

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

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

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

2.2.3 Pythia ntuple generation

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

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

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

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

2.3 Usage of Delphes 3.2

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

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

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

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

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

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

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

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

2.4 Integration of MadGraph 5.2 + Pythia 8.2 + Delphes 3.2

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

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

```
./script_Integration.sh
```

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

⁸d-sanz@uniandes.edu.co

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

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

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

```

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

```

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

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

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

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

2.5 Example of the integration scripts

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

1. Install the three programs and compile the code `hadronization02` of Pythia.
2. Download the folder `MG_pythia8_delphes_parallel` from the repository of the project.
3. Open the file `config_Integration.ini` and write all the installation folders in front of the corresponding variables. Use the path of the downloaded folder `RunCard_Template` as the directory of `RUNCARDFOLDER`, `MADGRAPHFILEFOLDER`, `PYTHIAPARAMFOLDER` and `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 6500GeV in lines 41 and 42.
- (c) In the file `input_pythia.cmd`, observe the same parameters presented in subsection 2.2.1. Additionally, the file includes some necessary settings to perform the *matching* procedure between MadGraph and Pythia. More information about it can be found at [7].

6. Execute the script by typing¹⁰:

```
./script_Integration.sh
```

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

Chapter 3

Analysis codes

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.

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`

Some important parameters of each program are defined in the form of global variables at the beginning of the corresponding code (lines 46 - 57). These parameters are not supposed to be modified frequently but are easy to change if necessary. A brief description of them is now presented:

- Variable `channel` is used to select if the channel under analysis corresponds to tops' or stops' production.
- `ISR_or_NOT` defines if the simulation presents or not an ISR jet.
- `Matching` is a boolean variable that should be set true if a matching has been performed between MadGraph and Pythia [7].
- Similar variables to those of the previous items exist for the histograms' files. (Those histograms will be explained soon). They specify the channel of the simulations performed to fill the histograms and if the matching procedure has been done in those simulations.
- Because sometimes I worked at the server and others at my pc, I used `atServer` to change easily between them. By toggling this variable, the user specifies the head folders where the histograms' files, the simulation for analysis and the matching results of such simulation are located. Furthermore, it also controls where will be the location of the tagging results.

All these variables are important as they allow handling with different simulations easily. However, this needs that the names of the folders as well as the name of the files follow a strict convention. In Table 3.1 the convention used to name files and folders is presented. A few rules should be taken into account when checking the name structure presented in Table 3.1:

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

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

Table 3.1: Naming convention of folders and files

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

Take into account these rules for managing files produced and read by the programs. Feel free to change this convention but remember that it should be changed in all codes. Other details to execute each program will be explained in the following sections, where additionally, the functionalities of each program are presented.

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 an event, in case it exists. Because of its importance, a complete explanation is presented below.

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.

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

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

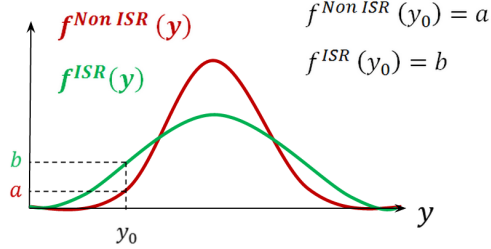


Figure 3.1: Probability distributions of a variable that distinguishes between ISR and Non ISR jets

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

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

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

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

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

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

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

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

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

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

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

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

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

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

3.2.2 From probability density functions to normalized histograms

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

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

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

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

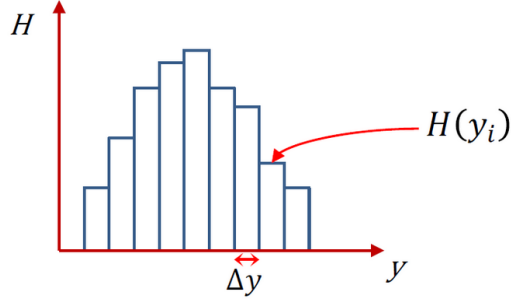


Figure 3.2: Shape of a histogram which does not exactly correspond with the probability density function

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

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

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

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

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

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

3.2.3 The Algorithm

Once the method has been prepared by selecting the distinguishing variables and by filling the histograms, the algorithm of Fig. 3.3 is applied for each

event. First, each jet in the event is studied and its probabilities of being ISR and Non ISR are determined from its kinematical variables and expressions 3.9 and 3.10.

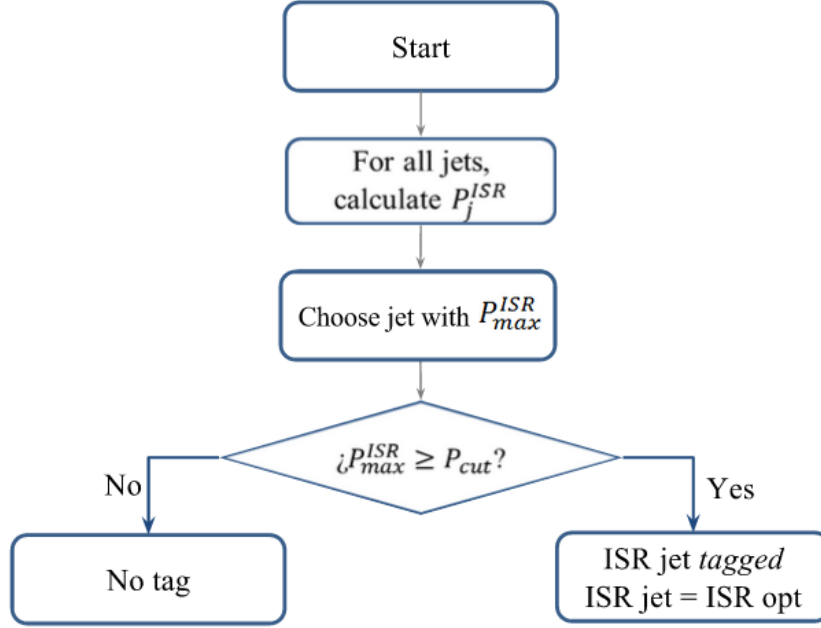


Figure 3.3: ISR jet tagging algorithm

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

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

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

3.2.4 The code

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

```
./ISR_tagging [N1] [N2] [N3] [pt_cut] [k_cut]
```

where all the parameters that follow `./ISR_tagging` 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.

On the other hand, the last two variables are used to perform an analysis of the tagging results. After executing the tagging algorithm with a probability cut `k_cut`, 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.

Other important parameters of this code are the global variables mentioned in section 3.1. You can change them according to the rules presented before and compile the code again to use those modifications. Additionally, the tagging code allows the analysis of several runs, which is possible to control by means of the *for* loop of line 254.

Other technical details of the tagging program can be found in the comments of the code.

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

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

3.3 Matching algorithm

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

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

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

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

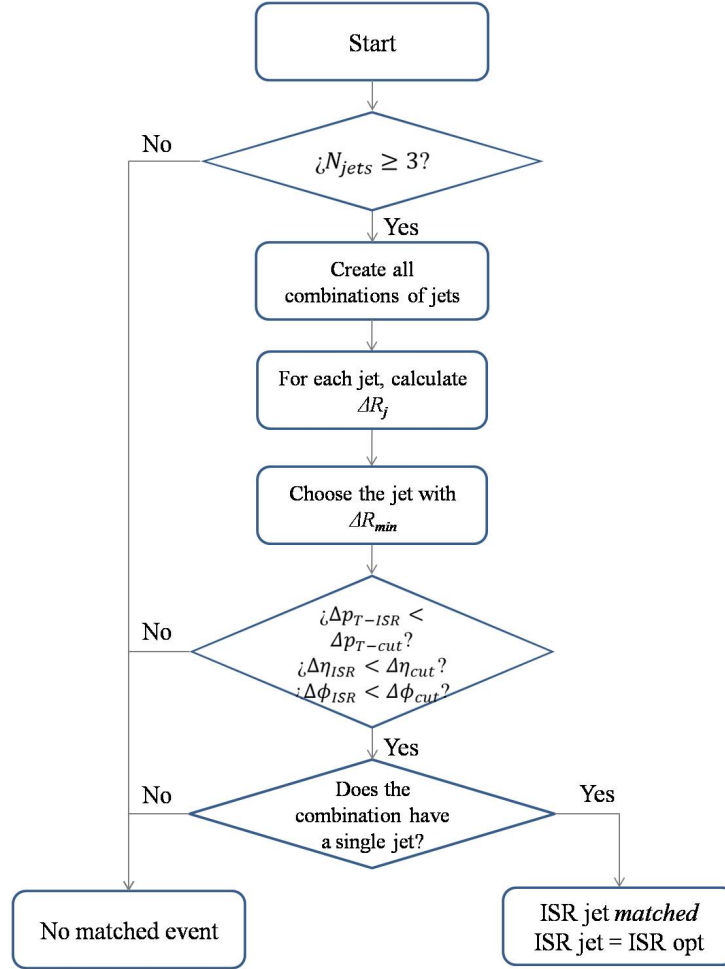


Figure 3.4: Matching algorithm between MadGraph and Pythia

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

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

```
./ISR_matching [000]
```

where the last three digits are optional and correspond to the number of

the simulation (its seed) to which the user wants to execute the matching. If no parameter is written, the simulation for analysis has seed 003.

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

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

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

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

3.4 ISR jet analysis code

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

The program takes a group of simulations and their corresponding matching results as inputs. Then, it creates histograms⁴ of kinematic variables and compares the distributions of ISR and Non ISR jets. To do so, several

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

³Check the structure of the name in section 3.1

⁴Root TH1 histograms

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.

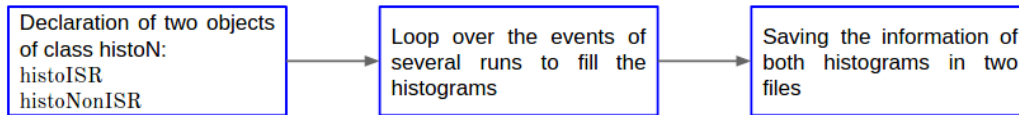


Figure 3.5: Procedure to create N-dimensional histograms

The compilation of the code is achieved by following the instructions of section 3.1. To execute the code, type:

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

where the parameters at the end are optional and correspond again 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

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 to any project which requires the usage of ROOT libraries and C++ programs.

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 li-

braries. 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 libraries ¹,
- `DelphesFunctions.h` with Delphes libraries.

5. In order 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

¹Uncomment line 41 in case you do not use 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. In this section it will presented how to configure 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. 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.
```
2. Uncompress the downloaded file:

```
tar -xzf eclipse-cpp-luna-SR2-linux-gtk-x86_64.tar.gz
```
3. Inside the new `eclipse` folder, open Eclipse. Type `./eclipse`
4. When asked to select a workspace, choose a directory where you want to save your future codes.
5. On the new window, select the option *Workbench*.
6. Create a new C++ project. Go to **File > New > C++ Project**
7. In the new window, change the project name, the project type, the Toolchains as illustrated in Fig.4.1. Then click on **Next**.
8. In the New Window, modify the author and any other comment as Fig. 4.2, for instance. Then click on **Next**.
9. In the last window, select **Debug** and **Release**. Then click on finish.

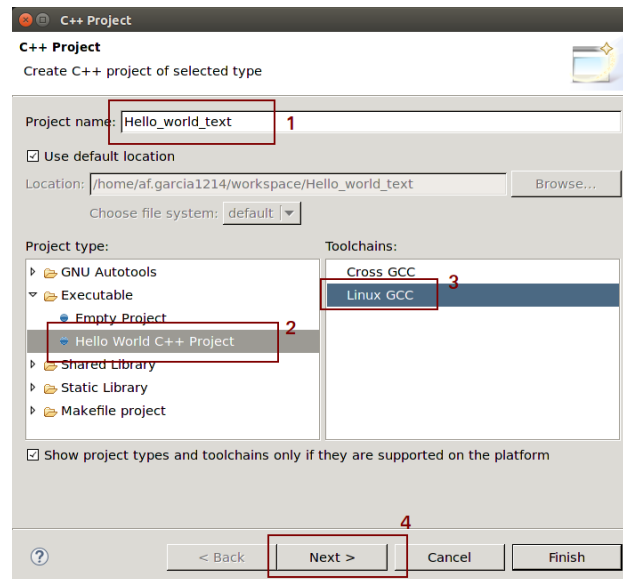


Figure 4.1: Configuration of a Hello-world program in Eclipse

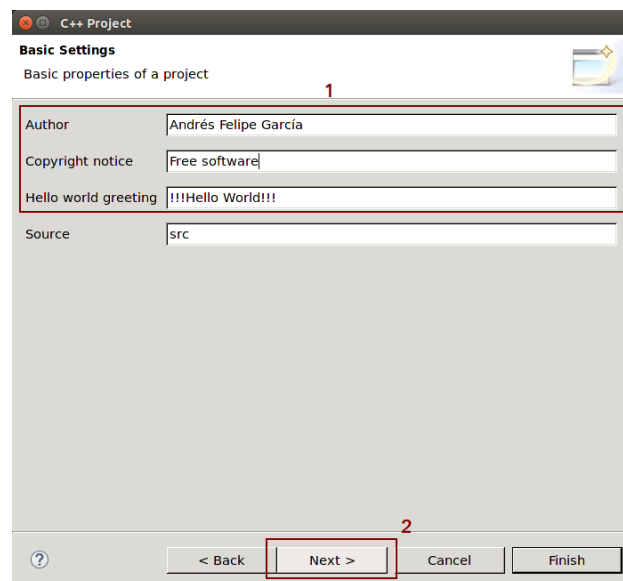
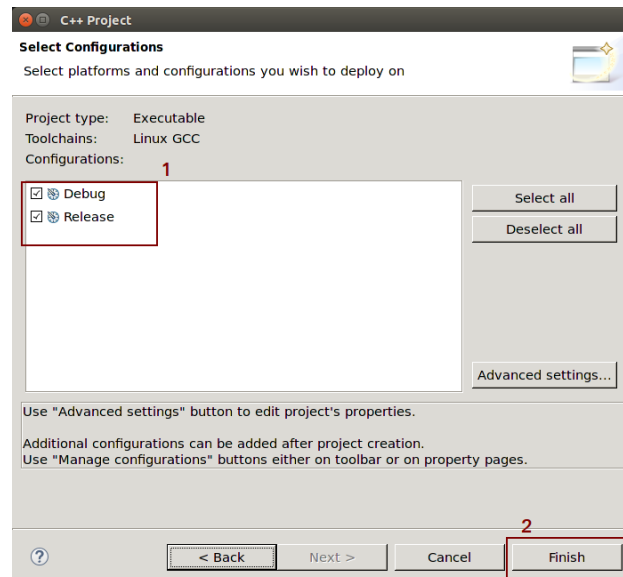


Figure 4.2: Setting the author in a eclipse project

Figure 4.3: Setting *release* and *debug* in eclipse

Appendices

Appendix A

Simulation codes and scripts

A.1 Pythia code: hadronization02.cc

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

```

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

```

```

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

```

```

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

```

```

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

```

158 }



A.2 Integration scripts: MadGraph + Pythia + Delphes

A.2.1 Configuration script: config_Integration.ini

```

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

```

```

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

```



```

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

```

A.2.2 Execution script: script_Integration.sh

```

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

```

```

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

```

```

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

```

```

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

```

Appendix B

Analysis codes

B.1 Tagging algorithm

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

```

22 6. Delta Phi_MET_others
23 7. Delta PT_others
24 8. Delta Eta_others
25
26 In order to choose them, the code should be run as
27 ./ISR_tagging N1 N2 N3, where N1 N2 and N3 are
28 the index of the 3 variables. If no parameter is
29 passed as parameter, N1 N2 and N3 will be 0,1 and 2
30 by default.
31
32 Additionally, the user can define a pt_cut and
33 probability cut k_cut to study the behavior of the
34 algorithm in a certain pt selection and to check
35 the MET boosting. In such case, the code should be
36 run as ./ISR_tagging N1 N2 N3 pt_cut k_cut
37 */
38
39
40 #include "ROOTFunctions.h"
41 #include "graphs_Funcs.h"
42 #include "functions.h"
43 #include "histoN.h"
44 #include "DelphesFunctions.h"
45
46 // Global Variables
47 const Double_t PI = TMath::Pi();
48
49 // Other simulations parameters
50 const Char_t channel = '_'; // 's' for sTops and '_' for
    Tops
51 const Char_t ISR_or_NOT[] = "WI"; // "WI" with ISR, "SI"
    without (Here it does not make any sense), "bb"
    bjets production
52 const Bool_t Matching = true; // True if a matching has
    been done between MG and Pythia, false otherwise
53
54 const Char_t channel_histo = '_'; // 's' for sTops and '
    _' for Tops (Which channel fills the histogram)
55 const Char_t ISR_or_NOT_histo[] = "WI"; // "WI" with ISR
    , "SI" without (Here it does not make any sense), "bb

```

```

    " bjets production (Which channel fills the
    histogram)
56 const Bool_t Matching_histo = true; // True if a
    matching has been done between MG and Pythia, false
    otherwise
57 const Bool_t atServer = true; // True if it is run at
    the server, false at the university's pc
58
59 int main(int argc, char **argv){
60     std::cout.precision(4);
61     // Counting time
62     Double_t initialTime = clock();
63     Double_t pt_cut = 0.0;
64     Double_t Jet_cut = 2;
65
66     cout << "\n *** Running the tagging Algorithm *** \n"
        << endl;
67
68     // Variables for initializing histograms
69     Int_t dims = 3;
70
71     /*
72     * Read inputs and set variables for analysis
73     */
74     Int_t var_index[3] = {0,1,2}; // Index of the 3
        variables for analysis. By default 0, 1 and 2
75     string variables[8] = {"PT","Abs(Eta)","Delta Phi_MET
        ","PT_ratio","Delta Eta_aver","Delta
        Phi_MET_others","Delta PT_leading","Delta
        Eta_leading"};
76     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
77
78
79     if (argc == 1) {
80         cout << "Running the algorithm with the default
            variables:" << endl;
81     }
82

```

```

83     if (argc >= 4){
84         cout << "Running the algorithm with the variables:
            " << endl;
85         for (Int_t ind = 0; ind < 3; ind ++){
86             var_index[ind] = atoi(argv[ind+1]);
87         }
88     }
89     if (argc >= 5) {
90         pt_cut = atof(argv[4]);
91     }
92     if (argc >= 6) {
93         Jet_cut = atof(argv[5]);
94     }
95
96     if ((argc >= 7) || (argc < 4 && argc > 1)) {
97         cout << "Error at calling this algorithm. Use as:"
            << endl;
98         cout << "\t ./ISR_tagging N1 N2 N3 [Pt_cut] [K_cut
            ] or just ./ISR_tagging" << endl;
99         cout << "Read the documentation at the beginning
            of the code for further information\n" << endl;
100        return 1;
101    }
102
103    cout << "Transverse momentum of the ISR: " << pt_cut
        << endl;
104
105    cout << "Var \t\t min_Value \t max_Value" << endl;
106    for (Int_t ind = 0; ind < 3; ind ++){
107
108        cout << var_index[ind] << ". " << variables[
            var_index[ind]] << endl;
109    }
110    cout << endl;
111
112    /*
113     * Initializing the 3-dimensional histogram
114     */
115    // Defining the names of the files
116    Char_t combination[] = "_____"; // Combination of

```



```

variables
117 for (Int_t ind = 0; ind < dims; ind++){
118     *(combination+(ind*2)+1) = (Char_t) (0x30 +
        var_index[ind]); // Int to char
119 }
120
121 Char_t *local_path_histos;
122 local_path_histos = (Char_t*) malloc(512*sizeof(
    Char_t));
123 if (atServer)
124     strcpy(local_path_histos, "/home/af.garcia1214/
        PhenoMCsamples/Results/histo_folder/"); // At
        the server
125 else
126     strcpy(local_path_histos, "/home/afgarcia1214/
        Documentos/Results_and_data/histo_folder/"); //
        At the University's pc
127
128 Char_t *head_folder_histos;
129 head_folder_histos = (Char_t*) malloc(512*sizeof(
    Char_t));
130 if (Matching_histo)
131     strcpy(head_folder_histos, "
        _Tops_histos_WI_Matching/");
132 else
133     strcpy(head_folder_histos, "_Tops_histos_WI/");
134 head_folder_histos[0] = channel_histo;
135 head_folder_histos[13] = ISR_or_NOT_histo[0];
136 head_folder_histos[14] = ISR_or_NOT_histo[1];
137
138 Char_t *info_ISR_name;
139 info_ISR_name = (Char_t*) malloc(sizeof(char)*512);
140 strcpy(info_ISR_name, local_path_histos);
141 strcat(info_ISR_name, head_folder_histos);
142 strcat(info_ISR_name, "info_histo_ISR");
143 strcat(info_ISR_name, combination);
144 strcat(info_ISR_name, ".txt");
145
146 Char_t *array_ISR_name;
147 array_ISR_name = (Char_t*) malloc(sizeof(char)*512);

```

```

148     strcpy(array_ISR_name, local_path_histos);
149     strcat(array_ISR_name, head_folder_histos);
150     strcat(array_ISR_name, "array_histo_ISR");
151     strcat(array_ISR_name, combination);
152     strcat(array_ISR_name, ".bn");
153
154     Char_t *info_Non_ISR_name;
155     info_Non_ISR_name = (Char_t*) malloc(sizeof(char)
156         *512);
157     strcpy(info_Non_ISR_name, local_path_histos);
158     strcat(info_Non_ISR_name, head_folder_histos);
159     strcat(info_Non_ISR_name, "info_histo_Non_ISR");
160     strcat(info_Non_ISR_name, combination);
161     strcat(info_Non_ISR_name, ".txt");
162
163     Char_t *array_Non_ISR_name;
164     array_Non_ISR_name = (Char_t*) malloc(sizeof(char)
165         *512);
166     strcpy(array_Non_ISR_name, local_path_histos);
167     strcat(array_Non_ISR_name, head_folder_histos);
168     strcat(array_Non_ISR_name, "array_histo_Non_ISR");
169     strcat(array_Non_ISR_name, combination);
170     strcat(array_Non_ISR_name, ".bn");
171
172     histoN* histoISR = new histoN(info_ISR_name,
173         array_ISR_name);
174     histoN* histoNonISR = new histoN(info_Non_ISR_name,
175         array_Non_ISR_name);
176
177     cout << "Entradas ISR: " << histoISR->getEntries() <<
178         endl;
179     cout << "Entradas FSR: " << histoNonISR->getEntries()
180         << endl;
181
182     // Input variables of each histogram
183     Double_t values[3] = {0.0, 0.0, 0.0};
184
185     /*
186     * MET histograms
187     */

```

```

182     TH1 *h_MET = new TH1F("Missing ET","All events"
        ,300,0,2000);
183     Char_t *name_histo_MET;
184     name_histo_MET = (Char_t*) malloc(sizeof(char)*512);
185     strcpy(name_histo_MET,"ISR jet PT > ");
186     Char_t pt_str[] = " ";
187     pt_str[0] = 0x30 + int(pt_cut/100)%10;
188     pt_str[1] = 0x30 + int(pt_cut/10)%10;
189     pt_str[2] = 0x30 + int(pt_cut)%10;
190     strcat(name_histo_MET,pt_str);
191     strcat(name_histo_MET,"-k = ");
192     Char_t k_str[] = " ";
193     k_str[0] = 0x30 + int(Jet_cut)%10;
194     k_str[1] = '.';
195     k_str[2] = 0x30 + int(Jet_cut*10)%10;
196     strcat(name_histo_MET,k_str);
197     TH1 *h_MET_hpt1 = new TH1F(name_histo_MET,"Missing ET
        high_ISR_pt-1",300,0.0,2000);
198
199     if (argc == 6)
200         cout << "The algorithm will evaluate the MET for a
            sample with PT > " << pt_str << " at k = " <<
            k_str << endl;
201     /*
202     * Tagging variables
203     */
204
205     cout << "Jet cut, k = " << Jet_cut << endl;
206
207     // Arrays with the number of tags, Misstags and
        events rejected
208     // Probability cut
209     Double_t Prob_cut = 0;
210     Double_t k_min = 1.2; // Minimum probability cut =
        k_min/num_jets
211     Double_t k_max = 3.0; // Maximum probability cut =
        k_max/num_jets
212     Int_t k_bins = 100; // Number of values of k between
        k_min and k_max
213     Double_t k_step = (Double_t) (k_max-k_min)/k_bins;

```

```

214     Double_t k_values[k_bins];
215     for(Int_t ind = 0; ind < k_bins; ind++){
216         k_values[ind] = k_min + k_step*ind;
217     }
218
219     // Tagging results
220     Int_t Num_Tags = 0;
221     Int_t Num_MissTags = 0;
222     Int_t Num_Rejected = 0;
223
224     Double_t Num_Tags_array[k_bins];
225     Double_t Num_MissTags_array[k_bins];
226     Double_t Num_Rejected_array[k_bins];
227     Double_t Num_Total_Jets[k_bins];
228
229     Double_t Num_Tags_array_hpt[k_bins];
230     Double_t Num_MissTags_array_hpt[k_bins];
231     Double_t Num_Rejected_array_hpt[k_bins];
232     Double_t Num_Total_Jets_hpt[k_bins];
233
234
235     for (Int_t ind = 0; ind < k_bins; ind++){
236         Num_Tags_array[ind] = 0;
237         Num_MissTags_array[ind] = 0;
238         Num_Rejected_array[ind] = 0;
239         Num_Total_Jets[ind] = 0;
240         Num_Tags_array_hpt[ind] = 0;
241         Num_MissTags_array_hpt[ind] = 0;
242         Num_Rejected_array_hpt[ind] = 0;
243         Num_Total_Jets_hpt[ind] = 0;
244     }
245
246     // Variables of the ISR tagging algorithm
247     Double_t H_ISR, H_Non_ISR, alpha;
248     Double_t prob_max = 0;
249     Double_t probISR = 0;
250     Double_t k_ISR = 0;
251     Double_t k_ISR_pos = 0; // Position of the ISR in the
                             // vector
252     Int_t ISR_tag_index = -1;

```

```

253
254 // Cycle over several runs. iRun corresponds to the
    // seed of the current run
255 for(int iRun = 1; iRun < 11; iRun ++){
256     // Create chains of root trees
257     TChain chain_Delphes("Delphes");
258
259     // Loading simulations from Delphes
260     Char_t *local_path;
261     local_path = (Char_t*) malloc(512*sizeof(Char_t));
262     if (atServer)
263         strcpy(local_path, "/home/af.garcia1214/
            PhenoMCsamples/Simulations/
            MG_pythia8_delphes_parallel/"); // At the
            server
264     else
265         strcpy(local_path, "/home/afgarcia1214/
            Documentos/Simulations/"); // At the
            University's pc
266
267     Char_t *head_folder;
268     head_folder = (Char_t*) malloc(512*sizeof(Char_t))
        ;
269     if (Matching)
270         strcpy(head_folder, "_Tops_Events_WI_Matching/");
        ;
271     else
272         strcpy(head_folder, "_Tops_Events_WI/");
273     head_folder[0] = channel;
274     head_folder[13] = ISR_or_NOT[0];
275     head_folder[14] = ISR_or_NOT[1];
276
277     Char_t current_folder[] = "_Tops_MG_1K_AG_WI_003/"
        ;
278     current_folder[0] = channel;
279     current_folder[15] = ISR_or_NOT[0];
280     current_folder[16] = ISR_or_NOT[1];
281
282     Char_t unidad = 0x30 + iRun%10;
283     Char_t decena = 0x30 + int(iRun/10)%10;

```

```

284         Char_t centena = 0x30 + int(iRun/100)%10;
285
286         current_folder[18] = centena;
287         current_folder[19] = decena;
288         current_folder[20] = unidad;
289
290         Char_t *file_delphes;
291         file_delphes = (Char_t*) malloc(512*sizeof(Char_t)
292                                     );
293         strcpy(file_delphes,local_path);
294         strcat(file_delphes,head_folder);
295         strcat(file_delphes,current_folder);
296         strcat(file_delphes,"Events/run_01/output_delphes.
297                 root");
298
299         cout << "Studying run: " << centena << decena <<
300                 unidad << endl;
301         cout << "\nReading the file: \nDelphes: " <<
302                 file_delphes << endl;
303
304         chain_Delphes.Add(file_delphes);
305         // Objects of class ExRootTreeReader for reading
306         // the information
307         ExRootTreeReader *treeReader_Delphes = new
308                 ExRootTreeReader(&chain_Delphes);
309
310         Long64_t numberOfEntries = treeReader_Delphes->
311                 GetEntries();
312
313         // Get pointers to branches used in this analysis
314         TClonesArray *branchJet = treeReader_Delphes->
315                 UseBranch("Jet");
316         TClonesArray *branchMissingET = treeReader_Delphes
317                 ->UseBranch("MissingET");
318
319         cout << endl;
320         cout << " Number of Entries Delphes = " <<
321                 numberOfEntries << endl;
322         cout << endl;
323

```

```

314 // particles, jets and vectors
315 MissingET *METpointer;
316 TLorentzVector *vect_currentJet = new
    TLorentzVector;
317 TLorentzVector *vect_auxJet = new TLorentzVector;
318 TLorentzVector *vect_leading = new TLorentzVector;
319 Jet *currentJet = new Jet;
320 Jet *auxJet = new Jet;
321
322 // Temporary variables
323 Double_t MET = 0.0; // Missing transverse energy
324 Double_t delta_phi = 0.0; // difference between
    the phi angle of MET and the jet
325 Double_t transverse_mass = 0.0; // Transverse mass
326 Double_t delta_PT_jet = 0.0; // |PT-<PT>|
327 Double_t PT_sum = 0.0; // sum(PT)
328 Double_t PT_aver = 0.0; // <PT>
329 Double_t Delta_eta_aver = 0.0; // sum_i|eta-eta_i
    |/(Nj-1)
330 Double_t Delta_phi_sum = 0.0; // sum delta_phi
331 Double_t Delta_phi_other_jets = 0.0; // Average of
    delta phi of other jets
332 Double_t PT_ratio = 0.0; // PT/PT_others
333 Double_t Delta_PT_leading = 0.0; // PT -
    PT_leading
334 Double_t Delta_Eta_leading = 0.0; // |Eta -
    Eta_leading|
335
336 // Jet with greatest PT
337 Double_t PT_max = 0;
338 Int_t posLeadingPT = -1;
339 Int_t ISR_greatest_PT = 0;
340 Double_t MT_leading_jet = 0.0; // Transverse mass
341
342 /*
343  * Some variables used through the code
344  */
345 Int_t ISR_jets[numberOfEntries];
346 Int_t NumJets = 0;
347

```

```

348     Char_t *local_path_binary;
349     local_path_binary = (Char_t*) malloc(512*sizeof(
        Char_t));
350     if (atServer)
351         strcpy(local_path_binary, "/home/af.garcia1214/
            PhenoMCsamples/Results/matching_Results/");
            // At the server
352     else
353         strcpy(local_path_binary, "/home/afgarcia1214/
            Documentos/Results_and_data/matching_Results
            /"); // At the University's pc
354
355     Char_t *head_folder_binary;
356     head_folder_binary = (Char_t*) malloc(512*sizeof(
        Char_t));
357     if (Matching)
358         strcpy(head_folder_binary, "
            _Tops_matches_WI_Matching/");
359     else
360         strcpy(head_folder_binary, "_Tops_matches_WI/");
361     head_folder_binary[0] = channel;
362     head_folder_binary[13] = ISR_or_NOT[0];
363     head_folder_binary[14] = ISR_or_NOT[1];
364
365     Char_t matching_name[] = "ISR_jets_Tops_WI_003.bn"
        ;
366     matching_name[8] = channel;
367     matching_name[14] = ISR_or_NOT[0];
368     matching_name[15] = ISR_or_NOT[1];
369
370     matching_name[17] = centena;
371     matching_name[18] = decena;
372     matching_name[19] = unidad;
373
374     Char_t * fileName;
375     fileName = (Char_t*) malloc(512*sizeof(Char_t));
376     strcpy(fileName, local_path_binary);
377     strcat(fileName, head_folder_binary);
378     strcat(fileName, matching_name);
379

```



```

380     if (ISR_or_NOT[0] != 'S'){ // != S means bb or WI
381         ifstream ifs(fileName,ios::in | ios::binary);
382
383         for (Int_t j = 0; j<numberOfEntries; j++){
384             ifs.read((Char_t *) (ISR_jets+j),sizeof(
385                 Int_t));
386         }
387         ifs.close();
388     }
389     else if (ISR_or_NOT[0] == 'S'){
390         for (Int_t j = 0; j<numberOfEntries; j++){
391             ISR_jets[j] = -2; // There is not ISR jet
392                                 but also there is not matching
393         }
394     }
395 }
396
397 /*
398  * Main cycle of the program
399  */
400 numberOfEntries = 100000;
401 for (Int_t entry = 0; entry < numberOfEntries; ++
402     entry){
403     // Progress
404     if(numberOfEntries>10 && (entry%((int)
405         numberOfEntries/10))==0.0){
406         cout<<"progress = "<<(entry*100/
407             numberOfEntries)<<"%\t";
408         cout<< "Time : "<< (clock()-initialTime)/
409             double_t(CLOCKS_PER_SEC)<<"s"<<endl;
410     }
411
412     // Load selected branches with data from
413     specified event
414     treeReader_Delphes->ReadEntry(entry);
415
416     // MET
417     METpointer = (MissingET*) branchMissingET->At
418         (0);
419     MET = METpointer->MET;
420 }

```

```

412     NumJets=branchJet->GetEntries();
413
414     // checking the ISR
415     if (NumJets < 3 || ISR_jets[entry] == -1)
416         continue;
417
418     h_MET->Fill(MET);
419
420     if (ISR_jets[entry] >= NumJets){
421         cout << "Error en el matching" << endl;
422         return 1;
423     }
424
425     // 3 PT ratio
426     PT_aver = 0.0;
427     PT_sum = 0.0;
428     PT_ratio = 0.0;
429
430     // 4 Delta Eta aver
431     Delta_eta_aver = 0.0;
432
433     // 5 Delta Phi others
434     Delta_phi_sum = 0.0;
435     Delta_phi_other_jets = 0.0;
436
437     // 6 Delta PT leading
438     PT_max = 0.0;
439     Delta_PT_leading = 0.0;
440     delta_PT_jet = 0.0; // If needed
441
442     // 7 Delta Eta leading
443     Delta_Eta_leading = 0.0;
444
445     // Reset Var_values (Not necessary)
446     for(Int_t ind = 0; ind < 8; ind++){
447         var_values[ind] = 0.0;
448         if (ind < dims) values[ind] = 0.0;
449     }
450
451     // Preliminary for. It is used to calculate

```

```

PT_aver and Delta_phi_sum
452   for (Int_t iJet = 0; iJet<NumJets; iJet++){
453       currentJet = (Jet*) branchJet->At(iJet);
454       vect_currentJet->SetPtEtaPhiM(currentJet->PT
           ,currentJet->Eta,currentJet->Phi,
           currentJet->Mass);
455       PT_sum += vect_currentJet->Pt();
456       delta_phi = deltaAng(vect_currentJet->Phi(),
           METpointer->Phi());
457       Delta_phi_sum += delta_phi;
458       // PT Leading jet
459       if(PT_max < vect_currentJet->Pt()){
460           PT_max = vect_currentJet->Pt();
461           posLeadingPT = iJet;
462       }
463   }
464
465   //PT_aver
466   PT_aver = PT_sum/NumJets;
467
468   // Leading PT
469   currentJet = (Jet*) branchJet->At(posLeadingPT)
           ;
470   vect_leading->SetPtEtaPhiM(currentJet->PT,
           currentJet->Eta,currentJet->Phi,currentJet->
           Mass);
471
472   // The best ISR candidate
473   TLorentzVector *vect_optimum = new
           TLorentzVector;
474
475   // Reset variables
476   probISR = 0.0;
477   k_ISR = 0.0;
478   prob_max = 0;
479   ISR_tag_index = -1;
480
481   for (Int_t iJet = 0; iJet<NumJets; iJet++){
482       currentJet = (Jet*) branchJet->At(iJet);
483       vect_currentJet->SetPtEtaPhiM(currentJet->PT

```

```

, currentJet->Eta, currentJet->Phi,
currentJet->Mass);
484
485 // 2 Delta Phi MET
486 delta_phi = deltaAng(vect_currentJet->Phi(),
METpointer->Phi);
487
488 // PT ratio
489 PT_ratio = vect_currentJet->Pt()*(NumJets-1)
/(PT_sum-vect_currentJet->Pt());
490
491 // 4 Delta Eta Aver
492 Delta_eta_aver = 0.0;
493 // For cycle used to calculate
Delta_eta_aver
494 for(Int_t iJet2 = 0; iJet2<NumJets; iJet2++)
{
495 auxJet = (Jet*) branchJet->At(iJet2);
496 vect_auxJet->SetPtEtaPhiM(auxJet->PT,
auxJet->Eta, auxJet->Phi, auxJet->Mass);
497 if (iJet2 != iJet) Delta_eta_aver +=
TMath::Abs(vect_auxJet->Eta()-
vect_currentJet->Eta());
498 }
499 Delta_eta_aver = Delta_eta_aver/(NumJets-1);
500
501 // 5 Delta Phi MET Others
502 Delta_phi_other_jets = (Delta_phi_sum-
delta_phi)/(NumJets-1);
503
504 // 6 Delta PT leading
505 Delta_PT_leading = vect_leading->Pt()-
vect_currentJet->Pt();
506
507 // 7 Delta Eta leading
508 Delta_Eta_leading = TMath::Abs(
vect_currentJet->Eta()-vect_leading->Eta
());
509
510 // Other variables

```

```

511     delta_PT_jet = TMath::Abs(vect_currentJet->
512         Pt()-PT_aver);
513     transverse_mass = sqrt(2*vect_currentJet->Pt
514         ()*MET*(1-cos(delta_phi)));
515
516     // Filling the array with the variables'
517     // values
518     var_values[0] = vect_currentJet->Pt();
519     var_values[1] = TMath::Abs(vect_currentJet->
520         Eta());
521     var_values[2] = delta_phi;
522     var_values[3] = PT_ratio;
523     var_values[4] = Delta_eta_aver;
524     var_values[5] = Delta_phi_other_jets;
525     var_values[6] = Delta_PT_leading;
526     var_values[7] = Delta_Eta_leading;
527
528     for (Int_t ind = 0; ind < dims; ind++){
529         int pos = *(var_index+ind);
530         values[ind] = *(var_values+pos);
531     }
532
533     // Comparing with histos
534     H_ISR = histoISR->getProbVal(values);
535     H_Non_ISR = histoNonISR->getProbVal(values);
536
537     if (H_ISR > 3e-7 || H_Non_ISR > 3e-7){
538         alpha = NumJets/(H_Non_ISR*(NumJets-1)+
539             H_ISR);
540         probISR = alpha*H_ISR/NumJets;
541
542         if(probISR > (1.0 + 1.0e-10)){
543             cout << setprecision(20) << "\n\t ***
544                 ERROR: La probabilidad no puede ser
545                 mayor a 1 ***" << endl;
546             return 1;
547         }
548
549         if (probISR >= prob_max){
550             prob_max = probISR;

```

```

544         vect_optimum->SetPtEtaPhiM(
            vect_currentJet->Pt(),
            vect_currentJet->Eta(),
            vect_currentJet->Phi(),
            vect_currentJet->M());
545         ISR_tag_index = iJet;
546     }
547 }
548 }
549
550 k_ISR = prob_max*NumJets;
551
552 // Check the tagging results
553 k_ISR_pos = findPosition(k_min,k_max,k_bins,
    k_ISR);
554
555 if(k_ISR == 0.0) k_ISR_pos = -1;
556
557 if (ISR_jets[entry] != -1 && ISR_or_NOT[0] != '
    S'){ // != S means bb or WI
558     // A comparison can be handled
559     for (Int_t ind = 0; ind < k_ISR_pos + 1; ind
        ++){
560         if (ISR_tag_index == ISR_jets[entry])
561             Num_Tags_array[ind]++;
562         else
563             Num_MissTags_array[ind]++;
564     }
565     for (Int_t ind = k_ISR_pos+1; ind < k_bins;
        ind++){
566         Num_Rejected_array[ind]++;
567     }
568 }
569 else if (ISR_jets[entry] == -2 && ISR_or_NOT[0]
    == 'S'){
570     for (Int_t ind = 0; ind < k_ISR_pos + 1; ind
        ++){
571         Num_MissTags_array[ind]++;
572     }
573     for (Int_t ind = k_ISR_pos+1; ind < k_bins;

```

```

        ind++){
574         Num_Rejected_array[ind]++;
575     }
576 }
577
578     if (ISR_tag_index != -1 && vect_optimum->Pt() >
        pt_cut && ISR_or_NOT[0] != 'S'){ // != S
        means bb or WI
579     for (Int_t ind = 0; ind < k_ISR_pos + 1; ind
        ++){
580         if (ISR_tag_index == ISR_jets[entry])
581             Num_Tags_array_hpt[ind]++;
582         else
583             Num_MissTags_array_hpt[ind]++;
584     }
585     for (Int_t ind = k_ISR_pos+1; ind < k_bins;
        ind++){
586         Num_Rejected_array_hpt[ind]++;
587     }
588 }
589
590     Prob_cut = Jet_cut/NumJets;
591     if(prob_max >= Prob_cut){
592         if (ISR_tag_index == ISR_jets[entry] &&
            ISR_or_NOT[0] != 'S') // != S means bb
            or WI
593             Num_Tags++;
594         else
595             Num_MissTags++;
596
597         // Cheching MET boosting
598         if(vect_optimum->Pt() > pt_cut){
599             h_MET_hpt1->Fill(MET);
600         }
601     }
602     else
603         Num_Rejected++;
604
605 }
606

```

```

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

```



```

        Num_MissTags_array_hpt[ind]/Num_Total_Jets_hpt[
        ind];
632     Num_Rejected_array_hpt[ind] =
        Num_Rejected_array_hpt[ind]/Num_Total_Jets_hpt[
        ind];
633 }
634
635 /*
636  * Writing results
637  */
638 Bool_t archivoExiste = false;
639
640 Char_t *local_path_results;
641 local_path_results = (Char_t*) malloc(512*sizeof(
        Char_t));
642 if (atServer)
643     strcpy(local_path_results, "/home/af.garcia1214/
        PhenoMCsamples/Results/resultsTagging/"); // At
        the server
644 else
645     strcpy(local_path_results, "/home/afgarcia1214/
        Documentos/Results_and_data/resultsTagging/");
        // At the University's pc
646
647 Char_t *head_folder_results;
648 head_folder_results = (Char_t*) malloc(512*sizeof(
        Char_t));
649 if (Matching)
650     strcpy(head_folder_results, "
        _Tops_result_WI_Matching/");
651 else
652     strcpy(head_folder_results, "_Tops_result_WI/");
653 head_folder_results[0] = channel;
654 head_folder_results[13] = ISR_or_NOT[0];
655 head_folder_results[14] = ISR_or_NOT[1];
656
657 Char_t outName[] = "_Tops_WI_Overall";
658 outName[0] = channel;
659 outName[6] = ISR_or_NOT[0];
660 outName[7] = ISR_or_NOT[1];

```

```

661
662     Char_t outNamept[] = "_Tops_WI_hpt-100";
663     outNamept[0] = channel;
664     outNamept[6] = ISR_or_NOT[0];
665     outNamept[7] = ISR_or_NOT[1];
666     outNamept[13] = 0x30 + int(pt_cut/100)%10;
667     outNamept[14] = 0x30 + int(pt_cut/10)%10;
668     outNamept[15] = 0x30 + int(pt_cut)%10;
669
670     Char_t *outFileTotal;
671     outFileTotal = (Char_t*) malloc(sizeof(char)*512);
672     strcpy(outFileTotal, local_path_results);
673     strcat(outFileTotal, head_folder_results);
674     strcat(outFileTotal, outName);
675     strcat(outFileTotal, combination);
676     strcat(outFileTotal, ".txt");
677
678     Char_t *outFileTotalpt;
679     outFileTotalpt = (Char_t*) malloc(sizeof(char)*512);
680     strcpy(outFileTotalpt, local_path_results);
681     strcat(outFileTotalpt, head_folder_results);
682     strcat(outFileTotalpt, outNamept);
683     strcat(outFileTotalpt, combination);
684     strcat(outFileTotalpt, ".txt");
685
686     ifstream my_file(outFileTotal);
687     if(my_file.good()){
688         archivoExiste = true;
689     }
690     my_file.close();
691
692     ofstream ofs_over(outFileTotal, ios::out);
693     if(!archivoExiste){
694         // If file already exists
695     }
696
697     ofs_over << "# Number of Tags, Misstags and Rejected
        as a function of k" << endl;
698     ofs_over << "# Number of Events " << Num_Total_Jets
        [0] << endl;

```

```

699     ofs_over << "# k_cut \t Tags \t MissTags \t Rejected
       \t Total_Events " << endl;
700
701
702     for (Int_t ind = 0; ind < k_bins; ind++){
703         ofs_over << setiosflags(ios::fixed) <<
           setprecision(6) << setw(6) << k_values[ind]
704         << "\t" << Num_Tags_array[ind] << "\t" <<
           Num_MissTags_array[ind] << "\t" <<
           Num_Rejected_array[ind]
705         << "\t" << setprecision(0) << Num_Total_Jets
           [ind] << endl;
706     }
707
708     if (argc >= 5){
709         ofstream ofs_pt(outFileTotalpt, ios::out);
710         ofs_pt << "# Number of Tags, Misstags and Rejected
           as a function of k. The ISR has pt > " <<
           pt_cut << endl;
711         ofs_pt << "# Number of Events " <<
           Num_Total_Jets_hpt[0] << endl;
712         ofs_pt << "# k_cut \t Tags \t MissTags \t Rejected
           \t Total_Events " << endl;
713         for (Int_t ind = 0; ind < k_bins; ind++){
714             ofs_pt << setiosflags(ios::fixed) <<
               setprecision(6) << setw(6) << k_values[ind]
715             << "\t" << Num_Tags_array_hpt[ind] << "\t"
               << Num_MissTags_array_hpt[ind] << "\t"
               << Num_Rejected_array_hpt[ind]
716             << "\t" << setprecision(0) <<
               Num_Total_Jets_hpt[ind] << endl;
717         }
718         ofs_pt.close();
719     }
720
721     if (argc == 6){
722         Char_t outNameMET[] = "_Tops_WI_MET_pt_000_k_2.0";
723         outNameMET[0] = channel;
724         outNameMET[6] = ISR_or_NOT[0];
725         outNameMET[7] = ISR_or_NOT[1];

```

```

726     outNameMET[16] = pt_str[0];
727     outNameMET[17] = pt_str[1];
728     outNameMET[18] = pt_str[2];
729     outNameMET[22] = k_str[0];
730     outNameMET[23] = k_str[1];
731     outNameMET[24] = k_str[2];
732
733     Char_t *outFileMET;
734     outFileMET = (Char_t*) malloc(sizeof(char)*512);
735     strcpy(outFileMET,local_path_results);
736     strcat(outFileMET,head_folder_results);
737     strcat(outFileMET,outNameMET);
738     strcat(outFileMET,combination);
739
740     Char_t *outFilehist;
741     outFilehist = (Char_t*) malloc(sizeof(char)*512);
742     strcpy(outFilehist,outFileMET);
743     strcat(outFilehist,".root");
744
745     TFile* hfile = new TFile("histos.root", "RECREATE"
746                               );
747     TCanvas *C = new TCanvas(outFileMET,"MET in a
748                               sample with high PT ISR jets",1280,720);
749     Present(h_MET,h_MET_hpt1,C,2,"MET [GeV]","Num.
750             Jets / Total");
751     C->Write();
752     C->Close();
753     hfile->Close();
754
755 }
756
757 ofs_over.close();
758
759 cout<<"Fin :)"<<endl;
760
761 return 0;
762 }

```

B.2 Matching algorithm

```

1  /*
2  -----
3  -----      Universidad de los Andes      -----
4  -----      Departamento de Fisica        -----
5  -----      Joven Investigador            -----
6  -----      Andres Felipe Garcia Albarracin -----
7  -----      Juan Carlos Sanabria Arenas    -----
8  -----
9
10 This algorithm looks for the ISR parton into the
11 pythia8 simulation file and then finds the
12 corresponding ISR jet
13
14 It also stores in a binary file the matching
15 results
16 */
17
18 #include <iostream>
19 #include "ROOTFunctions.h"
20 #include "graphs_Funcs.h"
21 #include "functions.h"
22 #include "DelphesFunctions.h"
23
24 using namespace std;
25 // Global Variables
26 const Double_t PI = TMath::Pi();
27
28 // Other simulations parameters
29 const Char_t channel = 's'; // 's' for sTops and '_' for
    Tops
30 const Char_t ISR_or_NOT[] = "WI"; // "WI" with ISR, "SI"
    without (Here it does not make any sense), "bb"
    bjets production
31 const Bool_t atServer = true; // True if it is run at
    the server, false at the university's pc
32 const Bool_t Matching = true; // True if a matching has
    been done between MG and Pythia, false otherwise
33

```

```

34 int main(int argc, char **argv){
35     std::cout.precision(4);
36     // Counting time
37     Double_t initialTime = clock();
38
39     // Create chains of root trees
40     TChain chain_Pythia("STDHEP");
41     TChain chain_Delphes("Delphes");
42
43     // Loading simulations from Delphes
44     Char_t *local_path;
45     local_path = (Char_t*) malloc(512*sizeof(Char_t));
46     if (atServer)
47         strcpy(local_path, "/home/af.garcia1214/
            PhenoMCsamples/Simulations/
            MG_pythia8_delphes_parallel/"); // At the
            server
48     else
49         strcpy(local_path, "/home/afgarcia1214/Documentos/
            Simulations/"); // At the University's pc
50
51     Char_t *head_folder;
52     head_folder = (Char_t*) malloc(512*sizeof(Char_t));
53     if (Matching)
54         strcpy(head_folder, "_Tops_Events_WI_Matching/");
55     else
56         strcpy(head_folder, "_Tops_Events_WI/");
57     head_folder[0] = channel;
58     head_folder[13] = ISR_or_NOT[0];
59     head_folder[14] = ISR_or_NOT[1];
60
61     Char_t current_folder[] = "_Tops_MG_1K_AG_WI_003/";
62     current_folder[0] = channel;
63     current_folder[15] = ISR_or_NOT[0];
64     current_folder[16] = ISR_or_NOT[1];
65
66     Char_t unidad = '3'; Char_t decena = '0'; Char_t
        centena = '0';
67
68     if (argc > 1){

```

```

69     cout << "The number of the simulation should
70         consist of 3 digits" << endl;
71     centena = argv[1][0];
72     decena = argv[1][1];
73     unidad = argv[1][2];
74     current_folder[18] = centena;
75     current_folder[19] = decena;
76     current_folder[20] = unidad;
77 }
78
79 Char_t *file_pythia;
80 file_pythia = (Char_t*) malloc(512*sizeof(Char_t));
81 strcpy(file_pythia,local_path);
82 strcat(file_pythia,head_folder);
83 strcat(file_pythia,current_folder);
84 strcat(file_pythia,"Events/run_01/output_pythia8.root
85         ");
86
87 Char_t *file_delphes;
88 file_delphes = (Char_t*) malloc(512*sizeof(Char_t));
89 strcpy(file_delphes,local_path);
90 strcat(file_delphes,head_folder);
91 strcat(file_delphes,current_folder);
92 strcat(file_delphes,"Events/run_01/output_delphes.
93         root");
94
95 if (argc > 1){
96     cout << "\nReading the files: \nPythia8: " <<
97         file_pythia << "\nDelphes: " << file_delphes <<
98         endl;
99 }
100 else
101     cout << "\nReading the default files: \nPythia8: "
102         << file_pythia << "\nDelphes: " <<
103         file_delphes << endl;
104
105 chain_Pythia.Add(file_pythia);
106 chain_Delphes.Add(file_delphes);
107
108 // Objects of class ExRootTreeReader for reading the

```

```

102     information
103     ExRootTreeReader *treeReader_Pythia = new
        ExRootTreeReader(&chain_Pythia);
104
105     ExRootTreeReader *treeReader_Delphes = new
        ExRootTreeReader(&chain_Delphes);
106
107     Long64_t numberOfEntries = treeReader_Pythia->
        GetEntries();
108     Long64_t numberOfEntries_Delphes = treeReader_Delphes
        ->GetEntries();
109
110     // Get pointers to branches used in this analysis
111     TClonesArray *branchParticlePythia =
        treeReader_Pythia->UseBranch("GenParticle");
112     TClonesArray *branchJet = treeReader_Delphes->
        UseBranch("Jet");
113     TClonesArray *branchMissingET = treeReader_Delphes->
        UseBranch("MissingET");
114
115     cout << endl;
116     cout << " Number of Entries Pythia = " <<
        numberOfEntries << endl;
117     cout << " Number of Entries Delphes = " <<
        numberOfEntries_Delphes << endl;
118     cout << endl;
119
120     // particles, jets and vectors
121     TRootGenParticle *particle_pythia;
122     TRootGenParticle *ISR_particle;
123     MissingET *METpointer;
124     TLorentzVector *vect_ISR_particle = new
        TLorentzVector;
125
126     // Temporary variables
127     Bool_t ISR_parton_found = false; // true if the
        initial ISR_parton (with status 43) was found
128     Int_t pos_ISR = -1; // position of the ISR_parton
        into the branchParticlePythia array
129     Double_t MET = 0.0; // Missing transverse energy

```



```

129  /*
130   * Some variables used through the code
131   */
132  Int_t NumEvents1ISRJet = 0;      // Number of events
133        where the number of ISR jets is 1
134  Int_t NumMatches = 0;          // Number of matches
135  Int_t NumJets = 0;
136  Int_t ISR_match_index = -1;
137  Double_t Cut_matching_DPT = 50.0;
138  Double_t Cut_matching_DEta = 0.4;
139  Double_t Cut_matching_DPhi = 0.4;
140  Double_t Cut_matching_Dy = 0.4;
141  Int_t ISR_jets[numberOfEntries];
142
143  /*
144   * Main cycle of the program. Cycle over the events
145   */
146  numberOfEntries = 100000;
147  for (Int_t entry = 0; entry < numberOfEntries; ++
148      entry){
149      // Progress
150      if(numberOfEntries>10 && (entry%((int)
151          numberOfEntries/10))==0.0){
152          cout<<"progress = "<<(entry*100/numberOfEntries
153              )<<"%\t";
154          cout<< "Time : "<< (clock()-initialTime)/
155              double_t(CLOCKS_PER_SEC)<<"s"<<endl;
156      }
157
158      // Load selected branches with data from specified
159      event
160      treeReader_Pythia->ReadEntry(entry);
161      treeReader_Delphes->ReadEntry(entry);
162
163      // By default, the ISR jet was not matched
164      ISR_jets[entry] = -1;
165
166      // MET
167      METpointer = (MissingET*) branchMissingET->At(0);
168      MET = METpointer->MET;

```

```

163
164     // Finding the ISR parton
165     ISR_parton_found = false;
166     pos_ISR = -1;
167     for(Int_t iPart = 0; iPart < branchParticlePythia
168         ->GetEntries(); iPart++){
169         particle_pythia = (TRootGenParticle*)
170             branchParticlePythia->At(iPart);
171         if( abs(particle_pythia->Status) == 43){
172             pos_ISR = iPart;
173             ISR_particle = (TRootGenParticle*)
174                 branchParticlePythia->At(pos_ISR);
175             ISR_parton_found = true;
176             // The following lines were used to check that
177             // everything was going well
178             // cout << pos_ISR << "\t\t" << ISR_particle->
179             Status << "\t\t" << ISR_particle->PID
180             // << "\t\t" << ISR_particle->M1 << "\t\t"
181             << ISR_particle->M2
182             // << "\t\t" << ISR_particle->D1 << "\t\t"
183             << ISR_particle->D2 << endl;
184             }
185         }
186
187     // If there is not ISR parton, pass to the next
188     // event
189     if (ISR_parton_found == false){
190         continue;
191     }
192
193     // Finding the last copy of the ISR_parton
194     ISR_parton_found = false;
195     while (!ISR_parton_found){
196         if (ISR_particle->D1 != ISR_particle->D2)
197             ISR_parton_found = true;
198         else{
199             pos_ISR = ISR_particle->D1;
200             if(pos_ISR != -1) // To avoid an incoherent
201                 event
202                 ISR_particle = (TRootGenParticle*)

```

```

194         branchParticlePythia->At(pos_ISR);
195     else
196         ISR_parton_found = true; // To end up the
197         while loop
198     }
199 }
200
201 if (pos_ISR == -1) // End the incoherent events
202     continue;
203
204 // Matching algorithm
205 // Matching between the ISR parton and a jet
206 // Auxiliary variables
207 Double_t R_min = 2.0;
208 Double_t r; // Current deltaR
209 ISR_match_index = -1;
210 Int_t mixJets = 0;
211 TLorentzVector *vect_Jet1 = new TLorentzVector();
212     // Four-momentum of the jet of the 1st
213     for
214 TLorentzVector *vect_Jetc = new TLorentzVector();
215     // Four-momentum of the jet of the 2nd, 3
216     rd ... for
217 TLorentzVector *vect_Jets = new TLorentzVector();
218     // Four-momentum of the sum of jets
219 TLorentzVector *vect_Jeto = new TLorentzVector();
220     // Four-momentum of the optimal
221     combination
222 Jet *jet = new Jet();
223 Jet *jet2 = new Jet();
224
225 NumJets = branchJet->GetEntries();
226 vect_ISR_particle->SetPtEtaPhiE(ISR_particle->PT,
227     ISR_particle->Eta, ISR_particle->Phi,
228     ISR_particle->E);
229
230 if (NumJets < 3) // Minimum 3 jets per event
231     continue;
232
233 // Finding the jet with the minimum R to the ISR

```

```

parton
223   for ( Int_t j = 0; j < NumJets; j++ ) {      //
      Loop over jets finding the one with the minimum
      R
224       jet = (Jet*) branchJet->At(j);
225       vect_Jet1->SetPtEtaPhiM(jet->PT, jet->Eta,
          jet->Phi, jet->Mass);
226       r = vect_ISR_particle->DeltaR(*vect_Jet1);
227       if ( r < R_min ) {
228           R_min = r;
229           ISR_match_index = j;
230           mixJets = 1;
231           *vect_Jeto = *vect_Jet1;
232       }
233       // Checking if there are two jets mixed
234       for ( Int_t k = j+1; k<NumJets; k++){
235           jet2 = (Jet*) branchJet->At(k);
236           vect_Jetc->SetPtEtaPhiM(jet2->PT, jet2->
              Eta, jet2->Phi, jet2->Mass);
237           *vect_Jets = *vect_Jet1 + *vect_Jetc;
238           r = vect_ISR_particle->DeltaR(*vect_Jets)
              ;
239           if ( r < R_min ) {
240               R_min = r;
241               ISR_match_index = j;
242               mixJets = 2;
243               *vect_Jeto = *vect_Jets;
244           }
245           // Checking if there are three jets mixed
246           for (Int_t m = k+1; m<NumJets; m++){
247               jet2 = (Jet*) branchJet->At(m);
248               vect_Jetc->SetPtEtaPhiM(jet2->PT, jet2->
                  Eta, jet2->Phi, jet2->Mass);
249               *vect_Jets = *vect_Jets + *vect_Jetc;
250               r = vect_ISR_particle->DeltaR(*vect_Jets)
                  ;
251               if ( r < R_min ) {
252                   R_min = r;
253                   ISR_match_index = j;
254                   mixJets = 3;

```

```

255         *vect_Jeto = *vect_Jets;
256     }
257     // Checking if there are four jets mixed
258     for (Int_t n = m+1; n<NumJets; n++){
259         jet2 = (Jet*) branchJet->At(n);
260         vect_Jetc->SetPtEtaPhiM(jet2->PT, jet2->
            Eta, jet2->Phi, jet2->Mass);
261         *vect_Jets = *vect_Jets + *vect_Jetc;
262         r = vect_ISR_particle->DeltaR(*vect_Jets)
            ;
263         if ( r < R_min ) {
264             R_min = r;
265             ISR_match_index = j;
266             mixJets = 4;
267             *vect_Jeto = *vect_Jets;
268         }
269     }
270     }
271     }
272     } // Loop over jets finding the one with the
        minimum R
273
274     if( (mixJets == 1) && (ISR_match_index >= 0) && (
        ISR_match_index < NumJets) ) {
275         NumEvents1ISRJet++;
276         Double_t Delta_PT = TMath::Abs(
            vect_Jeto->Pt() - vect_ISR_particle
            ->Pt());
277         Double_t Delta_Eta = TMath::Abs(
            vect_Jeto->Eta() - vect_ISR_particle
            ->Eta());
278         Double_t Delta_Phi = vect_Jeto->
            DeltaPhi(*vect_ISR_particle);
279         Double_t Delta_y = TMath::Abs(vect_Jeto
            ->Rapidity() - vect_ISR_particle->
            Rapidity());
280
281         if ( (Delta_PT > Cut_matching_DPT) || (
            Delta_Eta > Cut_matching_DEta) || (
            Delta_Phi > Cut_matching_DPhi ) || (

```

```

282         Delta_y > Cut_matching_Dy) ) {
283             ISR_jets[entry] = -1;
284         }
285         else {
286             NumMatches++;
287             ISR_jets[entry] =
288                 ISR_match_index;
289         }
290     }
291     if (ISR_jets[entry] >= NumJets){
292         cout << "Error en el matching" << endl;
293         return 1;
294     }
295 } // End of the cycle over the events
296
297 cout<<"progress = 100%\t";
298 cout<< "Time :"<< (clock()-initialTime)/double_t(
299     CLOCKS_PER_SEC)<<"s"<<endl;
300
301 /*
302  * Writing results
303  */
304 Char_t *local_path_results;
305 local_path_results = (Char_t*) malloc(512*sizeof(
306     Char_t));
307 if (atServer)
308     strcpy(local_path_results, "/home/af.garcia1214/
309         PhenoMCsamples/Results/matching_Results/"); //
310         At the server
311 else
312     strcpy(local_path_results, "/home/afgarcia1214/
313         Documentos/Results_and_data/matching_Results/");
314         // At the University's pc
315
316 Char_t *head_folder_results;
317 head_folder_results = (Char_t*) malloc(512*sizeof(
318     Char_t));
319 if (Matching)
320     strcpy(head_folder_results, "

```

```

        _Tops_matches_WI_Matching/");
313     else
314         strcpy(head_folder_results, "_Tops_matches_WI/");
315     head_folder_results[0] = channel;
316     head_folder_results[13] = ISR_or_NOT[0];
317     head_folder_results[14] = ISR_or_NOT[1];
318
319     Char_t matching_name[] = "ISR_jets_Tops_WI_003.bn";
320     matching_name[8] = channel;
321     matching_name[14] = ISR_or_NOT[0];
322     matching_name[15] = ISR_or_NOT[1];
323
324     if (argc > 1){
325         matching_name[17] = centena;
326         matching_name[18] = decena;
327         matching_name[19] = unidad;
328     }
329
330     Char_t * fileName;
331     fileName = (Char_t*) malloc(512*sizeof(Char_t));
332     strcpy(fileName, local_path_results);
333     strcat(fileName, head_folder_results);
334     strcat(fileName, matching_name);
335
336     if (argc > 1)
337         cout << "*** Writing the binary file..." <<
            fileName << endl;
338     else
339         cout<<"*** Writing the default binary file..."
            << fileName << endl;
340
341     ofstream ofs(fileName, ios::out|ios::binary);
342     if (!ofs){
343         cout << "Problemas al escribir el archivo" << endl
            ;
344     }
345     else{
346         for(Int_t j = 0; j<numberOfEntries; j++){
347             ofs.write((Char_t *) (ISR_jets+j), sizeof(Int_t)
            );

```

```

348     }
349 }
350 ofs.close();
351
352 cout << endl;
353 cout << "Number of events with a single ISR jet = "
        << NumEvents1ISRJet << endl;
354 cout << "Number of matches = " << NumMatches << endl;
355 cout << endl;
356
357 return 0;
358 }

```

B.3 ISR jet analysis

```

1  /*
2  -----
3  -----      Universidad de los Andes      -----
4  -----      Departamento de Fisica        -----
5  -----      Joven Investigador            -----
6  -----      Andres Felipe Garcia Albarracin -----
7  -----      Juan Carlos Sanabria Arenas    -----
8  -----
9
10 This algorithm studies the kinematic properties
11 of the ISR jets. It reads the results of the
12 matching algorithm
13 */
14
15
16 #include "ROOTFunctions.h"
17 #include "graphs_Funcs.h"
18 #include "functions.h"
19 #include "Rtypes.h"
20 #include "DelphesFunctions.h"
21
22 // Global Variables
23 const Double_t PI = TMath::Pi();
24

```



```

25 // Other simulations parameters
26 const Char_t channel = '_'; // 's' for sTops and '_' for
    Tops
27 const Char_t ISR_or_NOT[] = "WI"; // "WI" with ISR, "SI"
    without (Here it does not make any sense), "bb"
    bjets production
28 const Bool_t atServer = true; // True if it is run at
    the server, false at the university's pc
29 const Bool_t Matching = true; // True if a matching has
    been done between MG and Pythia, false otherwise
30
31
32 int main(int argc, char **argv){
33     std::cout.precision(4);
34     // Counting time
35     Double_t initialTime = clock();
36
37     /*
38      * Histograms
39      */
40     // All jets
41     TH1 *h_numberJet = new TH1F("Number Jets","Number
        Jets",11,-0.5,10.5);
42
43     // Non Isr jets
44     TH1 *h_jet_PT = new TH1F("Jet PT","Jet PT",
        201,0.0,600.0);
45     TH1 *h_jet_Eta = new TH1F("Jet Eta","Jet Eta",
        171,-5.0,5.0);
46     TH1 *h_jet_Phi = new TH1F("Jet Phi","Jet Phi",
        375,-3.5,3.5);
47     TH1 *h_jet_DPhi_MET = new TH1F("Jet - MET Delta_Phi",
        "Jet - MET Delta_Phi",300,0.0,4.0);
48     TH1 *h_jet_DPhi_MET_hpt = new TH1F("Jet - MET
        Delta_Phi_hpt","Jet - MET Delta_Phi_hpt"
        ,300,0.0,4.0);
49     TH1 *h_jet_MT = new TH1F("Jet Transverse mass","Jet
        Transverse Mass",201,0.0,600.0);
50     TH1 *h_jet_Delta_PT = new TH1F("Jet Delta-PT","Non
        ISR Delta-PT", 201,0.0,300.0);

```

```

51 TH1 *h_jet_PT_HT = new TH1F("Jet PT-HT ratio","Jet PT
    -HT ratio",201,-0.0025,1.0025);
52 TH1 *h_jet_PT_over_PT_others = new TH1F("Jet PT/
    PT_others","Jet PT/PT_others",401,-0.0025,2.0025);
53 TH1 *h_jet_Eta_over_Eta_others = new TH1F("Jet Eta/
    Eta_others","Jet Eta/Eta_others"
    ,401,-0.0025,2.0025);
54 TH1 *h_jet_DPhi_over_Phi_others = new TH1F("Jet Phi/
    Phi_others","Jet Phi/Phi_others"
    ,401,-0.0025,2.0025);
55 TH1 *h_jet_Delta_Eta = new TH1F("Jet Delta-Eta","Jet
    Delta-Eta", 171,0.0,5.0);
56 TH1 *h_jet_DPhi_MET_other = new TH1F("Jet - MET
    Delta_Phi other","Jet - MET Delta_Phi other"
    ,300,0.0,4.0);
57 TH1 *h_jet_multiplicity = new TH1F("Jet -
    Multiplicity","Jet - Multiplicity",101,-0.5,100.5)
    ;
58 TH1 *h_jet_DeltaR = new TH1F ("Jet - Delta_R","Jet -
    Delta_R",201,-0.0025,0.8025);
59 TH1 *h_jet_Delta_PT_leading = new TH1F("Delta PT:
    leading - Jet","Delta PT: leading - Jet",
    201,0.0,600.0);
60 TH1 *h_jet_Delta_Eta_leading = new TH1F("Delta Eta:
    Jet - leading","Delta Eta: Jet - leading",
    171,0.0,8.0);
61
62 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);
63
64 // ISR jets
65 TH1 *h_ISR_PT = new TH1F("ISR PT","ISR PT",
    201,0.0,600.0);
66 TH1 *h_ISR_Eta = new TH1F("ISR Eta","ISR Eta",
    171,-5.0,5.0);
67 TH1 *h_ISR_Phi = new TH1F("ISR Phi","ISR Phi",
    375,-3.5,3.5);
68 TH1 *h_ISR_DPhi_MET = new TH1F("ISR - MET Delta_Phi",
    "ISR - MET Delta_Phi",300,0.0,4.0);

```

```

69  TH1 *h_ISR_DPhi_MET_hpt = new TH1F("ISR - MET
    Delta_Phi_hpt","ISR - MET Delta_Phi_hpt"
    ,300,0.0,4.0);
70  TH1 *h_ISR_MT = new TH1F("ISR Transverse mass","ISR
    Transverse Mass",201,0.0,600.0);
71  TH1 *h_ISR_Delta_PT = new TH1F("ISR Delta-PT","ISR
    Delta-PT", 201,0.0,300.0);
72  TH1 *h_ISR_PT_HT = new TH1F("ISR PT-HT ratio","ISR PT
    -HT ratio",201,-0.0025,1.0025);
73  TH1 *h_ISR_PT_over_PT_others = new TH1F("ISR PT/
    PT_others","ISR PT/PT_others",401,-0.0025,2.0025);
74  TH1 *h_ISR_Eta_over_Eta_others = new TH1F("ISR Eta/
    Eta_others","ISR Eta/Eta_others"
    ,401,-0.0025,2.0025);
75  TH1 *h_ISR_DPhi_over_Phi_others = new TH1F("ISR Phi/
    Phi_others","ISR Phi/Phi_others"
    ,401,-0.0025,2.0025);
76  TH1 *h_ISR_Delta_Eta = new TH1F("ISR Delta-Eta","ISR
    Delta-Eta", 171,0.0,5.0);
77  TH1 *h_ISR_DPhi_MET_other = new TH1F("ISR - MET
    Delta_Phi other","ISR - MET Delta_Phi other"
    ,300,0.0,4.0);
78  TH1 *h_ISR_multiplicity = new TH1F("ISR -
    Multiplicity","ISR - Multiplicity",101,-0.5,100.5)
    ;
79  TH1 *h_ISR_DeltaR = new TH1F ("ISR - Delta_R","ISR -
    Delta_R",201,-0.0025,0.8025);
80  TH1 *h_ISR_Delta_PT_leading = new TH1F("Delta PT:
    leading - ISR","Delta PT: leading - ISR",
    201,0.0,600.0);
81  TH1 *h_ISR_Delta_Eta_leading = new TH1F("Delta Eta:
    ISR - leading","Delta Eta: ISR - leading",
    171,0.0,8.0);
82
83  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);
84
85  // MET
86  TH1 *h_MET = new TH1F("Missing ET","Missing ET"
    ,200,0,600);

```

```

87 TH1 *h_MET_hpt1 = new TH1F("Missing ET high_ISR_pt-1"
88 , "Missing ET high_ISR_pt-1", 200, 0.0, 600.0);
89 TH1 *h_MET_hpt2 = new TH1F("Missing ET high_ISR_pt-2"
90 , "Missing ET high_ISR_pt-2", 200, 0.0, 600.0);
91 TH1 *h_MET_hpt3 = new TH1F("Missing ET high_ISR_pt-3"
92 , "Missing ET high_ISR_pt-3", 200, 0.0, 600.0);
93 TH1 *h_MET_hpt4 = new TH1F("Missing ET high_ISR_pt-4"
94 , "Missing ET high_ISR_pt-4", 200, 0.0, 600.0);
95
96 TH2 *h2_dif_PTEta=new TH2F("
97   FSR_ISR_Jet_PT_Eta_Difference", "Difference between
98   FSR and ISR Jet PT Vs. Eta distributions"
99   , 201, -1.25, 501.25, 201, -4.02, 4.02);
100 TH2 *h2_dif_lead_PTEta=new TH2F("
101   Lead_ISR_Jet_PT_Eta_Difference", "Difference
102   between Lead and ISR Jet PT Vs. Eta distributions"
103   , 201, -1.25, 501.25, 201, -4.02, 4.02);
104
105 // Leading PT
106 TH1 *h_leading_PT = new TH1F("Leading PT", "Leading PT
107   ", 201, 0.0, 600.0);
108 TH1 *h_leading_MT = new TH1F("Leading Transverse mass
109   ", "Leading Transverse Mass", 201, 0.0, 600.0);
110 TH1 *h_leading_Eta = new TH1F("Leading Eta", "Leading
111   Eta", 171, -5.0, 5.0);
112 TH1 *h_leading_DPhi_MET = new TH1F("Leading - MET
113   Delta_Phi", "Leading - MET Delta_Phi", 300, 0.0, 4.0);
114
115 TH2 *h2_leading_PTEta=new TH2F("Leading_Jet_PT_Eta", "
116   Leading Jet PT Vs. Eta"
117   , 201, -1.25, 501.25, 201, -4.02, 4.02);
118
119 // Other variables
120 TH1 *h_HT = new TH1F("HT", "HT", 201, 0.0, 600.0);
121 TH1 *h_HT_R1 = new TH1F("HT_R1", "HT_R1"
122   , 51, -0.01, 1.01);
123 TH1 *h_HT_R2 = new TH1F("HT_R2", "HT_R2"
124   , 51, -0.01, 1.01);
125
126 // B tagging

```

```

109 TH1 *h_BTag = new TH1F("BTag","BTag",5,-0.5,4.5);
110 TH1 *h_BTag_PT = new TH1F("BTag PT","BTag PT",
    201,0.0,600.0);
111 TH1 *h_BTag_Eta = new TH1F("BTag Eta","BTag Eta",
    171,-5.0,5.0);
112 TH1 *h_BTag_DPhi_MET = new TH1F("BTag - MET Delta_Phi
    ","BTag - MET Delta_Phi",300,0.0,4.0);
113 TH1 *h_BTags_per_Event = new TH1F("BTags per event","
    BTags per event",5,-0.5,4.5);
114
115 // Further analysis
116 TH1 *h_ISR_PT_comp = new TH1F("ISR PT for comparison"
    ,"ISR PT for comparison with histo", 20,0.0,800.0)
    ;
117 TH1 *h_ISR_Eta_comp = new TH1F("ISR Eta for
    comparison","ISR Eta for comparison with histo",
    20,-4.2,4.2);
118 TH1 *h_ISR_DPhi_MET_comp = new TH1F("ISR Phi for
    comparison","ISR Phi for comparison with histo",
    20,0,PI);
119
120 // To check the histograms' creation
121 TH1 *hist_ISR_PT = new TH1F("ISR PT comp","ISR PT
    comp", 20,0.0,800.0);
122 TH1 *hist_ISR_Abs_Eta = new TH1F("ISR Abs Eta comp","
    ISR Abs Eta comp", 20,0.0,5.2);
123 TH1 *hist_ISR_DPhi_MET = new TH1F("ISR Delta Phi comp
    ","ISR Delta Phi comp", 20,0.0,PI);
124 TH1 *hist_ISR_PT_ratio = new TH1F("ISR PT/PT_others
    comp","ISR PT/PT_others comp",20,0.0,8.0);
125 TH1 *hist_ISR_Delta_Eta = new TH1F("ISR Delta-Eta
    comp","ISR Delta-Eta comp", 20,0.0,7.0);
126 TH1 *hist_ISR_DPhi_MET_other = new TH1F("ISR - MET
    Delta_Phi other comp","ISR - MET Delta_Phi other
    comp",20,0.0,PI);
127 TH1 *hist_ISR_Delta_PT_leading = new TH1F("Delta PT:
    leading - ISR comp","Delta PT: leading - ISR comp"
    , 20,0.0,500.0);
128 TH1 *hist_ISR_Delta_Eta_leading = new TH1F("Delta Eta
    : ISR - leading comp","Delta Eta: ISR - leading

```

```

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

137
138 for(int iRun = 1; iRun < 11; iRun ++){
139     // Create chains of root trees
140     TChain chain_Delphes("Delphes");
141
142     // Loading simulations from Delphes
143     Char_t *local_path;
144     local_path = (Char_t*) malloc(512*sizeof(Char_t));
145     if (atServer)
146         strcpy(local_path,"/home/af.garcia1214/
            PhenoMCsamples/Simulations/
            MG_pythia8_delphes_parallel/"); // At the
            server
147     else
148         strcpy(local_path,"/home/afgarcia1214/
            Documentos/Simulations/"); // At the
            University's pc
149
150     Char_t *head_folder;
151     head_folder = (Char_t*) malloc(512*sizeof(Char_t))

```

```

152         ;
153         if (Matching)
154             strcpy(head_folder, "_Tops_Events_WI_Matching/")
155             ;
156         else
157             strcpy(head_folder, "_Tops_Events_WI/");
158         head_folder[0] = channel;
159         head_folder[13] = ISR_or_NOT[0];
160         head_folder[14] = ISR_or_NOT[1];
161
162         Char_t current_folder[] = "_Tops_MG_1K_AG_WI_003/"
163         ;
164         current_folder[0] = channel;
165         current_folder[15] = ISR_or_NOT[0];
166         current_folder[16] = ISR_or_NOT[1];
167
168         Char_t unidad = 0x30 + iRun%10;
169         Char_t decena = 0x30 + int(iRun/10)%10;
170         Char_t centena = 0x30 + int(iRun/100)%10;
171
172         current_folder[18] = centena;
173         current_folder[19] = decena;
174         current_folder[20] = unidad;
175
176         Char_t *file_delphes;
177         file_delphes = (Char_t*) malloc(512*sizeof(Char_t)
178         );
179         strcpy(file_delphes, local_path);
180         strcat(file_delphes, head_folder);
181         strcat(file_delphes, current_folder);
182         strcat(file_delphes, "Events/run_01/output_delphes.
183         root");
184
185         cout << "\nReading the file: \nDelphes: " <<
186         file_delphes << endl;
187
188         chain_Delphes.Add(file_delphes);
189         // Objects of class ExRootTreeReader for reading
190         the information
191         ExRootTreeReader *treeReader_Delphes = new

```

```

185         ExRootTreeReader(&chain_Delphes);
186         Long64_t numberOfEntries = treeReader_Delphes->
            GetEntries();
187
188         // Get pointers to branches used in this analysis
189         TClonesArray *branchJet = treeReader_Delphes->
            UseBranch("Jet");
190         TClonesArray *branchMissingET = treeReader_Delphes
            ->UseBranch("MissingET");
191
192         cout << endl;
193         cout << " Number of Entries Delphes = " <<
            numberOfEntries << endl;
194         cout << endl;
195
196         // particles, jets and vectors
197         MissingET *METpointer;
198         TLorentzVector *vect_currentJet = new
            TLorentzVector;
199         TLorentzVector *vect_auxJet = new TLorentzVector;
200         TLorentzVector *vect_leading = new TLorentzVector;
201         Jet *currentJet = new Jet;
202         Jet *auxJet = new Jet;
203         TRefArray array_temp;
204
205         // Temporary variables
206         Double_t MET = 0.0; // Missing transverse energy
207         Double_t delta_phi = 0.0; // difference between
            the phi angle of MET and the jet
208         Double_t transverse_mass = 0.0; // Transverse mass
209         Double_t HT = 0.0; // Sum of jets' PT
210         Double_t HT_R1 = 0.0; // Sum of jets' PT which are
            in the same hemisphere of the ISR jet
            hemisphere
211         Double_t HT_R2 = 0.0; // Sum of jets' PT which are
            in the opposite hemisphere of the ISR jet
            hemisphere
212         Double_t ISR_Eta = 0.0; // Pseudorapidity of the
            ISR jet

```



```

213     Int_t number_Btags = 0; // Number of B jets per
        event
214     Int_t ISR_Btags = 0; // Number of BTags which are
        also ISR jets
215     Double_t delta_PT_jet = 0.0; // |PT-<PT>|
216     Double_t PT_sum = 0.0; // sum(PT)
217     Double_t PT_aver = 0.0; // <PT>
218     Double_t Delta_eta_aver = 0.0; // sum_i|eta-eta_i
        |/(Nj-1)
219     Double_t Delta_phi_sum = 0.0; // sum delta_phi
220     Double_t Delta_phi_other_jets = 0.0; // Average of
        delta phi of other jets
221     Double_t PT_ratio = 0.0; // PT/PT_others
222     Double_t Eta_ratio = 0.0; // Eta/Eta_others
223     Double_t Eta_sum = 0.0; // sum(Eta)
224     Double_t Delta_R = 0.0; // Size of the jet
225     Double_t Delta_phi_ratio = 0.0; // Delta_phi/
        Delta_phi_others
226     Double_t Delta_PT_leading = 0.0; // PT -
        PT_leading
227     Double_t Delta_Eta_leading = 0.0; // |Eta -
        Eta_leading|
228
229     /*
230     * Some variables used through the code
231     */
232     Int_t ISR_jets[numberOfEntries];
233     Int_t NumJets = 0;
234
235     Char_t *local_path_binary;
236     local_path_binary = (Char_t*) malloc(512*sizeof(
        Char_t));
237     if (atServer)
238         strcpy(local_path_binary, "/home/af.garcia1214/
        PhenoMCsamples/Results/matching_Results/");
        // At the server
239     else
240         strcpy(local_path_binary, "/home/afgarcia1214/
        Documentos/Results_and_data/matching_Results
        /"); // At the University's pc

```

```

241
242     Char_t *head_folder_binary;
243     head_folder_binary = (Char_t*) malloc(512*sizeof(
        Char_t));
244     if (Matching)
245         strcpy(head_folder_binary, "
            _Tops_matchs_WI_Matching/");
246     else
247         strcpy(head_folder_binary, "_Tops_matchs_WI/");
248     head_folder_binary[0] = channel;
249     head_folder_binary[13] = ISR_or_NOT[0];
250     head_folder_binary[14] = ISR_or_NOT[1];
251
252     Char_t matching_name[] = "ISR_jets_Tops_WI_003.bn"
        ;
253     matching_name[8] = channel;
254     matching_name[14] = ISR_or_NOT[0];
255     matching_name[15] = ISR_or_NOT[1];
256
257     matching_name[17] = centena;
258     matching_name[18] = decena;
259     matching_name[19] = unidad;
260
261     Char_t * fileName;
262     fileName = (Char_t*) malloc(512*sizeof(Char_t));
263     strcpy(fileName, local_path_binary);
264     strcat(fileName, head_folder_binary);
265     strcat(fileName, matching_name);
266
267     ifstream ifs(fileName, ios::in | ios::binary);
268
269     for (Int_t j = 0; j<numberOfEntries; j++){
270         ifs.read((Char_t *) (ISR_jets+j), sizeof(
            Int_t));
271     }
272     ifs.close();
273
274     // Jet with greatest PT
275     Double_t PT_max = 0;
276     Int_t posLeadingPT = -1;

```

```

277     Int_t ISR_greatest_PT = 0;
278     Double_t MT_leading_jet = 0.0; // Transverse mass
279
280     /*
281     * Main cycle of the program
282     */
283     numberOfEntries = 100000;
284     for (Int_t entry = 0; entry < numberOfEntries; ++
         entry){
285         // Progress
286         if(numberOfEntries>10 && (entry%((int)
            numberOfEntries/10))==0.0){
287             cout<<"progress = "<<(entry*100/
                numberOfEntries)<<"%\t";
288             cout<< "Time : "<< (clock()-initialTime)/
                double_t(CLOCKS_PER_SEC)<<"s"<<endl;
289         }
290
291         // Load selected branches with data from
            specified event
292         treeReader_Delphes->ReadEntry(entry);
293
294         // MET
295         METpointer = (MissingET*) branchMissingET->At
            (0);
296         MET = METpointer->MET;
297         h_MET->Fill(MET);
298
299         NumJets=branchJet->GetEntries();
300         h_numberJet->Fill(NumJets);
301
302         // checking the ISR
303         if (ISR_jets[entry] == -1 || NumJets < 3)
304             continue;
305
306         PT_max = 0;
307         posLeadingPT = -1;
308         HT = 0;
309         HT_R1 = 0;
310         HT_R2 = 0;

```

```

311         number_Btags = 0;
312
313         delta_PT_jet = 0.0;
314         PT_aver = 0.0;
315         PT_sum = 0.0;
316         Delta_eta_aver = 0.0;
317         Delta_phi_sum = 0.0;
318         Delta_phi_other_jets = 0.0;
319         Delta_phi_ratio = 0.0;
320         Delta_PT_leading = 0.0;
321         Delta_Eta_leading = 0.0;
322
323         PT_ratio = 0.0;
324         Eta_ratio = 0.0;
325         Eta_sum = 0.0;
326
327         Delta_R = 0.0;
328
329         if (ISR_jets[entry] >= NumJets){
330             cout << "Error en el matching" << endl;
331             return 1;
332         }
333
334         // Preliminary for. It is used to calculate
335         // PT_aver and Delta_phi_sum
336         for (Int_t iJet = 0; iJet<NumJets; iJet++){
337             currentJet = (Jet*) branchJet->At(iJet);
338             vect_currentJet->SetPtEtaPhiM(currentJet->PT
339                 ,currentJet->Eta,currentJet->Phi,
340                 currentJet->Mass);
341             delta_phi = deltaAng(vect_currentJet->Phi(),
342                 METpointer->Phi());
343             PT_sum += vect_currentJet->Pt();
344             Eta_sum += vect_currentJet->Eta();
345             Delta_phi_sum += delta_phi;
346             // HT
347             HT += vect_currentJet->Pt();
348             // HT ratios
349             if((vect_currentJet->Eta()*ISR_Eta) > 0)
350                 HT_R1 += vect_currentJet->Pt();

```

```

347         else
348             HT_R2 += vect_currentJet->Pt();
349             // PT Leading jet
350             if (PT_max < vect_currentJet->Pt()){
351                 PT_max = vect_currentJet->Pt();
352                 posLeadingPT = iJet;
353             }
354     }
355
356     //PT_aver
357     PT_aver = PT_sum/NumJets;
358
359     // Leading PT
360     currentJet = (Jet*) branchJet->At(posLeadingPT)
361         ;
362     vect_leading->SetPtEtaPhiM(currentJet->PT,
363         currentJet->Eta, currentJet->Phi, currentJet->
364         Mass);
365
366     // ISR jet
367     currentJet = (Jet*) branchJet->At(ISR_jets[
368         entry]);
369     vect_currentJet->SetPtEtaPhiM(currentJet->PT,
370         currentJet->Eta, currentJet->Phi, currentJet->
371         Mass);
372     ISR_Eta = vect_currentJet->Eta();
373
374     for (Int_t iJet = 0; iJet<NumJets; iJet++){
375         currentJet = (Jet*) branchJet->At(iJet);
376         vect_currentJet->SetPtEtaPhiM(currentJet->PT,
377             ,currentJet->Eta, currentJet->Phi,
378             currentJet->Mass);
379         delta_phi = deltaAng(vect_currentJet->Phi(),
380             METpointer->Phi());
381         transverse_mass = sqrt(2*vect_currentJet->Pt
382             ()*MET*(1-cos(delta_phi)));
383
384         // Correlated variables
385         delta_PT_jet = TMath::Abs(vect_currentJet->
386             Pt()-PT_aver);

```

```

376     Delta_phi_other_jets = (Delta_phi_sum -
377         delta_phi)/(NumJets-1);
378     PT_ratio = vect_currentJet->Pt()*(NumJets-1)
379         /(PT_sum-vect_currentJet->Pt());
380     Eta_ratio = vect_currentJet->Eta()*(NumJets
381         -1)/(Eta_sum-vect_currentJet->Eta());
382     Delta_phi_ratio = delta_phi*(NumJets-1)/(
383         Delta_phi_sum-delta_phi);
384
385     Delta_Eta_leading = TMath::Abs(
386         vect_currentJet->Eta()-vect_leading->Eta
387         ());
388     Delta_PT_leading = vect_leading->Pt()-
389         vect_currentJet->Pt();
390
391     Delta_eta_aver = 0.0;
392     // For cycle used to calculate
393     Delta_eta_aver
394     for(Int_t iJet2 = 0; iJet2<NumJets; iJet2++)
395     {
396         auxJet = (Jet*) branchJet->At(iJet2);
397         vect_auxJet->SetPtEtaPhiM(auxJet->PT,
398             auxJet->Eta,auxJet->Phi,auxJet->Mass);
399         if (iJet2 != iJet) Delta_eta_aver +=
400             TMath::Abs(vect_auxJet->Eta()-
401                 vect_currentJet->Eta());
402     }
403     Delta_eta_aver = Delta_eta_aver/(NumJets-1);
404     Delta_R = sqrt(pow(currentJet->DeltaEta,2)+
405         pow(currentJet->DeltaPhi,2));
406
407     // Multiplicity
408     array_temp = (TRefArray) currentJet->
409         Constituents;
410
411     if (iJet != ISR_jets[entry]){ // Non ISR
412         h_jet_PT->Fill(vect_currentJet->Pt());
413         h_jet_Eta->Fill(vect_currentJet->Eta());
414         h_jet_Phi->Fill(vect_currentJet->Phi());
415         h_jet_DPhi_MET->Fill(delta_phi);

```

```

402         h_jet_MT->Fill(transverse_mass);
403         h_jet_Delta_PT->Fill(delta_PT_jet);
404         h_jet_Delta_Eta->Fill(Delta_eta_aver);
405         h_jet_DPhi_MET_other->Fill(
            Delta_phi_other_jets);
406         h_jet_PT_HT->Fill(vect_currentJet->Pt()/
            HT);
407         h_jet_multiplicity->Fill(array_temp.
            GetEntries());
408         h_jet_PT_over_PT_others->Fill(PT_ratio);
409         h_jet_Eta_over_Eta_others->Fill(Eta_ratio
            );
410         h_jet_DeltaR->Fill(Delta_R);
411         h_jet_DPhi_over_Phi_others->Fill(
            Delta_phi_ratio);
412         h_jet_Delta_PT_leading->Fill(
            Delta_PT_leading);
413         h_jet_Delta_Eta_leading->Fill(
            Delta_Eta_leading);
414         if (vect_currentJet->Pt()>240)
415             h_jet_DPhi_MET_hpt->Fill(delta_phi);
416         h2_jet_PTEta->Fill(vect_currentJet->Pt(),
            vect_currentJet->Eta());
417
418         // For testing creating histo
419         hist_jet_PT->Fill(vect_currentJet->Pt());
420         hist_jet_Abs_Eta->Fill(TMath::Abs(
            vect_currentJet->Eta()));
421         hist_jet_DPhi_MET->Fill(delta_phi);
422         hist_jet_PT_ratio->Fill(PT_ratio);
423         hist_jet_Delta_Eta->Fill(Delta_eta_aver);
424         hist_jet_DPhi_MET_other->Fill(
            Delta_phi_other_jets);
425         hist_jet_Delta_PT_leading->Fill(
            Delta_PT_leading);
426         hist_jet_Delta_Eta_leading->Fill(
            Delta_Eta_leading);
427     }
428     else{ //ISR
429         h_ISR_PT->Fill(vect_currentJet->Pt());

```

```

430     h_ISR_Eta->Fill(vect_currentJet->Eta());
431     h_ISR_Phi->Fill(vect_currentJet->Phi());
432     h_ISR_DPhi_MET->Fill(delta_phi);
433     h_ISR_Eta_comp->Fill(vect_currentJet->Eta
        ());
434     h_ISR_PT_comp->Fill(vect_currentJet->Pt()
        );
435     h_ISR_DPhi_MET_comp->Fill(delta_phi);
436     h_ISR_Delta_PT->Fill(delta_PT_jet);
437     h_ISR_Delta_Eta->Fill(Delta_eta_aver);
438     h_ISR_DPhi_MET_other->Fill(
        Delta_phi_other_jets);
439     h_ISR_PT_HT->Fill(vect_currentJet->Pt()/
        HT);
440     h_ISR_multiplicity->Fill(array_temp.
        GetEntries());
441     h_ISR_PT_over_PT_others->Fill(PT_ratio);
442     h_ISR_Eta_over_Eta_others->Fill(Eta_ratio
        );
443     h_ISR_DeltaR->Fill(Delta_R);
444     h_ISR_DPhi_over_Phi_others->Fill(
        Delta_phi_ratio);
445     h_ISR_Delta_PT_leading->Fill(
        Delta_PT_leading);
446     h_ISR_Delta_Eta_leading->Fill(
        Delta_Eta_leading);
447     if (vect_currentJet->Pt()>120)
448         h_MET_hpt1->Fill(MET);
449     if (vect_currentJet->Pt()>200)
450         h_MET_hpt2->Fill(MET);
451     if (vect_currentJet->Pt()>240){
452         h_MET_hpt3->Fill(MET);
453         h_ISR_DPhi_MET_hpt->Fill(delta_phi);
454     }
455     if (vect_currentJet->Pt()>300)
456         h_MET_hpt4->Fill(MET);
457     h2_ISR_PTEta->Fill(vect_currentJet->Pt(),
        vect_currentJet->Eta());
458     // Transverse mass
459     h_ISR_MT->Fill(transverse_mass);

```



```

460
461         // For testing creating histo
462         hist_ISR_PT->Fill(vect_currentJet->Pt());
463         hist_ISR_Abs_Eta->Fill(TMath::Abs(
464             vect_currentJet->Eta()));
465         hist_ISR_DPhi_MET->Fill(delta_phi);
466         hist_ISR_PT_ratio->Fill(PT_ratio);
467         hist_ISR_Delta_Eta->Fill(Delta_eta_aver);
468         hist_ISR_DPhi_MET_other->Fill(
469             Delta_phi_other_jets);
470         hist_ISR_Delta_PT_leading->Fill(
471             Delta_PT_leading);
472         hist_ISR_Delta_Eta_leading->Fill(
473             Delta_Eta_leading);
474     }
475
476     // BTag
477     h_BTag->Fill(currentJet->BTag);
478     if (currentJet->BTag == 1){ // The current
479         jet is B Tagged
480         h_BTag_PT->Fill(vect_currentJet->Pt());
481         h_BTag_Eta->Fill(vect_currentJet->Eta());
482         h_BTag_DPhi_MET->Fill(delta_phi);
483         number_Btags++;
484
485         if (iJet == ISR_jets[entry]){ // If the
486             ISR jet is also a B jet
487             ISR_Btags++;
488         }
489     }
490
491     // Jet with greatest PT
492     if (posLeadingPT != -1){
493         h_leading_PT->Fill(PT_max);
494         if(posLeadingPT == ISR_jets[entry])
495             ISR_greatest_PT++;
496
497         currentJet = (Jet*) branchJet->At(
498             posLeadingPT);

```

```

492         vect_currentJet->SetPtEtaPhiM(currentJet->PT
           ,currentJet->Eta,currentJet->Phi,
           currentJet->Mass);
493         delta_phi = deltaAng(vect_currentJet->Phi(),
           METpointer->Phi);
494         MT_leading_jet = sqrt(2*vect_currentJet->Pt
           ()*MET*(1-cos(delta_phi)));
495         h_leading_MT->Fill(MT_leading_jet);
496
497         h_leading_Eta->Fill(vect_currentJet->Eta());
498         h_leading_DPhi_MET->Fill(delta_phi);
499
500         h2_leading_PTEta->Fill(vect_currentJet->Pt()
           ,vect_currentJet->Eta());
501     }
502
503     // HT
504     if (1 < HT_R1/HT || 1 < HT_R2/HT){
505         cout << "Error en el evento: " << entry <<
           endl;
506         cout << "HT: " << HT << "\tHT_R1: " << HT_R1
           << "\tHT_R2: " << HT_R2 << endl;
507         return 1;
508     }
509
510     h_HT->Fill(HT);
511     h_HT_R1->Fill(HT_R1/HT);
512     h_HT_R2->Fill(HT_R2/HT);
513     h_BTags_per_Event->Fill(number_Btags);
514
515 }
516
517 cout<<"progress = 100%\t";
518 cout<< "Time : "<< (clock()-initialTime)/double_t(
           CLOCKS_PER_SEC)<<"s"<<endl;
519 cout<< "Percentage of events where the ISR jet is
           the jet with greatest PT: " << (Double_t) (
           ISR_greatest_PT*100)/numberOfEntries << "%\n";
520 cout<< "Percentage of events where the ISR jet is
           tagged as Bjet: " << (Double_t) (ISR_Btags*100)

```

```

        /numberOfEntries << "%\n";
521
522     } // End run's for cicle
523
524     TFile* hfile = new TFile("./histos/histos.root", "
        RECREATE");
525     h_jet_DPhi_MET->Write();
526     h_jet_Eta->Write();
527     h_jet_PT->Write();
528     h_jet_Phi->Write();
529     h_jet_MT->Write();
530     h_jet_Delta_PT->Write();
531     h_jet_Delta_Eta->Write();
532     h_jet_DPhi_MET_other->Write();
533     h_jet_PT_HT->Write();
534     h_jet_multiplicity->Write();
535     h_jet_PT_over_PT_others->Write();
536     h_jet_Eta_over_Eta_others->Write();
537     h_jet_DeltaR->Write();
538     h_jet_DPhi_over_Phi_others->Write();
539     h_jet_Delta_Eta_leading->Write();
540     h_jet_Delta_PT_leading->Write();
541
542     h_ISR_DPhi_MET->Write();
543     h_ISR_Eta->Write();
544     h_ISR_PT->Write();
545     h_ISR_Phi->Write();
546     h_ISR_MT->Write();
547     h_ISR_Delta_PT->Write();
548     h_ISR_Delta_Eta->Write();
549     h_ISR_DPhi_MET_other->Write();
550     h_ISR_PT_HT->Write();
551     h_ISR_multiplicity->Write();
552     h_ISR_PT_over_PT_others->Write();
553     h_ISR_Eta_over_Eta_others->Write();
554     h_ISR_DeltaR->Write();
555     h_ISR_DPhi_over_Phi_others->Write();
556     h_ISR_Delta_Eta_leading->Write();
557     h_ISR_Delta_PT_leading->Write();
558

```

```

559     h_MET->Write();
560     h_MET_hpt1->Write();
561     h_MET_hpt2->Write();
562     h_MET_hpt3->Write();
563
564     h_leading_MT->Write();
565     h_leading_PT->Write();
566     h_leading_Eta->Write();
567     h_leading_DPhi_MET->Write();
568
569     h_HT->Write();
570     h_HT_R1->Write();
571     h_HT_R2->Write();
572
573     h_numberJet->Write();
574
575     h_BTag->Write();
576     h_BTag_PT->Write();
577     h_BTag_Eta->Write();
578     h_BTag_DPhi_MET->Write();
579     h_BTags_per_Event->Write();
580
581     h2_ISR_PTEta->Write();
582     h2_jet_PTEta->Write();
583     h2_dif_PTEta->Add(h2_ISR_PTEta,h2_jet_PTEta,1,-1);
584     h2_dif_PTEta->Write();
585
586     h2_dif_lead_PTEta->Add(h2_ISR_PTEta,h2_leading_PTEta
587         ,1,-1);
587     h2_dif_lead_PTEta->Write();
588
589     {
590         TCanvas *C = new TCanvas("Eta","Pseudorapidity"
591             ,1280,720);
591         Present(h_ISR_Eta,h_jet_Eta,C,1,"h","Num. Jets /
592             Total",122);
592         C->Write();
593         C->Close();
594
595         C = new TCanvas("Eta ISR vs BTag","Pseudorapidity

```

```

        ISR vs BTag",1280,720);
596 Present(h_ISR_Eta,h_BTag_Eta,C,1,"h","Num. Jets /
        Total",122);
597 C->Write();
598 C->Close();
599
600 C = new TCanvas("Eta ISR vs Leading","
        Pseudorapidity ISR vs Leading",1280,720);
601 Present(h_ISR_Eta,h_leading_Eta,C,1,"h","Num. Jets
        / Total",122);
602 C->Write();
603 C->Close();
604
605 C = new TCanvas("Transverse momentum","Transverse
        momentum",1280,720);
606 Present(h_ISR_PT,h_jet_PT,C,2,"PT [GeV]","Num.
        Jets / Total");
607 C->Write();
608 C->Close();
609
610 C = new TCanvas("Transverse momentum ISR vs
        Leading","Transverse momentum ISR vs Leading"
        ,1280,720);
611 Present(h_ISR_PT,h_leading_PT,C,2,"PT [GeV]","Num.
        Jets / Total");
612 C->Write();
613 C->Close();
614
615 C = new TCanvas("Transverse momentum ISR vs B_Tag"
        ,"Transverse momentum ISR vs B_Tag",1280,720);
616 Present(h_ISR_PT,h_BTag_PT,C,2,"PT [GeV]","Num.
        Jets / Total");
617 C->Write();
618 C->Close();
619
620 C = new TCanvas("Transverse momentum ISR, B_Tag,
        Leading","Transverse momentum ISR, B_Tag,
        Leading",1280,720);
621 Present_3(h_ISR_PT,h_BTag_PT,h_leading_PT,C,2,"PT
        [GeV]","Num. Jets / Total");

```

```

622     C->Write();
623     C->Close();
624
625     C = new TCanvas("Transverse momentum ISR, B_Tag,
        Leading LOG","Transverse momentum ISR, B_Tag,
        Leading LOG",1280,720);
626     Present_3(h_ISR_PT,h_BTag_PT,h_leading_PT,C,2,"PT
        [GeV]","Num. Jets / Total",12,12,true);
627     C->Write();
628     C->Close();
629
630     C = new TCanvas("Transverse mass Leading vs ISR
        Jet","Transverse mass Leading vs ISR Jet"
        ,1280,720);
631     Present(h_ISR_MT,h_leading_MT,C,2,"MT [GeV]","Num.
        Jets / Total");
632     C->Write();
633     C->Close();
634
635     C = new TCanvas("Transverse mass ISR vs Jet","
        Transverse mass ISR vs Jet",1280,720);
636     Present(h_ISR_MT,h_jet_MT,C,2,"MT [GeV]","Num.
        Jets / Total");
637     C->Write();
638     C->Close();
639
640     C = new TCanvas("Phi","Phi",1280,720);
641     Present(h_ISR_Phi,h_jet_Phi,C,3,"f","Num. Jets /
        Total",122);
642     C->Write();
643     C->Close();
644
645     C = new TCanvas("Delta Phi - Jet - MET","Delta Phi
        - Jet - MET",1280,720);
646     Present(h_ISR_DPhi_MET,h_jet_DPhi_MET,C,3,"Df","
        Num. Jets / Total",122);
647     C->Write();
648     C->Close();
649
650     C = new TCanvas("Delta Phi - Jet - MET - Btag","

```

```

        Delta Phi - Jet - MET - Btag",1280,720);
651 Present(h_ISR_DPhi_MET,h_BTag_DPhi_MET,C,3,"Df",
        Num. Jets / Total",122);
652 C->Write();
653 C->Close();
654
655 C = new TCanvas("Delta Phi - Jet - MET - leading",
        "Delta Phi - Jet - MET - leading",1280,720);
656 Present(h_ISR_DPhi_MET,h_leading_DPhi_MET,C,1,"Df",
        "Num. Jets / Total",122);
657 C->Write();
658 C->Close();
659
660 C = new TCanvas("MET > 120","MET > 120",1280,720);
661 Present(h_MET,h_MET_hpt1,C,2,"MET","Num. Jets /
        Total");
662 C->Write();
663 C->Close();
664
665 C = new TCanvas("MET > 200","MET > 200",1280,720);
666 Present(h_MET,h_MET_hpt2,C,2,"MET","Num. Jets /
        Total");
667 C->Write();
668 C->Close();
669
670 C = new TCanvas("MET > 240","MET > 240",1280,720);
671 Present(h_MET,h_MET_hpt3,C,2,"MET","Num. Jets /
        Total");
672 C->Write();
673 C->Close();
674
675 C = new TCanvas("HT ratio comparison","HT ratio
        comparison",1280,720);
676 Present(h_HT_R1,h_HT_R2,C,2,"HT","Num. Jets /
        Total");
677 C->Write();
678 C->Close();
679
680 C = new TCanvas("PT vs ETA - ISR","PT vs ETA - ISR
        ",1280,720);

```

```

681     Plot_Single_2D(h2_ISR_PTEta,C,2, "PT [GeV]", "h",
682                   12, 122);
683     C->Write();
684     C->Close();
685
686     C = new TCanvas("PT vs ETA - Jet","PT vs ETA - Jet
        ",1280,720);
687     Plot_Single_2D(h2_jet_PTEta,C,2, "PT [GeV]", "h",
688                   12, 122);
689     C->Write();
690     C->Close();
691
692     C = new TCanvas("PT vs ETA - Diff with any jet","
        PT vs ETA - Diff with any jet",1280,720);
693     Plot_Single_2D(h2_dif_PTEta,C,2, "PT [GeV]", "h",
694                   12, 122);
695     C->Write();
696     C->Close();
697
698     C = new TCanvas("PT vs ETA - leading","PT vs ETA -
        leading",1280,720);
699     Plot_Single_2D(h2_leading_PTEta,C,2, "PT [GeV]", "
        h", 12, 122);
700     C->Write();
701     C->Close();
702
703     C = new TCanvas("PT vs ETA - Diff with leading","
        PT vs ETA - Diff with leading",1280,720);
704     Plot_Single_2D(h2_dif_lead_PTEta,C,2, "PT [GeV]",
        "h", 12, 122);
705     C->Write();
706     C->Close();
707
708     C = new TCanvas("HT","HT",1280,720);
709     Plot_Single(h_HT,C,2, "HT [GeV]", "Num. Jets /
        Total", 12, 12);
710     C->Write();
711     C->Close();
712
713     C = new TCanvas("Number_of_B_Tags","Number of B

```



```

    Tags",1280,720);
711   Plot_Single(h_BTags_per_Event,C,2, "B Tags / event
    ", "Num. Jets / Total", 12, 12);
712   C->Write();
713   C->Close();
714
715   C = new TCanvas("Jet_multipliticity","Jet
    multiplicity",1280,720);
716   Present(h_ISR_multiplicity,h_jet_multiplicity,C,2,
    "Tracks","Num. Jets / Total");
717   C->Write();
718   C->Close();
719
720   C = new TCanvas("Delta_R_-_Jet_size","Delta R -
    Jet Size",1280,720);
721   Present(h_ISR_DeltaR,h_jet_DeltaR,C,1,"Delta_R","
    Num. Jets / Total");
722   C->Write();
723   C->Close();
724
725   // Correlated variables
726   C = new TCanvas("Cor_Delta_PT_Jet", "Delta PT jet"
    ,1280,720);
727   Present(h_ISR_Delta_PT,h_jet_Delta_PT,C,2,"PT [GeV
    ]","Num. Jets / Total");
728   C->Write();
729   C->Close();
730
731   C = new TCanvas("Cor_PT_proportion","PT proportion
    ",1280,720);
732   Present(h_ISR_PT_HT,h_jet_PT_HT,C,2,"PT/HT","Num.
    Jets / Total");
733   C->Write();
734   C->Close();
735
736   C = new TCanvas("Cor_Delta_Eta_Average","Delta Eta
    Average",1280,720);
737   Present(h_ISR_Delta_Eta,h_jet_Delta_Eta,C,2,"Dh","
    Num. Jets / Total",122);
738   C->Write();

```

```

739     C->Close();
740
741     C = new TCanvas("Cor_Delta_Phi_Jet_MET_other_jets"
742         ,"Delta Phi - Jet MET - other jets",1280,720);
743     Present(h_ISR_DPhi_MET_other,h_jet_DPhi_MET_other,
744         C,2,"Df","Num. Jets / Total",122);
745     C->Write();
746     C->Close();
747
748     C = new TCanvas("Cor_PT_over_<PT_other>","PT/<
749         PT_other>",1280,720);
750     Present(h_ISR_PT_over_PT_others,
751         h_jet_PT_over_PT_others,C,2,"PT/<PT>","Num.
752         Jets / Total");
753     C->Write();
754     C->Close();
755
756     C = new TCanvas("Cor_Eta_over_<Eta_other>","Eta/<
757         Eta_other>",1280,720);
758     Present(h_ISR_Eta_over_Eta_others,
759         h_jet_Eta_over_Eta_others,C,3,"h/<h>","Num.
760         Jets / Total",122);
761     C->Write();
762     C->Close();
763
764     C = new TCanvas("Cor_Delta_Phi_over_<
765         Delta_Phi_other>","Delta_Phi/<Delta_Phi_other>"
766         ,1280,720);
767     Present(h_ISR_DPhi_over_Phi_others,
768         h_jet_DPhi_over_Phi_others,C,3,"Df/<Df>","Num.
769         Jets / Total",122);
770     C->Write();
771     C->Close();
772
773     // Comparison with the leading Jet
774     C = new TCanvas("Leading_Delta_PT","Delta PT:
775         PT_leading-PT",1280,720);
776     Present(h_ISR_Delta_PT_leading,
777         h_jet_Delta_PT_leading,C,2,"(PT_leading - PT)",
778         "Num. Jets / Total");

```

```

764     C->Write();
765     C->Close();
766
767     C = new TCanvas("Leading_Delta_Eta","Delta Eta: |
768         Eta-Eta_leading|",1280,720);
769     Present(h_ISR_Delta_Eta_leading,
770         h_jet_Delta_Eta_leading,C,2,"|Eta - Eta_leading
771         |","Num. Jets / Total");
772     C->Write();
773     C->Close();
774
775 }
776
777 hfile->Close();
778
779 TFile* hfile2 = new TFile("./histos/histos2.root", "
780     RECREATE");
781 h_ISR_PT_comp->Write();
782 h_ISR_Eta_comp->Write();
783 h_ISR_DPhi_MET_comp->Write();
784
785 hist_ISR_PT->Write();
786 hist_ISR_Abs_Eta->Write();
787 hist_ISR_DPhi_MET->Write();
788 hist_ISR_PT_ratio->Write();
789 hist_ISR_Delta_Eta->Write();
790 hist_ISR_DPhi_MET_other->Write();
791 hist_ISR_Delta_PT_leading->Write();
792 hist_ISR_Delta_Eta_leading->Write();
793
794 hist_jet_PT->Write();
795 hist_jet_Abs_Eta->Write();
796 hist_jet_DPhi_MET->Write();
797 hist_jet_PT_ratio->Write();
798 hist_jet_Delta_Eta->Write();
799 hist_jet_DPhi_MET_other->Write();
800 hist_jet_Delta_PT_leading->Write();
801 hist_jet_Delta_Eta_leading->Write();
802
803 hfile2->Close();

```

```

800
801     return 0;
802 }

```

B.4 Creating histograms

```

1  /*
2  -----
3  -----      Universidad de los Andes      -----
4  -----      Departamento de Fisica        -----
5  -----      Joven Investigador            -----
6  -----      Andres Felipe Garcia Albarracin -----
7  -----      Juan Carlos Sanabria Arenas    -----
8  -----
9
10 This algorithm fills 2 N-dimensional histograms.
11 The histograms contain kinematic variables of ISR
12 jets and non ISR jets.
13
14 The user can choose 3 of 8 variables for filling
15 the histograms:
16 1. PT
17 2. Abs(Eta) // Eta is a pair function
18 3. Delta Phi_MET
19 4. PT_ratio
20 5. Delta Eta_aver
21 6. Delta Phi_MET_others
22 7. Delta PT_others
23 8. Delta Eta_others
24
25 In order to choose them, the code should be run as
26 ./Creating_histo N1 N2 N3, where N1 N2 and N3 are
27 the index of the 3 variables. If no parameter is
28 passed as parameter, N1 N2 and N3 will be 0,1 and 3
29 by default.
30 */
31
32
33 #include "ROOTFunctions.h"

```

```

34 #include "graphs_Funcs.h"
35 #include "functions.h"
36 #include "histoN.h"
37 #include "DelphesFunctions.h"
38
39 // Global Variables
40 const Double_t PI = TMath::Pi();
41
42 // Other simulations parameters
43 const Char_t channel_histo = '_'; // 's' for sTops and '
    '_' for Tops
44 const Char_t ISR_or_NOT_histo[] = "WI"; // "WI" with ISR
    , "SI" without (Here it does not make any sense), "bb
    " bjets production
45 const Bool_t atServer = true; // True if it is run at
    the server, false at the university's pc
46 const Bool_t Matching = true; // True if a matching has
    been done between MG and Pythia, false otherwise
47
48 int main(int argc, char **argv){
49     std::cout.precision(4);
50     // Counting time
51     Double_t initialTime = clock();
52
53     cout << "\n *** Creating histograms *** \n" << endl;
54
55     // Variables for initializing histograms
56     Int_t dims = 3;
57     Double_t min_Values[3] = {0,-5.2,0};
58     Double_t max_Values[3] = {800,5.2,PI};
59
60     /*
61      * Read inputs and set variables for analysis
62      */
63     Int_t var_index[3] = {0,1,2}; // Index of the 3
        variables for analysis. By default 0, 1 and 2
64     string variables[8] = {"PT","Abs(Eta)","Delta Phi_MET
        ","PT_ratio","Delta Eta_aver","Delta
        Phi_MET_others","Delta PT_leading","Delta
        Eta_leading"};

```

```

65     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
66     // Min and maximun values of the eight variables
67     Double_t var_min_values[8] =
        {0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0};
68     Double_t var_max_values[8] = {800,5.2,PI,8.0,7.0,PI
        ,500,6.5};
69
70     if (argc == 4){
71         cout << "Filling histograms with the variables:"
            << endl;
72         for (Int_t ind = 0; ind < 3; ind ++){
73             var_index[ind] = atoi(argv[ind+1]);
74         }
75         cout << endl;
76     }
77     else if (argc == 1) {
78         cout << "Filling histograms with the default
            variables:" << endl;
79     }
80     else {
81         cout << "Error at calling this algorithm. Use as:"
            << endl;
82         cout << "\t ./Creating_histo N1 N2 N3 or ./
            Creating_histo" << endl;
83         cout << "Read the documentation at the beginning
            of the code for further information\n" << endl;
84         return 1;
85     }
86
87     cout << "Var \t\t min_Value \t max_Value" << endl;
88     for (Int_t ind = 0; ind < 3; ind ++){
89         min_Values[ind] = var_min_values[var_index[ind]];
90         max_Values[ind] = var_max_values[var_index[ind]];
91         cout << var_index[ind] << ". " << variables[
            var_index[ind]] <<
92             "\t" << min_Values[ind] << "\t" <<
            max_Values[ind] << endl;
93     }

```

```

94     cout << endl;
95
96
97     /*
98     * Initializing the 3-dimensional histogram
99     */
100    Int_t bins[3] = {20,20,20};
101    histoN* histoISR = new histoN(dims,min_Values,
102                                  max_Values,bins);
103    histoN* histoNonISR = new histoN(dims,min_Values,
104                                      max_Values,bins);
105    // Input variables of each histogram
106    Double_t values[3] = {0.0,0.0,0.0};
107
108    for(int iRun = 100; iRun < 261; iRun++){
109        // Create chains of root trees
110        TChain chain_Delphes("Delphes");
111
112        // Loading simulations from Delphes
113        Char_t *local_path;
114        local_path = (Char_t*) malloc(512*sizeof(Char_t));
115        if (atServer)
116            strcpy(local_path, "/home/af.garcia1214/
117                               PhenoMCsamples/Simulations/
118                               MG_pythia8_delphes_parallel/"); // At the
119                               server
120        else
121            strcpy(local_path, "/home/afgarcia1214/
122                               Documentos/Simulations/"); // At the
123                               University's pc
124
125        Char_t *head_folder;
126        head_folder = (Char_t*) malloc(512*sizeof(Char_t))
127            ;
128        if (Matching)
129            strcpy(head_folder, "_Tops_Events_WI_Matching/");
130            ;
131        else
132            strcpy(head_folder, "_Tops_Events_WI/");
133        head_folder[0] = channel_histo;

```

```

125     head_folder[13] = ISR_or_NOT_histo[0];
126     head_folder[14] = ISR_or_NOT_histo[1];
127
128     Char_t current_folder[] = "_Tops_MG_1K_AG_WI_003/"
        ;
129     current_folder[0] = channel_histo;
130     current_folder[15] = ISR_or_NOT_histo[0];
131     current_folder[16] = ISR_or_NOT_histo[1];
132
133     Char_t unidad = 0x30 + iRun%10;
134     Char_t decena = 0x30 + int(iRun/10)%10;
135     Char_t centena = 0x30 + int(iRun/100)%10;
136
137     current_folder[18] = centena;
138     current_folder[19] = decena;
139     current_folder[20] = unidad;
140
141     Char_t *file_delphes;
142     file_delphes = (Char_t*) malloc(512*sizeof(Char_t)
        );
143     strcpy(file_delphes,local_path);
144     strcat(file_delphes,head_folder);
145     strcat(file_delphes,current_folder);
146     strcat(file_delphes,"Events/run_01/output_delphes.
        root");
147
148     cout << "Writing run: " << centena << decena << unidad <<
        endl;
149     cout << "\nReading the file: \nDelphes: " <<
        file_delphes << endl;
150
151     chain_Delphes.Add(file_delphes);
152     // Objects of class ExRootTreeReader for reading
        the information
153     ExRootTreeReader *treeReader_Delphes = new
        ExRootTreeReader(&chain_Delphes);
154
155     Long64_t numberOfEntries = treeReader_Delphes->
        GetEntries();
156

```



```

157 // Get pointers to branches used in this analysis
158 TClonesArray *branchJet = treeReader_Delphes->
    UseBranch("Jet");
159 TClonesArray *branchMissingET = treeReader_Delphes
    ->UseBranch("MissingET");
160
161 cout << endl;
162 cout << " Number of Entries Delphes = " <<
    numberOfEntries << endl;
163 cout << endl;
164
165 // particles, jets and vectors
166 MissingET *METpointer;
167 TLorentzVector *vect_currentJet = new
    TLorentzVector;
168 TLorentzVector *vect_auxJet = new TLorentzVector;
169 TLorentzVector *vect_leading = new TLorentzVector;
170 Jet *currentJet = new Jet;
171 Jet *auxJet = new Jet;
172
173 // Temporary variables
174 Double_t MET = 0.0; // Missing transverse energy
175 Double_t delta_phi = 0.0; // difference between
    the phi angle of MET and the jet
176 Double_t transverse_mass = 0.0; // Transverse mass
177 Int_t numMatches = 0; // Number of matched jets
178 Double_t delta_PT_jet = 0.0; // |PT-<PT>|
179 Double_t PT_sum = 0.0; // sum(PT)
180 Double_t PT_aver = 0.0; // <PT>
181 Double_t Delta_eta_aver = 0.0; // sum_i|eta-eta_i
    |/(Nj-1)
182 Double_t Delta_phi_sum = 0.0; // sum delta_phi
183 Double_t Delta_phi_other_jets = 0.0; // Average of
    delta phi of other jets
184 Double_t PT_ratio = 0.0; // PT/PT_others
185 Double_t Delta_PT_leading = 0.0; // PT -
    PT_leading
186 Double_t Delta_Eta_leading = 0.0; // |Eta -
    Eta_leading|
187

```

```

188 // Jet with greatest PT
189 Double_t PT_max = 0;
190 Int_t posLeadingPT = -1;
191 Int_t ISR_greatest_PT = 0;
192 Double_t MT_leading_jet = 0.0; // Transverse mass
193
194 /*
195  * Some variables used through the code
196  */
197 Int_t ISR_jets[numberOfEntries];
198 Int_t NumJets = 0;
199
200 Char_t *local_path_binary;
201 local_path_binary = (Char_t*) malloc(512*sizeof(
202     Char_t));
203 if (atServer)
204     strcpy(local_path_binary, "/home/af.garcia1214/
205         PhenoMCsamples/Results/matching_Results/"); //
206         At the server
207 else
208     strcpy(local_path_binary, "/home/afgarcia1214/
209         Documentos/Results_and_data/matching_Results
210         /"); // At the University's pc
211
212 Char_t *head_folder_binary;
213 head_folder_binary = (Char_t*) malloc(512*sizeof(
214     Char_t));
215 if (Matching)
216     strcpy(head_folder_binary, "
217         _Tops_matches_WI_Matching/");
218 else
219     strcpy(head_folder_binary, "_Tops_matches_WI/");
220 head_folder_binary[0] = channel_histo;
221 head_folder_binary[13] = ISR_or_NOT_histo[0];
222 head_folder_binary[14] = ISR_or_NOT_histo[1];
223
224 Char_t matching_name[] = "ISR_jets_Tops_WI_003.bn"
225     ;
226 matching_name[8] = channel_histo;
227 matching_name[14] = ISR_or_NOT_histo[0];

```

```

220     matching_name[15] = ISR_or_NOT_histo[1];
221
222     matching_name[17] = centena;
223     matching_name[18] = decena;
224     matching_name[19] = unidad;
225
226     Char_t * fileName;
227     fileName = (Char_t*) malloc(512*sizeof(Char_t));
228     strcpy(fileName,local_path_binary);
229     strcat(fileName,head_folder_binary);
230     strcat(fileName,matching_name);
231
232     ifstream ifs(fileName,ios::in | ios::binary);
233
234     for (Int_t j = 0; j<numberOfEntries; j++){
235         ifs.read((Char_t *) (ISR_jets+j),sizeof(
            Int_t));
236     }
237     ifs.close();
238
239     /*
240     * Main cycle of the program
241     */
242     numberOfEntries = 100000;
243     for (Int_t entry = 0; entry < numberOfEntries; ++
        entry){
244         // Progress
245         if(numberOfEntries>10 && (entry%((int)
            numberOfEntries/10))==0.0){
246             cout<<"progress = "<<(entry*100/
                numberOfEntries)<<"%\t";
247             cout<< "Time : "<< (clock()-initialTime)/
                double_t(CLOCKS_PER_SEC)<<"s"<<endl;
248         }
249
250         // Load selected branches with data from
            specified event
251         treeReader_Delphes->ReadEntry(entry);
252
253         // MET

```

```

254     METpointer = (MissingET*) branchMissingET->At
        (0);
255     MET = METpointer->MET;
256
257     NumJets=branchJet->GetEntries();
258
259     // checking the ISR
260     if (ISR_jets[entry] == -1 || NumJets < 3)
261         continue;
262
263     if (ISR_jets[entry] >= NumJets){
264         cout << "Error en el matching" << endl;
265         return 1;
266     }
267
268     // 3 PT ratio
269     PT_aver = 0.0;
270     PT_sum = 0.0;
271     PT_ratio = 0.0;
272
273     // 4 Delta Eta aver
274     Delta_eta_aver = 0.0;
275
276     // 5 Delta Phi others
277     Delta_phi_sum = 0.0;
278     Delta_phi_other_jets = 0.0;
279
280     // 6 Delta PT leading
281     PT_max = 0.0;
282     Delta_PT_leading = 0.0;
283     delta_PT_jet = 0.0; // If needed
284
285     // 7 Delta Eta leading
286     Delta_Eta_leading = 0.0;
287
288     // Reset Var_values (Not necessary)
289     for(Int_t ind = 0; ind < 8; ind++){
290         var_values[ind] = 0.0;
291         if (ind < dims) values[ind] = 0.0;
292     }

```

```

293
294     // Preliminary for. It is used to calculate
        PT_aver and Delta_phi_sum
295     for (Int_t iJet = 0; iJet<NumJets; iJet++){
296         currentJet = (Jet*) branchJet->At(iJet);
297         vect_currentJet->SetPtEtaPhiM(currentJet->PT
            ,currentJet->Eta,currentJet->Phi,
            currentJet->Mass);
298         PT_sum += vect_currentJet->Pt();
299         delta_phi = deltaAng(vect_currentJet->Phi(),
            METpointer->Phi());
300         Delta_phi_sum += delta_phi;
301         // PT Leading jet
302         if(PT_max < vect_currentJet->Pt()){
303             PT_max = vect_currentJet->Pt();
304             posLeadingPT = iJet;
305         }
306     }
307
308     numMatches++;
309
310     //PT_aver
311     PT_aver = PT_sum/NumJets;
312
313     // Leading PT
314     currentJet = (Jet*) branchJet->At(posLeadingPT)
        ;
315     vect_leading->SetPtEtaPhiM(currentJet->PT,
        currentJet->Eta,currentJet->Phi,currentJet->
        Mass);
316
317     for (Int_t iJet = 0; iJet<NumJets; iJet++){
318         currentJet = (Jet*) branchJet->At(iJet);
319         vect_currentJet->SetPtEtaPhiM(currentJet->PT
            ,currentJet->Eta,currentJet->Phi,
            currentJet->Mass);
320
321         // 2 Delta Phi MET
322         delta_phi = deltaAng(vect_currentJet->Phi(),
            METpointer->Phi());

```

```

323
324 // PT ratio
325 PT_ratio = vect_currentJet->Pt()*(NumJets-1)
           /(PT_sum-vect_currentJet->Pt());
326
327 // 4 Delta Eta Aver
328 Delta_eta_aver = 0.0;
329 // For cycle used to calculate
           Delta_eta_aver
330 for(Int_t iJet2 = 0; iJet2<NumJets; iJet2++)
    {
331         auxJet = (Jet*) branchJet->At(iJet2);
332         vect_auxJet->SetPtEtaPhiM(auxJet->PT,
           auxJet->Eta,auxJet->Phi,auxJet->Mass);
333         if (iJet2 != iJet) Delta_eta_aver +=
           TMath::Abs(vect_auxJet->Eta()-
           vect_currentJet->Eta());
334     }
335 Delta_eta_aver = Delta_eta_aver/(NumJets-1);
336
337 // 5 Delta Phi MET Others
338 Delta_phi_other_jets = (Delta_phi_sum-
           delta_phi)/(NumJets-1);
339
340 // 6 Delta PT leading
341 Delta_PT_leading = vect_leading->Pt()-
           vect_currentJet->Pt();
342
343 // 7 Delta Eta leading
344 Delta_Eta_leading = TMath::Abs(
           vect_currentJet->Eta()-vect_leading->Eta
           ());
345
346 // Other variables
347 delta_PT_jet = TMath::Abs(vect_currentJet->
           Pt()-PT_aver);
348 transverse_mass = sqrt(2*vect_currentJet->Pt
           ()*MET*(1-cos(delta_phi)));
349
350 // Filling the array with the variables'

```

```

351         values
352         var_values[0] = vect_currentJet->Pt();
353         var_values[1] = TMath::Abs(vect_currentJet->
354             Eta());
355         var_values[2] = delta_phi;
356         var_values[3] = PT_ratio;
357         var_values[4] = Delta_eta_aver;
358         var_values[5] = Delta_phi_other_jets;
359         var_values[6] = Delta_PT_leading;
360         var_values[7] = Delta_Eta_leading;
361
362         for (Int_t ind = 0; ind < dims; ind++){
363             int pos = *(var_index+ind);
364             values[ind] = *(var_values+pos);
365         }
366
367         if (iJet != ISR_jets[entry]){
368             // Non ISR jet
369             histoNonISR->fill(values);
370         }
371         else{
372             // ISR jet
373             histoISR->fill(values);
374         }
375     }
376 }
377
378     cout<<"progress = 100%\t";
379     cout<<"Time : "<< (clock()-initialTime)/double_t(
380         CLOCKS_PER_SEC)<<"s"<<endl;
381     cout<<"\nNumber of Written Events: "<<numMatches<<
382         endl;
383 } // End run's for cicle
384
385 /*
386  * Writing the histogram
387  */
388 // Counting time
389 Double_t partialTime = clock();

```

```

387     cout<< "Time building the histogram:"<< (partialTime-
        initialTime)/double_t(CLOCKS_PER_SEC)<<"s"<<endl;
388
389     // Writing the histogram
390     cout<<"Min value 1: "<<min_Values[0]<<endl;
391     Int_t* freq;
392     for(Int_t j = 0; j<dims; j++){
393         cout<<"ISR Jets - Events of the dimension:\t"<<j<<
            endl;
394         freq = histoISR->getHistDim(j);
395         for(Int_t i = 0; i<bins[j];i++){
396             cout<<"\t"<<freq[i];
397             if(i>0 && ((i+1)%10 == 0.0))
398                 cout<<endl;
399         }
400         cout<<endl;
401     }
402
403     cout<<endl<<"\t\t ***"<<endl<<endl;
404     for(Int_t j = 0; j<dims; j++){
405         cout<<"Non ISR Jets - Events of the dimension:\t"
            <<j<<endl;
406         freq = histoNonISR->getHistDim(j);
407         for(Int_t i = 0; i<bins[j];i++){
408             cout<<"\t"<<freq[i];
409             if(i>0 && ((i+1)%10 == 0.0))
410                 cout<<endl;
411         }
412         cout<<endl;
413     }
414
415     cout<<"Entries: "<<histoISR->getEntries()<<endl;
416
417     /*
418     * Creating histograms
419     */
420     cout<<"\nWriting..."<<endl;
421
422     // Defining the names of the files
423     Char_t combination[] = "_____"; // Combination of

```



```

variables
424 for (Int_t ind = 0; ind < dims; ind++){
425     *(combination+(ind*2)+1) = (Char_t) (0x30 +
        var_index[ind]); // Int to char
426 }
427
428 Char_t *local_path_results;
429 local_path_results = (Char_t*) malloc(512*sizeof(
    Char_t));
430 if (atServer)
431     strcpy(local_path_results, "/home/af.garcia1214/
        PhenoMCsamples/Results/histo_folder/"); // At
        the server
432 else
433     strcpy(local_path_results, "/home/afgarcia1214/
        Documentos/Results_and_data/histo_folder/"); //
        At the University's pc
434
435 Char_t *head_folder_results;
436 head_folder_results = (Char_t*) malloc(512*sizeof(
    Char_t));
437 if (Matching)
438     strcpy(head_folder_results, "
        _Tops_histos_WI_Matching/");
439 else
440     strcpy(head_folder_results, "_Tops_histos_WI/");
441 head_folder_results[0] = channel_histo;
442 head_folder_results[13] = ISR_or_NOT_histo[0];
443 head_folder_results[14] = ISR_or_NOT_histo[1];
444
445 Char_t *info_ISR_name;
446 info_ISR_name = (Char_t*) malloc(sizeof(char)*512);
447 strcpy(info_ISR_name, local_path_results);
448 strcat(info_ISR_name, head_folder_results);
449 strcat(info_ISR_name, "info_histo_ISR");
450 strcat(info_ISR_name, combination);
451 strcat(info_ISR_name, ".txt");
452
453 Char_t *array_ISR_name;
454 array_ISR_name = (Char_t*) malloc(sizeof(char)*512);

```

```

455     strcpy(array_ISR_name, local_path_results);
456     strcat(array_ISR_name, head_folder_results);
457     strcat(array_ISR_name, "array_histo_ISR");
458     strcat(array_ISR_name, combination);
459     strcat(array_ISR_name, ".bn");
460
461     Char_t *info_Non_ISR_name;
462     info_Non_ISR_name = (Char_t*) malloc(sizeof(char)
         *512);
463     strcpy(info_Non_ISR_name, local_path_results);
464     strcat(info_Non_ISR_name, head_folder_results);
465     strcat(info_Non_ISR_name, "info_histo_Non_ISR");
466     strcat(info_Non_ISR_name, combination);
467     strcat(info_Non_ISR_name, ".txt");
468
469     Char_t *array_Non_ISR_name;
470     array_Non_ISR_name = (Char_t*) malloc(sizeof(char)
         *512);
471     strcpy(array_Non_ISR_name, local_path_results);
472     strcat(array_Non_ISR_name, head_folder_results);
473     strcat(array_Non_ISR_name, "array_histo_Non_ISR");
474     strcat(array_Non_ISR_name, combination);
475     strcat(array_Non_ISR_name, ".bn");
476
477     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;
478
479     histoISR->writeClass((Char_t*) info_ISR_name, (Char_t
         *) array_ISR_name);
480     histoNonISR->writeClass((Char_t*) info_Non_ISR_name, (
         Char_t*) array_Non_ISR_name);
481     cout<< "Time writing the file:"<< (clock()-
         partialTime)/double_t(CLOCKS_PER_SEC)<<"s"<<endl;
482
483     cout<<"Fin :)"<<endl;
484
485     return 0;
486 }

```

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>.