Documentation of the project: ISR jet tagging

Author:
Andrés Felipe García Albarracín

Advisor:
Juan Carlos Sanabria, Ph.D.

July 2, 2015

Contents

Chapter 1

Introduction

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

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

In this project, three of those programs were used: MadGraph 5.2 (MadEvent) [1], Pythia 8.2 [2] [3] and Delphes 3.2 [4] with the aim of simulating proton - proton collision events. The description of those programs and their particular purposes in the project are described in chapter 2. In addition, chapter 2 includes the explanation of the codes and the scripts that were developed both to integrate those programs, and to run the simulations under specific conditions.

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

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

Chapter 2

Simulation chain

"*Divide et impera*",
"Divide and conquer"

Philip II of Macedon

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

As presented before, MadGraph and Pythia handle the Physics of the collision. Again, there is more than a single program for this task, and now the reason to use two programs lies on the limits of the theoretical models. At the very first moment of the collision when the Energy Density of the System is high enough, perturbative Quantum Chromo-Dynamics (pQCD), Quantum Electro-Dynamics (QED) and ElectroWeak Theory are the most accurate models known so far. MadGraph, and specifically MadEvent, use them to calculate the transverse sections of a particular channel defined by the user. From this calculation and the Monte Carlo models, it randomly establishes the kinematic variables of the resulting particles of the collision.

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

Although the usage of several programs for the simulation means better results, it also implies the challenge of connecting them. This task has already been done inside the MadGraph package, which connects MadEvent + Pythia 6 + Delphes / PGS[1]. However, the version of Pythia included there (v.6) is old and does not offer the possibility of controlling ISR emissions as the last one (v.8) does. As ISR emissions were the main focus of the project, it was convenient to use Pythia 8 instead of Pythia 6 and therefore to develop the integration of MadGraph 5.2 with Pythia 8.2 and Delphes 3.2.

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

2.1 Usage of MadGraph 5.2

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

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

been correctly installed [2].

1. In the folder where MadGraph has been installed, type:
   `./bin/mg5_aMC`

2. Once MadGraph has been initialized, import the Standard Model parameters:
   `import model sm`

3. Generate the event $p\,p\ \to t\ \tilde{t}$:
   `generate p p > t t~`

4. Create an output folder where all the simulation files will be saved, in this case test_t_tbar:
   `output test_t_tbar`

5. Launch the Feynman diagrams production:
   `launch -m`
   and select the number of cores you want to use for the simulation

6. Turn off Pythia and other programs[3]. You can switch off and on by typing the number before the program (type 1 to toggle pythia, for instance). Then, press enter.

7. Modify the `run_card.dat` file by typing 2. Write `:32` and press enter to go to line 32, then type `i` and press enter to modify the file. Change the number of events from 10000 to 1000. Press `Esc` and write `:wq` to write and quit.

8. Press enter to run the simulation

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

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

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

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

```
import model sm
generate p p > t t~
output test_t_tbar -f
launch -m
2
pythia=OFF
Template/LO/Cards/run_card.dat
models/sm_v4/param_card.dat
```

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

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

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

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

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

2.2 Usage of Pythia 8.2

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

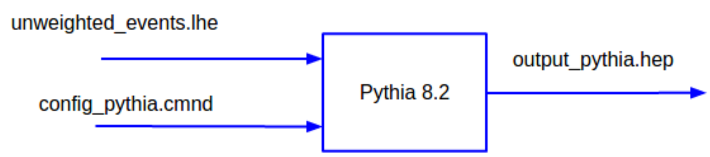

Figure 2.1

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

2.2.1 Code Usage

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

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

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

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

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

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

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

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

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

```
make hadronization02
```

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

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

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

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

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

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

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

! Hadronization:
PartonLevel:FSR = off ! switch final state radiation
PartonLevel:ISR = on ! switch initial state radiation
PartonLevel:MPI = off ! switch off multiparton interactions
Random:setSeed = on ! For random seed
Random:seed = 1 ! any number between 1 and 900,000,000
```

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

2.2.2 The code

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

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

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

With the definition of the pointer `myUserHooks` and its inclusion in the object `pythia`, the configuration stage finishes. Then, the execution starts by initializing the simulation at line 109. Basically, the simulation consists of the *for* loop of lines 111-125, where each iteration corresponds to the generation of a new event through the call of method `pythia.next()`. Observe that if the latter method returns `false`, either pythia has reached the end of the input file (from MadGraph), or an error has happened and the execution should stop if the maximum number of errors is reached. Once this has been verified, each cycle ends by writing the event in the output `.hep` file.

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

2.2.3 Pythia ntuple generation

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

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

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

***

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

2.3 Usage of Delphes 3.2

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

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

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

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

***

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

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

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

These three files are the final result of the simulation and as it will be presented later, the latter two will be used in this project. The procedure to obtain them has been presented and despite being straightforward, it is cumbersome as it requires several times the user intervention. Simulating would be a tedious task when several runs need to be executed such as the situation that this project deals with. Therefore, it was necessary to create an script that involved the three steps of the simulation. This script, originally written by Diego A. Sanz[8] to run MadGraph alone, was modified to include Pythia 8.2 and Delphes 3.2, and it is the topic of the next section.

2.4 Integration of MadGraph 5.2 + Pythia 8.2 + Delphes 3.2

To integrate MadGraph 5.2 with Pythia 8.2 and Delphes 3.2 two scripts were written, which can be found in the Appendix A.2 and in the repository of the project[9] at the folder `Codes/Simulation/MG_pythia8_delphes_parallel`. Those scripts allow parallel simulations taking advantage of the computing capabilities of the machine where the user is working.

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

```
./script_Integration.sh
```

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

[8] d-sanz@uniandes.edu.co

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

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

- Because the scripts execute parallel simulations, it is necessary to specify two folders where they will be saved: `EVENTSFOLDER` is the name of the head directory where all simulations will be saved, and `NAMESUBFOLDER` is the generic name of the folders that contain each simulation and that are located at `EVENTSFOLDER`. Thus, simulation #3 is saved in `EVENTSFOLDER/NAMESUBFOLDER3`.
- In total, each execution of `script_Integration.sh` run simulations from `INIRUN` to `ENDRUN`. Each of them consists of `NUMEVENTSRUN` events and its seed is the simulation number.
- Because MadGraph can develop some parallel calculations, `CORESNUMBER` is the number of cores devoted to each MadGraph run. Be aware that the total number of parallel runs times `CORESNUMBER` needs to be less or equal than the number of cores of your machine. Once MadGraph has been executed, only one core of `CORESNUMBER` is used to run Pythia and Delphes, because they only manage one thread.
- There are two sequences inside `script_Integration.sh` . The first one copies and modifies the run and param cards according to each simulation (it changes the seed, for instance). At the end of this sequence, those copies are located at the folders `/RunCards/` and `/ParamCards/` inside `EVENTSFOLDER`. When configuring `config_Integration.ini`, it is extremely important to use the templates of the files:
  - `run_card.dat`
  - `mgFile.mg5`
  - `input_pythia.cmnd`

  provided at the folder `Codes/Simulation/MG_pythia8_delphes_parallel /RunCard_Template` of the repository, as the script looks for certain variables defined in such templates and replace them with the specific parameters of each simulation.
- The second sequence inside `script_Integration.sh` runs the simulations. As it can be verified in Appendix A.2.2, it:

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

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

2.5 Example of the integration scripts

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

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

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

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

   (a) Open `/RunCard_Template/mgFile.mg5` and check the details of the MadGraph simulation. Observe, for instance, line 4 where the channel is specified.

   (b) Open `run_card.dat` and verify that the energy per beam is $6500 GeV$ in lines 41 and 42.

   (c) In the file `input_pythia.cmnd`, observe the same parameters presented in subsection 2.2.1. Additionally, the file includes some necessary settings to perform the *matching* procedure between MadGraph and Pythia. More information about it can be found at [7].

6. Execute the script by typing[10]:

   `./script_Integration.sh`

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

Chapter 3

Analysis 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_matchs_WI_*Matching*
Matching file	Matching information of a specific run	ISR_jetssTops_WI_005.bn
Histograms' folder	All histograms' head folders	histo_folder
Histograms' head folder	Histograms' files of a particular simulation (channel)	sTops_histos_WI_*Matching*
Histograms' files	Information of the N-dimensional histograms. Each histogram consists of 4 files: A binary and a plain text file for both ISR and Non ISR jets.	array_histo_ISR_0_1_2.bn array_histo_Non_ISR_0_1_2.bn info_histo_ISR_0_1_2.txt info_histo_Non_ISR_0_1_2.txt
Tagging folder	All tagging head folders	resultsTagging
Tagging head folder	Tagging result files of a particular simulation	sTops_result_WI_*Matching*
Tagging result files	Efficiency of the tagging algorithm for a particular channel and a specific selection of analysis variables.	sTops_WI_Overall_0_1_2.txt sTops_WI_hpt-050_0_1_2.txt sTops_WI_MET_pt_050_ k_2.0_0_1_2.png

Table 3.1: Naming convention of folders and files

3. '_*Matching*' appears if the matching procedure between MadGraph and Pythia has been done. If not, it does not appear in the name.

4. The sequence of numbers '_0_1_2' corresponds to the set of variables used for the analysis (Those variables will be explained later on).

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

3.2.1 The method

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

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

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

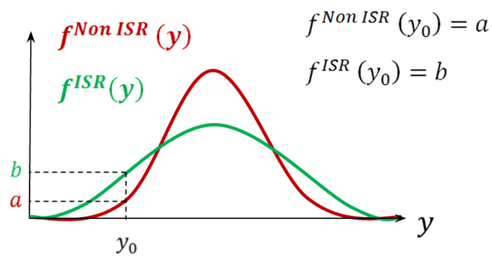

Figure 3.1: Probability distributions of a variable that distinguishes between ISR and Non ISR jets

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

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

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

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

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

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

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

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

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

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

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

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

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

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

3.2.2 From probability density functions to normalized histograms

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

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

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

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

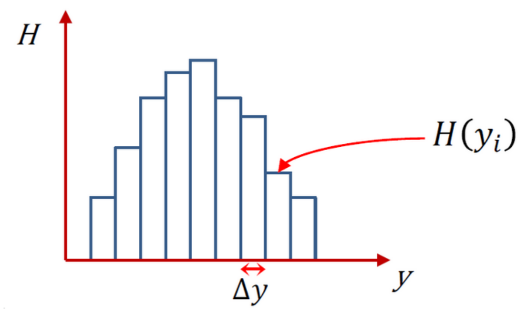

Figure 3.2: Shape of a histogram which does not exactly correspond with the probability density function

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

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

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

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

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

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

3.2.3 The Algorithm

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

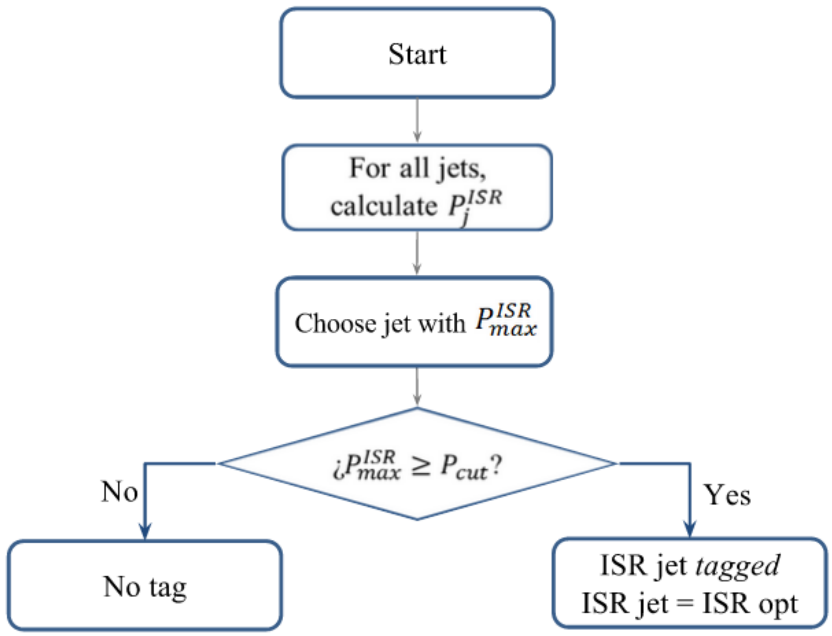

Figure 3.3: ISR jet tagging algorithm

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

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

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

3.2.4 The code

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

```
./ISR_tagging [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

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

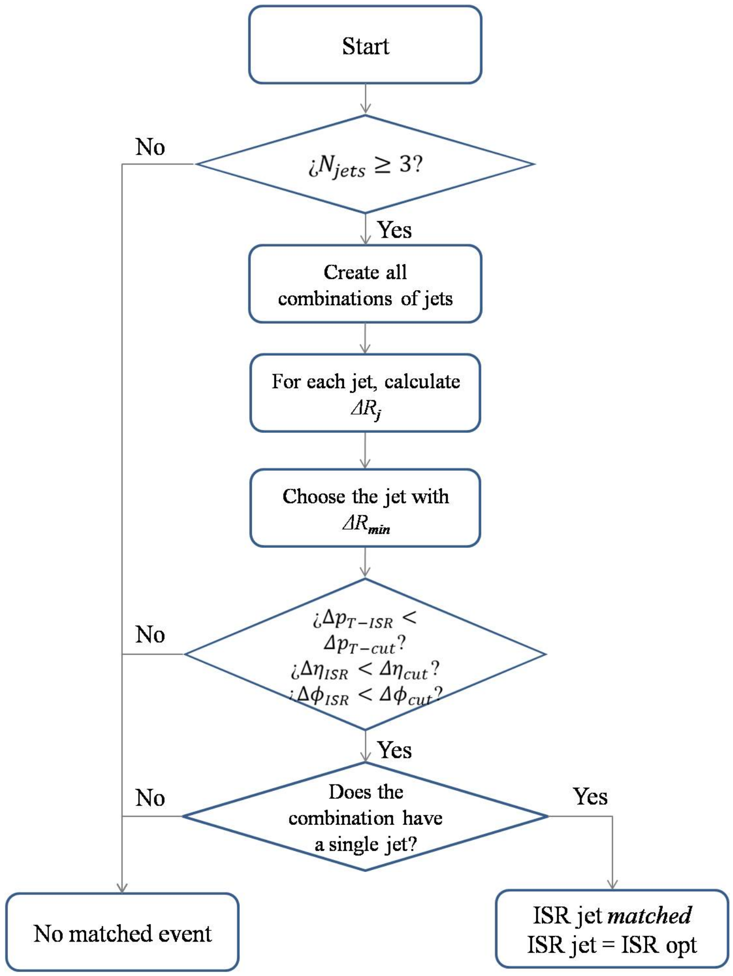

Figure 3.4: Matching algorithm between MadGraph and Pythia

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

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

```
./ISR_matching [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 the 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 !!-_/-!! and in the repository of the project, in the folder `Codes/Codes_analysis/ISR_jet_analysis_FV`.

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

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

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

Appendices

Appendix A

Simulation codes and scripts

A.1 Pythia code: hadronization02.cc

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

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

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

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

```
hadronization01

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

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

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

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

// Write own derived UserHooks class.

class MyUserHooks : public UserHooks {

public:

   // Constructor.
   MyUserHooks() { }

   // Destructor.
   ~MyUserHooks() { }

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

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

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

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


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

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

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

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

```

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

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

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

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

  // Initialize simulation
  pythia.init();

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

    // Fill stdhep file
```

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

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

}

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

```
}
```

A.2 Integration scripts: MadGraph + Pythia + Delphes

A.2.1 Configuration script: config_Integration.ini

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

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

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

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

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

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

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

A.2.2 Execution script: `script_Integration.sh`

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

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

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

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

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

## second sequence for each run, where the madgraph is called for each of the madgraph files (mgParallelFile_i.mg5). Pythia8 and Delphes are also executed
sequ2 () {
    source config_Integration.ini
         ## run madgraph with the corresponding madgraph file .mg5. all the messages are thrown to /dev/null
    ## Madgraph execution
    $1/bin/mg5_aMC -f $2/RunCards/mgFile_$4.mg5 # &> /dev/null
         ## sleep for 1s. Important, for the wait order to work
         sleep 1s
         ## wait for previous subprocesses to finish
         wait
    # Uncompress .lhe.gz file
    gzip -d $2/$3_$4/Events/run_01/unweighted_events.lhe.gz

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

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

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

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

}

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

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

Appendix B

Analysis codes

B.1 Tagging algorithm

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

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

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

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

In order to choose them, the code should be run as
./ISR_tagging N1 N2 N3, where N1 N2 and N3 are
the index of the 3 variables. If no parameter is
passed as parameter, N1 N2 and N3 will be 0,1 and 2
by default.

Additionally, the user can define a pt_cut and
probability cut k_cut to study the behavior of the
algorithm in a certain pt selection and to check
the MET boosting. In such case, the code should be
run as ./ISR_tagging N1 N2 N3 pt_cut k_cut
*/


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

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

// Other simulations parameters
const Char_t channel = '_'; // 's' for sTops and '_' for Tops
const Char_t ISR_or_NOT[] = "WI"; // "WI" with ISR, "SI" without (Here it does not make any sense), "bb" bjets production
const Bool_t Matching = true; // True if a matching has been done between MG and Pythia, false otherwise

const Char_t channel_histo = '_'; // 's' for sTops and '_' for Tops (Which channel fills the histogram)
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)
const Bool_t Matching_histo = true; // True if a matching has been done between MG and Pythia, false otherwise
const Bool_t atServer = true; // True if it is run at the server, false at the university's pc

int main(int argc, char **argv){
   std::cout.precision(4);
   // Counting time
   Double_t initialTime = clock();
   Double_t pt_cut = 0.0;
   Double_t Jet_cut = 2;

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

   // Variables for initializing histograms
   Int_t dims = 3;

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


   if (argc == 1) {
      cout << "Running the algorithm with the default variables:" << endl;
   }

```

```
   if (argc >= 4){
      cout << "Running the algorithm with the variables: " << endl;
      for (Int_t ind = 0; ind < 3; ind ++){
         var_index[ind] = atoi(argv[ind+1]);
      }
   }
   if (argc >= 5) {
      pt_cut = atof(argv[4]);
   }
   if (argc >= 6) {
      Jet_cut = atof(argv[5]);
   }

   if ((argc >= 7) || (argc < 4 && argc > 1)) {
      cout << "Error at calling this algorithm. Use as:" << endl;
      cout << "\t ./ISR_tagging N1 N2 N3 [Pt_cut] [K_cut] or just ./ISR_tagging" << endl;
      cout << "Read the documentation at the beginning of the code for further information\n" << endl;
      return 1;
   }

   cout << "Transverse momentum of the ISR: " << pt_cut << endl;

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

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

   /*
    * Initializing the 3-dimensional histogram
    */
   // Defining the names of the files
   Char_t combination[] = "______"; // Combination of
```

```
      variables
   for (Int_t ind = 0; ind < dims; ind ++){
      *(combination+(ind*2)+1) = (Char_t) (0x30 +
         var_index[ind]); // Int to char
   }

   Char_t *local_path_histos;
   local_path_histos = (Char_t*) malloc(512*sizeof(
      Char_t));
   if (atServer)
      strcpy(local_path_histos,"/home/af.garcia1214/
         PhenoMCsamples/Results/histo_folder/"); // At
         the server
   else
      strcpy(local_path_histos,"/home/afgarcia1214/
         Documentos/Results_and_data/histo_folder/"); //
          At the University's pc

   Char_t *head_folder_histos;
   head_folder_histos = (Char_t*) malloc(512*sizeof(
      Char_t));
   if (Matching_histo)
      strcpy(head_folder_histos,"
         _Tops_histos_WI_Matching/");
   else
      strcpy(head_folder_histos,"_Tops_histos_WI/");
   head_folder_histos[0] = channel_histo;
   head_folder_histos[13] = ISR_or_NOT_histo[0];
   head_folder_histos[14] = ISR_or_NOT_histo[1];

   Char_t *info_ISR_name;
   info_ISR_name = (Char_t*) malloc(sizeof(char)*512);
   strcpy(info_ISR_name,local_path_histos);
   strcat(info_ISR_name,head_folder_histos);
   strcat(info_ISR_name,"info_histo_ISR");
   strcat(info_ISR_name,combination);
   strcat(info_ISR_name,".txt");

   Char_t *array_ISR_name;
   array_ISR_name = (Char_t*) malloc(sizeof(char)*512);
```

```
    strcpy(array_ISR_name,local_path_histos);
    strcat(array_ISR_name,head_folder_histos);
    strcat(array_ISR_name,"array_histo_ISR");
    strcat(array_ISR_name,combination);
    strcat(array_ISR_name,".bn");

    Char_t *info_Non_ISR_name;
    info_Non_ISR_name = (Char_t*) malloc(sizeof(char)
       *512);
    strcpy(info_Non_ISR_name,local_path_histos);
    strcat(info_Non_ISR_name,head_folder_histos);
    strcat(info_Non_ISR_name,"info_histo_Non_ISR");
    strcat(info_Non_ISR_name,combination);
    strcat(info_Non_ISR_name,".txt");

    Char_t *array_Non_ISR_name;
    array_Non_ISR_name = (Char_t*) malloc(sizeof(char)
       *512);
    strcpy(array_Non_ISR_name,local_path_histos);
    strcat(array_Non_ISR_name,head_folder_histos);
    strcat(array_Non_ISR_name,"array_histo_Non_ISR");
    strcat(array_Non_ISR_name,combination);
    strcat(array_Non_ISR_name,".bn");

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

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

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

    /*
    * MET histograms
    */
```

```
    TH1 *h_MET = new TH1F("Missing ET","All events",300,0,2000);
    Char_t *name_histo_MET;
    name_histo_MET = (Char_t*) malloc(sizeof(char)*512);
    strcpy(name_histo_MET,"ISR jet PT > ");
    Char_t pt_str[] = "   ";
    pt_str[0] = 0x30 + int(pt_cut/100)%10;
    pt_str[1] = 0x30 + int(pt_cut/10)%10;
    pt_str[2] = 0x30 + int(pt_cut)%10;
    strcat(name_histo_MET,pt_str);
    strcat(name_histo_MET,"-k = ");
    Char_t k_str[] = "   ";
    k_str[0] = 0x30 + int(Jet_cut)%10;
    k_str[1] = '.';
    k_str[2] = 0x30 + int(Jet_cut*10)%10;
    strcat(name_histo_MET,k_str);
    TH1 *h_MET_hpt1 = new TH1F(name_histo_MET,"Missing ET high_ISR_pt-1",300,0.0,2000);

    if (argc == 6)
       cout << "The algorithm will evaluate the MET for a sample with PT > " << pt_str << " at k = " << k_str << endl;
    /*
     * Tagging variables
     */

    cout << "Jet cut, k = " << Jet_cut << endl;

    // Arrays with the number of tags, Misstags and events rejected
    // Probability cut
    Double_t Prob_cut = 0;
    Double_t k_min = 1.2; // Minimum probability cut = k_min/num_jets
    Double_t k_max = 3.0; // Maximum probability cut = k_max/num_jets
    Int_t k_bins = 100; // Number of values of k between k_min and k_max
    Double_t k_step = (Double_t) (k_max-k_min)/k_bins;
```

```
    Double_t k_values[k_bins];
    for(Int_t ind = 0; ind < k_bins; ind ++){
       k_values[ind] = k_min + k_step*ind;
    }

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

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

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


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

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

```

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

     // Loading simulations from Delphes
     Char_t *local_path;
     local_path = (Char_t*) malloc(512*sizeof(Char_t));
     if (atServer)
        strcpy(local_path,"/home/af.garcia1214/
           PhenoMCsamples/Simulations/
           MG_pythia8_delphes_parallel/"); // At the
           server
     else
        strcpy(local_path,"/home/afgarcia1214/
           Documentos/Simulations/"); // At the
           University's pc

     Char_t *head_folder;
     head_folder = (Char_t*) malloc(512*sizeof(Char_t))
        ;
     if (Matching)
        strcpy(head_folder,"_Tops_Events_WI_Matching/")
           ;
     else
        strcpy(head_folder,"_Tops_Events_WI/");
     head_folder[0] = channel;
     head_folder[13] = ISR_or_NOT[0];
     head_folder[14] = ISR_or_NOT[1];

     Char_t current_folder[] = "_Tops_MG_1K_AG_WI_003/"
        ;
     current_folder[0] = channel;
     current_folder[15] = ISR_or_NOT[0];
     current_folder[16] = ISR_or_NOT[1];

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

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

      current_folder[18] = centena;
      current_folder[19] = decena;
      current_folder[20] = unidad;

      Char_t *file_delphes;
      file_delphes = (Char_t*) malloc(512*sizeof(Char_t));
      strcpy(file_delphes,local_path);
      strcat(file_delphes,head_folder);
      strcat(file_delphes,current_folder);
      strcat(file_delphes,"Events/run_01/output_delphes.root");

          cout << "Studying run: "<<centena<<decena<<unidad<<endl;
      cout << "\nReading the file: \nDelphes: " << file_delphes << endl;

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

      Long64_t numberOfEntries = treeReader_Delphes->GetEntries();

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

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

```

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

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

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

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

```

```
    Char_t *local_path_binary;
    local_path_binary = (Char_t*) malloc(512*sizeof(Char_t));
    if (atServer)
        strcpy(local_path_binary,"/home/af.garcia1214/PhenoMCsamples/Results/matching_Results/"); // At the server
    else
        strcpy(local_path_binary,"/home/afgarcia1214/Documentos/Results_and_data/matching_Results/"); // At the University's pc

    Char_t *head_folder_binary;
    head_folder_binary = (Char_t*) malloc(512*sizeof(Char_t));
    if (Matching)
        strcpy(head_folder_binary,"_Tops_matchs_WI_Matching/");
    else
        strcpy(head_folder_binary,"_Tops_matchs_WI/");
    head_folder_binary[0] = channel;
    head_folder_binary[13] = ISR_or_NOT[0];
    head_folder_binary[14] = ISR_or_NOT[1];

    Char_t matching_name[] = "ISR_jets_Tops_WI_003.bn";
    matching_name[8] = channel;
    matching_name[14] = ISR_or_NOT[0];
    matching_name[15] = ISR_or_NOT[1];

    matching_name[17] = centena;
    matching_name[18] = decena;
    matching_name[19] = unidad;

    Char_t * fileName;
    fileName = (Char_t*) malloc(512*sizeof(Char_t));
    strcpy(fileName,local_path_binary);
    strcat(fileName,head_folder_binary);
    strcat(fileName,matching_name);

```

```
      if (ISR_or_NOT[0] != 'S'){ // != S means bb or WI
         ifstream ifs(fileName,ios::in | ios::binary);

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

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

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

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

```

```
        NumJets=branchJet->GetEntries();

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

        h_MET->Fill(MET);

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

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

        // 4 Delta Eta aver
        Delta_eta_aver = 0.0;

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

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

        // 7 Delta Eta leading
        Delta_Eta_leading = 0.0;

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

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

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

        //PT_aver
        PT_aver = PT_sum/NumJets;

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

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

        // Reset variables
        probISR = 0.0;
        k_ISR = 0.0;
        prob_max = 0;
        ISR_tag_index = -1;

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

```
                   ,currentJet->Eta,currentJet->Phi,
                   currentJet->Mass);

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

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

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

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

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

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

                // Other variables
```

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

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

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

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

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

               if(probISR > (1.0 + 1.0e-10)){
                  cout << setprecision(20) << "\n\t *** ERROR: La probabilidad no puede ser mayor a 1 ***" << endl;
                  return 1;
               }

               if (probISR >= prob_max){
                  prob_max = probISR;
```

```
                    vect_optimum->SetPtEtaPhiM(
                       vect_currentJet->Pt(),
                       vect_currentJet->Eta(),
                       vect_currentJet->Phi(),
                       vect_currentJet->M());
                    ISR_tag_index = iJet;
                 }
              }
           }

           k_ISR = prob_max*NumJets;

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

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

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

```
            ind++){
            Num_Rejected_array[ind]++;
         }
      }

      if (ISR_tag_index != -1 && vect_optimum->Pt()>pt_cut && ISR_or_NOT[0] != 'S'){  // != S means bb or WI
         for (Int_t ind = 0; ind < k_ISR_pos + 1; ind++){
            if (ISR_tag_index == ISR_jets[entry])
               Num_Tags_array_hpt[ind]++;
            else
               Num_MissTags_array_hpt[ind]++;
         }
         for (Int_t ind = k_ISR_pos+1; ind < k_bins; ind++){
            Num_Rejected_array_hpt[ind]++;
         }
      }

      Prob_cut = Jet_cut/NumJets;
      if(prob_max >= Prob_cut){
         if (ISR_tag_index == ISR_jets[entry] && ISR_or_NOT[0] != 'S')  // != S means bb or WI
            Num_Tags++;
         else
            Num_MissTags++;

         // Cheching MET boosting
         if(vect_optimum->Pt()>pt_cut){
            h_MET_hpt1->Fill(MET);
         }
      }
      else
         Num_Rejected++;

   }
```

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

    } // End run's for cicle

    /*
     * Tagging results
     */

    Int_t Num_Studied = Num_Tags + Num_MissTags + Num_Rejected;

    cout << "Number of compared events (between the matching and tagging algorithms) : " << Num_Studied << endl;
    cout << "Per. Tags: \t" << ((Double_t)Num_Tags/Num_Studied)*100 << "%" << endl;
    cout << "Per. MissTags: \t" << ((Double_t)Num_MissTags/Num_Studied)*100 << "%" << endl;
    cout << "Per. Rejected: \t" << ((Double_t)Num_Rejected/Num_Studied)*100 << "%" << endl;

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

```
        Num_MissTags_array_hpt[ind]/Num_Total_Jets_hpt[
        ind];
    Num_Rejected_array_hpt[ind] =
        Num_Rejected_array_hpt[ind]/Num_Total_Jets_hpt[
        ind];
  }

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

  Char_t *local_path_results;
  local_path_results = (Char_t*) malloc(512*sizeof(
     Char_t));
  if (atServer)
    strcpy(local_path_results,"/home/af.garcia1214/
        PhenoMCsamples/Results/resultsTagging/"); // At
         the server
  else
    strcpy(local_path_results,"/home/afgarcia1214/
        Documentos/Results_and_data/resultsTagging/");
        // At the University's pc

  Char_t *head_folder_results;
  head_folder_results = (Char_t*) malloc(512*sizeof(
     Char_t));
  if (Matching)
    strcpy(head_folder_results,"
        _Tops_result_WI_Matching/");
  else
    strcpy(head_folder_results,"_Tops_result_WI/");
  head_folder_results[0] = channel;
  head_folder_results[13] = ISR_or_NOT[0];
  head_folder_results[14] = ISR_or_NOT[1];

  Char_t outName[] = "_Tops_WI_Overall";
  outName[0] = channel;
  outName[6] = ISR_or_NOT[0];
  outName[7] = ISR_or_NOT[1];
```

```

  Char_t outNamept[] = "_Tops_WI_hpt-100";
  outNamept[0] = channel;
  outNamept[6] = ISR_or_NOT[0];
  outNamept[7] = ISR_or_NOT[1];
  outNamept[13] = 0x30 + int(pt_cut/100)%10;
  outNamept[14] = 0x30 + int(pt_cut/10)%10;
  outNamept[15] = 0x30 + int(pt_cut)%10;

  Char_t *outFileTotal;
  outFileTotal = (Char_t*) malloc(sizeof(char)*512);
  strcpy(outFileTotal,local_path_results);
  strcat(outFileTotal,head_folder_results);
  strcat(outFileTotal,outName);
  strcat(outFileTotal,combination);
  strcat(outFileTotal,".txt");

  Char_t *outFileTotalpt;
  outFileTotalpt = (Char_t*) malloc(sizeof(char)*512);
  strcpy(outFileTotalpt,local_path_results);
  strcat(outFileTotalpt,head_folder_results);
  strcat(outFileTotalpt,outNamept);
  strcat(outFileTotalpt,combination);
  strcat(outFileTotalpt,".txt");

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

  ofstream ofs_over(outFileTotal,ios::out);
  if(!archivoExiste){
        // If file already exists
  }

  ofs_over << "# Number of Tags, Misstags and Rejected
     as a function of k" << endl;
  ofs_over << "# Number of Events " << Num_Total_Jets
     [0] << endl;
```

```
    ofs_over << "# k_cut \t Tags \t MissTags \t Rejected
       \t Total_Events " << endl;


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

    if (argc >= 5){
       ofstream ofs_pt(outFileTotalpt,ios::out);
       ofs_pt << "# Number of Tags, Misstags and Rejected
           as a function of k. The ISR has pt > " <<
          pt_cut << endl;
       ofs_pt << "# Number of Events " <<
          Num_Total_Jets_hpt[0] << endl;
       ofs_pt << "# k_cut \t Tags \t MissTags \t Rejected
           \t Total_Events " << endl;
       for (Int_t ind = 0; ind < k_bins; ind ++){
          ofs_pt << setiosflags(ios::fixed) <<
             setprecision(6) << setw(6) << k_values[ind]
                << "\t" << Num_Tags_array_hpt[ind] << "\t
                   " << Num_MissTags_array_hpt[ind] << "\
                   t" << Num_Rejected_array_hpt[ind]
                << "\t" << setprecision(0) <<
                   Num_Total_Jets_hpt[ind] << endl;
       }
       ofs_pt.close();
    }

    if (argc == 6){
       Char_t outNameMET[] = "_Tops_WI_MET_pt_000_k_2.0";
       outNameMET[0] = channel;
       outNameMET[6] = ISR_or_NOT[0];
       outNameMET[7] = ISR_or_NOT[1];
```

```
        outNameMET[16] = pt_str[0];
        outNameMET[17] = pt_str[1];
        outNameMET[18] = pt_str[2];
        outNameMET[22] = k_str[0];
        outNameMET[23] = k_str[1];
        outNameMET[24] = k_str[2];

        Char_t *outFileMET;
        outFileMET = (Char_t*) malloc(sizeof(char)*512);
        strcpy(outFileMET,local_path_results);
        strcat(outFileMET,head_folder_results);
        strcat(outFileMET,outNameMET);
        strcat(outFileMET,combination);

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

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

    }

    ofs_over.close();

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

    return 0;
}
```

B.2 Matching algorithm

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

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

It also stores in a binary file the matching
results
*/

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

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

// Other simulations parameters
const Char_t channel = 's'; // 's' for sTops and '_' for Tops
const Char_t ISR_or_NOT[] = "WI"; // "WI" with ISR, "SI" without (Here it does not make any sense), "bb" bjets production
const Bool_t atServer = true; // True if it is run at the server, false at the university's pc
const Bool_t Matching = true; // True if a matching has been done between MG and Pythia, false otherwise

```

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

   // Create chains of root trees
   TChain chain_Pythia("STDHEP");
   TChain chain_Delphes("Delphes");

   // Loading simulations from Delphes
   Char_t *local_path;
   local_path = (Char_t*) malloc(512*sizeof(Char_t));
   if (atServer)
      strcpy(local_path,"/home/af.garcia1214/PhenoMCsamples/Simulations/MG_pythia8_delphes_parallel/"); // At the server
   else
      strcpy(local_path,"/home/afgarcia1214/Documentos/Simulations/"); // At the University's pc

   Char_t *head_folder;
   head_folder = (Char_t*) malloc(512*sizeof(Char_t));
   if (Matching)
      strcpy(head_folder,"_Tops_Events_WI_Matching/");
   else
      strcpy(head_folder,"_Tops_Events_WI/");
   head_folder[0] = channel;
   head_folder[13] = ISR_or_NOT[0];
   head_folder[14] = ISR_or_NOT[1];

   Char_t current_folder[] = "_Tops_MG_1K_AG_WI_003/";
   current_folder[0] = channel;
   current_folder[15] = ISR_or_NOT[0];
   current_folder[16] = ISR_or_NOT[1];

   Char_t unidad = '3'; Char_t decena = '0'; Char_t centena = '0';

   if (argc > 1){
```

```
        cout << "The number of the simulation should consist of 3 digits" << endl;
        centena = argv[1][0];
        decena = argv[1][1];
        unidad = argv[1][2];
        current_folder[18] = centena;
        current_folder[19] = decena;
        current_folder[20] = unidad;
    }

    Char_t *file_pythia;
    file_pythia = (Char_t*) malloc(512*sizeof(Char_t));
    strcpy(file_pythia,local_path);
    strcat(file_pythia,head_folder);
    strcat(file_pythia,current_folder);
    strcat(file_pythia,"Events/run_01/output_pythia8.root");

    Char_t *file_delphes;
    file_delphes = (Char_t*) malloc(512*sizeof(Char_t));
    strcpy(file_delphes,local_path);
    strcat(file_delphes,head_folder);
    strcat(file_delphes,current_folder);
    strcat(file_delphes,"Events/run_01/output_delphes.root");

    if (argc > 1){
        cout << "\nReading the files: \nPythia8: " << file_pythia << "\nDelphes: " << file_delphes << endl;
    }
    else
        cout << "\nReading the default files: \nPythia8: " << file_pythia << "\nDelphes: " << file_delphes << endl;

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

    // Objects of class ExRootTreeReader for reading the
```

```
   information
ExRootTreeReader *treeReader_Pythia = new ExRootTreeReader(&chain_Pythia);
ExRootTreeReader *treeReader_Delphes = new ExRootTreeReader(&chain_Delphes);

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

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

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

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

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

```

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

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

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

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

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

```

      // Finding the ISR parton
      ISR_parton_found = false;
      pos_ISR = -1;
      for(Int_t iPart = 0; iPart < branchParticlePythia->GetEntries(); iPart++){
         particle_pythia = (TRootGenParticle*) branchParticlePythia->At(iPart);
         if( abs(particle_pythia->Status) == 43){
            pos_ISR = iPart;
            ISR_particle = (TRootGenParticle*) branchParticlePythia->At(pos_ISR);
            ISR_parton_found = true;
//          The following lines were used to check that everything was going well
//          cout << pos_ISR << "\t\t" << ISR_particle->Status << "\t\t" << ISR_particle->PID
//             << "\t\t" << ISR_particle->M1 << "\t\t" << ISR_particle->M2
//             << "\t\t" << ISR_particle->D1 << "\t\t" << ISR_particle->D2 << endl;
         }
      }

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

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

```
                  branchParticlePythia->At(pos_ISR);
            else
               ISR_parton_found = true; // To end up the while loop
         }
      }

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

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

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

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

      // Finding the jet with the minimum R to the ISR
```

```
         parton
    for ( Int_t j = 0; j < NumJets; j++ ) {     // Loop over jets finding the one with the minimum R
          jet = (Jet*) branchJet->At(j);
          vect_Jet1->SetPtEtaPhiM(jet->PT, jet->Eta, jet->Phi, jet->Mass);
          r = vect_ISR_particle->DeltaR(*vect_Jet1);
          if ( r < R_min ) {
                R_min = r;
                ISR_match_index = j;
                mixJets = 1;
                *vect_Jeto = *vect_Jet1;
          }
          // Checking if there are two jets mixed
          for ( Int_t k = j+1; k<NumJets; k++){
             jet2 = (Jet*) branchJet->At(k);
             vect_Jetc->SetPtEtaPhiM(jet2->PT, jet2->Eta, jet2->Phi, jet2->Mass);
             *vect_Jets = *vect_Jet1 + *vect_Jetc;
             r = vect_ISR_particle->DeltaR(*vect_Jets);
             if ( r < R_min ) {
                R_min = r;
                ISR_match_index = j;
                mixJets = 2;
                *vect_Jeto = *vect_Jets;
             }
          // Checking if there are three jets mixed
          for (Int_t m = k+1; m<NumJets; m++){
             jet2 = (Jet*) branchJet->At(m);
             vect_Jetc->SetPtEtaPhiM(jet2->PT, jet2->Eta, jet2->Phi, jet2->Mass);
             *vect_Jets = *vect_Jets + *vect_Jetc;
             r = vect_ISR_particle->DeltaR(*vect_Jets);
             if ( r < R_min ) {
                R_min = r;
                ISR_match_index = j;
                mixJets = 3;
```

```
                  *vect_Jeto = *vect_Jets;
                         }
            // Checking if there are four jets mixed
            for (Int_t n = m+1; n<NumJets; n++){
               jet2 = (Jet*) branchJet->At(n);
               vect_Jetc->SetPtEtaPhiM(jet2->PT, jet2->
                  Eta, jet2->Phi, jet2->Mass);
               *vect_Jets = *vect_Jets + *vect_Jetc;
               r = vect_ISR_particle->DeltaR(*vect_Jets)
                  ;
               if ( r < R_min ) {
                  R_min = r;
                  ISR_match_index = j;
                  mixJets = 4;
                  *vect_Jeto = *vect_Jets;
               }
            }
                           }
                     }
      }     // Loop over jets finding the one with the
         minimum R

      if( (mixJets == 1) && (ISR_match_index >= 0) && (
         ISR_match_index < NumJets) ) {
                NumEvents1ISRJet++;
                Double_t Delta_PT = TMath::Abs(
                   vect_Jeto->Pt() - vect_ISR_particle
                   ->Pt());
                Double_t Delta_Eta = TMath::Abs(
                   vect_Jeto->Eta() - vect_ISR_particle
                   ->Eta());
                Double_t Delta_Phi = vect_Jeto->
                   DeltaPhi(*vect_ISR_particle);
                Double_t Delta_y = TMath::Abs(vect_Jeto
                   ->Rapidity() - vect_ISR_particle->
                   Rapidity());

                if ( (Delta_PT > Cut_matching_DPT) || (
                   Delta_Eta > Cut_matching_DEta) || (
                   Delta_Phi > Cut_matching_DPhi ) || (
```

```
                    Delta_y > Cut_matching_Dy) ) {
                         ISR_jets[entry] = -1;
                 }
                 else {
                         NumMatches++;
                         ISR_jets[entry] =
                            ISR_match_index;
                 }
         }

         if (ISR_jets[entry] >= NumJets){
            cout << "Error en el matching" << endl;
            return 1;
         }
   } // End of the cycle over the events

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

   /*
    * Writing results
    */
   Char_t *local_path_results;
   local_path_results = (Char_t*) malloc(512*sizeof(
      Char_t));
   if (atServer)
      strcpy(local_path_results,"/home/af.garcia1214/
         PhenoMCsamples/Results/matching_Results/"); //
         At the server
   else
   strcpy(local_path_results,"/home/afgarcia1214/
      Documentos/Results_and_data/matching_Results/");
      // At the University's pc

   Char_t *head_folder_results;
   head_folder_results = (Char_t*) malloc(512*sizeof(
      Char_t));
   if (Matching)
      strcpy(head_folder_results,"
```

```
         _Tops_matchs_WI_Matching/");
   else
      strcpy(head_folder_results,"_Tops_matchs_WI/");
   head_folder_results[0] = channel;
   head_folder_results[13] = ISR_or_NOT[0];
   head_folder_results[14] = ISR_or_NOT[1];

   Char_t matching_name[] = "ISR_jets_Tops_WI_003.bn";
   matching_name[8] = channel;
   matching_name[14] = ISR_or_NOT[0];
   matching_name[15] = ISR_or_NOT[1];

   if (argc > 1){
      matching_name[17] = centena;
         matching_name[18] = decena;
         matching_name[19] = unidad;
        }

   Char_t * fileName;
   fileName = (Char_t*) malloc(512*sizeof(Char_t));
   strcpy(fileName,local_path_results);
   strcat(fileName,head_folder_results);
   strcat(fileName,matching_name);

   if (argc > 1)
      cout << "*** Writing the binary file...:" <<
         fileName << endl;
      else
         cout<<"*** Writing the default binary file...:"
             << fileName << endl;

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

```
        }
    }
    ofs.close();

    cout << endl;
    cout << "Number of events with a single ISR jet = "
        << NumEvents1ISRJet <<endl;
    cout << "Number of matches = " << NumMatches << endl;
    cout << endl;

    return 0;
}
```

Bibliography

[1] J. Alwall, R. Frederix, S. Frixione, V. Hirschi, F. Maltoni, et al. The automated computation of tree-level and next-to-leading order differential cross sections, and their matching to parton shower simulations. *JHEP*, 1407:079, 2014.

[2] Torbjörn Sjöstrand, Stefan Ask, Jesper R. Christiansen, Richard Corke, Nishita Desai, et al. An Introduction to PYTHIA 8.2. *Comput.Phys.Commun.*, 191:159–177, 2015.

[3] Torbjorn Sjostrand, Stephen Mrenna, and Peter Skands. Pythia 6.4 physics and manual. *JHEP*, 05:026, 2006.

[4] J. de Favereau et al. DELPHES 3, A modular framework for fast simulation of a generic collider experiment. *JHEP*, 1402:057, 2014.

[5] Lynn Garren. *StdHep 5.06.01 Monte Carlo Standardization at FNAL Fortran and C Implementation.* Fermilab, 11 2006. Available at http://cepa.fnal.gov/psm/stdhep/.

[6] O. Tange. Gnu parallel - the command-line power tool. *;login: The USENIX Magazine*, 36(1):42–47, Feb 2011.

[7] Simon De Visscher and Johan Alwall. Introduction to jet-parton matching in mg/me, 03 2011. Available at https://cp3.irmp.ucl.ac.be/projects/madgraph/wiki/IntroMatching.