Documentation of the project: ISR jet tagging

Author:
Andrés Felipe García Albarracín

Advisor:
Juan Carlos Sanabria, Ph.D.

June 30, 2015

Contents

1 Introduction **1**

2 Simulation chain **3**

2.1 Usage of MadGraph 5.2 4

2.2 Usage of Pythia 8.2 7

2.2.1 Code Usage 7

2.2.2 The code 10

2.2.3 Pythia ntuple generation 11

2.3 Usage of Delphes 3.2 12

2.4 Integration of MadGraph 5.2 + Pythia 8.2 + Delphes 3.2 . . . 13

2.5 Example of the integration scripts 15

3 Analysis codes **17**

3.1 The ISR jet tagging method 17

3.1.1 The method 17

3.1.2 From probability density functions to normalized histograms 19

3.1.3 The Algorithm 21

3.2 Matching algorithm 22

Appendices **24**

A Pythia code **25**

B Integration scripts **31**

B.1 Configuration script: `config_Integration.ini` 31

B.2 Execution script: `script_Integration.sh` 33

Bibliography **38**

Chapter 1

Introduction

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

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

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

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

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

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

Chapter 2

Simulation chain

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

Philip II of Macedon

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

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

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

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

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

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

2.1 Usage of MadGraph 5.2

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

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

been correctly installed [2].

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

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

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

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

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

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

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

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

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

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

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

2.2 Usage of Pythia 8.2

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

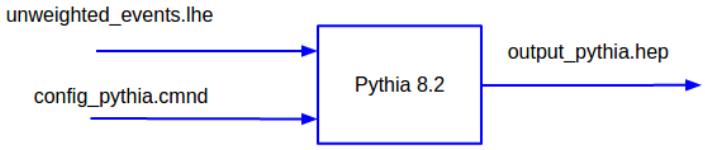

Figure 2.1

The functionality of the black box of 2.1 is done by a program written in C++, which is based on the examples provided by Pythia developers [3]. The code is called `hadronization02.cc`, was written in C++ and can be found at Appendix A. 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`:

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

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

```
make hadronization02
```

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

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

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

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

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

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

2.2.2 The code

Having explained the procedure to compile and use the hadronization program, this subsection presents the code and what it does. As stated before, the code can be found in the Appendix A 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, 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 B 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 B.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 $6500GeV$ 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 The ISR jet tagging method

3.1.1 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:

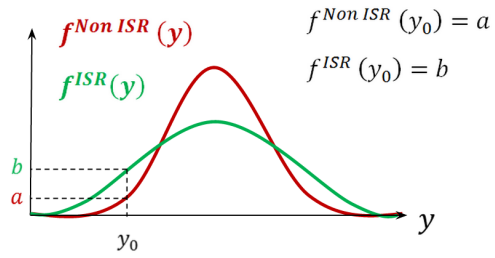

Figure 3.1: Probability distributions of a variable that distinguishes between ISR and Non ISR jets

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

3.1.2 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 presented in Fig. 3.2.

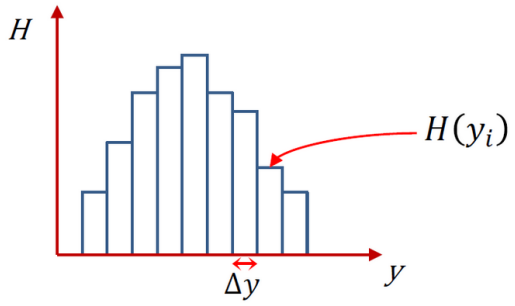

Figure 3.2: Shape of a histogram which does not exactly correspond with the probability density function

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).

3.1.3 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.

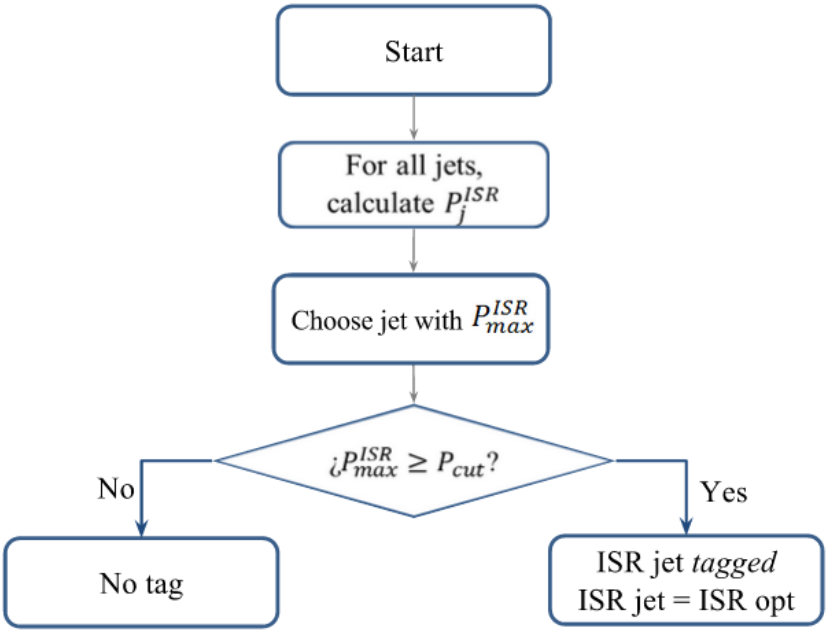

Figure 3.3: ISR jet tagging algorithm

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.

***

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.

3.2 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. TATATÁ. In practical matters, after knowing the ISR parton in Pythia, it looks for the closest

[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].

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. If the matched jet is too far from the ISR parton or if it is a combination of several jets, the method does not reports any match.

Appendices

Appendix A

Pythia code: hadronization02.cc

```
1  // Copyright (C) 2015 Torbjorn Sjostrand.
2  // PYTHIA is licenced under the GNU GPL version 2, see
      COPYING for details.
3  // Please respect the MCnet Guidelines, see GUIDELINES
      for details.
4
5  /*
6  -------     Universidad de los Andes      -------
7  -------      Departamento de Fisica       -------
8  -------   Proyecto Joven Investigador     -------
9  -------  Andres Felipe Garcia Albarracin  -------
10 -------    Juan Carlos Sanabria Arenas    -------
11
12 This code develops pythia hadronization. Takes as
13 parameter a .cmnd file, where a .lhe file from MadGraph
14 and other parameters are specified. Then the code
15 produces .hep files after making the hadronization
16
17 Obs: The class MyUserHooks is written in order to
18 veto all the ISR emissions produced after the
19 first ISR parton. It is an extension of the code
20 hadronization01
```

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

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

```
                    endl;
88                  return 1;
89          }
90
91          // Specify the name of the output file
92          if (argc > 2 ) strcpy(fileout,argv[2]);
93          else strcpy(fileout,"output_pythia8.hep\0");
94
95          // Especify the number of events
96          int nEvent = pythia.mode("Main:numberOfEvents");
                // For reading only
97          int nAbort = 10; // Maximum number of failures
               accepted
98          int iAbort = 0; // Abortions counter
99
100         // Necessary stdhep functions
101         int istr(0);
102         int ierr = StdHepXdrWriteOpen(fileout, title,
               nEvent, istr);
103
104         // Set up to do a user veto and send it in.
105         MyUserHooks* myUserHooks = new MyUserHooks();
106         pythia.setUserHooksPtr( myUserHooks);
107
108         // Initialize simulation
109         pythia.init();
110
111         // Begin event loop; generate until none left in
                input file.
112         for (int iEvent = 0; iEvent < nEvent ; ++iEvent)
                {
113                 // Generate events, and check whether
                       generation failed.
114                 if (!pythia.next()) {
115                         // If failure because reached
                               end of file then exit event
                               loop.
116                         if (pythia.info.atEndOfFile())
                               break;
117                         // First few failures write off
```

```
                                as "acceptable" errors, then
                                quit.
118                         if (++iAbort < nAbort) continue;
119                         break;
120                 }
121
122                 // Fill stdhep file
123                 fill_stdhep(iEvent+1,pythia.event);
124                 ierr = StdHepXdrWrite(1,istr);
125         }
126
127         StdHepXdrEnd(istr);
128         pythia.stat();
129         cout << ierr;
130         delete myUserHooks;
131         return 0;
132
133 }
134
135 // This functions writes in stdhep format. It was
      written by Steve Mrenna
136 void fill_stdhep(int i, Event &e)
137 {
138    int num = e.size();
139    hepevt_.nevhep = i;
140    hepevt_.nhep = num;
141    for (int j = 0; j < num; j++) {
142       hepevt_.idhep[j] = e[j].id();
143       hepevt_.isthep[j] = e[j].statusHepMC();
144       hepevt_.jmohep[j][0] = (e[j].mother1()>0) ? e[j].
             mother1()+1 : 0;
145       hepevt_.jmohep[j][1] = (e[j].mother2()>0) ? e[j].
             mother2()+1 : 0;
146       hepevt_.jdahep[j][0] = (e[j].daughter1()>0) ? e[j
             ].daughter1()+1 : 0;
147       hepevt_.jdahep[j][1] = (e[j].daughter2()>0) ? e[j
             ].daughter2()+1 : 0;
148       hepevt_.phep[j][0] = e[j].px();
149       hepevt_.phep[j][1] = e[j].py();
150       hepevt_.phep[j][2] = e[j].pz();
```

```
151         hepevt_.phep[j][3] = e[j].e();
152         hepevt_.phep[j][4] = e[j].m();
153         hepevt_.vhep[j][0] = e[j].xProd();
154         hepevt_.vhep[j][1] = e[j].yProd();
155         hepevt_.vhep[j][2] = e[j].zProd();
156         hepevt_.vhep[j][3] = e[j].tProd();
157     }
158 }
```

Appendix B

Integration scripts: MadGraph + Pythia + Delphes

B.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
```

```
45 PYTHIAPARAMFOLDER=
46 ## PYTHIAPARAM IS THE NAME OF THE .cmnd FILE THAT SERVES
       AS PARAMETER TO PYTHIA
47 PYTHIAPARAM="input_pythia.cmnd"
48
49 ## *** Delphes
50 ## DIRECTORY OF DELPHES EXECTUABLE
51 DELPHESFOLDER=
52 ## DELPHES .EXE
53 DELPHESEXE="DelphesSTDHEP"
54 ## DELPHESCARDFOLDER IS THE NAME OF THE FOLDER WHERE THE
       DELPHES CARD IS LOCATED (check at the Delphes folder
      : /cards/)
55 DELPHESCARDFOLDER=
56 ## DELPHESCARD IS THE NAME OF THE .lct FILE THAT SERVES
      AS PARAMETER TO DELPHES
57 DELPHESCARD="delphes_card_CMS.tcl"
58
59 ## EXROOTANALYSIS
60 ## DIRECTORY OF EXROOTANALYSIS
61 EXROOTFOLDER=
62 ## EXROOT .EXE (STDHEP ---> .ROOT)
63 EXROOTEXE="ExRootSTDHEPConverter"
```

B.2 Execution script: `script_Integration.sh`

```
1  #!/bin/bash
2  # ------------------------------------------------
3  # -------     Universidad de los Andes      -------
4  # -------      Departamento de Fisica       -------
5  # -------        Joven Investigador         -------
6  # -------  Andres Felipe Garcia Albarracin  -------
7  # -------    Diego Alejandro Sanz Becerra   -------
8  # -------    Juan Carlos Sanabria Arenas    -------
9  # ------------------------------------------------
10 # This file executes parallel simulations with the
      programs: MadGraph 5.2 + Pythia 8.2 + Delphes 3.2
11 # Based on Diego Sanz's execution file:
      scriptMGParallelV2.sh
```

```
12
13 # Load the parameter file
14 source config_Integration.ini
15 ## make the RunCards Folder in the EVENTSFOLDER
16 mkdir ${EVENTSFOLDER}/RunCards
17 ## make the ParamCard Folder in the EVENTSFOLDER
18 mkdir ${EVENTSFOLDER}/ParamCard
19 ## copy the param card supplied to the EVENTSFOLDER/
   ParamCard and name it param_card.dat
20 cp ${PARAMCARDFOLDER}/${PARAMCARDFILE} ${EVENTSFOLDER}/
   ParamCard/param_card.dat
21
22 ## first sequence for each run, where the madgraph files
    and the run cards are created
23 sequ () {
24         ## copy the run card frame to the RunCards
              directory and append the seed (counter $i)
25         cp ${RUNCARDFOLDER}/${RUNCARDFILE} ${
              EVENTSFOLDER}/RunCards/run_card_$i.dat
26         ## copy the MadGraph file to the RunCards
              directory as mgParallelFile_$i
27         cp ${MADGRAPHFILEFOLDER}/${MADGRAPHFILE} ${
              EVENTSFOLDER}/RunCards/mgFile_$i.mg5
28         ## copy the parameter pythia file to the
              RunCards directory
29         cp ${PYTHIAPARAMFOLDER}/${PYTHIAPARAM} ${
              EVENTSFOLDER}/RunCards/input_pythia_$i.cmnd
30         ## copy the delphes card to the RunCards
              directory *** Delphes card is the same for
              all runs
31         cp ${DELPHESCARDFOLDER}/${DELPHESCARD} ${
              EVENTSFOLDER}/RunCards/${DELPHESCARD}
32         ## change all the instances of SEED to the
              counter $i on the file run_card_$i.dat
33         sed -i "s/SEED/$i/g" ${EVENTSFOLDER}/RunCards/
              run_card_$i.dat
34         ## change all the instances of SEED to the
              counter $i on the file mgParallelFile_$i.mg5
35         sed -i "s/SEED/$i/g" ${EVENTSFOLDER}/RunCards/
              mgFile_$i.mg5
```

```
36          ## change all the instances of SEED to the
               counter $i on the file input_pythia_$i.cmnd
37          sed -i "s/SEED/$i/g" ${EVENTSFOLDER}/RunCards/
               input_pythia_$i.cmnd
38          ## change all the instances of RUNEVENTSNUM to
               $NUMEVENTSRUN on the file run_card_$i.dat
39          sed -i "s/RUNEVENTSNUM/$NUMEVENTSRUN/g" ${
               EVENTSFOLDER}/RunCards/run_card_$i.dat
40          ## change all the instances of FOLDEREVENTS to
               $EVENTSFOLDER on the file mgParallelFile.mg5
41          sed -i "s|FOLDEREVENTS|$EVENTSFOLDER|g" ${
               EVENTSFOLDER}/RunCards/mgFile_$i.mg5
42          ## change all the instances of NUMBERCORES to
               $CORESNUMBER on the file mgParallelFile.mg5
43          sed -i "s|NUMBERCORES|$CORESNUMBER|g" ${
               EVENTSFOLDER}/RunCards/mgFile_$i.mg5
44          ## change all the instances of SUBFOLDERNAME to
               $NAMESUBFOLDER on the file mgParallelFile_$i.
               mg5
45          sed -i "s|SUBFOLDERNAME|$NAMESUBFOLDER|g" ${
               EVENTSFOLDER}/RunCards/mgFile_$i.mg5
46          ## change all the instances of RESULTSFOLDER to
               the name of the folder where the results are
               located
47          sed -i "s|RESULTSFOLDER|${EVENTSFOLDER}/${
               NAMESUBFOLDER}_$i/Events/run_01|g" ${
               EVENTSFOLDER}/RunCards/input_pythia_$i.cmnd
48          ## change all the instances of RUNEVENTSNUM to
               $NUMEVENTSRUN on the file parameter pythia
               file
49          sed -i "s/RUNEVENTSNUM/$NUMEVENTSRUN/g" ${
               EVENTSFOLDER}/RunCards/input_pythia_$i.cmnd
50  }
51
52  ## second sequence for each run, where the madgraph is
       called for each of the madgraph files (
       mgParallelFile_i.mg5). Pythia8 and Delphes are also
       executed
53  sequ2 () {
54          source config_Integration.ini
```

```
55          ## run madgraph with the corresponding madgraph
               file .mg5. all the messages are thrown to /
               dev/null
56          ## Madgraph execution
57          $1/bin/mg5_aMC -f $2/RunCards/mgFile_$4.mg5 # &>
                /dev/null
58          ## sleep for 1s. Important, for the wait order
               to work
59          sleep 1s
60          ## wait for previous subprocesses to finish
61          wait
62          # Uncompress .lhe.gz file
63          gzip -d $2/$3_$4/Events/run_01/unweighted_events
               .lhe.gz
64
65          ## Pythia 8 execution
66          ${PYTHIA8FOLDER}/${PYTHIA8EXE} $2/RunCards/
               input_pythia_$4.cmnd $2/$3_$4/Events/run_01/
               output_pythia8.hep # &> /dev/null
67
68          ## Delphes execution
69          ${DELPHESFOLDER}/${DELPHESEXE} $2/RunCards/${
               DELPHESCARD} $2/$3_$4/Events/run_01/
               output_delphes.root $2/$3_$4/Events/run_01/
               output_pythia8.hep
70
71          ## ExRootAnalysis execution
72          ${EXROOTFOLDER}/${EXROOTEXE} $2/$3_$4/Events/
               run_01/output_pythia8.hep $2/$3_$4/Events/
               run_01/output_pythia8.root
73
74          ## Remove unnecessary files
75          rm $2/$3_$4/Events/run_01/output_pythia8.hep
76
77  }
78
79  export -f sequ
80  export -f sequ2
81  ## start PARAMETERS variable
82  PARAMETERS=""
```

```
83 ## loop to execute sequence "sequ" for all the values
      from $INIRUN to $ENDRUN
84 for i in `seq ${INIRUN} ${ENDRUN}`; do  # {21,28}; do ##
       `seq ${INIRUN} ${ENDRUN}`; do
85         ## execute sequ
86         sequ
87         ## concatenate the variable PARAMETERS with the
              current value of $i
88         PARAMETERS="$PARAMETERS ${i}"
89 done
90
91 ## execute gnuparallel. Use %% as the replacement string
       instead of {}.
92 parallel -0 -I %% --gnu "sequ2 ${MADGRAPHFOLDER} ${
      EVENTSFOLDER} ${NAMESUBFOLDER} %%" ::: $PARAMETERS
```

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.