

Documentation of the project: ISR jet tagging

Author:

Andrés Felipe García Albarracín

Advisor:

Juan Carlos Sanabria, Ph.D.

June 16, 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	8
	Appendices	10
A	Pythia code	11

Chapter 1

Introduction

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

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

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

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

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

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

Chapter 2

Simulation chain

“Divide et impera”,
“Divide and conquer”

Philip II of Macedon

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

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

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

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

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

Throughout this chapter, the codes and scripts written to achieve the simulation will be explained. One section is devoted to each program and another one presents the script that connects the three programs. Finally, the last section of this chapter presents the procedure known as Matching between MadGraph and Pythia, which ensures that the physics calculations made by each program correspond to the Energy scale that each one of them should handle.

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

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

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

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

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

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

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

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

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

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

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

2.2 Usage of Pythia 8.2

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

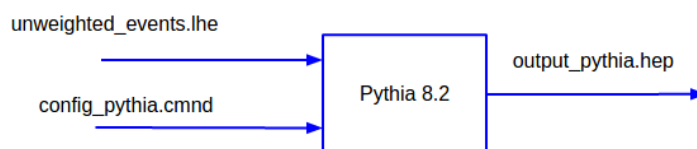


Figure 2.1

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

2.2.1 Code Usage

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

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

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

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

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

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

and these others at the end:

```
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 $@ $(CXXCOMMON) $(INCS) $(LOCAL) -L$(
   PREFIX_LIB) -Wl,-rpath $(PREFIX_LIB) -lpythia8
```

After doing so, the code is compiled by typing:

```
make hadronization02
```

2.2.2 The code

. Overall, the code can be described in three steps:

1. Input file:

Pythia is configured by means of an input file that the user needs to specify when executing the code. It is a plain text file with extension `.cmd` and contains parameters for the Pythia execution, such as the `.lhe` file produced by MadGraph and the hadronization conditions. An example of this file is:

```

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      = 100000  ! number of events
   generated (It needs to be <= Number of events
   generated in MG)
7  Init:showChangedParticleData = off  ! not useful info
   here
8  Next:numberShowInfo      = 1      ! 1 to show info , 0
   to not
9  Next:numberShowEvent     = 0      ! Especificy the number
   of events that will be listed as output
10
11 // Read .lhe file
12 Beams:frameType = 4 ! To take a MG file as input
13 Beams:LHEF = unweighted_events.lhe ! MG .lhe file
14
15 ! Hadronization:
16 PartonLevel:FSR = off ! switch final state radiation
17 PartonLevel:ISR = on ! switch initial state radiation
18 PartonLevel:MPI = off ! switch off multiparton
   interactions
19 !Random:setSeed = on ! For random seed
20 Random:seed = 1 ! any number between 1 and 900,000,000

```

2. Hadronization and Veto ISR

3. Output

The code can be compiled using the **Makefile** available at the folder examples. It is necessary to have installed Pythia including a HepMC package⁸. Additionally, the **Makefile**

⁸Again, information to install Pythia 8.2 and HepMC can be found at <http://goo.gl/vigBdj>

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 .cmdnd 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
52                         // shower emission after each emission
53                         // and allow the emission
54                         // to be vetoed by the next
55                         // method.
56     }
57
58     // Analyze each emission and asks for the number of
59     // the ISR emissions so far, in order
60     // 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: \nNo parameters file has
            passed as parameter. \nAbort" << endl;
88         return 1;

```

```

89         }
90
91         // Specify the name of the output file
92         if (argc > 2 ) strcpy(fileout ,argv[2]);
93         else strcpy(fileout ,"output_pythia8.hep\0");
94
95         // Especificy the number of events
96         int nEvent = pythia.mode("Main:numberOfEvents"); //
           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())
117                    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
136     // by Steve Mrenna
137     void fill_stdhep(int i, Event &e)
138     {
139         int num = e.size();
140         hepevt_.nevhep = i;
141         hepevt_.nhep = num;
142         for (int j = 0; j < num; j++) {
143             hepevt_.idhep[j] = e[j].id();
144             hepevt_.isthep[j] = e[j].statusHepMC();
145             hepevt_.jmohep[j][0] = (e[j].mother1()>0) ? e[j].
146                 mother1()+1 : 0;
147             hepevt_.jmohep[j][1] = (e[j].mother2()>0) ? e[j].
148                 mother2()+1 : 0;
149             hepevt_.jdahep[j][0] = (e[j].daughter1()>0) ? e[j].
150                 daughter1()+1 : 0;
151             hepevt_.jdahep[j][1] = (e[j].daughter2()>0) ? e[j].
152                 daughter2()+1 : 0;
153             hepevt_.phep[j][0] = e[j].px();
154             hepevt_.phep[j][1] = e[j].py();
155             hepevt_.phep[j][2] = e[j].pz();
156             hepevt_.phep[j][3] = e[j].e();
157             hepevt_.phep[j][4] = e[j].m();
158             hepevt_.vhhep[j][0] = e[j].xProd();
159             hepevt_.vhhep[j][1] = e[j].yProd();
160             hepevt_.vhhep[j][2] = e[j].zProd();

```

```
156         hepevt_.vhpe[j][3] = e[j].tProd();  
157     }  
158 }
```

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.