Monte Carlo Tutorial, Cairns Tropical Workshop 2013

Martin White, Csaba Balazs

July 8, 2013

1 Introduction

The purpose of this tutorial is to teach you how to take a generic physics model (specified as a Lagrangian), and simulate the expected output at the Large Hadron Collider, culminating in some simple plots of LHC distributions. We will install and use the following programs:

- Madgraph 5 (matrix element generator)
- Pythia 8 (Monte Carlo generator)
- Delphes 3.0.9 (LHC detector simulation)
- ROOT (heinous High Energy Physics software framework)

In addition, we will use pre-prepared output from a Mathematica package called Feynrules, which takes Lagrangians and spits out the information required by Madgraph.

The tutorial is a cut down version of that given at the 6^{th} MC4BSM workshop at Cornell in March 2012 (`http://www.phys.ufl.edu/~matchev/mc4bsm6/`). The original link contains more detail and examples if you want to explore further what we cannot cover in 1.5 hours.

2 Before you begin: Instructions for code installation

All of the code for todays tutorial is in the following tarball:
`https://dl.dropboxusercontent.com/u/52286446/TutorialCode.tgz`
Download the file to your computer, and unpack it as follows:

```
cd $HOME
tar xvf TutorialCode.tgz
```

This will produce a directory called TutorialCode in your home directory. Successful compilation requires the following:

- The `automake` package
- Python 2.6 or higher (but not 3.X)

Please ensure that you have these installed and working before attempting the following installation steps.

We will now proceed to build each piece of code required to complete the tutorial. Feel free to skip packages that you already have (ROOT being the most obvious example).

2.1 Installing ROOT

ROOT is a general purpose statistics and plotting package developed for High Energy Physicists. In the following, we will need it in order to interpret the output of the DELPHES detector simulation. From the `TutorialCode` directory, unpack the ROOT source as follows:

```
tar xvf root_v5.34.09.source.tar.gz
```

Now type the following:

```
cd root
./configure --prefix=$HOME/TutorialCode --etcdir=$HOME/TutorialCode
make
make install
```

Note that this will install ROOT into the `TutorialCode` directory. You may speed it up if you have a multicore machine by using `make -j4` for, e.g., a build on 4 cores.

Having built ROOT, you need to set it up using the following command:

```
. $HOME/TutorialCode/bin/thisroot.sh
```

Note that this will need to be done each time you start a new shell session.

2.2 Installing HepMC

HepMC provides a standard library for storing event data from Monte Carlo simulations. Events are stored as large ASCII files with a specific formatting of the output text. We will use HepMC to interface the DELPHES detector simulation with the Pythia Monte Carlo event generator.

To install the HepMC library, type the following in the `TutorialCode` directory:

```
tar xvf HepMC-2.06.08.tar.gz
cd HepMC-2.06.08
./bootstrap
./configure --prefix=$HOME/TutorialCode -with-momentum=GEV -with-length=MM
make
make install
```

Note that the `bootstrap` step will only work if you have automake installed. You may get some latex errors (the configure process attempts to also build the documentation), in which case hit enter until they disappear!

2.3 Installing Pythia

Pythia is a Monte Carlo event generator. Starting from the `TutorialCode` directory, one may install Pythia as follows. Note that we have to tell Pythia where HepMC lives, and thus it is essential that you complete the instructions in Section 2.2 before attempting to install Pythia.

```
tar xvf pythia8176.tgz
cd pythia8176
./configure --with-hepmc=$HOME/TutorialCode --with-hepmcversion=2.06.08
make
```

2.4 Installing DELPHES

If you have not setup ROOT, do so now via:

```
. $HOME/TutorialCode/bin/thisroot.sh
```

Now, from the `TutorialCode` directory, type the following:

```
tar xvf Delphes-3.0.9.tar.gz
cd Delphes-3.0.9
./configure --prefix=‘pwd‘
make
```

3 A toy model of new physics

We are now ready to start our tutorial, and will start by adding some physics to the Standard Model. For no reason at all, we will add two real scalar fields ϕ^1 and ϕ^2 that are singlets under all SM gauge groups. Their mass terms in the Lagrangian are:

$$\mathcal{L}_{s.m.} = -\frac{m_1^2}{2}\phi_1^2 - \frac{m_2^2}{2}\phi_2^2 - m_{12}^2\phi_1\phi_2 \tag{1}$$

We will call the mass eigenstates Φ_1 and Φ_2, and their eigenmasses M_1 and M_2. We will further assume that $M_1 < M_2$.

In addition to the scalars, we will add two new Dirac fields, U and E. Their SM quantum numbers are those of the SM u_R and e_R respectively. These fields have mass terms:

$$\mathcal{L}_{f.m.} = M_U\bar{U}U + M_E\bar{E}E \tag{2}$$

They interact with the scalars via:

$$\mathcal{L}_{Yuk} = \lambda_1\phi_1\bar{U}P_Ru + \lambda_2\phi_2\bar{U}P_Ru + \lambda_1'\phi_1\bar{E}P_Re + \lambda_2'\bar{E}P_Re \tag{3}$$

where u and e are the SM up-quark and electron fields. Note that there is a $\mathcal{Z}_2$ symmetry under which all fields we added ($\phi_{1,2}$,U,E) flip sign, while all SM fields do not. Thus, all of the new particles must be pair-produced, and the lightest new particle is stable (what does this remind you of?). This same symmetry prevents $U-u$ and $E-e$ mixing via Yukawas with the SM Higgs.

We will assume the following ordering of masses:

$$M_u > M_2 > M_L > M_1 \tag{4}$$

so that ϕ_1 is the lightest new particle. This will appear as missing transverse energy in the detector. The goal of the tutorial is to simulate the process:

$$pp \to \bar{U}U \tag{5}$$

at the 8 TeV LHC, along with the subsequent U decays:

$$U \to u\Phi_1 \tag{6}$$

$$U \to u\Phi_2, \Phi_2 \to eE, E \to e\Phi_1 \tag{7}$$

4 Step 1: Feynrules

The first step on the journey to LHC events is to take our Lagrangian and generate the required *Feynman rules* for the new particle interactions predicted by our theory. Though this can of course be done analytically (and the results put into an event generator by hand), there now exist a variety of tools to automate the process.

Feynrules is a Mathematica package into which one may enter Lagrangians for new physics. The package includes an implementation of the Standard Model, such that one may easily describe physics that interacts with currently observed particles. We will not consider the use of Feynrules in detail in this tutorial due to a) time constraints and b) the fact that not everybody has Mathematica on their laptop. We suggest instead that, if you are interested, you refer to `http://www.phys.ufl.edu/~matchev/mc4bsm6/` for the full details of how to implement the Lagrangian specified above.

For our purposes, it is sufficient to know how to use the *output* of Feynrules. Feynrules is interfaced to a variety of Monte Carlo tools such as *CalcHEP*, *Sherpa* and *Madgraph 5*. In the last case, one must export the Feynman rules in "UFO" format.

5 Step 2: Madgraph 5

In this section we will:

- Use the model generated by Feynrules
- Learn how to generate processes in Madgraph
- Generate the process $pp \to \bar{U}U$, with the U decays set correctly
- Generate some events for later use in Pythia

Madgraph 5 (MG5) is a general purpose matrix element calculator and event generator, interfaced to Feynrules via the "UFO" interface.

5.1 Running Madgraph 5

Madgraph 5 is written in python, and does not need to be installed (though you will need Python 2.6 or higher, excluding Python 3.X). From the `TutorialCode` directory, extract the code via:

```
tar xvf MadGraph5_v1.5.10.tar.gz
```

Now you can run MG5 like this:

```
cd MadGraph5_v1_5_10
./bin/mg5
```

You can get a quick overview of the MG5 syntax by typing:

```
tutorial
```

at the interactive prompt.

5.2 Loading our toy model

We are now ready to load our toy model into Madgraph. First, you will need the Feynrules output which has been magically prepared for you. In the `TutorialCode directory` type:

```
tar xvf MC4BSM_2012_UFO.tgz
mv MC4BSM_2012_UFO MadGraph5_v1_5_10/models/
```

Now “cd” into the Madgraph directory and open Madgraph as above. You can load our toy model by typing:

```
import model MC4BSM_2012_UFO
```

To see the particles that exist in the model, type `display particles` . You will see some general information, followed by a list of all of the particles of the model. Some of these you will recognise as the particles of the standard model, and they are supplemented by the new particles we have added:

```
p1 p2 uv uv~ ev ev~
```

Make sure that you understand the correspondence with the new particles introduced above. When you first import a new model, it is a good idea to perform some basic tests. To do this, quit MG5 (type “exit”), and reopen it in debug mode:

```
./bin/mg5 --debug
```

You can check the Lorentz and gauge invariance of processes. For example, let’s check this for $pp \to U\bar{U}$:

```
import model MC4BSM_2012_UFO
check p p > uv uv~
```

You will see that the model passes the tests (which include calculations of matrix elements in the Feynman and unitary gauges). You will further see the use of the letter “p” which stands for the constituent partons in the proton. MG5 automatically knew that it should take all possible parton-level processes in the above calculation.

5.3 Calculating widths and branching ratios

Enough of the preliminaries- let’s get to work on LHC physics. An essential ingredient of our LHC physics calculations will be the width and branching ratios for our new particle decays. These are essential for telling Pythia (or the MG5 event generator) how often our new particles are made, and how often they decay to other particles in our model. The way this works in MG5 is to specify the decay channel, and then type a series of commands to perform the required calculations. Open MG5 (debug mode is no longer required) and type the following:

```
import model sm
import model MC4BSM_2012_UFO
generate uv > u p1
add process uv > u p2
add process p2 > ev e+
add process p2 > ev~ e-
add process ev > e- p1
output
launch
```

You will see that we have had to add explicitly each process we specified in Section 3. You will get prompted when you press enter on the last command about editing two files called "param_card.dat" and "run_card.dat". Press enter at the prompt to ignore this, and you will see that an output file `param_card.dat` will be written, in the following location:

```
$HOME/TutorialCode/MadGraph5_v1_5_10/PROC_MC4BSM_2012_UFO_0/Events/run_01/param_card.dat
```

It is worth looking at this file and convincing yourself that you understand the output (this is a good point to ask an instructor if you are stuck). For example, look at the block of the file under the heading "INFORMATION FOR MASS". This is a simple list of all of the particles in the model. The first column gives a unique identifying code for the particle (called a "PDG ID" code, which is standard across generators). The first line, for example, represents the b quark. The second column gives the mass, and the rest (following the "#" symbol) is reserved for comments. You should be able to identify our new particles (whose masses were specified at the Feynrules stage). Note that changing the masses in this file would allow you to rerun the model for a different set of parameters without having to reenter the model in Feynrules.

There is in fact one change we need to make in this file. When the model was put into Feynrules, a width of 1 was specified for the Φ_1 decay. The particle should be stable, however, and we can enforce this by hand by changing the width. Find the block headed "Decay widths" and change the width of the particle with PDG_ID 9000006 to 0. All other widths have been recalculated since we specified the decay process in MG5 (the Φ_1 had no such process to be specified because it has no allowed decay process).

5.4 Step 3: Generating events

5.4.1 General issues

We are now ready to generate events for our new processes. There are two ways to do this. The first is to pass our width and branching ratio information to an external Monte Carlo generator (in this case Pythia 8), and get the generator to generate events for us using code generated by MG5 to handle the matrix element calculation. The second method is to use MG5 itself to perform the decay, in which case we can be sure that spin correlations are handled correctly. In the latter case, we will still need to pass the events through Pythia, because the event generator in MG5 does not implement parton shower and hadronization effects that are implemented in Pythia. In this tutorial, we will only consider the second method for generating events.

5.4.2 Generating events in Madgraph with decays

We want to use MG5 to both produce the $U\bar{U}$ particles and decay them. We start off as before by importing the model:

```
import model sm
import model MC4BSM_2012_UFO
```

We then specify the processes that we want to generate, and we enter the decay pattern explicitly:

```
generate p p > uv uv~ , uv > u p1, uv~ > u~ p1
```

Note the syntax here- we specify the production process, then specify the decay process for the particles, separated by commas. Here we have $U\bar{U}$ production, where both U particles decay to a Φ_1 particle. The command "generate" wipes the slate clean. To add further processes, we use the "add process" command. For more help on the generate syntax, type "help generate".

We would now like to add U particles decaying to Φ_2 particles, with the subsequent decay of the Φ_2's. Before we do so, it is helpful to define a shorthand for the electron/positron and for the $E/\bar{E}$:

```
define l e+ e-
define lv ev ev~
```

Now we can add the missing processes:

```
add process p p > uv uv~, uv > u p1, (uv~ > u~ p2, (p2 > l lv, lv > l p1))
add process p p > uv uv~, uv~ > u~ p1, (uv > u p2, (p2 > l lv, lv > l p1))
add process p p > uv uv~, (uv > u p2, (p2 > l lv, lv > l p1)),
 (continue on previous line) (uv~ > u~ p2, (p2 > l lv, lv > l p1))
```

Finally, we run (after the first command below, take note of the directory that is printed to the screen- this is where the output will be stored):

```
output
launch
```

Running the last command will lead to a user prompt in which you can edit various settings. One allowed option is to enter a path to a parameter card file that contains settings for the model. Point it to the file that you produced in section 5.3 (have you remembered to hack the width for the Φ_1?).

Next, enter "2" so that we hack the run parameters. This opens a file which configures various settings for the generation. Those which interest us are the collider parameters. Find the block headed "Collider type and energy" and you will see that we are set to use a proton-proton collider with 7000 GeV per beam. This is the 14 TeV LHC! What we would like is an 8 TeV LHC, so change the energy of each beam to 4000 GeV. Note that the file has been opened in the vim editor, which is rather hardcore. Press "INSERT" to go into editing mode, make your changes and then press "ESC". Then type ":w" to save the changes, and ":q" to quit the editor. You will be returned to the Madgraph prompt, and if you hit enter the generation will start. A browser window will open, allowing you to monitor the process.

When the generation has finished, look in the following directory:

```
$HOME/TutorialCode/MadGraph5_v1_5_10/PROC_MC4BSM_2012_UFO_1/Events/run_01
```

The event files are tarballed ascii files that contain four vectors and other information (e.g. the mother and daughter) for each particle produced in our LHC collision events. Crucially, partons in the final state remain as partons, and have not yet formed jets. We need to add a simulation of the complicated physics that turns these proto-events into more realistic LHC events.

6 Step 4: Pythia 8

In this section we will:

- Run Pythia on the events generated by Madgraph
- Produce an output file upon which we can run a detector simulation

6.1 Writing a C++ program to run Pythia

To run Pythia, you need to write a simple C++ program to call various functions that initialize the generator and run any processes that you are interested in. Pythia comes with a large library of built in physics (e.g. the Standard Model, the Minimal Supersymmetric Standard Model), but we ignore most of this in order to run our toy model. Examples of test programs are contained in the `examples` directory within the main Pythia directory.

From the `TutorialCode` directory, `cd` into this directory, and move across the events generated by Madgraph (we will also unpack them):

```
cd pythia8176/examples/
mv $HOME/TutorialCode/MadGraph5_v1_5_10/PROC_MC4BSM_2012_UFO_1/Events/run_01/events.lhe.gz .
gunzip events.lhe.gz
```

We now need to write the program that will process these events. Open a new file `mymain.cc` in the `examples` subdirectory with a text editor, e.g. Emacs. Then type the following lines:

```
// Headers and Namespaces.
#include "Pythia.h" // Include Pythia headers.
#include "HepMCInterface.h"
#include "HepMC/GenEvent.h"
#include "HepMC/IO_GenEvent.h"

using namespace Pythia8; // Let Pythia8:: be implicit.
int main() { // Begin main program.

  // Interface for conversion from Pythia8::Event to HepMC event.
  HepMC::I_Pythia8 ToHepMC;

  // Specify file where HepMC events will be stored.
  HepMC::IO_GenEvent ascii_io("hepmc-cairns.dat", std::ios::out);
  // Set up generation.
  Pythia pythia; // Declare Pythia object
  pythia.readString("Beams:frameType = 4"); // Beam info in LHEF.
  pythia.readString("Beams:LHEF = events.lhe");
  pythia.init(); // Initialize; incoming pp beams is default.
  // Generate event(s).
  for (int iEvent = 0; iEvent < 10000; ++iEvent) {

    if (!pythia.next())continue;

      // Construct new empty HepMC event and fill it.
      // Units will be as chosen for HepMC build, but can be changed
      // by arguments, e.g. GenEvt( HepMC::Units::GEV, HepMC::Units::MM)
    HepMC::GenEvent* hepmcevt = new HepMC::GenEvent();
    ToHepMC.fill_next_event( pythia, hepmcevt );

    // Write the HepMC event to file. Done with it.

    ascii_io << hepmcevt;
    delete hepmcevt;
    if (pythia.info.atEndOfFile()) break; // Exit at enf of LHEF.

  }

  pythia.stat(); // Print run statistics.
  return 0;
}
```

This is another good point to ask your instructor if something isn't clear. Next you need to edit the `Makefile` (the one in the examples subdirectory) so that it knows what to do with `mymain.cc` . The lines:

```
# Create an executable for one of the normal test programs
main00 main01 main02 main03 ... main09 main10 main10 \
```

and the three next lines enumerate the main programs. Edit the last of these lines to include also `mymain` :

```
main31 ... main40 mymain: \
```

Since mymain will require HepMC, we also need to edit these lines of the Makefile:

```
ifneq (x$(HEPMCLOCATION),x)
  main41 main42: \
```

to say:

```
ifneq (x$(HEPMCLOCATION),x)
  main41 main42 mymain: \
```

Now you can compile the program above by typing:

```
make mymain
```

We are now ready to run the program. First, type this (if you are using bash shell, otherwise use the equivalent for the shell that you are using):

```
export LD_LIBRARY_PATH=$HOME/TutorialCode/HepMC/lib:${LD_LIBRARY_PATH}
```

This tells Pythia where the HepMC is located. Now we run the program with:

```
./mymain.exe > mymain.out
```

This has run over all 10,000 events and produced a HepMC output file that we can process using a detector simulation.

7 Step 5: Delphes

In this section, we will:

- Apply an LHC detector simulation to the $pp \to \bar{U}U$ events that we have produced.
- Make some simple plots of kinematic distributions of the particles produced in the decay of the U particles.

7.1 Using DELPHES

DELPHES reads in events that have been produced by a Monte Carlo simulation, and simulates the passage of particles through an LHC-style detector. The parameters of the detector are highly configurable, and do not automatically correspond t to those used in a given LHC analysis (hence care must be taken in practise).

DELPHES is capable of reading several different event formats, and we will use the HepMC format dumped by Pythia in the previous section. DELPHES is configured by using a parameter card file, such as those found in the `examples` directory. We will use the file `delphes_card_ATLAS.tcl` for the following.

First, let's take a look at the most interesting parts of the parameter card file. The first section tells DELPHES which of its modules need to be run. There are sections called things like "Charged hadron tracking efficiency" and "Electron tracking efficiency" that contain parameterisations of the detector efficiency as functions of p_T and η. In principle, these should match ATLAS, though it would be wise to check the details for any serious work. You will also find paramaterisations of the momentum resolution. A particularly relevant part for this tutorial is the block headed "Calorimeter" which contains the calorimeter parameters. This includes a list of particle PDG_ID codes (look for "default energy fractions"), along with the fraction of energy deposited by each particle in the electromagnetic and hadronic calorimeters. Crucially, this must be set to zero for our new particles, such that they are treated as missing energy rather than observable particles (technically this is only relevant for the Φ_1, but it does no harm to add the whole list). Therefore, append the following to the bottom of the "add EnergyFraction" section:

```
add EnergyFraction {9000006} {0.0 0.0}
add EnergyFraction {9000007} {0.0 0.0}
add EnergyFraction {9000008} {0.0 0.0}
add EnergyFraction {9000009} {0.0 0.0}
```

Other notable parts of the parameter card file including the isolation settings for electrons and muons, and the parameters that configure the jet algorithms used by DELPHES. These vary wildly from analysis to analysis in ATLAS, and care must be taken if you are trying to reproduce the results of any particular paper.

We are now ready to run DELPHES with a "ATLAS-like" configuration. `cd` into the `Delphes-3.0.9` directory from the `TutorialCode` directory and type the following:

```
./DelphesHepMC examples/delphes_card_ATLAS.tcl cairns.root $HOME/TutorialCode/pythia8176/examples/hepmc-cairns.dat
```

where the three arguments are explained as follows:

- **examples/delphes_card_ATLAS.tcl:** The DELPHES configuration file that we have just edited to ensure that our new particles are invisible (did you definitely do it?).
- **cairns.root:** The name of the output ROOT file that you wish to produce. This will contain a ROOT ntuple with the four vectors of the reconstructed objects (electrons, muons, jets, etc). This file must not already exist.
- **hepmcout41.dat:** The name of the input file, which should contain Monte Carlo events formatted in the HepMC format.

Having run DELPHES, we have technically completed our analysis chain. An experimentalist looking for a new ATLAS analysis might now proceed by generating Standard Model background events, and trying to find functions of the four vectors of the final state particles that discriminate between the signal and background. A theorist interested in seeing whether current LHC analyses constrain our toy model would code up the most relevant LHC analyses, and compare the number of signal events that pass the analysis cuts to the limits set by ATLAS and CMS.

We do not have time to pursue either of these options in detail (indeed, both would lead to research papers). But we can make some simple distributions from the DELPHES output to convince ourselves that things look sensible.

We now have the reconstructed particle information in ROOT ntuple format. The simplest way to look at the information is to open the ntuple interactively in ROOT using:

```
root cairns.root
```

This will lead you to an interactive ROOT prompt from which we can plot basic histograms of the particle data. For example, to plot the distribution of the number of jet (for all events), type:

```
Delphes->Draw("Jet_size");
```

You should see a histogram that peaks at 3 jets, with a reasonable tail to high values. Next, let's look at the number of electrons:

```
Delphes->Draw("Electron_size");
```

Finally, plot the distribution of the number of muons via:

```
Delphes->Draw("Muon_size");
```

There are no muons in the events! Why does this make sense?

Although interactive mode is suitable for plotting simple distributions, the best way of performing analysis on ROOT data is to use a ROOT macro. DELPHES comes with a couple of ROOT macros in the `examples` directory. For example, `Example1.C` plots a histogram of the jet pt and the invariant mass of electron pairs. Assuming you are in the `Delphes-3.0.9` directory, run the macro using:

```
root -l examples/Example1.C\(\"cairns.root\"\)
```

This will produce two histograms:

- The jet p_T distribution
- The invariant mass of the pair of electrons/positrons with the highest transverse momentum, for events with at least 2 electrons or positrons

One will appear in a window, and you can see the list of histograms by typing `.ls` (note the full stop) at the ROOT prompt. You can then draw histograms using:

```
histogramname->Draw();
```

If you are new to ROOT, you may wish to spend some time looking at this macro and understanding how it works. Where are the histograms booked? Can you work out which number is the number of bins, and how to set the axis range? How would you plot the leading jet p_T distribution? Most experimentalists in the room will know the answer, so now is a great time to strike up a conversation if you are stuck.

The dilepton invariant mass plot has an interesting feature, with evidence of a kinematic limit near 100 GeV. For the cascade decay process $\Phi_2 \to eE \to ee\Phi_1$, the dilepton invariant mass is bounded by above by the expression:

$$m_{ll}^{max} = \sqrt{\frac{(m_{\phi_2}^2 - m_E^2)(m_E^2 - m_{\phi_1}^2)}{m_E^2}} \tag{8}$$

The masses for the new particles are in the Madgraph parameter card produced earlier. As an exercise, calculate the value of the upper bound on the dilepton invariant mass, and see if it agrees with your histogram.

Use any remaining time that you have to either ask about any parts of the tutorial that you did not understand, or try plotting other distributions from the DELPHES output.