Table of Contents

Madgraph and Delphes Tutorial

This tutorial shows how to generate collision events with MadGraph5_aMC@NLO, as well as how to shower, hadronize, and simulate the response of a detector like CMS with Pythia and Delphes. These tools are invaluable for phenomenology studies of physics beyond the SM, calculating cross sections, and generating parton level events to pass to other software for full MC generation. I will cover the basic operations as well as many useful tricks I've learned to save time and customize the event generation, as well as introduce some resources that I've found helpful.

To see the physics theory and statistics underlying these packages, check out the lectures at https://cp3.irmp.ucl.ac.be/projects/madgraph/wiki/MGTutorial⬚

Here is the color key for line highlighting:

```
GREY: For commands.
```

```
GREEN: For what the output of a command should look like in your terminal.
```

Several acronyms are used throughout:

Acronym	Definition
MG	MadGraph
CLI	Command Line Interface
MC	Monte Carlo

Download and Installation

Prerequisites: Python v2.6 or newer and ROOT. If you're new to ROOT, see the section below titled Installing ROOT.

Navigate your browser to http://madgraph.hep.uiuc.edu/⬚. If you have not already set up an account, under the Downloads tab at the top of the page, click needs account, and create an account. Once you are able to log in, navigate to the Downloads page and click on the link in the left hand column to download MadGraph5_aMC@NLO.

NOTE: As newer versions of these packages are released, the version numbers in this tutorial will need to be updated in the commands. Since there are library dependencies between some of these packages and ROOT, it is usually not possible to have the latest versions of everything. The working combination I found for this tutorial is: MG5_aMC_v2_2_3, Delphes-3.2.0, root-5.34.28 for Linux and root-6.02.08 for Mac.

Move the tar file to whichever directory you wish the installation to live (I keep a Software directory in my home directory). In this directory, untar the file with:

```
tar -xf MG5_aMC_v2.2.3.tar.gz
```

```
cd MG5_aMC_v2_2_3
```

The next instructions are slightly different for Linux vs Mac.

Linux

The next step is to install the pythia-pgs package. These packages can be downloaded from the same MadGraph site and installed manually, but it is easier to use the MG Command Line Interface (CLI):

```
./bin/mg5_aMC
```

If a newer version of MG is available, you will be asked if you want to update. Enter y or n and press enter to continue.

```
install pythia-pgs
```

Once the installation is complete, you can quit the CLI with:

```
quit
```

Next, we'll install the latest version of Delphes from the developer's site, https://cp3.irmp.ucl.ac.be/projects/delphes ☑, rather than the MG site or CLI. Or we can install it just from MG.

In your main MG directory:

```
wget http://cp3.irmp.ucl.ac.be/downloads/Delphes-3.2.0.tar.gz
tar -xf Delphes-3.2.0.tar.gz
cd Delphes-3.2.0/
make

or
./bin/mg5_aMC
install Delphes
```

In the file input/mg5_configuration.txt, change the delphes_path variable on line 122 from

```
# delphes_path = ./Delphes
```

to

```
delphes_path = ./Delphes-3.2.0
```

If there was a problem with the installation of these packages, red text or an error message will be displayed. Make sure you're using a compatible version of ROOT and the environment is set up correctly (see Installing ROOT section below). You can also check if the error has been resolved on the MG forum at https://answers.launchpad.net/mg5amcnlo ☑. If this doesn't resolve the issue, try using the exact versions used in this tutorial or ask for help.

Mac

Download the pythia-pgs package directly from http://madgraph.hep.uiuc.edu/ ☑ and move it to your main MG directory. In your main MG directory:

```
tar -xf pythia-pgs_V2.4.3.tar.gz
cd pythia-pgs
sudo make
```

My test system was missing a GFortran compiler, which I obtained and installed from https://gcc.gnu.org/wiki/GFortranBinaries#MacOS ☑

If the compilation fails at the STDHEP compilation step, the patch can be found at:

```
curl -O http://www.physics.ucdavis.edu/~conway/stdhep.tar
```

Next, we'll install Delphes. In your main MG directory:

```
curl -O http://cp3.irmp.ucl.ac.be/downloads/Delphes-3.2.0.tar.gz
tar -xf Delphes-3.2.0.tar.gz
cd Delphes-3.2.0/
make
```

In the file input/mg5_configuration.txt, change the delphes_path variable on line 122 from

```
# delphes_path = ./Delphes
```

to

```
delphes_path = ./Delphes-3.2.0
```

If there was a problem with the installation of these packages, red text or an error message will be displayed. Make sure you're using a compatible version of ROOT and the environment is set up correctly (see Installing ROOT section below). You can also check if the error has been resolved on the MG forum at https://answers.launchpad.net/mg5amcnlo🔗. If this doesn't resolve the issue, try using the exact versions used in this tutorial or ask for help.

Madgraph Tutorial

In the MG5_aMC_v2_2_3 directory, enter the CLI:

```
./bin/mg5_aMC
```

Exploring the model

By default the standard model is loaded. To see this enter:

```
display particles
```

Most of the particle names are intuitive, except a, which is a photon.

```
display multiparticles
```

Defining multiparticles can save a lot of typing. You can define your own multiparticles with, for example, to create a multiparticle for the electroweak bosons:

```
define v = w+ w- z a
```

Multiparticle definitions can be used recursively. For example, to include b quarks in the jet definition:

```
define j = j b b~
```

Last, to see the vertices in the model:

```
display interactions
```

Enter q and press enter to exit the interactions file.

Generating a process

To generate a collision process, for protons colliding at the LHC, for example:

```
generate p p > l+ l-
```

This is called the Drell-Yan process, in which a pair of leptons are produced from a Z boson or photon decay. To see the diagrams for this process enter:

```
display diagrams
```

The s-channel mediator for a process can be specified explicitly:

```
generate p p > z > l+ l-
```

```
display diagrams
```

Alternatively, a particle can be excluded from all diagrams with a slash:

```
generate p p > l+ l- /a
```

```
display diagrams
```

For generating more complicated processes, the syntax needs to be entered carefully. For processes with multiple intermediate particles, the decays are separated with commas, and parentheses are used to specify decay chains. For example, Higgs production in association with a Z boson decaying to neutrinos, with the Higgs decaying to four leptons is:

```
generate p p > z h, z > vl vl~, (h > z l+ l-, z > l+ l-)
```

Intermediate particles are constrained to be as close to on-shell as possible. More details on the generate syntax can be found at http://madgraph.hep.uiuc.edu/EXAMPLES/example_mg5.html

An independent process can be added to the generation with, for example:

```
add process p p > w+ h, w+ > l+ vl, (h > z l+ l-, z > l+ l-)
add process p p > w- h, w- > l- vl~, (h > z l+ l-, z > l+ l-)
```

Importing a Model

When we want to simulate physics beyond the standard model, a model with additional particles and/or interactions can be imported. The standard MadGraph installation comes with several models including MSSM, NMSSM, and Randall-Sundrum. To include additional Higgs interactions:

```
import model heft
```

```
display interactions
```

Outputting and Launching a Process

Once you have imported the model heft and generated a process, say

```
generate p p > z h, z > vl vl~, h > a a
```

(Why is the model heft needed to generate this process and not just the SM (at tree level)? Hint: heft stands for Higgs effective field theory. Go back and display the interactions for sm and heft.)

you can save it to a subdirectory in your MG5_aMC_v2_2_3 working directory with:

```
output ppTOzh_zTOvv_hTOaa_13TeV
```

You can name the output directory whatever you want, but I prefer to use this format which mimics the format of the generate command. I also find it's a good habit to append the output name with parameters of the generation, which will be covered later in the tutorial, such as the center of mass energy 13 TeV.

Details of the process and event generation are stored in an html file, which can be opened with:

```
open index.html
```

This file will be opened in Mozilla Firefox, the default browser, if it is installed. You can view the subprocesses and Feynman diagrams for the process under Process Information.

We are now ready to launch the event generation:

```
launch ppTOzh_zTOvv_hTOaa_13TeV
```

This launches the MadEvent CLI. You are presented with options for what packages you want to run during the generation:

```
The following switches determine which programs are run:
 1 Run the pythia shower/hadronization:                   pythia=OFF
 2 Run PGS as detector simulator:                            pgs=OFF
 3 Run Delphes as detector simulator:                    delphes=OFF
 4 Decay particles with the MadSpin module:              madspin=OFF
 5 Add weight to events based on coupling parameters:   reweight=OFF
  Either type the switch number (1 to 5) to change its default setting,
  or set any switch explicitly (e.g. type 'madspin=ON' at the prompt)
  Type '0', 'auto', 'done' or just press enter when you are done.
 [0, 1, 2, 3, 4, 5, auto, done, pythia=ON, ... ][60s to answer]
```

For now, let's only run madevent, the default option, to calculate the cross section for the process. Enter 0 and press enter.

The next menu lists the set of Cards used in the run. Cards are configuration files where the parameters of the run are specified. Let's explore these Cards and identify some commonly varied parameters. Enter 1 and press enter to open the parameter card param_card.dat. The default editor is VIM, which if you haven't learned before, now is your chance!

Particles are specified by the PDG particle ID. A key can be found at http://pdg.lbl.gov/2002/montecarlorpp.pdf⬀

Scroll through the param card and try to identify what the various blocks specify (masses, couplings, widths, etc). When you are ready to exit, hold shift and press zz to exit.

Next, press 2 and enter to open the run card, run_card.dat. Scroll through and try to identify what the various parameters are. What is the default center of mass energy? Reduce the number of events in the run from 10000 to 1000 to shorten the time of the run. (If you are new to VIM, press i to enter INSERT mode, then you can edit text normally. Change nevents to 1000, press esc, then ZZ to exit).

Press 0 and enter to launch the run. A browser window will pop up displaying the progress of the run and the calculated cross section. It's normal for multiple warning messages to be displayed related to the browser display. I usually just close the browser so I can watch the progress of the run on the command line.

The results of the run should look something like:

```
  === Results Summary for run: run_02 tag: tag_1 ===

    Cross-section :   0.0001165 +- 8.076e-07 pb
    Nb of events :   1000
```

Let's see if this result is reasonable. The ZH (NNLO QCD + NLO EW) production cross section can be found from https://twiki.cern.ch/twiki/bin/view/LHCPhysics/CERNYellowReportPageAt14TeV#HZ_Process for Higgs mass of 125 GeV is 0.8830 pb. (This value is for 14 TeV, the originally planned operating energy of the LHC, so if you want an exact comparison, regenerate your MG cross section changing the beam energies to 7000 GeV each. The result is 0.0001263 +- 9.197e-07 pb). The Higgs to diphoton branching ratio can be found at https://twiki.cern.ch/twiki/bin/view/LHCPhysics/CERNYellowReportPageBR#Higgs_2_gauge_bosons for Higgs mass of 125 GeV is 2.29E-3. Pull out your PDG and find the Z to invisible branching fraction (about 20%). Putting this all together, our theoretical cross section is sigma(ZH)*Br(H>aa)*Br(Z>inv) = 0.0004 pb. Our value is about a factor of 4 off. Let's try to do better.

The cross sections calculated by MG for processes including decays tend to be off at LO, so one trick to get a more accurate result is to only calculate the production cross section and use the PDG plus combinatorial factors to calculate the full cross section. In our case, do:

```
import model heft
generate p p > z h
output ppTOzh_14TeV
launch ppTOzh_14TeV
```

Edit the run card to run at center of mass energy 14 TeV. You should get something like:

```
=== Results Summary for run: run_01 tag: tag_1 ===

   Cross-section :   0.6424 +- 0.002715 pb
   Nb of events :  1000
```

Multiplying this by the branching ratios listed above yields 0.0002942, not bad for comparing LO to NNLO.

Bonus: Calculate the uncertainty on our theoretical cross section value and compare it to the uncertainty of the full cross section calculation given by MG and the cross section we calculated using the MG production cross section.

Bonus: Madgraph now has the ability to calculate NLO QCD corrections by appending [QCD] to the generate command. Try this and compare the cross section to our previous values.

Avoiding the CLI

The MG CLI is useful when you're getting started since it walks you through step-by-step and allows you to explore the model, but when you start to generate a lot of MC, scanning over the parameter space of a new model, for example, you'll get really tired of editing cards manually and pressing 3 enter 0 enter 2 enter all day. For this reason, let's go through a few ways to reduce the time spent in the CLI, and how to start automating event generation.

First let's talk about what the output and launch CLI commands do. In your main MG working directory enter:

```
ls Template/LO/
ls ppTOzh_zTOvv_hTOaa_13TeV/
```

Look similar? The output CLI command copies these template files and incorporates the process you generated.

Remember editing the param_card.dat and run_card.dat? Enter:

```
ls Template/LO/Cards/
ls Template/Common/Cards/
ls ppTOzh_zTOvv_hTOaa_13TeV/Cards/
```

So, the output CLI command copies these template Cards to the directory you created in the CLI. If you want to change a default parameter for all of your MC, say the beam energies, you can change it once in the template Card instead of manually every time you output a new process.

Suppose you needed to generate MC over a range of values for a single parameter, such as the mass of a new particle. Since the generate command will be the same for each of these processes, there is no need to go in to the CLI for every process, just do it once, copy and rename the output directory, and change the value of this single parameter in the appropriate Card. This saves a ton of time, especially when you have models with multiple parameters that you want to scan over.

Next, let's talk about what the CLI launch command does. Enter:

```
cd ppTOzh_zTOvv_hTOaa_13TeV/
ls bin/
```

This is a set of executables. These can be executed from this directory, with, for example:

```
./bin/generate_events
```

This should look familiar. This executable launches madevent the same way the launch command did in the CLI. This means you can launch any process from the output directory instead of the main MG CLI.

Taking this a step further, the madevent executable takes optional arguments so that you can bypass the CLI altogether:

syntax: generate_events [run_name] [options]) -- Launch the full chain of script for the generation of events Including possible plotting, shower and detector resolution. Those steps are performed if the related program are installed and if the related card are present in the Cards directory. -- local options: -f : Use default for all questions. --laststep= : argument might be parton/pythia/pgs/delphes and indicate the last level to be run. --session options: Note that those options will be kept for the current session --cluster : Submit to the cluster. Current cluster: condor --multicore : Run in multi-core configuration --nb_core=X : limit the number of core to use to X.

So, you bypass the CLI for the launching stage altogether with a command like

```
./bin/generate_events testrun_01 -f --laststep=parton
```

Be careful that you are confident in the parameters that you set for the run if you use automated commands like this. I still find it useful to sometimes go through the generation process in the CLI just to second check the parameters of the run.

As a last note on avoiding the CLI and automating your event generation, it is possible to do everything after the first generate and output CLI commands in a script, copying and renaming the original output folder, modifying parameters with utilities like sed, and submitting jobs to a cluster's batch system. Setting this up once can save significant time, especially considering the iterative nature of these analyses.

Delphes Tutorial

Generating Events

This section assumes a basic familiarity with ROOT, but if you are new to it, I'll give commands explicitly and explain as we go along. Since we installed Delphes manually instead of using the CLI, the default cards in Template/Common/Cards/ are old and need to be updated. We'll start with a basic configuration for a generic cylindrical detector for this tutorial. From your main MG directory:

```
mv Template/Common/Cards/delphes_card_default.dat Template/Common/Cards/delphes_card_default.dat.
cp Delphes-3.2.0/cards/delphes_card_FCC_basic.tcl Template/Common/Cards/delphes_card_default.dat
```

Enter the CLI and launch our example process:

```
./bin/mg5_aMC
launch ppTOzh_zTOvv_hTOaa_13TeV
```

Now, turn on Delphes by pressing 3 and enter. Notice that Pythia turns on automatically, as it is a required prerequisite for Delphes. You now have the option of viewing and editing the Pythia and Delphes Cards. Open and scroll through each of these to get an idea of what they are doing. Pythia showers and hadronizes the parton level particles, and Delphes simulates the response of a generic cylindrical detector. An easy modification to make in the delphes_card.dat is to "trim" the output root file by commenting out unneeded branches in the TreeWriter module.

Once you are done editing the Cards, launch the job with at least 1000 events. If there is an error during the generation, most likely during the Pythia or Delphes steps if you've made it this far in the tutorial, the most likely culprit is an incompatibility between Delphes and some ROOT libraries. I went through several iterations of Delphes and ROOT versions to get this to run properly.

If you are still having problems getting Delphes to run, I have attached a sample file with 100 events so you can proceed with the tutorial. If the generation finished without error, you can browse through the generated ROOT file:

```
root
TBrowser b
```

and navigate to MG5_aMC_v2_2_3 -> ppTOzh_zTOvv_hTOaa_13TeV -> Events -> run_01 -> tag_1_delphes_events.root. Double click on the root file and the TTree Delphes;1 should appear. If it doesn't, there was probably a silent error when Delphes ran, and you should open tag_1_delphes.log in this same directory to debug the issue.

Double click on Delphes;1 and explore the different branches and leafs. The branch Particle contains parton level kinematics, the branches with particle names contain their respective kinematic distributions, and MissingET contains the MET value and direction for each event.

Just like with MG, you can avoid the CLI when running Delphes by running the executables directly with command line options on the Pythia hep files. This is useful, for example, when you are comparing samples generated with different Delphes configurations but the same MG and Pythia settings:

```
cd Delphes-3.2.0/
gunzip ../ppTOzh_zTOvv_hTOaa_13TeV/Events/run_01/tag_1_pythia_events.hep.gz
./DelphesSTDHEP cards/delphes_card_FCC_basic.tcl ppTOzh_zTOvv_hTOaa_13TeV.root ../ppTOzh_zTOvv_hT
```

Macro-based Analysis

Delphes comes with several very helpful examples for using the Delphes ROOT libraries to loop through ntuples, do basic analysis steps like applying cuts, and to fill histograms. Let's look at a couple of these examples, and add some code to make them more useful:

```
cd Delphes-3.2.0/
ls examples/
cp examples/Example1.C examples/Analyzer1.C
```

Open examples/Analyzer1.C and skim through the code. The branch names used in the pointers are the same as the branches we saw in the ROOT TBrowser earlier, so if you want to access other branches, add a pointer here. Next, histograms are instantiated, or booked. The main piece of code is the for loop accessing events in the ROOT TTree, called the event loop. It is in the event loop that cuts can be applied and histograms are filled. Change the name of the macro from

```
void Example1(const char *inputFile)
```

to

```
void Analyzer1(const char *inputFile)
```

Exit out and run this example on the ppTOzh_zTOvv_hTOaa_13TeV samples we produced earlier:

```
root -l examples/Analyzer1.C'("../ppTOzh_zTOvv_hTOaa_13TeV/Events/run_01/tag_1_delphes_events.roo
```

Note that the library paths are set up such that the macros in the examples directory must be run from the main Delphes directory. This is another point where Delphes and ROOT incompatibilities can pop up if you get an error message about failing to load libraries. Again, this example was found to work with Delphes-3.2.0 and root-5.34.28.

If the macro runs properly, the jet PT is printed out and a not very interesting di-electron invariant mass histogram will be displayed with only a few events. This makes sense as we didn't produce any leptons in our final state, so there will be very few events with two electrons.

The macro has two histograms being displayed, but since they are not placed on different TCanvases, the first one is overwritten by the second. Let's fix this by modifying the following at the bottom of the Analyzer:

```
  TCanvas * canvas1 = new TCanvas("canvas1");
  TCanvas * canvas2 = new TCanvas("canvas2");
  canvas1->cd();
  histJetPT->Draw();
  canvas2->cd();
  histMass->Draw();
```

Run the Analyzer again. Now two not very interesting histograms are displayed, the jet pT and the di-electron invariant mass. You can see from the Jet pT values printed out that our histogram x-axis doesn't go high enough. Change the upper limit from 100 to 250:

```
TH1 *histJetPT = new TH1F("jet_pt", "jet P_{T}", 100, 0.0, 250.0);
```

A more scalable solution to displaying histograms is to write them to a ROOT file. Add the following to the bottom of the Analyzer macro:

```
  TFile *file1 = new TFile("histos.root", "UPDATE");
  histJetPT->Write();
  histMass->Write();
  file1->Close();
```

The UPDATE option will add the histograms to the file each time you run the macro, appending a number to the title. You can change this option to REPLACE to instead replace the histograms. You can TBrowse your file to view the histograms.

On a side note, two common histogram tasks are 1) making nice looking plots and 2) comparing histograms. If you've ever attempted 1), you know this code can be even longer and more complex than the macro that generated the histogram. I find it's most efficient and clean to factor my analysis code into a histogram filling Analyzer script and a separate plot making script, which reads in and manipulates plots from a TFile that I write out in the Analyzer like we've done above. To accomplish 2) in this framework, I write the TFile out with the UPDATE option and change the name of the histogram when running over different ntuples. This would be used to compare the same kinematic distributions, say MET, for different processes.

Let's fill some more interesting histograms for this process, the MET and di-photon invariant mass. Add the branch name pointers below the Jet and Electron pointers:

```
TClonesArray *branchPhoton  = treeReader->UseBranch("Photon");
TClonesArray *branchMET      = treeReader->UseBranch("MissingET");
```

Book the histograms:

```
TH1 *histDiPhotMass = new TH1F("DiPhotMass", "M_{inv}(a_{1}, a_{2})", 100, 40.0, 140.0);
TH1 *histMET              = new TH1F("MET", "MET", 100, 0.0, 300.0);
```

Inside the event loop, add the object identification and histogram filling code:

```
  if(branchMET->GetEntries() > 0)
  {
    Met = (MissingET *) branchMET->At(0);
    histMET->Fill(Met->MET);
  }
  if(branchPhoton->GetEntries() > 1)
  {
    phot1 = (Photon *) branchPhoton->At(0);
    phot2 = (Photon *) branchPhoton->At(1);
    if(phot1->PT < 20 | phot2->PT < 20 | abs(phot1->Eta) > 2.5 | abs(phot2->Eta) > 2.5) continu
    histDiPhotMass->Fill(((phot1->P4()) + (phot2->P4())).M());
  }
```

Note the selection cuts enforced on the photons to clean up the events, each photon must have PT > 20 and absEta < 2.5. Where the other histograms are written to our TFile, write out the new ones:

```
histMET->Write();
histDiPhotMass->Write();
```

TBrowse the output file to check out our new distributions. You should see the Higgs mass reconstructed by the diphoton invariant mass, and a feature in the MET around the Z mass from the neutrinos which escape the detector.

As a final step, let's normalize these distributions to a benchmark scenario of 300 fb^-1 of integrated luminosity at 13 TeV so that we can determine the event yield. The event weight factor is cross section * branching ratio * integrated luminosity / number of generated events. Let's use our theoretical cross section * branching ratio = 0.0004 pb that we found earlier. Before your event loop, calculated the weight factor and create your yield variable:

```
Double_t weight = 0.0004 * 300 * 1E3 / numberOfEntries;
Double_t yield = 0;
Double_t y2 = 0;
```

Then add the weight factor to your histogram Fills:

```
histMET->Fill(Met->MET, weight);
```

and

```
histDiPhotMass->Fill(((phot1->P4()) + (phot2->P4())).M(), weight);
```

At the end of your event loop, update the yield variables:

```
yield += weight;
y2 += weight*weight;
```

After the event loop, add some print statements to show the event yield and selection efficiency:

```
cout << "Event yield:   " << yield << " +/- " << sqrt(y2) << endl;
cout << "Selection Eff: " << yield / (weight*numberOfEntries) << endl;
```

Rerun the macro, TBrowse the output file, and make sure the printed out numbers make sense. If you messed something up during the addition of code, the final version I ended up with is attached.

This tutorial gives you the basic tools needed to perform phenomenology studies. All you need now is an interesting BSM signal model, generate the SM backgrounds that mimic this signal, normalize the distributions, and calculate the sensitivity.

Exercise

1.First generate p p > z h, z > l+ l-, h > b b~ process and p p > t t~, t > b l+ vl, t~ > b~ l- vl~, then plot the invariant mass of bottom pairs and lepton pairs. What is the difference between these two processes? The example of di-photon invariant mass is given. Here is how to run the example. (Remember to change the lib and input directory in the code)

```
root -l dipho_test.C
```

2.Based on one of the processes in exercise 1, plot and compare the kinematics in parton level and reco level. Try to see how well the particle is reconstructed. Plot the delta R between the parton level leptons to their reconstructed leptons. Plot the invariant mass for dilepton and dibottom for parton level, compare that to what you got in exercise 1. Example is given about how to plot the deltaR between the parton level bottom and reconstructed bottom. Here is how to run the code.

```
root -l
root[0].x b_dR.C("path/to/rootfile/name.root")
```

For parton level information, they are stored in the brach "Particle" if you use TBrowser to check the root file. This is how we call the particle branch:

```
TClonesArray *branchParticle = treeReader->UseBranch("Particle");
```

This is how we define the parton level particle

```
GenParticle *par;
for( Int_t i=0; i<branchParticle->GetEntries() ; i++){
    par = (GenParticle *) branchParticle->At(i);
}
```

There are several unique variables for GenParticle. "Status=3" identifies the "hard part" of the interaction, i.e. the partons that are used in the matrix element calculation, including immediate decays of resonances. "PID" shows which particle it is. The PID value could be found at http://pdg.lbl.gov/2002/montecarlorpp.pdf.

Exercise   11

Installing ROOT

By far the easiest way to install ROOT is from the pre-compiled binaries. This is sufficient for most users, unless you want to include an optional package, such as PyROOT, in which case you will need to install from source. The instructions are slightly different for Linux vs Mac.

Linux

First, make sure you have all of the prerequisite packages installed listed at
https://root.cern.ch/drupal/content/build-prerequisites☒. If you're using Ubuntu 10 or newer:

```
sudo apt-get install git dpkg-dev make g++ gcc binutils libx11-dev libxpm-dev \
        libxft-dev libxext-dev
```

The version of ROOT confirmed to be compatible with Delphes-3.2.0 used in this tutorial is 5.34, which can be downloaded from https://root.cern.ch/drupal/content/production-version-534☒. Download the version for your OS and move the tar file to the directory you want ROOT to live (I have a Software/ directory in my home directory), then:

```
tar -xf root_v5.34.28.Linux-ubuntu14-x86_64-gcc4.8.tar.gz
mv root root-5.34.28
```

Since you need to source the ROOT setup script every time you open a new terminal, add it to your .bashrc, which is executed every time a new shell is opened:

```
echo source ~/Software/root-5.34.28/bin/thisroot.sh >> ~/.bashrc
```

Open a new terminal and test the installation with:

```
root
```

and exit with:

```
.q
```

If you want to suppress the annoying pop-up welcome window, add the -l option to the root start up command. To add this to your .bashrc:

```
echo alias root='root -l' >> ~/.bashrc
```

Mac

First, make sure you have all of the prerequisite packages installed listed at
https://root.cern.ch/drupal/content/build-prerequisites☒. The Xcode package can be installed easily from the App store.

The version of ROOT confirmed to be compatible with Delphes-3.2.0 used in this tutorial is 6.02.08, which can be downloaded from https://root.cern.ch/drupal/content/production-version-602☒. Download the version for your OS and move the tar file to the directory you want ROOT to live (I have a Software/ directory in my home directory), then:

```
tar -xf root_v6.02.08.macosx64-10.10-clang60.tar.gz
mv root root-6.02.08
```

Since you need to source the ROOT setup script every time you open a new terminal, add it to your .bash_profile, which is executed every time a new shell is opened:

```
echo source ~/Software/root-6.02.08/bin/thisroot.sh >> ~/.bash_profile
```

Open a new terminal and test the installation with:

```
root
```

and exit with:

```
.q
```

If you want to suppress the annoying pop-up welcome window, add the -l option to the root start up command. To add this to your .bash_profile:

```
alias root='root -l'
```

-- DustinRayBurns - 2015-04-02

This topic: CMSPublic > MadgraphTutorial
Topic revision: r8 - 2016-05-24 - ZhangqierWang