MadGraph5_aMC@NLO

CTEQ School 2014

July 14, 2014

1 Introduction

MadGraph5_aMC@NLO [3] started as the project MadGraph [2] (*i.e.* Madison Graphs) in 1994 as a joint collaboration of T. Stelzer and W.F.Long. At that time Tim Stelzer had started coding by hand the Leading-Order scattering amplitude for a specific process he was studying for his Ph.D. thesis. While buried in the details of his implementation, he realized that hundreds of people would repeat this tedious task for many different processes and models and that he would better invest his time in a more radical solution to these sort of problems; namely the automation of amplitude computations!

The project had a tremendous success, and many people quickly joined the effort to further develop the project and implement MadEvent [5, 6] to perform the integration; *i.e.* producing parton-level events using the amplitudes automatically generated.

More recently, MadGraph was rewritten from scratch in python to become MadGraph5 [7]. Thanks to the great flexibility of this new structure, MadGraph5 became the host of many modular simulation tools. Finally, the MadLoop [4], MadFKS [8] and aMC@NLO [9] ventures merged with MadGraph5 to give MadGraph5_aMC@NLO, bringing together the best of both worlds: flexible automation and accuracy. You can now forget about all other acronyms since, as you are about to witness by yourself, the MG5_aMC framework transparently switches upon need from one tool to another.

The MadGraph5_aMC@NLO framework can perform both fixed order and parton shower simulations, but in the latter case, the parton shower is not implemented directly by MG5_aMC but via an interface to a third-party program (such as Pythia for example, a code that you will learn about later). In this tutorial, we will focus on using standalone MG5_aMC in the context of a fixed-order computation.

Finally, NLO corrections are one way of improving the simulation accuracy but, as you may know by now, merging procedures (*i.e.* combining simulations of increasing parton-level multiplicities) can be added on top to even further refine kinematic shapes. MG5_aMC implements several merging techniques, and in particular at NLO the FxFx [10] approach; we won't cover that aspect of the framework here, but you should be aware of this possibility.

MadGraph5_aMC@NLO is installed on the TASI14 virtual machine in /opt/hep, and the interactive interface from which the framework is steered can be launched from anywhere by running the executable mg5_amc. If you want to get familiar with the tool, you can run an interactive tutorial by typing tutorial in this interface (you can use tab completion for checking the available tutorials dedicated to specific uses of MG5_aMC).

Now, let's dive in and simulate $pp \to Hj$ at fixed order NLO QCD with MadGraph5_aMC@NLO!

2 Setting up the physics model

To start the interactive interface steering MG5_aMC , change to the directory tutorials/intro/mg5_amc and type the command in your terminal (don't type the '$' of course)

```
$ mg5_aMC
```

This brings you to an environment very much like your bash terminal; in particular it features history tracking with the up/down arrow keys, tab completion, and contextual help with the help command. In what follows, we will differentiate the commands to be typed within the MG5_aMC interface and those to be executed directly in the bash shell by using the prefixes MG5_aMC> and $ respectively.

To get familiar with this interface, you can try a couple of commands already, such as display particles, display multiparticles, display interactions, help (or even try out the tutorial, if you have time).

In this tutorial, we want to study $pp \to Hj$; but if you type display interactions g g h, you will notice that it is not found! The reason being that the Standard Model loaded when starting the interface doesn't have this interaction because it comes through loops only. However, it is possible to integrate out the top (in the infinite top mass approximation) and build and effective interaction between the Higgs and two (as well as three) gluons. We refer to this model as the Higgs Effective Field Theory (HEFT). MG5_aMC puts a strong emphasis on flexibility, especially regarding the type of physics model supported. Therefore, all the model information is independent of MG5_aMC and stored in individual standalone python modules following the UFO [1] format. These model modules are folders placed in /opt/hep/mg5/models and they are formally not part of MG5_aMC. They should be seen as plugins defined by the user to specify his model of interest. Of course, the MG5_aMC distribution comes with a variety of predefined models, including the 'heft' one. However, the version of the HEFT model we need for this tutorial must have additional information to be able to handle QCD NLO corrections (such as the UV counterterms for example). In this tutorial, we provide you with this NLO model in /tutorials/intro/mg5_amc/HEFT, but we left it up to you to place it in the MG5_aMC distribution with the command

```
$ tar -xzf ~/tutorials/intro/mg5_amc/HEFT.tar.gz
$ sudo cp -R ~/tutorials/intro/mg5_amc/HEFT /opt/hep/mg5/models/nlo_heft
```

To execute the command above, you must first exit the MG5_aMC interface with the crtl+d key combination or by typing exit. Alternatively, you could execute this shell command directly from within the MG5_aMC interface by preceding it with an exclamation mark.

You are now ready to import this new model, by typing (in the MG5_aMC interface that you should restart if you have left it)

```
MG5_aMC> import model nlo_heft-heft
```

Notice that the suffix -heft is added to the name of the model; this is a name of a *restriction card* (i.e. the file restrict_heft.dat in the /opt/hep/mg5/models/nlo_heft UFO directory) which specifies the default values for the model parameters that MG5_aMC should assume when loading the model. This allows for some optimizations by hardcoding assumptions on the model parameters that we will keep fixed. For example, the heft restriction card specifies that we only consider a diagonal CKM matrix (irrelevant for the purpose of our process) and massless b quarks. This last point is of importance because the original model loaded was the SM with massive b-quarks so that the original definition of the multiparticle labels 'p' (for 'proton') and 'j' (for 'jet') did not include the b-quark. You must now correct for this by redefining them so as to include it with

```
MG5_aMC> define p = p b b~
MG5_aMC> define j = p
```

(the tilde suffix denotes an antiparticle). We are now ready to generate our process $pp \to Hj$, but just before we proceed, try to type display particles again, and see if you can find the new name of the Higgs boson in this model.

3 Process generation

The generation of the code happens in two steps. First, by typing

```
MG5_aMC> generate p p > x0 j [QCD]
```

MG5_aMC creates an internal representation of the process. The generate syntax used above is minimal, and it can be refined to specify plainly of options (coupling orders, s-channel, forbidden particles, etc...). Read more on this by typing help generate. The only non trivial part of the command above is [QCD], which specifies that we want to include both tree-level LO contributions and the real-emission and virtual NLO ones[1]. You can see all the Feynman diagrams generated by typing display diagrams <born/loop/real>, but beware that it will open one pdf file per production channel and diagram category. Can you find the diagrams featuring the QCD ghost contribution? Now, you can write out to disk the process generated. This means that the numerical code for the phase-space integration and the computations of the required amplitudes will be output to the folder of your choice, from which you will be able to run your $pp \to hj$ simulation. Type the command

```
MG5_aMC> output pp_hj
```

where pp_hj is the user-defined name of the folder which is created and contains the numerical code created. Explore a bit the structure of this pp_hj directory, can you see where is located the code for the various amplitudes? Can you find the subfolders belonging to each production channel? Don't hesitate to ask questions about what is what or where is located some part of the simulation you are particularly keen on exploring. We are now ready to *launch* the simulation. Now that the process information has been written out on disk, you could exit the MG5_aMC interface and restart it later to steer the numerical code previously output.

4 Launching simulations

In order to plot the parton level observables of his choice, the MG5_aMC user must[2] code a subroutine that fills histograms for the desired observables, using as input the kinematic configuration and ME weights generated by MG5_aMC during the Monte-Carlo integration procedure. The template code pp_hj/FixedOrderAnalysis/analysis_td_template.f is heavily commented and is an excellent basis for starting the coding of a new analysis. For the purpose of this tutorial, we coded the analysis for plotting the Higgs rapidity and transverse momentum in the fortran file /tutorials/intro/mg5_amc/analysis_td_pp_hj.f that you should copy to the pp_hj/FixedOrderAnalysis folder:

```
cp ~/tutorials/intro/mg5_amc/analysis_td_pp_hj.f ~/tutorials/intro/mg5_amc/pp_hj/FixedOrderAnalysis/
```

The simulation can now be initiated and steered by typing

```
MG5_aMC> launch pp_hj
```

where pp_hj could have been the name of *any* directory that has been output by MG5_aMC during this interactive session or a previous one. The following panel should now be presented to you:

```
The following switches determine which operations are executed:
 1 Perturbative order of the calculation:                        order=NLO
 2 Fixed order (no event generation and no MC@[N]LO matching):   fixed_order=OFF
 3 Shower the generated events:                                  shower=ON
```

[1] In a soon to be public version [QCD] can be [QED] (or even [QCD QED]) to include electroweak correction (mixed QCD/QED/EW corrections respectively). Also by using the [virt=QCD] syntax instead, MG5_aMC generates the code for the loop Matrix Element (ME) only (using MadLoop standalone) without the code for the real-emission and the integration routines. This useful for using the MG5_aMC loop ME with another external Monte Carlo tool. Ask me for more details if your are interested.

[2] For NLO predictions matched to parton showers, one can write out unweighted events which can be reprocessed later by any third-party tool (such as root, MadAnalysis5, rivet, etc...) to plot any observable. This is not a possibility for NLO fixed-order run, ask us why!

```
 4 Decay particles with the MadSpin module:                          madspin=OFF
  Either type the switch number (1 to 4) to change its default setting,
  or set any switch explicitly (e.g. type 'order=LO' at the prompt)
  Type '0', 'auto', 'done' or just press enter when you are done.
 [0, 1, 2, 3, 4, auto, done, order=LO, order=NLO, ... ][60s to answer]
```

Since we are interested in a fixed order simulation, you can type `2` and see the switch `fixed_order` turning to 'ON'. Notice that the switch '`shower`' automatically turned off, do you understand why? We will start by producing the LO distributions so type `1` to switch to LO and then proceed to the next step by typing `0` or the key. The next panel allows you to specify all the simulation parameters:

```
Do you want to edit a card (press enter to bypass editing)?
  1 / param      : param_card.dat
  2 / run        : run_card.dat
  3 / FO_analyse : FO_analyse_card.dat
 you can also
   - enter the path to a valid card or banner.
   - use the 'set' command to modify a parameter directly.
     The set option works only for param_card and run_card.
     Type 'help set' for more information on this command.
   - call an external program (ASperGE/MadWidth/...).
     Type 'help' for the list of available command
 [0, done, 1, param, 2, run, 3, FO_analyse, enter path][60s to answer]
```

You can have a look at the `param_card.dat` by typing `1`; this is where the physical values of the model parameters can be set and you can leave all parameters there unchanged. The `run_card.dat`, opened by typing `2`, specifies the generation-level cuts and general settings of the run. For this tutorial, you should modify the following parameters:

- `req_acc_FO` : This is the desired relative MC accuracy for the inclusive cross-section. To make sure that your simulation goes through sufficiently quickly, set this parameter to something between 0.01 and 0.03.
- `ebeam1` and `ebeam2` : We want to study 8 TeV collisions, so set both these values to 4000.
- `pdlabel` : To use the PDF sets from the `lhapdf` interface and not the internal `MG5_aMC` ones, set this parameter to 'lhapdf'.
- `lhaid` : Let's use the NLO CT10 PDF sets, whose id is 11000 (c.f. `/opt/hep/share/lhapdf/PDFsets.index`). Set 11000 here then.
- `fixed_ren_scale`, `fixed_fac_scale`, `fixed_QES _scale` : Fixed scales are perfectly fine for the process at hand here. So set `T` for all three of these entries.
- `muR_ref_fixed`, `muF1_ref_fixed`, `muF2_ref_fixed`, `QES_ref_fixed` : We shall use the unique scale $\mu = m_H$ here, so set all these to `125.0`
- `reweight_scale`, `reweight_PDF`: `MG5_aMC` has the option of automatically computing the scale and PDF variations (i.e. uncertainties) via reweighing techniques [11] (i.e. no need any longer for rerunning the entire simulation from scratch with different scales or PDF sets). No reason not to benefit from that, so set both entries to `.true.`.
- `PDF_set_min`, `PDF_set_max`: These are the min and max error set ID for the PDF considered. In our case we use CT10, and these should be `11001` and `11052` respectively.

The other parameters can keep their default values, but have a look at them, especially the generation level cuts. Do you understand them all? Why don't we find here some of the cuts that we are used to have in LO Monte-Carlos, for example the minimum deltaR between a photon and a jet?

Finally, in the FO_analyse_card.dat card, we must instruct MG5_aMC to use the analysis written in analysis_td_pp_hj.f (see beginning of this section) by setting the parameter FO_ANALYSE to analysis_td_pp_hj.o.

We are all set, and you can now press the enter key to start the simulation. Pay attention to what MG5_aMC writes out on the screen, and ask us about what you don't understand. For example, what does mean the "Poles successfully cancel for 20 points over 20 (tolerance=1.0e-05)" printout that you will see during the NLO run? You can also follow the progression of the run (although in a less detailed way) from the local webpage which should have opened automatically at the beginning of the run.

Once MG5_aMC displays the summary information of the run, the simulation is over. Before continuing, let's take detour and present an alternative way of steering MG5_aMC simulations; type the following command

```
MG5_aMC> launch -i pp_hj
```

Notice that the prompt changed to pp_hj>. This is because your interface is now bound to the pp_hj process output and the available commands (that you can list by typing 'help') are now different and all related to a type of simulation you can launch for this specific process output. We refer to this bound interface as the *run interface*, as opposed to the *master interface* which is the one that you can start at any time using the shell command mg5_aMC. The run interface of pp_hj can be accessed either by typing launch -i pp_hj from within the master interface (as we just did), or by running the executable pp_hj/bin/aMCatNLO. Finally you can exit the run interface (and return to the master one) at any time using the key combination crtl+d or by typing exit.

We must now re-perform this simulation, but this time including NLO QCD corrections. For this, simply type

```
pp_hj> launch NLO -f
```

from the pp_hj run interface. The '-f' argument bypasses the possible modifications of the cards and directly launches the simulation using the setup last used. Once this NLO run finishes you should be able to infer from the two summaries of the LO and NLO simulations what is the K-factor for that process; does its value surprise you?

5 Plotting the Higgs rapidity and transverse momentum

The entries of the histograms for the observables $y(H)$ and $p_T(H)$ we want to plot are stored in the file MADatNLO.top placed in the pp_hj/Events/run_<n>[_LO] directories where n numbers your simulations and the suffix '_LO' stresses which one was LO only. Copy these files to the mg5_amc directory (while renaming the MADatNLO.top file for the LO simulation to MADatLO.top) , by typing (from within /tutorials/intro/mg5_amc/)

```
$ cp pp_hj/Events/run_01_LO/MADatNLO.top MADatLO.top
$ cp pp_hj/Events/run_02/MADatNLO.top .
```

You can now simply execute the script

```
$ ./makeplots.py
```

that uses gnuplot and the plotting card plotter.gnuplot to create the pdf file plot.pdf which should automatically open. If you have already been through the sherpa tutorial at this stage, you should see the matched and merged sherpa predictions superimposed in black.

We are now back to physics and many questions should arise:

- What is the "kink" around $p_T(H) = 10GeV$ which seems to affect MG5_aMC results?
- Why is the scale dependence flat before this kink, and acquires some shape only after it?
- Why does the LO prediction *not* cover the phase-space region before the kink?

- Why is there a bin up immediately followed by a bin down in the kink region?
- Why is the p_T spectrum of fixed order NLO prediction softer than the matched ones for $p_T < 10GeV$
- Why does it look like NLO corrections have more impact on the shape of the p_T distribution in the case of fixed order simulations than in the case of matched ones?
- Why does the shape of the Higgs rapidity distribution looks more similar across the four types of prediction?

As you can see, the MC accuracy error bars are bigger in the regions of the phase-space contributing less. What could one do to improve the resolution in those regions, other than simply waiting longer after have increased `req_acc`?

6 Playing with the setup

If you still have time, try changing some parameters of the simulation and assessing their impact on the plots created in the previous section. Here are a couple of ideas:

- How much do things change when switching the default dynamic scale?
- When using a PDF set from a different group, does it fall within the error band computed using the CT 10 error set?
- How does the agreement with matched predictions change as you change the minimum jet p_T?

And finally, you can play around with `MG5_aMC` and try to generate a couple of different processes, using the various tweaks of the `generate` syntax, and seeing what are the diagrams generated.

References

[1] C. Degrande, C. Duhr, B. Fuks, D. Grellscheid, O. Mattelaer and T. Reiter, "UFO - The Universal FeynRules Output," Comput. Phys. Commun. **183**, 1201 (2012) [arXiv:1108.2040 [hep-ph]].

[2] T. Stelzer and W. F. Long, "Automatic generation of tree level helicity amplitudes," Comput. Phys. Commun. **81**, 357 (1994) [hep-ph/9401258].

[3] J. Alwall, R. Frederix, S. Frixione, V. Hirschi, F. Maltoni, O. Mattelaer, H. -S. Shao and T. Stelzer *et al.*, "The automated computation of tree-level and next-to-leading order differential cross sections, and their matching to parton shower simulations," arXiv:1405.0301 [hep-ph].

[4] V. Hirschi, R. Frederix, S. Frixione, M. V. Garzelli, F. Maltoni and R. Pittau, "Automation of one-loop QCD corrections," JHEP **1105**, 044 (2011) [arXiv:1103.0621 [hep-ph]].

[5] F. Maltoni and T. Stelzer, "MadEvent: Automatic event generation with MadGraph," JHEP **0302**, 027 (2003) [hep-ph/0208156].

[6] J. Alwall, P. Demin, S. de Visscher, R. Frederix, M. Herquet, F. Maltoni, T. Plehn and D. L. Rainwater *et al.*, "MadGraph/MadEvent v4: The New Web Generation," JHEP **0709**, 028 (2007) [arXiv:0706.2334 [hep-ph]].

[7] J. Alwall, M. Herquet, F. Maltoni, O. Mattelaer and T. Stelzer, "MadGraph 5 : Going Beyond," JHEP **1106**, 128 (2011) [arXiv:1106.0522 [hep-ph]].

[8] R. Frederix, S. Frixione, F. Maltoni and T. Stelzer, "Automation of next-to-leading order computations in QCD: The FKS subtraction," JHEP **0910**, 003 (2009) [arXiv:0908.4272 [hep-ph]].

[9] S. Frixione, F. Stoeckli, P. Torrielli, B. R. Webber and C. D. White, "The MCaNLO 4.0 Event Generator," arXiv:1010.0819 [hep-ph].

[10] R. Frederix and S. Frixione, "Merging meets matching in MC@NLO," JHEP **1212**, 061 (2012) [arXiv:1209.6215 [hep-ph]].

[11] R. Frederix, S. Frixione, V. Hirschi, F. Maltoni, R. Pittau and P. Torrielli, "Four-lepton production at hadron colliders: aMC@NLO predictions with theoretical uncertainties," JHEP **1202**, 099 (2012) [arXiv:1110.4738 [hep-ph]].

LUND UNIVERSITY

For tutorials
at Summer Schools

PYTHIA 8 Merging Tutorial

Torbjörn Sjöstrand
Department of Theoretical Physics, Lund University

Stefan Prestel
Theory group, DESY

1 Introduction

The objective of this exercise is to teach you the basics of how to use the PYTHIA 8.1 event generator to study multi-jet backgrounds. As you become more familiar you will better understand the tools at your disposal, and can develop your own style to use them. Within this first exercise it is not possible to describe the physics models used in the program; for this we refer to the PYTHIA 8.1 brief introduction [1], to the full PYTHIA 6.4 physics description [2] (and all the further references found in them), and [5, 6, 7] concerning matrix element merging.

PYTHIA 8 is, by today's standards, a small package. As such, it should be noted that PYTHIA8 includes a selection $2 \to 1$ and $2 \to 2$ processes, as well as a limited variety of $2 \to 3$ processes, but does not contain a general matrix element generator. New processes, particularly for two or more additional jets, can be made available in form of Les Houches Event (LHE) files. This means that to estimate backgrounds with many jets, you can use a matrix element generator like e.g. MADGRAPH/MADEVENT to improve the PYTHIA8 description of well-separated jets.

2 Getting started: Installing Pythia 8

If you would like to install PYTHIA 8 on your private machine, and you have a C++ compiler, here is how to install the latest PYTHIA 8 version on a Linux/Unix/MacOSX system as a standalone package.

For this tutorial, most installation steps have already been taken. You can find the PYTHIA 8 source code under

```
PYTHIADIR=/opt/hep/share/Pythia8
```

on your virtual machine. PYTHIA 8 has been installed so that you can directly change into one of the tutorial dirctories

```
INTRODIR=~/tutorials/intro/pythia
HIGGSDIR=~/tutorials/higgs/pythia
BOOSTDIR=~/tutorials/boost/pythia
```

and start directly with Step 6 below. Alternatively, the following steps allow you to install PYTHIA 8 from the source code.

While PYTHIA can be run standalone, it can also be interfaced with a set of other libraries. One example is HEPMC, which is the standard format used by experimentalists to store generated events. Since the the location of external libraries is naturally installation-dependent, it is not possible to give a fool-proof linking procedure, but some hints are given below.

1. In a browser, go to

   `http://www.thep.lu.se/~torbjorn/Pythia.html`

2. Download the (current) program package

   `pythia81xx.tgz`

   to a directory of your choice (e.g. by right-clicking on the link).

3. In a terminal window, `cd` to where `pythia81xx.tgz` was downloaded, and type

   ```
   tar xvfz pythia81xx.tgz
   ```

   This will create a new (sub)directory `pythia81xx` where all the PYTHIA 8 source files are now ready and unpacked. From now on, we will call this directory[1] `PYTHIADIR`.

4. Move to this directory (`cd $PYTHIADIR`). If you are only interested in directly producing plots from PYTHIA event records, you can directly go to the next step. If you want to produce and store HEPMC event output, configure the program in preparation for the compilation by typing

   ```
   ./configure --with-hepmc=$HEPMC_PATH
   ```

   where the directory-tree `$HEPMC_PATH` would depend on your local HEPMC installation. Should `configure` not recognise the version number you can supply that with an optional argument, like

   ```
   ./configure --with-hepmc=$HEPMC_PATH--with-hepmcversion=2.06.06
   ```

   For this tutorial, we will use a trick to keep the size of inputs small: We will allow PYTHIA to read compressed Les Houches event files. For this, configure PYTHIA with the options

[1] On your virtual machine, you can find the PYTHIA 8 source files under `/opt/hep/share/Pythia8`.

```
./configure --enable-gzip --with-boost=$BOOST_PATH \
   --with-zlib=$ZLIB_PATH --with-hepmc=$HEPMC_PATH \
   --with-hepmcversion=2.06.06
```

in order to link to the `gzip` and `boost` libraries. For this school, you can use

```
BOOST_PATH=/usr
ZLIB_PATH=/usr/lib/i386-linux-gnu
HEPMC_PATH=/opt-hep
```

5. Do a `make`. This will take 2–3 minutes (computer-dependent). The PYTHIA 8 libraries are now compiled and ready for physics.

6. For test runs, `cd` to the `examples/` subdirectory, or one of the tutorial directories. An `ls` reveals a list of programs, `mainNN`, with `NN` from `01` through `90`. These example programs each illustrate an aspect of PYTHIA 8. For a list of what they do, see the `README` file in the same directory or look at the online documentation. Initially only compile one or two of them to check that the installation works. Once you have worked your way though the introductory exercises in the next sections you can return and study the programs and their output in more detail.
   If you want to produce HEPMC output when installing PYTHIA 8 for the source code, do either of

   ```
   source config.csh
   source config.sh
   ```

   the former when you use the csh or tcsh shells, otherwise the latter. (Use `echo $SHELL` if uncertain.). If you are not interested in HEPMC, this step can be skipped. To execute one of the test programs, do

   ```
   make mainNN
   ./mainNN.exe
   ```

   The output is now just written to the terminal, `stdout`. To save the output to a file instead, do `./mainNN.exe > mainNN.out`, after which you can study the test output at leisure by opening `mainNN.out`. See Appendix A for an explanation of the event record that is listed in several of the runs.

7. If you open the file

   ```
   $PYTHIADIR/htmldoc/Welcome.html
   ```

   you will gain access to the online manual, where all available methods and parameters are described. Use the left-column index to navigate among the topics, which are then displayed in the larger right-hand field.

To produce a matrix-element improved prediction with PYTHIA 8, you will also need to supply Les Houches Event (LHE) files as input. You will have to generate these yourself, with your own cuts, and your preferred matrix element generator. For the sake of this introductory tutorial, we supplied sample LHE files under

```
INTRODIR=~/tutorials/intro/pythia
```

You can use these as input for the programs you will develop in this tutorial. The names of the files reflect the process and the generation settings[2]

`ggh_cc_tree_0.lhe.gz`: $pp \to H$ (leading order, $\mu_f = m_H, \mu_r = m_H$)
`ggh_ll_tree_0.lhe.gz`: $pp \to H$ (leading order, $\mu_f = \frac{1}{2}m_H, \mu_r = \frac{1}{2}m_H$)
`ggh_cc_tree_1.lhe.gz`: $pp \to H + 1$ parton (leading order, $\mu_f = m_H, \mu_r = m_H$)
`ggh_hh_tree_1.lhe.gz`: $pp \to H + 1$ parton (leading order, $\mu_f = 2m_H, \mu_r = 2m_H$)
...

These files come with no guarantees and should only be used for this tutorial. for the first part of the tutorial, you can stick to the input events generated with $\mu_f = m_H$ and $\mu_r = m_H$, i.e. the "`ggh_cc`"-files.

3 Pythia 8 standalone

When using Pythia, you are expected to write the main program yourself, for maximal flexibility and power. Several examples of such main programs are included with the code, to illustrate common tasks and help getting started. You will also see how the parameters of a run can be read in from a file, so that the main program can be kept fixed. Many of the provided main programs therefore allow you to create executables that can be used for different physics studies without recompilation, but potentially at the cost of some flexibility.

3.1 Simple LHC Events

The focus of the next sessions will be on extracting signals from Standard Model backgrounds. So, to get to know the Pythia 8 syntax, we will generate one $pp \to H + \text{jets}$ event at the LHC, using Pythia standalone to produce all jets.

Open a new file `mymain.cc` in the `INTRODIR` subdirectory with a text editor, e.g. Emacs. Then type the following lines (here with explanatory comments added)[3]:

```
// Headers and Namespaces.
#include "Pythia.h"      // Include Pythia headers.
using namespace Pythia8; // Let Pythia8:: be implicit.

int main() {             // Begin main program.

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

[2]The usage of the different input files will be explained in the text.

[3]Experienced Pythia 8 users might be surprised by the initialisation on a Les Houches reader object. Using this slightly more complicated initialisation, `gzip`-compressed files will become readable.

```
  // Declare Les Houches event file reader.
  LHAupLHEF lhareader("./ggh_cc_tree_0.lhe.gz");
  // Initialise pythia on LHE file for qqbar-> W
  pythia.init(&lhareader);

  // Generate event(s).
  // Generate an(other) event. Fill event record.
  pythia.next();

  // End main program with error-free return.
  return 0;
}
```

Next you need to edit the `Makefile` (the one in the `INTRODIR` subdirectory) so it knows what to do with `mymain.cc`. The line

```
# Create an executable for one of the normal test programs

main00 main01 main02 main03 ...  main09 main10 main10 \
```

together with the following three lines enumerate the main programs that do not need any external libraries. Edit the last of these lines to include also `mymain`:

```
 main40 mymain: \
```

Now you can compile and run your main program by typing

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

You can then study `mymain.out`, especially the example of an event record. At this point you need to turn to Appendix A for a brief overview of the information stored in the event record.

An important part of the event record is that many copies of the same particle may exist, but only those with a positive status code are still present in the final state. For illustration, consider a gluon produced in the first initial state splitting of the $gg \to H$ hard interaction. Initially, this parton will have a positive status code. When a shower branching changes properties of this gluon, then the new, changed gluon is added at the bottom of the then-current event record, but the old g is not removed. It is marked as decayed, however, by negating its status code. At any stage of the shower there is thus only one "current" copy of this gluon. When you understand the basic principles, see if you can find several copies of the a parton produced in the hard interaction, and check the status codes to figure out why each new copy has been added. Also note how the mother/daughter indices tie together the various copies.

3.2 A first realistic analysis

We will now gradually expand the skeleton `mymain` program from above, towards what would be needed for a more realistic analysis setup.

- Generally, we wish to generate more than one event. To do this, introduce a loop around `pythia.next()` and `pythia.event.list()`, so the code now reads

```
for (int iEvent = 0; iEvent < 5; ++iEvent) {
  pythia.next();
  pythia.event.list();
}
```

  Hereafter, we will call this the **event loop**. The program will now generate and print 5 events; each call to `pythia.next()` resets the event record and fills it with a new event. Once you start generating many events, it might be convenient to remove the `pythia.event.list()` call. By default, PYTHIA 8 will still print a record of the very first event.

- To obtain statistics on the number of events generated of the different kinds, and the estimated cross sections, add

```
pythia.stat();
```

  just before the end of the program.

- During the run you may receive problem messages. These come in three kinds:
  - a *warning* is a minor problem that is automatically fixed by the program, at least approximately;
  - an *error* is a bigger problem, that is normally still automatically fixed by the program, by backing up and trying again;
  - an *abort* is such a major problem that the current event could not be completed; in such a rare case `pythia.next()` is `false` and the event should be skipped.

  Thus the user need only be on the lookout for aborts. During event generation, a problem message is printed only the first time it occurs. The above-mentioned `pythia.stat()` will then tell you how many times each problem was encountered over the entire run.

- Looking at the `pythia.event.list()` listing for a few events at the beginning of each run is useful to make sure you are generating the right kind of events, at the right energies, etc. For real analyses, however, you need automated access to the event record. The Pythia event record provides many utilities to make this as simple and efficient as possible. To access all the particles in the event record, insert the following loop after `pythia.next()` (but fully enclosed by the **event loop**)

```
for (int i = 0; i < pythia.event.size(); ++i)
  cout << "i = " << i << ", id = " << pythia.event[i].id() <<
endl;
```

  which we will call the **particle loop**. Inside this loop, you can access the properties of each particle `pythia.event[i]`. For instance, the method `id()` returns the PDG identity code of a particle (see Appendix A). The `cout` statement, therefore,

will give a list of the PDG code of every particle in the event record. All methods that give particle properties can be found following the "Particle properties" link in the section "Study output" in the left-hand menu in the manual, or in the file `/path/to//Pythia/pythia8160/htmldoc/ParticleProperties.html`.

- If you are e.g. only interested in final state partons, the `isFinal()` and `isParton()` methods can be applied to the event record entry:

```
for (int i = 0; i < pythia.event.size(); ++i)
  if(pythia.event[i].isFinal() && pythia.event[i].isParton())
       cout << "i = " << i << ", id = " <<
pythia.event[i].id() << endl;
```

  This will only print the PDG code of final state gluons, (anti)quarks and diquarks.

- In addition to the particle properties in the event listing, there are also methods that return many derived quantities for a particle, such as transverse momentum (`pythia.event[i].pT()`). Use this method to find the transverse momentum of the H-boson after the evolution, i.e. the last H-boson in the event record (which is of course cheating a bit, since in an experimental environment, it is a lot more complicated to isolate H-candidates).

- We now want to generate more events, say 1000, to study the shape the $p_{T,\mathrm{H}}$-spectrum. Inside PYTHIA is a very simple histogramming class, that can be used for rapid check/debug purposes. To book the histograms, insert before the **event loop**

```
Hist HistPTH("pT of H-boson", 50, 0., 500.);
```

  where the last three arguments are the number of bins, the lower edge and the upper edge of the histogram, respectively. As an exercise, fill this histogram for each event with the transverse momentum of the H-boson after the evolution. For this, initialise a variable before the **particle loop**, and find the p_T inside a **particle loop**:

```
double pT = 0.;
for (int i = pythia.event.size(); i > 0; --i)
  if( pythia.event[i].idAbs() == 25 ) {
       pT = pythia.event[i].pT();
       break;
  }
```

  Then, before the end of the **event loop**, insert

```
HistPTH.fill(pT);
```

  to fill the histogram. To arrive at a correctly normalised histogram, include

```
HistPTH *= pythia.info.sigmaGen() / pythia.info.nAccepted();
```

  after the **event loop** and the `pythia.stat()` call. In this way, the sum of the heights of all histogram bins will give the correct cross section, irrespectively of

how many events you requested[4]. Finally, to print the histograms to the terminal, add a line like

```
cout << HistPTH;
```

For comparison with merged results, it might be useful to save the output of this run.

4 Tree-level merged predictions

The main program we have constructed in the previous section has one drawback: All radiation will be produced by Pythia 8. This will give reliable results for soft and collinear configurations, but less so for multiple hard, well-separated jets. To model both soft/collinear and well-separated jets at the same time, we need to include matrix element calculations – which describe the production of multiple hard jets nicely – into the jet evolution of the parton shower. This can be done by supplying LHE files to Pythia 8, which will then internally be processed to perform a smooth transition from n-jet to $n+1$-jet events. Several main programs illustrating matrix element + parton shower merging (MEPS) are included in the Pythia 8 distribution, offering a range of different merging schemes:

- Tree-level merging: MLM-style jet matching with Madgraph or Alpgen, CKKW-L merging [5], and unitarised ME+PS merging (UMEPS) [6];
- Next-to-leading order merging: NL^3 merging, unitarised NLO+PS merging (UNLOPS) [7].

Here, we will get to know the CKKW-L, UMEPS and UNLOPS schemes by manipulating some example main programs[5] We will use H+ ≤ 2 jets as an example.

4.1 CKKW-L merging

CKKW-L merging [5] was the first merging scheme available in Pythia 8. For this school, we supply the example main program `main-ckkwl.cc`. This program uses the LH files in the examples directory to produce a result that simltaneously describes $\mathrm{H} + 0, 1, 2$ jet observables with leading-order matrix elements, while also including PS resummation (i.e. arbitrarily many PS emissions). CKKW-L is described in the online manual under **Link to other programs → CKKW-L merging**.

Take as an example one-jet observables (e.g. the transverse momentum of the jet in events with *exactly* one jet). In this case, we want to take a "hard" jet from the $\mathrm{pp} \to \mathrm{H}j$ matrix element (ME), while "soft" jets should be modelled by emissions off $\mathrm{pp} \to \mathrm{H}$ states, generated by Pythia. This means that in order to smoothly merge these two samples,

[4]Pythia cross sections are given in units of mb. If you instead prefer e.g. pb then multiply by `1e9`.

[5]A more concise tutorial on how to write a CKKW-L main program from scratch can be found at `http://home.thep.lu.se/~torbjorn/pythia8/mergingworksheet8160.pdf` .

we have to know in which measure "hard" is defined, and which value of this measure separates the hard and soft regions. In `main-ckkwl.cc`, these definitions are

```
pythia.readString("Merging:doPTLundMerging = on");
pythia.readString("Merging:TMS = 15");
```

This will enable the merging procedure, with the merging scale defined through the evolution $p_\perp$ of Pythia 8. Such a definition fixes what we mean when we talk about "hard" and "soft" jets[6]:

$$\begin{aligned} \text{Hard jets:} &\quad \min\{p_{\perp evol}(\text{Hard jet})\} > t_{\text{MS}} \\ \text{Soft jets:} &\quad \min\{p_{\perp evol}(\text{Soft jet})\} < t_{\text{MS}} \end{aligned}$$

Then, to define that we want to produce results for Higgs production, we set

```
pythia.readString("Merging:Process = pp>h");
```

and allow for full flexibility in the the Higgs decay products, by simply disregarding the decay products in the merging:

```
pythia.readString("Merging:mayRemoveDecayProducts = on");
```

Finally, we want to include the pre-generated ME events for up to two additional jets. This is communicated to Pythia by setting

```
pythia.readString("Merging:nJetMax = 2");
```

Then, `main-ckkwl.cc` will start generating events, reading from the LHE file with the highest parton multiplicity first. Each event after the merging comes with a weight

```
double weight = pythia.info.mergingWeight();
```

which contains Sudakov factors to remove the double counting between samples of different multiplicity, α_s-ratios to incorporate α_s-running, and ratios of parton distributions to include variable factoriation scales. This weight *must* be used to fill histogram bins. As an exercise, convince yourself that the variation in this weight is moderate.

`main-ckkwl.cc` finishes by normalising the histograms for each sub-sample by the correct cross section

```
SubHistPTH *= 1e9 * pythia.info.sigmaGen() / pythia.info.nAccepted();
```

and then adding the histograms

```
HistPTH    += SubHistPTH;
```

[6]There is of course a plethora of possilbe hardness criteria. For this reason, CKKW-L merging in Pythia 8 allows you to define your very own merging scale (please consult the manual for details).

You can compile and `main-ckkwl.cc` by issuing the commands

```
make main-ckkwl
./main-ckkwl.exe
```

Now that we understand the main program, let us (a) compare to the full result of simply showering the Higgs production process, (b) check in which $p_\perp$ regions which jet multiplicity contributes, (c) vary the merging scale value `TMS`.

If you would like to get your hands dirty on some CKKW-L details, check out the collection of main programs for CKKW-L merging (`main81.cc-main85.cc`) that Pythia 8 offers. Each of these require a different level of knowledge about the merging procedure. `main85.cc` is intended as a black box. However, before you get stuck on CKKW-L, it is useful to know that other, more modern, merging schemes are available in Pythia 8.

4.2 Unitarised ME+PS merging

One main disadvantage of classical merging schemes is the matrix element calculations that are included into the parton shower might contain large logarithmic contributions that cannot be cancelled by the shower resummation. These terms can, particularly for small merging scales, lead to changes of the inclusive cross section after multi-jet merging. A better shower would ameliorate this issues. Still, such problems reveal that the merging scale value is not a purely technical parameter, but should be chosen in a sensible range.

However, it is possible to demote the merging scale to a technical parameter without having to improve the PS resummation. Rather, improved approximate virtual corrections can be employed, resulting in a scheme called unitarised ME+PS merging, short UMEPS [6]. The basic idea is simple: Subtract what you add, then the inclusive cross section will stay fixed. The way this "add-subtract" scheme is facilitated (without disturbing the description of well-separated jets) closely follows the prescription how (un-merged) parton showers preserve the inclusive cross section. UMEPS is described in the online manual under **Link to other programs → UMEPS merging**.

For this school, we supply a sample main program[7] for UMEPS merging, `main-umeps.cc`. The main program starts by setting the merging scale value

```
pythia.readString("Merging:TMS = 15");
```

Note that the actual merging scale definition will be set on a sample-by-sample basis. Then, the core scattering is defined

```
pythia.readString("Merging:Process = pp>h");
pythia.readString("Merging:mayRemoveDecayProducts = on");
```

and the maximal number of additional partons in the matrix element calculation is set

[7]The standard UMEPS sample main program is `main86.cc`.

```
pythia.readString("Merging:nJetMax = 2");
```

The event generation code starts after these settings. Note that this code is split into an "additive" part (the loop over `njetcounterLO`), followed by a "subtractive" part (the loop over `njetcounterCT`).

In the additive part of UMEPS, the merging scale is defined by

```
pythia.readString("Merging:doUMEPSTree = on");
```

This is currently the only merging scale definition for UMEPS, and corresponds to separating hard and soft jets according to $\min\{p_{\perp evol}\}$. The setting

```
pythia.readString("Merging:nRecluster = 0");
```

is also necessary to ensure that the algorithm works as expected (see below). Histograms bins for observables in this additive part should, as in CKKW-L, be filled with the weight containing Sudakov-, $\alpha_{\rm s}$ and PDf-factors,

```
double weight = pythia.info.mergingWeight();
```

and the resulting sub-sample histogram should added to the to-be merged histogram.

In the subtractive part of UMEPS, we define the merging scale by

```
pythia.readString("Merging:doUMEPSSubt = on");
pythia.readString("Merging:nRecluster = 1");
```

This enables the trick that is at the very heat of UMEPS: In a parton shower, approximate virtual corrections V_{ps} and approximate real emissions R_{ps} cancel because $V_{ps} = -\int R_{ps}$. Thus, the inclusive cross section is not changed by the parton shower. This cancellation is however spoiled when merging a real emission matrix element R_{me}, since $R_{me} \neq R_{ps}$. The subtractive part of UMEPS puts $V_{umeps} = -\int R_{me}$, thus explicitly enforcing the parton-shower-like cancellation. The necessary integration is enabled by putting `Merging:nRecluster = 1`. As always, the result should be histogrammed with the weight

```
double weight = pythia.info.mergingWeight();
```

Now, the resulting sub-sample histograms should be subtracted from the to-be merged result.

You can compile and `main-umeps.cc` by issuing the commands

```
make main-umeps
./main-umeps.exe
```

Once we have understood `main-umeps.cc`, let us find the average weight (for one particular number of additional ME partons) of the additive parts of UMEPS, and compare this to the average weight (of the same parton multiplicity) in CKKW-L. Further, change the merging scale value in UMEPS, and try to understand the changes. Compare the merged results between CKKW-L and UMEPS.

If you have finished these challenges, you should take the opportunity to try a few other options. Below are given some examples, but feel free to pick something else that you would be more interested in.

In the next session, you will move one step further in the history of merging schemes. To assess the theoretical uncertainties of current merging methods, you will use the best method we have so far in Pythia 8: NLO multi-jet merging in the UNLOPS scheme.

5 Next-to-leading order merging

The most recent development in combining fixed-order calculations and parton shower resummation is next-to-leading order merging. This is different from NLO matching (e.g. POWHEG and MC@NLO) because it allows to describe different parton multiplicities simultaneously at NLO accuracy. NLO merging schemes are the successor of tree-level merging schemes.

In this tutorial, we will assess the uncertainties of the unitarised NLO+PS merging method (UNLOPS, [7]). For this, we will depart from our usual practise of simply printing histograms to the terminal. Instead, we will use a sample main program that uses HepMC output, which can then be analysed with RIVET. We will briefly describe the main program used for this study, and come back to the actual uncertainty estimates below.

5.1 main88.cc

UNLOPS merging is the direct extention of UMEPS to NLO accuracy. For this school, we would like our simulation to describe $\mathrm{H}+0$ and $\mathrm{H}+1$ jet simultaneously with NLO accuracy, describe $\mathrm{H}+2$ jets with tree-level accuracy, and take all further jets from the PS approximation. This means we have to consider two types of input calculations: Tree-level and NLO inputs. For this school, we supply tree-level event samples produced with MadGraph, and NLO event samples produced with POWHEG-BOX. UNLOPS then processes these samples in the following way:

(1) Use tree-level matrix elements for n partons as "seeds" for higher-order corrections (of $\mathcal{O}\left(\alpha_{\mathrm{s}}^{n+2}\right)$ and beyond), making sure that no $\mathcal{O}\left(\alpha_{\mathrm{s}}^{n+1}\right)$ are produced.

(2) Add the NLO samples, making sure that no higher orders (of $\mathcal{O}\left(\alpha_{\mathrm{s}}^{n+2}\right)$ and beyond) are produced.

(3) Unitarise everything, making sure that no unwanted $\mathcal{O}\left(\alpha_{\mathrm{s}}^{n+1}\right)$ terms are produced, i.e. ensure that the inclusive cross section is given by the $\mathrm{H}+0$ NLO result.

This is done internally in `main88.cc`, the sample main program we will use to generate NLO merged results. In the event generation phase, `main88.cc` uses the tree-level input file twice (once for Step (1) and once for Step (3)), as well as using the NLO (POWHEG) inputs twice (once for Step (2) and once for Step (3)). `main88.cc` is described in detail in the online manual under **Link to other programs → NLO merging**[8].

`main88.cc` reads a settings file (e.g. `main88.cmnd`) for the necessary settings. It is important to set the switches

```
// Definition core process for merging
Merging:Process                 = ?
Merging:mayRemoveDecayProducts  = ?
// Maximal number of additional LO jets.
Merging:nJetMax                 = ?
// Maximal number of additional NLO jets.
Merging:nJetMaxNLO              = ?
// Merging scale value.
Merging:TMS                     = ?
// Values of (fixed) scales in the matrix element calculation.
Merging:muFacInME               = ?
Merging:muRenInME               = ?
// Values of (fixed) scales for the PS lowest multiplicity process.
Merging:muFac                   = ?
Merging:muRen                   = ?
```

in such an input file. The process definition, maximal number of additional tree-level jets and merging scale value have already been discussed in the CKKW-L section. In addition, you now need to set the maximal number of additional jets for which an NLO event sample is available, and the renormalisation and factorisation scales with which the event samples have been produced. Finally, it is also necessary to set te scales which would be used in default Pythia 8 to evaluate the core scattering (i.e. for H production, the mass m_{H}). In the case of wimpy showers, the value of `Merging:muFac` further sets the shower starting scale in UNLOPS.

In general, you also need to generate tree-level- and POWHEG event samples as input for `main88.cc`. For this school, we have included some pre-calculated samples in the `HIGGSDIR` directory. The main program `main88.cc` assumes, to allow for a streamlined file parsing, a particular naming convention. All POWHEG event samples should be called

```
myLHEF_powheg_njets.lhe.gz
```

where `myLHEF` is a free process idetifier, which is assumed to be identical for all samples belonging to one particular process. `njets` should give the number of additional partons that are described at NLO accuracy (i.e. *not* counting real-emission partons). POWHEG

[8] In fact, we have slightly changed `main88.cc` to be able to read compressed LHE files.

events for H + j@NLO could for example be called `higgs_powheg_1.lhe.gz`. Tree-level inputs are called

```
myLHEF_tree_njets.lhe.gz
```

A legitimate name for a tree-level sample with two additional jets could e.g. be `higgs_tree_2.lhe.gz`.

To use `main88.cc`, go into the `HIGGSDIR` directory and compile the main program

```
cd $HIGGSDIR
make main88
```

and run by issuing a command of the form

```
./main88.exe myInputFile myLHEF myHEPMCoutput
```

`main88.cc` will then, consecutively, read tree-level and POWHEG event samples, and produce HepMC events.

To get used to the program, produce only small number of events, and run

```
./main88.exe main88.cmnd ggh_cc myHEPMCoutput
```

Can you identify the Steps (1)-(3) through the terminal output? Why is step (3) applied both to tree-level and POWHEG samples?

5.2 Uncertainty estimates

In this part, we will try to estimate the scale uncertainty of the UNLOPS method. For this, we will vary μ_f and μ_r in the matrix element calculation, while keeping the scales in the parton shower resummation unchanged. In the `HIGGSDIR` directory, you can find event files produced with three different scale settings:

- Files whose name contains `ll` have been generated with $\mu_f = \mu_r = \frac{1}{2} m_{\mathrm{H}}$
- Files whose name contains `cc` have been generated with $\mu_f = \mu_r = m_{\mathrm{H}}$
- Files whose name contains `hh` have been generated with $\mu_f = \mu_r = 2 m_{\mathrm{H}}$

Let us now investigate the scale uncertainty of the (H+0, 1@NLO)+(H+2@LO) prediction. We will use the analysis `MC_HJETS` of RIVET to do this.

For each (fixed-order) scale setting, you will need to change `Merging:muFacInME` and `Merging:muRenInME` in your input file and use the correct LHE samples.

We use HepMC events to parse the Pythia 8 output to RIVET. However, HepMC event files quickly become prohibitively large. This is why RIVET allows the usage

of `fifo` pipes to pass the events at generator run time, without having to store large intermediate files. PYTHIA 8 will write a single event to the pipe, and RIVET will read and analyse this event. To make this as simple as possible, you can use the BASH script `run.sh`. This script allows to run both PYTHIA 8 and RIVET in the background of one common terminal. To use `run.sh`, make it executable

```
chmod +x run.sh
```

and infer it by

```
./run.sh
```

`main88.cc` needs to be compiled before running the script. You will have to change the name of LHE files, log-files and `aida` output files in the script, and the scales in the settings file, if you want to generate histograms with different scale choices.

Now, try to do the following

- Run `main88.cc`, for the central scales, with only (H + 0@NLO)+(H + 1, 2@LO), and with H + 0, 1@NLO)+(H + 2@LO). Check the exclusive jet multiplicity of the `MC_HJETS` analysis. Can you explain the differences?
- Run `main88.cc`, with H + 0, 1@NLO)+(H + 2@LO) for the three scale choices
  (a) $\mu_f = \mu_r = \frac{1}{2} m_{\mathrm{H}} = 62.5$ GeV
  (b) $\mu_f = \mu_r = m_{\mathrm{H}} = 125$ GeV
  (c) $\mu_f = \mu_r = 2 m_{\mathrm{H}} = 250$ GeV

  Display the results together, and try to understand the differences.
- Switch between wimpy and power showers by changing the settings `TimeShower:pTmaxMatch` and `SpaceShower:pTmaxMatch`.

Finally, compare the UNLOPS results to the results of the other Event Generators. This is most conveniently done if you display the variations as uncertainty band. You can modify the script

```
~/tutorials/higgs/plotit.sh
```

for this purpose. Type

```
cd ~/tutorials/higgs/
./plotit.sh -help
```

to find out about the script.

This concludes the generator-specific part of the tutorials. Don't hesitate to contact us [8] if you have further questions.

Bonus: Some interesting settings

You are now free to play with further options in the input file (or use the `pythia.readString` method directly in your code), and make changes such as:

- `Tune:pp = 5` (or other values between 1 and 7)
  Different combined tunes, in particular to radiation and multiple interactions parameters. In part this reflects that no generator is perfect, and also not all data is perfect, so different emphasis will result in different optima. Currently, the default tune for LHC is Tune 4C, (which can also be explicityly set by `Tune:pp = 5`). What happens if you switch to Tune A2, which is a recent tune that can be used by putting `Tune:pp = 7`?

- `PartonLevel:FSR = off`
  switch off final-state radiation.

- `PartonLevel:ISR = off`
  switch off initial-state radiation.

- `PartonLevel:MPI = off`
  switch off multiple interactions.

- `TimeShower:pTmaxMatch = 1`
  wimpy final state showers (default).

- `SpaceShower:pTmaxMatch = 1`
  wimpy initial state showers (non-default).

For debugging your code for instance, it might be reasonable to switch off multiple interactions, so that you can produce plots more quickly.

The philosophy of Pythia is to make every parameter available to the user. A complete list of changeable settings can be found in the online manual. Have a look at the switches related to MEPS merging. As additional challenge, think about which switches are dangerous, i.e. will maximally corrupt your predictions. Can you isolate a singularly bad setting? Why does the prediction deteriorate with the changes?

A The Event Record

The event record is set up to store every step in the evolution from an initial low-multiplicity partonic process to a final high-multiplicity hadronic state, in the order that new particles are generated. The record is a vector of particles, that expands to fit the needs of the current event (plus some additional pieces of information not discussed here). Thus `event[i]` is the `i`'th particle of the current event, and you may study its properties by using various `event[i].method()` possibilities.

The `event.list()` listing provides the main properties of each particles, by column:

- `no`, the index number of the particle (`i` above);
- `id`, the PDG particle identity code (method `id()`);
- `name`, a plain text rendering of the particle name (method `name()`), within brackets for initial or intermediate particles and without for final-state ones;
- `status`, the reason why a new particle was added to the event record (method `status()`);
- `mothers` and `daughters`, documentation on the event history (methods `mother1()`, `mother2()`, `daughter1()` and `daughter2()`);
- `colours`, the colour flow of the process (methods `col()` and `acol()`);
- `p_x`, `p_y`, `p_z` and `e`, the components of the momentum four-vector (p_x, p_y, p_z, E), in units of GeV with $c = 1$ (methods `px()`, `py()`, `pz()` and `e()`);
- `m`, the mass, in units as above (method `m()`).

For a complete description of these and other particle properties (such as production and decay vertices, rapidity, $p_\perp$, etc), open the program's online documentation in a browser (see Section 2, point 6, above), scroll down to the "Study Output" section, and follow the "Particle Properties" link in the left-hand-side menu. For brief summaries on the less trivial of the ones above, read on.

A.1 Identity codes

A complete specification of the PDG codes is found in the Review of Particle Physics [3]. An online listing is available from

`http://pdg.lbl.gov/2008/mcdata/mc_particle_id_contents.html`

A short summary of the most common `id` codes would be

1	d	11	e^-	21	g	211	π^+	111	π^0	213	ρ^+	2112	n
2	u	12	ν_e	22	γ	311	K^0	221	η	313	K^{*0}	2212	p
3	s	13	μ^-	23	Z^0	321	K^+	331	η'	323	K^{*+}	3122	Λ^0
4	c	14	ν_μ	24	W^+	411	D^+	130	K^0_L	113	ρ^0	3112	Σ^-
5	b	15	τ^-	25	H^0	421	D^0	310	K^0_S	223	ω	3212	Σ^0
6	t	16	ν_τ			431	D^+_s			333	ϕ	3222	Σ^+

Antiparticles to the above, where existing as separate entities, are given with a negative sign.

Note that simple meson and baryon codes are constructed from the constituent (anti)quark codes, with a final spin-state-counting digit $2s + 1$ (K^0_L and K^0_S being exceptions), and with a set of further rules to make the codes unambiguous.

A.2 Status codes

When a new particle is added to the event record, it is assigned a positive status code that describes why it has been added, as follows:

code range	explanation
11 – 19	beam particles
21 – 29	particles of the hardest subprocess
31 – 39	particles of subsequent subprocesses in multiple interactions
41 – 49	particles produced by initial-state-showers
51 – 59	particles produced by final-state-showers
61 – 69	particles produced by beam-remnant treatment
71 – 79	partons in preparation of hadronization process
81 – 89	primary hadrons produced by hadronization process
91 – 99	particles produced in decay process, or by Bose-Einstein effects

Whenever a particle is allowed to branch or decay further its status code is negated (but it is *never* removed from the event record), such that only particles in the final state remain with positive codes. The `isFinal()` method returns `true/false` for positive/negative status codes.

A.3 History of parton shower branchings

The two mother and two daughter indices of each particle provide information on the history relationship between the different entries in the event record. The detailed rules depend on the particular physics step being described, as defined by the status code. As an example, in a $2 \to 2$ process $ab \to cd$, the locations of a and b would set the mothers of c and d, with the reverse relationship for daughters. When the two mother or daughter indices are not consecutive they define a range between the first and last entry, such as a string system consisting of several partons fragment into several hadrons.

There are also several special cases. One such is when "the same" particle appears as a second copy, e.g. because its momentum has been shifted by it taking a recoil in the dipole picture of parton showers. Then the original has both daughter indices pointing to the same particle, which in its turn has both mother pointers referring back to the original. Another special case is the description of ISR by backwards evolution, where the mother is constructed at a later stage than the daughter, and therefore appears below in the event listing.

If you get confused by the different special-case storage options, the two `pythia.event.motherList(i)` and `pythia.event.daughterList(i)` methods are able to return a `vector` of all mother or daughter indices of particle `i`.

A.4 Colour flow information

The colour flow information is based on the Les Houches Accord convention [4]. In it, the number of colours is assumed infinite, so that each new colour line can be assigned a new

separate colour. These colours are given consecutive labels: 101, 102, 103, ... A gluon has both a colour and an anticolour label, an (anti)quark only (anti)colour.

While colours are traced consistently through hard processes and parton showers, the subsequent beam-remnant-handling step often involves a drastic change of colour labels. Firstly, previously unrelated colours and anticolours taken from the beams may at this stage be associated with each other, and be relabelled accordingly. Secondly, it appears that the close space–time overlap of many colour fields leads to reconnections, i.e. a swapping of colour labels, that tends to reduce the total length of field lines.

References

[1] T. Sjöstrand, S. Mrenna and P. Skands, Comput. Phys. Comm. **178** (2008) 852 [arXiv:0710.3820]

[2] T. Sjöstrand, S. Mrenna and P. Skands, JHEP **05** (2006) 026 [hep-ph/0603175]

[3] Particle Data Group, C. Amsler et al., Physics Letters **B667** (2008) 1

[4] E. Boos et al., in the Proceedings of the Workshop on Physics at TeV Colliders, Les Houches, France, 21 May - 1 Jun 2001 [hep-ph/0109068]

[5] L. Lönnblad and S. Prestel, JHEP **03** (2012) 019, `arxiv:1109.4829 [hep-ph]`

[6] L. Lönnblad and S. Prestel, JHEP **02** (2013) 094, `arxiv:1211.4827 [hep-ph]`

[7] L. Lönnblad and S. Prestel, JHEP **03** (2013) 166, `arxiv:1211.7278 [hep-ph]`

[8] For merging related questions, email `stefan.prestel@thep.lu.se`
In case of general problems, contact us under `pythia8@projects.hepforge.org`

Sherpa Tutorial

CTEQ School 2014

July 14, 2014

1 Introduction

Sherpa is a complete Monte-Carlo event generator for particle production at lepton-lepton, lepton-hadron, and hadron-hadron colliders [1]. The simulation of higher-order perturbative QCD effects, including NLO corrections to hard processes and resummation as encoded in the parton shower, is emphasized in Sherpa. QED radiation, underlying events, hadronization and hadron decays can also be simulated. Alternatively, Sherpa can be used for pure parton-level NLO QCD calculations with massless or massive partons.

Many reactions at the LHC suffer from large higher-order QCD corrections. The correct simulation of Bremsstrahlung in these processes is essential. It can be tackled either in the parton-shower approach, or using fixed-order calculations. Sherpa combines both these methods using a technique known as Matrix Element + Parton Shower merging (ME+PS). Details are described in Ref. [2] and have been discussed in the lectures. This tutorial will show you how to use the method in Sherpa.

Sherpa is installed on the virtual machine in `/opt/hep`, and its documentation is found online [3]. Example setups are located in `/opt/hep/share/SHERPA-MC/Examples`.

2 The Input File

Sherpa is steered using input files, which consist of several sections. A comprehensive list of all input parameters for Sherpa is given in the Sherpa manual [3]. For the purpose of this tutorial, we will focus on the most relevant ones.

Change to the directory `tutorials/intro/sherpa` and open the file `Run.dat` in an editor. Have a look at the section which is delimited by the tags `(run){` and `}(run)` (We will call this section the `(run)` section in the following). You will find the specification of the collider, i.e. its beam type and center-of-mass energy, as well as a couple of other parameters, which will be explained later.

The `(processes)` section specifies, which reactions are going to be simulated. Particles are identified by their PDG codes, i.e. 1 stands for a down-quark, -1 stands for an anti-down, 2 for an up-quark, etc. The special code 93 represents a "container", which comprises all light quarks, b-quarks, and the gluon. It is also called the "jet" container.

3 Running Sherpa

Have you found out which physics process is going to be generated? You can verify your guess by running the command line

```
Sherpa
```

When run for the first time, Sherpa will produce diagram information for the calculation of hard scattering processes. It will also produce source code for computing the hard scattering cross sections, which must be compiled using

```
./makelibs
```

Have a look at the Feynman diagram, which contributes to the simulation of the hard process:

```
plot_graphs.sh graphs/
firefox graphs/index.html
```

Is this the diagram you expect? If not, which one is missing?

Next, run

```
Sherpa -e1 -o3
```

The option `-e1` instructs Sherpa to produce one event, and the option `-o3` will print the result of event generation on the screen. You will see Sherpa's internal event record. Search for `Signal Process` inside the output and check incoming and outgoing particles. What happens to the unstable particle after it has been produced in the hard scattering? Is this result physically meaningful?

4 Analyses

By default, Sherpa does not store or analyze events. Run Sherpa with the following command to perform a simple analysis of the Higgs-boson kinematics:

```
Sherpa -a1
```

The configuration file for the analysis is `Analysis.dat`. It instructs Sherpa to produce a few simple observables for diagnostic purposes. The output will be a directory called `Analysis/`, which contains histogram files in plain text format. Use the script provided to plot these histograms

```
./plot_results.sh
```

Have a look a the plots by opening the file `plots.ps` with `gv`. Can you explain why there are no differences in the rapidity distribution and large differences in the transverse momentum distribution?

5 MC@NLO matching

The current runcard lets Sherpa generate events at lowest order in the strong coupling. To improve the description of real radiative corrections, we can include higher-multiplicity tree-level contributions in the simulation. This is done by changing the process specification

```
Process 93 93 -> 25;
```

to

```
Process 93 93 -> 25;
NLO_QCD_Mode MC@NLO;
```

The second line instructs Sherpa to compute cross section at NLO and match the NLO calculation to the parton shower using the MC@NLO method. The essence of this method is a modified subtraction, which removes the approximate NLO corrections generated by the parton shower from the hard cross section calculation before processing events in the parton shower.

Run the new setup. New source code will be created which you need to compile using

```
./makelibs
```

Run the setup again. Why does Sherpa compute two cross sections?

Produce a separate event sample with MC@NLO matching and analyze it, but store the analysis results in a different directory than the previous runs. This can be done with the command

```
Sherpa -a1 -AAnalysis_MCNLO/
```

Compare the new results to your previous ones using the plot script. Do you observe any differences? Why?

6 ME+PS merging

The current runcard lets Sherpa generate events at NLO in the strong coupling. To improve the description of real radiative corrections only, we can compute higher-multiplicity tree-level contributions in the simulation. This is done by changing the process specification

```
Process 93 93 -> 25;
NLO_QCD_Mode MC@NLO;
```

to

```
Process 93 93 -> 25 93{1};
```

The last entry instructs Sherpa to produce up to one additional "jet" using hard matrix elements and combine the respective process with the leading-order process. This is known as Matrix Element + Parton Shower merging (ME+PS), or the CKKW method. The essence of the method is a separation of hard from soft radiative corrections, achieved using phase-space slicing by means of a variable called the jet criterion. The slicing parameter is called the merging cut.

Let us assume we want to classify jets of more than 20 GeV transverse momentum as hard. In Sherpa, the corresponding merging cut would be specified as

```
CKKW sqr(60/E_CMS);
```

Therefore, the complete `(processes)` section for the merged event sample reads:

```
(processes){
  Process 93 93 -> 25 93{1};
  CKKW sqr(60/E_CMS);
  Order_EW 1;
  End process;
}(processes);
```

If you like, you can have a look at the Feynman graphs again, which contribute to the ME+PS merged event sample. In this case, you should not remove the line `Print_Graphs graphs;` from the `(processes)` section, and rerun the plot command from Sec. 3.

Run the new setup. Why does Sherpa compute another cross section?

Produce a separate event sample with ME+PS merging and analyze it, but store the analysis results in a different directory than the previous runs. This can be done with the command

```
Sherpa -a1 -AAnalysis_MEPS/
```

Compare the new results to your previous ones using the plot script. Do you observe any differences? Why?

7 Playing with the setup

Here are a few suggestions to try if you have time left.

8 Unstable particles

Check the `(run)` section of the input file again. Add the setting `HARD_DECAYS 1;` which instructs Sherpa to automatically decay unstable particles. None of the particles produced in the hard scattering process is set unstable, so this option by itself will have no effect. Verify this by checking the screen output of Sherpa during runtime. Search for the `'List of Particle Data'`.

Now that you know about the problem, you can remedy the situation by including the following line in the `(run)` section

```
STABLE[25] 0;
```

8.1 Changing the functional form of scales

You may have noticed the following line in the runcard

```
CORE_SCALE VAR{sqr(125)};
```

It invokes Sherpa's interpreter to compute the renormalization and factorization scale for the $pp \to h$ production process as the mass of the Higgs boson (Note that all scales have dimension GeV^2, hence the scale is squared using the `sqr` function).

You can change this to a different value. For example, you could use the transverse mass of the Higgs boson:

```
CORE_SCALE VAR{MPerp2(p[2])};
```

Note that when you change the scale, the cross section will change and needs to be recomputed! You can instruct Sherpa not to pick up the old result and compute a new one by launching it with the commandline option `-r Result_NewScale/`. New cross section information will then be stored in the database `Results_NewScale.db`, while the old is still present in the directory `Results.db`.

How do the observables change with the new scale and why? What is the effect of ME+PS merging when using this scale?

8.2 Hadronization and underlying event

So far, multiple parton interactions, which provide a model for the underlying event, have not been simulated. Also, hadronization has not been included in order to speed up event generation. You can enable both by removing or commenting the line

```
MI_HANDLER None; FRAGMENTATION Off;
```

How do the predictions for the observables change and why?

8.3 Variation of the merging cut

ME+PS merging is based on phase-space slicing, and the slicing cut is called the merging scale. Is is an unphysical parameter, and no observable should depend sizeably on its precise value. Verify this by varying the merging cut by a factor of two up and down.

Note that the initial cross sections that Sherpa computes will be different for different values of the merging cut (Why?). You should therefore instruct Sherpa to use different result directories for each run in the test. The result directory is specified on the command line with option `-r`, for example

```
Sherpa -r MyResultDirectory/
```

8.4 Other hard scattering processes

Try to generate other hard scattering processes, like the production of a Drell-Yan muon pair (Hint: The PDG id for the muon is `13`.) You will have to insert an additional section into the runcard, which reads

```
(selector){
  Mass 13 -13 66 116;
}(selector);
```

Why is this section needed?

9 Outlook

You may ask yourself what the meaningful variations are to assess the uncertainty of your Sherpa predictions. We will try to answer this question in the next tutorial.

References

[1] T. Gleisberg et al. "Event generation with SHERPA 1.1." JHEP **02** (2009) 007.

[2] A. Buckley et al. "General-purpose event generators for LHC physics." Phys. Rept. **504** (2011) 145.

[3] https://sherpa.hepforge.org/doc/SHERPA-MC-2.1.1.html.

NTuples Tutorial

CTEQ School 2014

July 14, 2014

1 NTuples with Sherpa

The purpose of this tutorial is to generate Root NTuples which store NLO fixed-order events that can be reweighted in order to change scales and PDFs. The tutorial is composed of two parts:

- The production of NTuples
- The usage of existing NTuples in a simple analysis

Root NTuples are a convenient way to store the result of cumbersome fixed-order calculations [1].

When using NTuples, one needs to bear in mind that every calculation involving jets in the final state is exclusive in the sense that a lower cutoff on the jet transverse momenta must be imposed. It is therefore necessary to check whether the event sample stored in the NTuple is sufficiently inclusive before using it. Similar remarks apply when photons are present in the NLO calculation or when cuts on leptons have been applied at generation level to increase efficiency. Every NTuple should therefore be accompanied by an appropriate documentation.

In this tutorial we will generate NTuples for the process

$$pp \to h\, j$$

We identify parton-level jets using the anti-k_T algorithm with $R = 0.4$. We require the transverse momentum of these jets to be $p_{T,j} > 20$ GeV. No other cuts are applied at generation level.

We will use Sherpa [2] for the production and analysis of the NTuples. Sherpa is installed on the virtual machine in `/opt/hep`, and its documentation is found online [3]. Example setups are located in `/opt/hep/share/SHERPA-MC/Examples`.

2 NTuple production

Sherpa is steered using input files, which consist of several sections. A comprehensive list of all input parameters for Sherpa is given in the online manual [3]. Some of them have been explained in the Sherpa introduction.

Change to the directory `tutorials/intro/ntuples` and open the file `Run.B-like.dat` in an editor. Have a look at the section which is delimited by the tags `(run){` and `}(run)`. You will find the specification of the collider, i.e. its beam type and center-of-mass energy, as well as a couple of parameters, specific to NLO event generation, like the scale definition and the NTuple file name. The `(selector)` section defines the jet cut. The `(processes)` section contains three different process specifications, corresponding to Born, virtual and integrated subtraction contribution. Currently it is not possible to generate both real-emission and born-type NTuples at the same time, which is why you will find a separate input file, `Run.R-like.dat`, for the real-emission type contributions. This also allows to produce more events for a certain event type if the statistics are too low. Typically, more events are needed for the real-emission type contribution thank for the Born type.

Start Sherpa using the command line

```
Sherpa -f Run.B-like.dat
```

Sherpa will first create source code for its matrix-element calculations. This process will stop with a message instructing you to compile. Do so by running

```
./makelibs
```

Launch Sherpa again, using

```
Sherpa -f Run.B-like.dat
```

Sherpa will then compute the Born, virtual and integrated subtraction contribution to the NLO cross section and generate events. These events are analyzed and stored in a Root NTuple file called `NTuple_B-like.root`. We will use this NTuple later to compute an NLO uncertainty band.

The real-emission contribution, including subtraction terms, to the NLO cross section is computed using

```
Sherpa -f Run.R-like.dat
```

Events are generated, analyzed and stored in the Root NTuple file `NTuple_R-like.root`

The two analyses of events with Born-like and real-emission-like kinematics need to be merged using the following command

```
Combine_Analysis -m=add Analysis/HTp/sum Analysis/HTp/BVI Analysis/HTp/RS
```

The result can then be plotted and displayed

```
./plot_results.sh
gv plots.ps
```

3 Usage of NTuples in Sherpa

Next we will compute the NLO uncertainty band using Sherpa. To this end, we make use of the Root NTuples generated in the previous steps.

First we re-evaluate the events with the scale increased by a factor 2:

```
Sherpa -f Reweight.B-like.dat
Sherpa -f Reweight.R-like.dat
```

Then we re-evaluate the events with the scale decreased by a factor 2:

```
Sherpa -f Reweight.B-like.dat SCF:=0.25 -A Analysis/025HTp/BVI
Sherpa -f Reweight.R-like.dat SCF:=0.25 -A Analysis/025HTp/RS
```

The two contributions are combined using

```
Combine_Analysis -m=add Analysis/4HTp/sum Analysis/4HTp/BVI Analysis/4HTp/RS
Combine_Analysis -m=add Analysis/025HTp/sum Analysis/025HTp/BVI Analysis/025HTp/RS
```

Now we can display all three analyses in one plot

```
./plot_results.sh
```

References

[1] Z. Bern et el. "Ntuples for NLO Events at Hadron Colliders." Comput. Phys. Commun. **185** (2014) 1443.

[2] T. Gleisberg et al. "Event generation with SHERPA 1.1." JHEP **02** (2009) 007.

[3] https://sherpa.hepforge.org/doc/SHERPA-MC-2.1.1.html.