1 Pythia 8 tutorial

1.1 Introduction

This exercise corresponds to the Pythia 8 part of the more general MC tutorial given at the MC4BSM workshop at DESY 2013 [2][1]. It is for this reason focused on the particular task within this tutorial, however, should still serve as a good starting point to get familiar with the basics of how to use the Pythia 8 event generator. Much of these instructions were based on earlier Pythia 8 tutorials, which can be found on the homepage [5] and often include additional material than what is covered here.

Within this first exercise it is not possible to describe the physics models used in the program; for this we refer to the online manual [6], Pythia 8.1 brief introduction [1], to the full Pythia 6.4 physics description [3], and to all the further references found in them.

Finally, a good way to continue after the tutorial is often to chose a particular physics study your interest and then start to explore the different simulation and analysis aspects, using the different example programs together with the online manual, along the lines of the tutorial.

1.2 Installation and pre-workshop exercise

Pythia 8 is, by today's standards, a small package. It is completely self-contained, and is therefore easy to install for standalone usage, e.g. if you want to have it on your own laptop, or if you want to explore physics or debug code without any danger of destructive interference between different libraries.

Denoting a generic Pythia 8 version `pythia81xx` (at the time of writing `xx` = 62), here is how to install Pythia 8 on a Linux/Unix/MacOSX system as a standalone package.

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 source files are now ready and unpacked.
4. Move to this directory (`cd pythia81xx`) and do a `make`. This will take ∼3 minutes (computer-dependent). The Pythia 8 libraries are now compiled and ready for physics.
5. For test runs, `cd` to the `examples/` subdirectory. An `ls` reveals a list of programs, `mainNN`, with `NN` from `01` through `28` (and beyond). These example programs each illustrate an aspect of Pythia 8. For a list of what they do, see the "Sample Main Programs" page in the online manual (point 6 below).

[1]It is based on last years tutorial, see `arxiv.org/abs/arXiv:1209.0297` for description of phsyics process

In order to test that the program installed successfully, most examples are suitable, however, for this tutorial (using version 8.175) a good choice would be `main11.cc`, which reads in a test LHEF called `ttbar.lhe`.
To execute this test program, do

```
make main11
./main11.exe
```

The output is now just written to the terminal, `stdout`, and if everything worked one should see (specific for main11.cc) a text based histogram, showing the a charge particle multiplicity, at the end of the program output. To save the output to a file instead, do `./main11.exe > main11.out`, after which you can study the test output at leisure by opening `main11.out`. See Appendix A for a brief explanation of the event record.

6. If you use a web browser to open the file

   ```
   pythia81xx/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.

1.3 On-site exercise

The task at hand for the Pythia 8 part of the tutorial is to read in the events, provided in LHEF format from the earlier steps, and continue the event simulation following the hard scatter generation. Given that the LHEF includes both the information for initialization and of the actual events, this is very straight forward. Copy first the LHEF `events_with_decay.lhe.gz` (can be downloaded via
`wget http://home.thep.lu.se/~jesper/events_with_decay.lhe.gz`) into the `examples` directory and unpack it using,

```
gunzip events_with_decay.lhe.gz
```

The LHEF should now be readable just like any ASCII file.

The BSM model as well as the hard process is described in detail in the earlier steps of this tutorial, but as a short reminder the events consists of, $pp \to U\bar{U}$, production at the LHC with $E_{cm} = 8$ TeV. The U fermions have a mass of 500 GeV and the two following decay scenarios are possible:

$$U \to u\ \Phi_1; \tag{1.1}$$

$$U \to u\ \Phi_2 \to u\ e\ E \to u\ e\ e\ \Phi_1. \tag{1.2}$$

The second step is to create our own main program. Open a new file `mymain.cc` in the `examples` subdirectory with a text editor, e.g. Emacs. Then type the following lines (here with explanatory comments added):

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

```
int main() {             // Begin main program.

  // Set up generation.
  Pythia pythia;         // Declare Pythia object
  pythia.readString("Beams:frameType = 4"); // Beam info in LHEF.
  pythia.readString("Beams:LHEF = events_with_decay.lhe");
  pythia.init(); // Initialize; incoming pp beams is default.

  // Generate event(s).
  for (int iEvent = 0; iEvent < 1; ++iEvent) {
    if (!pythia.next()) {
      if (pythia.info.atEndOfFile()) break; // Exit at enf of LHEF.
      continue; // Skip event in case of problem.
    }
  }

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

This will use one event from the LHEF and pass it through the remaining Pythia 8 simulation. Go through the lines in the program and try to understand them by consulting the online manual [6].

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

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

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

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

Now it should work as before with the other examples:

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

whereafter you can study `mymain.out`, especially the example of a complete event record (preceded by initialization information and by kinematical-variable listing for the same event). For this reason Appendix A contains a brief overview of the information stored in the event record.

At this point you have in principle achieved your goal, the first event in the LHEF have been fully simulated by Pythia 8, using the default values of all settings. On the other hand, and despite only having one event, we only have access to the information of the individual particles in the event record, so interesting for technical validation but not much of real physics interest yet.

1.4 A simple jet analysis

We will now gradually expand the skeleton `mymain` program from above, in order to make a simple analysis, including jet reconstruction. The common standard, used in the experimental communities, is to produce fully simulated MC events in the `HepMc` format, either for analysis or further detector simulation. However, in order to keep the software infrastructure at a minimum we will in this part only use the analysis functionality already available within PYTHIA 8. For further information regarding how to produce `HepMc`, the reader is referred to the online manual [6] together with other tutorials on the home page [5]. The final version of `mymain.cc`, with all steps included, is kept in Appendix B. Hence one can look there in case the instructions are unclear, however, that should of course be considered as the last resort, if getting completely stuck.

The BSM events generated in this tutorial will always include jets, from the U decay, and missing Φ_1 particles. They will some times also contain electrons, depending on the U decay channel. For this reason we start with reconstructing jets from the final state particles in the event. For this we use the PYTHIA 8 `SlowJet` program (found in the manual under *Event Analysis*), which is a simpler version of the commonly used `FastJet` jet finder. The `SlowJet` program supports both the kT, anti-kT, and Cambridge/Aachen algorithms, where we will use the anti-kT which is a common choice within the LHC experiments. For further information about the jet reconstruction algorithms we recommend the online manual together with its references to the `FastJet` program.

Insert the following line before the event loop, in order to create a `SlowJet` object,

```
SlowJet sJets(-1, 0.4, 50., 5., 1, 2);
```

The first argument specifies the anti-kT algorithm, the second is the related R measure which roughly corresponds to the radius of the jet cone in (y,ϕ). The next two corresponds to, minimum p_T and maximum $|\eta|$, acceptance cuts usually implied by the experiment of interest. The last two arguments specifies respectively to use all final state particles and to use their actual mass in the reconstruction. The selection of which particles to include is normally done more carefully, however, in a trade–off between illustrating the physics without spending too much time on technical details and infrastructure, we chose a rather unsophisticated approach here. Just after this line, you should also create the following two histograms,

```
Hist nJ( "Nr Jets", 10, -0.5, 9.5);
Hist j1pT( "Leading Jet pT", 100, 0., 1000.);
```

which corresponds to our final analysis results.

The next step is to reconstruct jets from the final state particles in each event according to the jet algorithm we just defined. After that we also want to fill our histograms, once per event, and at the end of the program print them to our output. To do this, include the following lines inside the event loop, but at the end so that `pythia.next()` has been executed,

```
// Jet analysis.
if (sJets.analyze(pythia.event)) {
  nJ.fill(sJets.sizeJet());
```

```
      if(sJets.sizeJet() > 0)
         j1pT.fill(sJets.pT(0));
    }
```

The `pythia` member `.event` corresponds to the event record and you can try to find out the meaning of these lines by using the online manual. To write the histograms to the output include the following line at the very end, just before the `return` statement,

```
    std::cout « nJ « j1pT;
```

Now increase the number of events in the event loop, *e.g.* to 1000, compile and run the program. At the end of the output one should now see the number of jets and leading jet p_T distributions, in traditional text based histograms.

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.

What we did so far is clearly not what we want to do, since first we include the Φ_1 which should be invisible and second the electrons produced in the decay chains, potentially also with high p_T, are included.

For the purpose of this tutorial we use a rather ugly trick in order to eliminate particles with respect to the jet reconstruction. We simply loop through the event record and when we find a Φ_1 (`id == 9000006`) we set its status to a negative value, since the jet reconstruction only considers final state particles, *i.e.* with positive status. In order to do this, insert the following lines, just before the jet analysis part,

```
  for (int iPart = 0; iPart < pythia.event.size(); ++iPart) {
     if (pythia.event[iPart].idAbs() == 9000006) {
       int stat = pythia.event[iPart].status();
       if (stat > 0) pythia.event[iPart].status(-stat);
     }
  }
```

These lines hence illustrate both how to loop over the particles in the event record and how to access their properties, in this case the id and status codes. Now compile and run the program again. The mean number of jets should now have been reduced by approximately two (from 4.8 to 3.1), due to the two Φ_1 in the events. The leading p_T jet should now dominantly originate from the quarks in the U decay and the mean p_T therefore is around half of the U mass, 500 GeV.

We now leave it as an exercise to also eliminate any final state electrons in the event and see how the distributions are affected. Another easy thing to investigate would be to see how the number of reconstructed jets varies with the minimum p_T requirement specified for the jet finder.

1.5 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 plaintext 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 1.2, 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.

1.5.1 Identity codes

A complete specification of the PDG codes is found in the Review of Particle Physics [7]. 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_L^0	113	ρ^0	3112	Σ^-
5	b	15	τ^-	25	H^0	421	D^0	310	K_S^0	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_L^0 and K_S^0 being exceptions), and with a set of further rules to make the codes unambiguous.

1.5.2 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 multiparton 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.

1.5.3 History information

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`.

1.5.4 Color flow information

The colour flow information is based on the Les Houches Accord convention [8]. 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 relabeled 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.

1.6 The jet analysis program

This is the final `mymain.cc` at the end of the tutorial,

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

int main() { // Begin main program.

  // Set up generation.
  Pythia pythia; // Declare Pythia object
  pythia.readString("Beams:frameType = 4"); // Beam info in LHEF.
  pythia.readString("Beams:LHEF = events_with_decay.lhe");
  pythia.init(); // Initialize; incoming pp beams is default.

  SlowJet sJets(-1, 0.4, 50., 5., 1, 2);
  Hist nJ( "Nr Jets", 10, -0.5, 9.5);
  Hist j1pT( "Leading Jet pT", 100, 0., 1000.);

  // Generate event(s).
  for (int iEvent = 0; iEvent < 1000; ++iEvent) {
    if (!pythia.next()) {
      if (pythia.info.atEndOfFile()) break; // Exit at enf of LHEF.
      continue; // Skip event in case of problem.
    }
    // Loop through event record.
    for (int iPart = 0; iPart < pythia.event.size(); ++iPart) {
      if (pythia.event[iPart].idAbs() == 9000006) {
        int stat = pythia.event[iPart].status();
        if (stat > 0) pythia.event[iPart].status(-stat);
      }
      if (pythia.event[iPart].idAbs() == 11) {
        int stat = pythia.event[iPart].status();
        if (stat > 0) pythia.event[iPart].status(-stat);
      }
    }

    // Jet analysis.
    if (sJets.analyze(pythia.event)) {
      nJ.fill(sJets.sizeJet());
      if (sJets.sizeJet() > 0)
         j1pT.fill(sJets.pT(0));
    }


  }
  pythia.stat(); // Print run statistics.

  std::cout << nJ << j1pT;

  return 0;
}
```

References

[1] T. Sjostrand, S. Mrenna and P. Z. Skands, "A Brief Introduction to PYTHIA 8.1," Comput. Phys. Commun. **178**, 852 (2008) [arXiv:0710.3820 [hep-ph]].

[2] http://www.phys.ufl.edu/∼matchev/mc4bsm6/

[3] T. Sjostrand, S. Mrenna and P. Z. Skands, "PYTHIA 6.4 Physics and Manual," JHEP **0605**, 026 (2006) [hep-ph/0603175].

[4] http://physics.ucdavis.edu/∼conway/research/software/pgs/pgs4-general.htm.

[5] http://www.thep.lu.se/∼torbjorn/Pythia.html

[6] The Pythia 8 online manual is both available on the home page [5] (for the current version) as well as distributed together with the source code, *e.g.* open the file `pythia81xx/htmldoc/Welcome.html` in a browser, as discussed in section 2.

[7] C. Amsler *et al.* [Particle Data Group Collaboration], "Review of Particle Physics," Phys. Lett. B **667**, 1 (2008).

[8] 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]