For tutorials
at Summer Schools
or self-study
December 2016

PYTHIA + DIRE Worksheet

Stefan Prestel
Theoretical Physics, Fermilab

(Note: This worksheet heavily relies on the PYTHIA 8.2
Worksheet by T. Sjöstrand, SP and P. Skands)

1 Introduction

The PYTHIA 8.2 program is a standard tool for the generation of high-energy collisions (specifically, it focuses on centre-of-mass energies greater than about 10 GeV), comprising a coherent set of physics models for the evolution from a few-body high-energy ("hard") scattering process to a complex multihadronic final state. The particles are produced in vacuum. Simulation of the interaction of the produced particles with detector material is not included in PYTHIA but can, if needed, be done by interfacing to external detector-simulation codes.

The PYTHIA 8.2 code package contains a library of hard interactions and models for initial- and final-state parton showers, multiple parton-parton interactions, beam remnants, string fragmentation and particle decays. It also has a set of utilities and interfaces to external programs.

The parton showers in PYTHIA 8.2 encode an evolution of the hard, high-energy, core scattering system to smaller energies. PYTHIA 8.2 offers three different shower implementations. The "default" model is built on collinear splitting functions and a dipole-like generation of real emission phase space[1], supplemented with *matrix-element corrections* for important processes. The VINCIA shower plugin greatly extends the matrix-element correction formalism (through multiple emissions), while also offering a better model of color coherence. The DIRE shower plugin – which we will use in this tutorial – attempts to connect the shower evolution more closely to factorization, by carefully handling collinear and soft phase space regions. As an important consequence, DIRE enables PYTHIA 8.2 to handle Deep Inelastic Scattering (DIS) events[2].

[1]The evolution of an on-shell n-particle system to an on-shell $(n+1)$-particle system by parton splitting is achieved by distributing the "recoil" of a splitting to a spectator momentum.

[2]Note that the DIS model in PYTHIA 8.2 is not yet complete, as photo-production and diffractive modeling are still missing. Thus the reach in Q^2 and W^2 is restricted.

The objective of this exercise is to teach you the basics of how to use the PYTHIA 8.2 event generator in conjunction with the DIRE parton shower evolution plugin to study various physics aspects. 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 8.2 introduction [1], to the full PYTHIA 6.4 physics description [2], and to all the further references found in them. The physics of DIRE is described in [3].

PYTHIA 8 and DIRE are, by today's standards, small packages. PYTHIA 8 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. Section 2 describes the installation procedure, which is what we will need for this introductory session. It does presuppose a working Unix-style environment with C++ compilers and the like; check Appendix D if in doubt.

When you use PYTHIA or DIRE, 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. Section 3 gives you a simple step-by-step recipe how to write a minimal main program, that can then gradually be expanded in different directions, e.g. as in Section 4.

In Section 5 you will learn how to install the DIRE shower plugin. As a first application, in Section 6, you will use DIRE in the evolution of Drell-Yan scattering events. Section 7 is then devoted to studying DIS, culminating in the assessment of renormalization scale uncertainties.

The final section provides suggestions for optional further studies, which can help further familiarize yourself with the PYTHIA. These suggestions and can be addressed in any order.

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 HEPMC library location is installation-dependent it is not possible to give a fool-proof linking procedure, but some hints are provided for the interested in Appendix C. Further main programs included with the PYTHIA code provide examples of linking, e.g., to ALPGEN, MADGRAPH, POWHEG, FASTJET, ROOT, and the Les Houches Accords LHEF, LHAPDF and SLHA.

Appendix A contains a brief summary of the event-record structure, and Appendix B some notes on simple histogramming and jet finding. Appendices C and D have already been mentioned.

2 Pythia installation

Denoting a generic PYTHIA 8 version `pythia82xx` (at the time of writing `xx` = 19), here is how to install PYTHIA 8 on a Linux/Unix/MacOSX system as a standalone package (assuming you have standard Unix-family tools installed, see Appendix D).

1. In a browser, go to
   ```
   http://home.thep.lu.se/Pythia
   ```
2. Download the (current) program package
   ```
   pythia82xx.tgz
   ```
   to a directory of your choice (e.g. by right-clicking on the link).
3. In a terminal window, `cd` to where `pythia82xx.tgz` was downloaded, and type
   ```
   tar xvfz pythia82xx.tgz
   ```
   This will create a new (sub)directory `pythia82xx` where all the Pythia source files are now ready and unpacked.
4. Move to this directory (`cd pythia82xx`) and do a `make`. This will take ~3 minutes (computer-dependent). The Pythia 8 library is now compiled and ready for physics.
5. For test runs, `cd` to the `examples/` subdirectory. An `ls` reveals a list of programs. 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). To execute one of the test programs, do
   ```
   make mainNN
   ./mainNN
   ```
   The output is now just written to the terminal, `stdout`. To save the output to a file instead, do `./mainNN > outNN`, after which you can study the test output at leisure by opening `outNN`. See Appendix A for an explanation of the event record that is listed in several of the runs.
6. If you use a web browser to open the file
   ```
   pythia82xx/share/Pythia8/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.

3 A "Hello World" program

We will now generate a single $\mathrm{gg} \to \mathrm{t}\bar{\mathrm{t}}$ event at the LHC, using Pythia standalone.

Open a new file `mymain01.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 "Pythia8/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("Top:gg2ttbar = on"); // Switch on process.
  pythia.readString("Beams:eCM = 8000."); // 8 TeV CM energy.
  pythia.init(); // Initialize; incoming pp beams is default.

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

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

The `examples/Makefile` has been set up to compile all `mymainNN.cc`, `NN` $= 01 - 99$, and link them to the `lib/libpythia8.a` library, just like the `mainNN.cc` ones. Therefore you can compile and run `mymain01` as before:

```
make mymain01
./mymain01 > myout01
```

If you want to pick another name, or if you need to link to more libraries, you have to edit `examples/Makefile` appropriately.

Thereafter you can study `myout01`, especially the example of a complete event record (preceded by initialization information, and by kinematical-variable and hard-process listing for the same event). 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. To exemplify, consider a top quark produced in the hard interaction, initially with positive status code. When later a shower branching $\mathrm{t} \to \mathrm{tg}$ occurs, the new t and g are added at the bottom of the then-current event record, but the old t 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 the top. After the shower, when the final top decays, $\mathrm{t} \to \mathrm{bW}^+$, also that copy receives a negative status code. When you understand the basic principles, see if you can find several copies of the top quarks, 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.

4 A first realistic analysis

We will now gradually expand the skeleton `mymain01` program from above, towards what would be needed for a more realistic analysis setup.

- First, let us switch to generate Drell-Yan lepton pair events. For this, replace the line
  ```
  pythia.readString("Top:gg2ttbar = on");
  ```
  by
  ```
  pythia.readString("WeakSingleBoson:ffbar2gmZ = on");
  ```

- Now we wish to generate more than one event. To do this, introduce a loop around `pythia.next()`, so the code now reads
  ```
  for (int iEvent = 0; iEvent < 5; ++iEvent) {
    pythia.next();
  }
  ```
  Hereafter, we will call this the *event loop*. The program will now generate 5 events; each call to `pythia.next()` resets the event record and fills it with a new event. To list more of the events, you also need to add
  ```
  pythia.readString("Next:numberShowEvent = 5");
  ```
  along with the other `pythia.readString` commands.
- To obtain statistics on the number of events generated of the different kinds, and the estimated cross sections, add a
  ```
  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 (except for a few special cases). The above-mentioned `pythia.stat()` will then tell you how many times each problem was encountered over the entire run.
- Studying the event 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.1). The `cout` statement, therefore, will give a list of the PDG code of every particle in the event record.

- As mentioned above, the event listing contains all partons and particles, traced through a number of intermediate steps. Eventually, the intermediate resonance will decay ($Z/\gamma^* \to f\bar{f}$), and by implication it is the last Z/γ^* copy in the event record that defines the definitive Z/γ^* production kinematics, just before the decay. You can obtain the location of this final particle e.g. by inserting a line just before the particle loop
  ```
  int iZ = 0;
  ```
  and a line inside the particle loop
  ```
  if (pythia.event[i].id() == 23) iZ = i;
  ```
  The value of `iZ` will be set every time a Z/γ^* is found in the event record. Note that PYTHIA 8 will *book-keep* the intermediate particle as Z-boson. However, all contributions from Z, γ^*, and interferences are included in the calculation. When the particle loop is complete, `iZ` will point to the final "Z" in the event record (which can be accessed as `pythia.event[iZ]`).
- 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[iZ].pT()`, and pseudorapidity, `pythia.event[iZ].eta()`. Use these methods to print out the values for the final Z found above.
- We now want to generate more events, say 1000, to view the shape of these distributions. Inside PYTHIA is a very simple histogramming class, see Appendix B.1, that can be used for rapid check/debug purposes. To book the histograms, insert before the event loop
  ```
  Hist pT("Drell-Yan transverse momentum", 30, 0., 30.);
  Hist eta("Drell-Yan pseudorapidity", 100, -5., 5.);
  ```
  where the last three arguments are the number of bins, the lower edge and the upper edge of the histogram, respectively. Now we want to fill the histograms in each event, so before the end of the event loop insert
  ```
  pT.fill( pythia.event[iZ].pT() );
  eta.fill( pythia.event[iZ].eta() );
  ```
  Finally, to write out the histograms, after the event loop we need a line like
  ```
  cout << pT << eta;
  ```
  Do you understand why the η distribution looks the way it does? Propose and study a related but alternative measure and compare.
- As a final standalone exercise, consider plotting the charged multiplicity of events. You then need to have a counter set to zero for each new event. Inside the particle loop this counter should be incremented whenever the particle `isCharged()` and `isFinal()`. For the histogram, note that it can be treacherous to have bin limits at integers, where roundoff errors decide whichever way they go. In this particular case only even numbers are possible, so 100 bins from -1 to 399 would still be acceptable.

5 Dire installation

Now that you have installed Pythia 8, and already familiarized yourself with the concept of Pythia programs, you should move on to install the Dire shower plugin. Dire acts as a replacement and extension of the Pythia parton shower evolution, allowing to use Pythia 8 also for deep inelastic scattering scenarios. Dire can be installed by following the steps below.

1. In a browser, go to
   `https://direforpythia.hepforge.org`
2. From the Downloads tab, download the (current) program package
   `DIRE-current.tar.gz`
   to the Pythia 8 `examples` directory, and in a terminal window `cd` into that directory. Execute
   `tar xvfz DIRE-current.tar.gz`
3. Move to the resulting directory (`cd DIRE-1.xxx`, at the time of writing `xxx` = 500)). An `ls` reveals the Dire source code as well as a list of example programs (`dire00.cc` - `dire05.cc`). Contrary to the `Pythia` 8 examples, the `Dire` programs are designed to highlight different event generator inputs and outputs. The directory further contains the example "input files" `lep.cmnd`, `dis.cmnd` and `lhc.cmnd` with lists of input settings. You can look at and modify these files to test different Pythia+Dire setups.
4. For test runs, do
   ```
   make dire05
   ./dire05 myinput.cmnd
   ```
   where `myinput.cmnd` can be any of the examples, or your personal input file.

6 Modeling the Drell-Yan transverse momentum

Dire inherits a large fraction of Pythia 8's functionality. This includes for example the facilities to perform simple analysis tasks. The main program `dire05.cc` is intended as a convenient option to implement and organize your event analyses. To aid organization, the program code includes a simplistic analysis class, which can be found under `plugins/analyses/AnalysisDummy.h`. This file encodes a simple histogramming of the charged hadron multiplicity. Before moving to a more serious task, let us familiarize ourselves with the structure of the analysis, with running the code, and plotting the output. For this, follow the steps

```
me@host:$ make dire05
me@host:$ ./dire05 lhc.cmnd
me@host:$ gnuplot  -e "plot 'nch_vs_pt.dat' using 1:2 w boxes; pause -1"
```

The last command will open a display with the number of charged hadrons plotted as a function of hadron transverse momentum. Note that by using the input file `lhc.cmnd`, you have selected to generate Drell-Yan lepton pair production events.

Now, as a warm-up, implement the Drell-Yan $p_\perp$ analysis of section 4 in this framework[3]. After obtaining first results, it is time to study how the modeling of the $p_\perp$ is influenced by choices in the event generation. Remember that the Drell-Yan $p_\perp$ is mainly generated by *a*) Recoil of the Drell-Yan pair against emissions from initial-state lines in the perturbative parton shower evolution, and *b*) non-vanishing non-perturbative $p_\perp$ of incoming partons ("primordial k_T"). In the following, try to investigate the correlations between these mechanisms. To do this, you can change the input settings in `lhc.cmnd`[4]. Can you (or your study group) answer the following questions?

1. What is the influence of the reference value $\alpha_s(M_Z)$ on the $p_\perp$ spectrum? Can you explain how the $p_\perp$ distribution changes?
   Hint: Investigate the dependence on the parameter `SpaceShower:alphaSvalue`.
2. What is the dependence on the (collinear) parton distributions?
   Hint: Investigate the dependence on the PDF set `PDF:pSet`.
3. What is the effect of including/changing the primordial k_T?
   Hint: Investigate the dependence on `BeamRemnants:primordialKT` and `BeamRemnants:primordialKThard`.
4. The transition between perturbative and non-perturbative modeling is governed by a cut-off on initial state radiation. How sensitive is the $p_\perp$ spectrum on where this transition takes place?
   Hint Investigate the dependence on the parameter `SpaceShower:pTmin`

When performing these studies, it is good to keep the PYTHIA 8 online manual `pythia82xx/share/Pythia8/htmldoc/Welcome.html` at hand, in particular the sections "Spacelike Showers", "PDF Selection" and "Beam Remnants".

7 Modeling Deep Inelastic Scattering

In this section, we turn our attention to Deep Inelastic Lepton-Hadron Scattering (DIS). The description of such scatterings has become available by introducing the DIRE shower plugin. You can produce DIS events by e.g. running

```
me@host:$ ./dire05 dis.cmnd
```

The input file is set up to produce HERA II-like events. No specific cuts on the virtuality of the exchanged neutral boson or on y are applied. Below, we will include such cuts in a realistic DIS analysis. We will implement this analysis in `plugins/analyses/DIS.h`[5].

[3]It might be convenient to leave the template `plugins/analyses/AnalysisDummy.h` untouched, and instead create a copy (e.g. called `plugins/analyses/DYpT.h`) and include this file in the `dire05.cc` code (i.e. replace `#include "plugins/analyses/AnalysisDummy.h"` by
`#include "plugins/analyses/DYpT.h"`

[4]Again, it might be useful to keep an unchanged copy of this file for later.

[5]Copy `plugins/analyses/AnalysisDummy.h` to a file called `plugins/analyses/DIS.h`) and include this file in the `dire05.cc` code (i.e. replace `#include "plugins/analyses/AnalysisDummy.h"` by
`#include "plugins/analyses/DIS.h"`

7.1 Finding the proton, incoming lepton and scattered lepton

Some observables used to define DIS are

$$\begin{aligned} Q^2 &= -(p_{\ell\ in} - p_{\ell\ out})^2 \\ W^2 &= (p_{h\ in} + p_{\ell\ in} - p_{\ell\ out})^2 \\ y &= \frac{p_{h\ in} \cdot (p_{\ell\ in} - p_{\ell\ out})}{p_{h\ in} \cdot p_{\ell\ in}} \end{aligned}$$

Your first task will be to isolate these four-vectors from the PYTHIA 8 event record. First print an event record by adding

```
e.list();
```

to the function `MyAnalysis::fill` of `DIS.h`. You will need to remove and recompile the *executable* `dire05` after each change of `DIS.h`. Use the information in Appendix A to isolate the incoming hadron, incoming lepton and the scattered lepton. Add the corresponding code to the `fill` function, e.g.

```
// Declare particle postitions.
int iProton(0), iInElectron(0), iOutLepton(0);
for ( int i=0; i < e.size(); ++i) {
  // Code to find the position of the incoming proton in the event.
  // Code to find the position of the incoming electron in the event.
  // Code to find the position of the outgoing electron in the event.
}
```

You can then obtain the four-momenta of the particles by e.g. using `e[iProton].p()`. The product of two four-vectors can be obtained by using `e[i].p()*e[j].p()` and a convenient way to obtain an invariant mass is `(e[i].p() + e[j].p()).m2Calc()`. Using these hints, impose the cuts 5 GeV$^2 < Q^2 <$ 100 GeV^2 and $0.01 < y < 0.9$.

7.2 DIS analysis

As a first step in the analysis, investigate how the Q^2 and y cuts change the hadron multiplicity. For this, you can add new histograms to your analysis, which you only fill once certain conditions (cuts) are met.

We begin by adding a single histogram

```
histograms.insert(make_pair("nch_vs_pt_highQ2",
  Hist("nch_vs_pt_highQ2",100,0.0,100.0)));
```

to the `MyAnalysis::init()` function, which we only want to fill if $Q^2 >$ 50 GeV^2. Assuming that the Q^2 variable in your code is called `Q2`, update your particle loop:

```
for ( int i=0; i < e.size(); ++i )
  if ( e[i].isFinal() && e[i].isCharged() && e[i].isHadron() ) {
    double pt = e[i].pT();
    // Fill histogram.
    histograms["nch_vs_pt"].fill ( pt, w );
    if (Q2 > 50 ) histograms["nch_vs_pt_highQ2"].fill ( pt, w );
  }
```

At the end of the event generation, the corresponding histogram will directly be printed to a file called `nch_vs_pt_highQ2.dat`. You can plot both multiplicity histograms in one figure with a bit of `gnuplot` magic:

```
me@host:$ gnuplot  -e "plot 'nch_vs_pt.dat' using 1:2 w boxes title 'regular Q2',\
                          'nch_vs_pt_highQ2.dat' using 1:2 w boxes title 'high Q2';\
                           pause -1"
```

Note that this command should occupy a single terminal line, without line breaks or "\" symbol. With this knowledge, you can try to extend your DIS analysis. Be creative! Some suggestions are listed below.

1. How does the rapidity of (charged) hadrons change with cuts?
2. What is the dependence on $\alpha_s(M_Z)$ and on the parton distributions?
3. How do initial-state and final-state radiation change the distributions?
   Hint: Switch on/off `PartonLevel:ISR` and `PartonLevel:FSR`
4. Construct jets from the final state particles, and histogram their transverse momenta. After how many emissions does the $p_\perp$ of the hardest jet remain (approximately) fixed? Why?
   Hint: The `dire00.cc` code uses a jet algorithm. The number of emissions in Dire can be limited by using `DireSpace:nFinalMax` and `DireTimes:nFinalMax`.
5. Force all quarks except the top quark to be massless. How does the hadron multiplicity change? How does the multiplicity of π, K, D change?
   Hint: Particle masses can be set in the input settings file my using the syntax `pdgid:m0 = 0.0`, e.g. `4:m0 = 0.0` will yield a vanishing charm quark mass. Note that you might also have to change the switch `ShowerPDF:usePDFmasses`.

In the previous sections, you have become acquainted with some of the uncertainties of the event generator. However, so far, we have not considered approaching this in a systematic fashion. For the perturbative evolution in Dire, this is (at least partially) possible. Thus, as a final exercise, let us use the in-built parton shower variations to assess the renormalization scale uncertainty. A detailed example can be found in `dire03.cc`, and the corresponding input settings are part of `dis.cmnd`. Following this example, update your `DIS.h` analysis to handle these variations. (*Hint:* Call `fill()` multiple times, to produce one histogram for each variation/weight.) How large are the variations in your observables? Which observables are more sensitive to the variations, and why?

8 Further studies

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

- One popular misconception is that the energy and momentum of a B meson has to be smaller than that of its mother b quark, and similarly for charm. The fallacy is twofold. Firstly, if the b quark is surrounded by nearby color-connected gluons, the B meson may also pick up some of the momentum of these gluons. Secondly, the concept of smaller momentum is not Lorentz-frame-independent: if the other end of the b color force field is a parton with a higher momentum (such as a beam remnant) the “drag” of the hadronization process may imply an acceleration in the lab frame (but a deceleration in the beam rest frame).
  To study this, simulate b production, e.g. the process `HardQCD:gg2bbbar`. Identify B/B* mesons that come directly from the hadronization, for simplicity those with status code -83 or -84. In the former case the mother b quark is in the `mother1()` position, in the latter in `mother2()` (study a few event listings to see how it works). Plot the ratio of B to b energy to see what it looks like.
- One of the characteristics of multiparton-interactions (MPI) models is that they lead to strong long-range correlations, as observed in data. That is, if many hadrons are produced in one rapidity range of an event, then most likely this is an event where many MPI’s occurred (and the impact parameter between the two colliding protons was small), and then one may expect a larger activity also at other rapidities.
  To study this, select two symmetrically located, one unit wide bins in rapidity (or pseudorapidity), with a variable central separation Δy: $[\Delta y/2, \Delta y/2 + 1]$ and $[-\Delta y/2 - 1, -\Delta y/2]$. For each event you may find n_F and n_B, the charged multiplicity in the “forward” and “backward” rapidity bins. Suitable averages over a sample of events then gives the forward–backward correlation coefficient

$$\rho_{FB}(\Delta y) = \frac{\langle n_F \, n_B \rangle - \langle n_F \rangle \langle n_B \rangle}{\sqrt{(\langle n_F^2 \rangle - \langle n_F \rangle^2)(\langle n_B^2 \rangle - \langle n_B \rangle^2)}} = \frac{\langle n_F \, n_B \rangle - \langle n_F \rangle^2}{\langle n_F^2 \rangle - \langle n_F \rangle^2} \, ,$$

  where the last equality holds for symmetric distributions such as in pp and $\overline{\mathrm{p}}$p.
  Compare how $\rho_{FB}(\Delta y)$ changes for increasing $\Delta y = 0, 1, 2, 3, \ldots$, with and without MPI switched on (`PartonLevel:MPI = on/off`) for minimum-bias events (`SoftQCD:minBias = on`).

Note that the Pythia homepage contains two further tutorials, in addition to older editions of the current one. These share some of the introductory material, but then put the emphasis on two specific areas:

- a merging tutorial, showing the step-by-step construction of a relevant main program, and more details on possible merging approaches than found in Section 6 of the current manual; and
- a BSM tutorial, describing how you can input events from Beyond-the-Standard-model scenarios into Pythia.

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 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 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 [7]. An online listing is available from

`http://pdg.lbl.gov/2014/reviews/rpp2014-rev-monte-carlo-numbering.pdf`

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_L^0 and K_S^0 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 (see the online manual for the meaning of each specific code):

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.

A.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 it in the event listing.

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

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 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.

B Some facilities

The Pythia package contains some facilities that are not part of the core generation mission, but are useful for standalone running, notably at summer schools. Here we give some brief info on histograms and jet finding.

B.1 Histograms

For real-life applications you may want to use sophisticated histogramming programs like ROOT, which however take much time to install and learn. Within the time at our disposal, we therefore stick with the very primitive `Hist` class. Here is a simple overview of what is involved.

As a first step you need to declare a histogram, with name, title, number of bins and x range (from, to), like

```
Hist pTH("Higgs transverse momentum", 100, 0., 200.);
```

Once declared, its contents can be added by repeated calls to fill,

```
pTH.fill( 22.7, 1.);
```

where the first argument is the x value and the second the weight. Since the weight defaults to 1 the last argument could have been omitted in this case.

A set of overloaded operators have been defined, so that histograms can be added, subtracted, divided or multiplied by each other. Then the contents are modified accordingly bin by bin. Thus the relative deviation between two histograms `data` and `theory` can be found as

```
diff = (data - theory) / (data + theory);
```

assuming that `diff`, `data` and `theory` have been booked with the same number of bins and x range.

Also overloaded operations with double real numbers are available. Again these four operations are defined bin by bin, i.e. the corresponding amount is added to, subtracted from, multiplied by or divided by each bin. The double number can come before or after the histograms, with obvious results. Thus the inverse of a histogram result is given by `1./result`. The two kind of operations can be combined, e.g.

```
     allpT = ZpT + 2.  * WpT
```

A histogram can be printed by making use of the overloaded `<<` operator, e.g.

```
     cout << ZpT;
```

The printout format is inspired by the old HBOOK one. To understand how to read it, consider the simplified example

```
          3.50*10^ 2  9
          3.00*10^ 2  X   7
          2.50*10^ 2  X  1X
          2.00*10^ 2  X6 XX
          1.50*10^ 2  XX5XX
          1.00*10^ 2  XXXXX
          0.50*10^ 2  XXXXX

            Contents
              *10^ 2  31122
              *10^ 1  47208
              *10^ 0  79373

            Low edge  --
              *10^ 1  10001
              *10^ 0  05050
```

The key feature is that the `Contents` and `Low edge` have to be read vertically. For instance, the first bin has the contents $3 * 10^2 + 4 * 10^1 + 7 * 10^0 = 347$. Correspondingly, the other bins have contents 179, 123, 207 and 283. The first bin stretches from $-(1 * 10^1 + 0 * 10^0) = -10$ to the beginning of the second bin, at $-(0 * 10^1 + 5 * 10^0) = -5$.

The visual representation above the contents give a simple impression of the shape. An `X` means that the contents are filled up to this level, a digit in the topmost row the fraction to which the last level is filled. So the `9` of the first column indicates this bin is filled 9/10 of the way from $3.00 * 10^2 = 300$ to $3.50 * 10^2 = 350$, i.e. somewhere close to 345, or more precisely in the range 342.5 to 347.5.

The printout also provides some other information, such as the number of entries, i.e. how many times the histogram has been filled, the total weight inside the histogram, the total weight in underflow and overflow, and the mean value and root-mean-square width (disregarding underflow and overflow). The mean and width assumes that all the contents is in the middle of the respective bin. This is especially relevant when you plot a integer quantity, such as a multiplicity. Then it makes sense to book with limits that are half-integers, e.g.

```
     Hist multMPI( "number of multiparton interactions", 20, -0.5, 19.5);
```

so that the bins are centered at 0, 1, 2, ..., respectively. This also avoids ambiguities which bin gets to be filled if entries are exactly at the border between two bins. Also note that the `fill( xValue)` method automatically performs a cast to double precision where necessary, i.e. `xValue` can be an integer.

Histogram values can also be output to a file

```
pTH.table("filename");
```

which produces a two-column table, where the first column gives the center of each bin and the second one the corresponding bin content. This may be used for plotting e.g. with Gnuplot.

B.2 Jet finding

The `SlowJet` class offer jet finding by the $k_{\perp}$, Cambridge/Aachen and anti-$k_{\perp}$ algorithms. By default it is now a front end to the FJcore subset, extracted from the FastJet package [8] and distributed as part of the PYTHIA package, and is therefore no longer slow. It is good enough for basic jet studies, but does not allow for jet pruning or other more sophisticated applications. (An interface to the full FastJet package is available for such uses.)

You set up `SlowJet` initially with

```
SlowJet slowJet( pow, radius, pTjetMin, etaMax);
```

where `pow = -1` for anti-$k_{\perp}$ (recommended), `pow = 0` for Cambridge/Aachen, `pow = 1` for $k_{\perp}$, while `radius` is the R parameter, `pTjetMin` the minimum $p_{\perp}$ of jets, and `etaMax` the maximum pseudorapidity of the detector coverage.

Inside the event loop, you can analyze an event by a call

```
slowJet.analyze( pythia.event );
```

The jets found can be listed by `slowJet.list();`, but this is only feasible for a few events. Instead you can use the following methods:

`slowJet.sizeJet()` gives the number of jets found,
`slowJet.pT(i)` gives the $p_{\perp}$ for the `i`'th jet, and
`slowJet.y(i)` gives the rapidity for the `i`'th jet.

The jets are ordered in falling $p_{\perp}$.

C Interface to HepMC

The standard HEPMC event-record format is frequently used in the MCnet school training sessions, notably since it is required for comparisons with experimental data analyses implemented in the Rivet package. Then a ready-made installation is used. However, for the ambitious, here is sketched how to set up the PYTHIA interface, assuming you already have installed HEPMC. A similar procedure is required for interfacing to other external libraries, so the points below may be of more general usefulness.

To begin with, you need to go back to the installation procedure of section 2 and insert/redo some steps.

1. Move back to the main `pythia82xx` directory (`cd ..` if you are in `examples`).
2. Configure the program:
   ```
   ./configure --with-hepmc2=path
   ```

where the directory-tree `path` would depend on your local installation. If the library is in a standard location you can omit the `=path` part.

3. Use `make` as before, to make the configure information available in the `examples/Makefile.inc` file, and move back to the `examples` subdirectory.
4. You can now also use the `main41.cc` and `main42.cc` examples to produce HepMC event files. The latter may be most useful; it presents a slight generalisation of the command-line-driven main program you constructed in Section 5. After you have built the executable you can run it with

```
./main42 infile hepmcfile > main42.out
```

where `infile` is an input "card" file (like `mymain01.cmnd`) and `hepmcfile` is your chosen name for the output file with HepMC events.

Note that the above procedure is based on the assumption that you will be running your main programs from the `examples` subdirectory. For experts there is a `make install` step to install the library and associated components in locations of your choice, and a `bin/pythia8-config` script to help you link to the library from anywhere.

D Preparations before starting the tutorial

Normally, you will run this tutorial on your own (laptop or desktop) computer. It is therefore important to make sure that you will be able to extract, compile, and run the code.

Pythia is not a particularly demanding package by modern standards, but some basic facilities such as Emacs (or an equivalent editor), gcc (g++), make, and tar must be available on your system. Below, we give some very basic instructions for standard installations on Linux, Mac OS X, and Windows platforms, respectively.

In the context of summer schools, students are strongly recommended to make sure that the above-mentioned facilities have been properly installed before traveling to the school, especially if the school is in a location which is likely to offer limited bandwidth.

D.1 Linux (Ubuntu)

The default tutorial instructions are intended for Linux (or other Unix-based) platforms, so this should be the easiest type of system to work with. The presence of the required development tools should be automatic on most Linux distributions.

Nonetheless, it seems that at least default installations of Ubuntu 12 do not include the full set of tools. These can be obtained by installing the "build-essential" package, by opening a terminal window and typing

```
sudo apt-get install build-essential
```

D.2 Max OS X

Mac OS X does not include code development tools by default, but they can relatively easily be obtained by installing Apple's Xcode package, which is free of charge from the App Store; just type "xcode" in the search field to find it. Note that downloading and installing Xcode and the Command Line Tools that come with it can take quite some time, and if you don't already have an Apple ID it will take even longer, so this should be done well before starting the tutorial.

With Xcode installed, you will also be able to use MacPorts (`www.macports.org`), a convenient package management system for Macs, which makes it very easy to install and maintain compiler suites, LaTeX, Root, and many other packages. Emacs is not part of the Xcode Command Line Tools, so is another useful example.

D.3 Windows

Unfortunately Microsoft Windows is not currently supported. If you don't have access to a regular Linux environment, e.g. via dual boot on your Windows laptop, we are aware of three possible approaches to take. We have no direct experience with either of them, however, so cannot help you in case of trouble.

- Install Linux in a Virtual Machine (VM) on your Windows system, and then work within this virtual environment as on any regular Linux platform. You could e.g. download the VirtualBox
  `https://www.virtualbox.org/`
  and install either Ubuntu or CernVM (Scientific Linux)
  `http://cernvm.cern.ch/`
  on it. If you install an Ubuntu VM, please see the instructions above for Ubuntu systems.

- Install the Cygwin package, intended to allow Linux apps to run under Windows, see
  `https://www.cygwin.com/`
  Be sure to install the Dev tools, which appears in the list of options to include, but won't be installed by default. Then put the `pythia82xx` folder in the `Cygwin/home` directory, and compile and work with it as usual. (The `include/Pythia8Plugins/execinfo.h` file provides dummy versions of methods needed for proper compilation.)

- The nuget.org website
  `http://www.nuget.org/packages/Pythia8/`
  contains pre-built Pythia packages ready to be used under Windows Visual Studio.

Note that linking with other libraries may involve further problems, in particular for the dynamic loading of LHAPDF. The exercises here only rely on Pythia standalone, however.

References

[1] T. Sjöstrand, S. Ask, J.R. Christiansen, R. Corke, N. Desai, P. Ilten, S. Mrenna, S. Prestel, C. Rasmussen and P.Z. Skands, arXiv:1410.3012 [hep-ph]

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

[3] S. Höche and S. Prestel, EPJC **75** (2015) 461 arXiv:1506.05057 [hep-ph]

[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] J. Alwall et al., Comput. Phys. Commun. **176** (2007) 300 [hep-ph/0609017]

[6] J. Butterworth et al., arXiv:1405.1067 [hep-ph]

[7] Particle Data Group, K.A. Olive et al., Chin.Phys. **C38** (2014) 090001

[8] M. Cacciari, G.P. Salam and G. Soyez, Eur. Phys. J. C72 (2012) 1896 [arXiv:1111.6097 [hep-ph]]