LUND UNIVERSITY

Monte Carlo School
Physics at the Terascale
21–24 April 2008
DESY, Hamburg

PYTHIA 8 Worksheet

Torbjörn Sjöstrand
Department of Theoretical Physics, Lund University

1 Introduction

Pythia 8 is, by today's standards, a small package. It is completely selfcontained, and is therefore easy to install for standalone usage, e.g. if you want to have it on your own laptop. The Pythia package also comes with an interface to HepMC (and LHAPDF). Section 2 describes how to install Pythia, both standalone and for use with HepMC. If you do not already have HepMC installed, or if you are not interested in learning how to install Pythia 8, you can skip this section, and instead make use of the existing Pythia 8 installation, found on
`/afs/desy.de/group/alliance/mcg/public/MCGenerators/pythia8/107/i586_rhel40/lib`

When you use Pythia you are expected to write the main program yourself, for maximal flexibility and power. Section 3 gives you a simple step-by-step recipe how to write a minimal such program, that could then gradually be expanded in different directions, e.g. as in Section 4.

There already exist some readymade main programs, however, where you only supply an input "cards file" with your specific instructions for the run. How to do this for the exercise of today is described in Section 5. If you want to get to the physics as rapidly as possible you can therefore skip right to it.

Finally section 6 gives some hints how to study other possible physics topics.

2 Installation

Here is how to install Pythia 8 on a Linux/Unix/MacOS system. The baseline is for a standalone installation, while the three points typeset in italics provide the additional instructions if you want to be able to link to an existing installation of HepMC. (You will have to look elsewhere if you want help with installing HepMC.) We denote the changing-with-time sub-subversion number `xx`, currently `07`.

- Use your web browser to go to
  `http://www.thep.lu.se/~torbjorn/Pythia.html`

- Right-click with your mouse on the link to (the current)
  `pythia81xx.tgz`
  and decide where you want to download the file. (The left-click default location is installation-dependent, and could be your home directory or the Desktop one.)
- When in the directory where `pythia81xx.tgz` was downloaded, type
  `tar xvfz pythia81xx.tgz`
  This will create a new (sub)directory `pythia81xx` where all files are unpacked.
- Move to this directory (`cd pythia81xx`).
- *HepMC: (version 2 only) type*
  `./configure --with-hepmc=path`
  where `path` *would depend on your local configuration. For instance, at DESY a* `path`
  `/afs/desy.de/group/alliance/mcg/public/HepMC/2.03.06/i586_rhel40`
  should work. (Should `configure` *not recognize the version number you can supply that with an optionat argument,* `--with-hepmcversion=2.03.06`*)*
- Compile the program with
  `make`
  This will take 1–3 minutes (computer-dependent).
- For test runs, move to the `examples` subdirectory.
- *HepMC: 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.)*
- The programs `mainNN`, with `NN` from `01` through `27` can now be executed with
  `make mainNN`
  `./mainNN.exe > outfile`
  You can study the contents of the `outfile`.
- *HepMC: you can similarly use main programs* `NN = 31` *and* `32`, *which also produce event files* `hepmcoutNN.dat`.
- For the ambitious: Further `configure` possibilites exist, see the README files in the `pythia81xx` directory and the `examples` subdirectory for details how to proceed.

3 A "Hello World" program

Assume you want to generate a single gg $\to$ $t\bar{t}$ event at the LHC, using PYTHIA standalone.

To do this, first open a new file `mymain.cc` in the `examples` subdirectory. Then type the following lines (here interspersed with comments):

```
#include "Pythia.h"
```

to include the header file with classes and methods we need to use;

```
using namespace Pythia8;
```

to save us from having to type the `Pythia8::` qualifier all the time;

```
int main() {
```

to begin the main program;

```
Pythia pythia;
```
to declare an instance `pythia` of the main `Pythia` class;
```
pythia.readString("Top:gg2ttbar = on");
```
to send in a message to `pythia` to switch on the process `Top:gg2ttbar`;
```
pythia.init( 2212, 2212, 14000.);
```
to tell `pythia` to initialize for a collision process between two incoming protons (PDG code 2212) at 14000 GeV c.m. energy;
```
pythia.next();
```
to tell `pythia` to generate the next event (in this run the only one);
```
pythia.event.list();
```
to list the event information stored in the `event` member of `pythia`;
```
return 0; }
```
to finish the main program with an error-free return.

Next you need to edit the `Makefile` (the one in the `examples` subdirectory) to make it know 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 two next enumerate the main programs that do not need any external libraries. Edit the last of these lines to include also `mymain`:
```
 main21 main22 main23 main24 main25 main26 main 27 mymain: \
```

Now it should work as before with the other examples:
```
make mymain
./mymain.exe > outfile
```
whereafter you can study `outfile`.

Alternatively, if you did not install PYTHIA 8 standalone, you can link to the existing DESY installation. Then replace the previous two paragraphs by
```
source /afs/desy.de/user/m/mcpythia/public/pythia8-setup-mymain.sh
./mymain.exe > outfile
```
which is a simplified variant of the `pythia8-setup-main32.sh` script in section 5 (it does not copy a main program).

4 A first realistic analysis

The main program can thereafter be gradually expanded.

- To allow the process $q\overline{q} \to t\overline{t}$ as well add a second `pythia.readString` call with argument `"Top:qqbar2ttbar = on"`.
- To give some variety of event types, introduce a loop over 5 events (to begin with), i.e. `pythia.next()` should be executed this many times.
- For the analysis you need to be able to access the information in the event record, as shown by the `pythia.event.list()` listing. To loop over all particles in it, use
  ```
  for (int i = 0; i < pythia.event.size(); ++i) {
  ```
  with a matching closing curly bracket further down.
- Inside the loop you can access the properties of particle `pythia.event[i]`. For instance, the method `id()` returns the PDG identity code, so if you insert
  ```
  cout << "i = " << i << " id = " << pythia.event[i].id() << endl;
  ```

you should get a table of identity codes that matches the one in the event listing.

- The event listing contains all partons and particles, traced through a number of intermediate steps. For instance, several copies of the top may well appear in the listing, as the addition of showers shifts the top around. It is the last copy of top that gives the "final" answer. You can thus obtain the location of this top e.g. by a line
  ```
  int iTop = 0;
  ```
  before the loop over the particles in the event, and
  ```
  if (pythia.event[i].id() == 6) iTop = i;
  ```
  inside of it.
- 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()`, and pseudorapidity, `pythia.event[i].eta()`. Use these methods to print out the values for the final top found above.
- We now want to generate more events, say 1000, to view the shape of these distribution. You then need to remove the listing and printing for each event. In its place book two histograms (a very simple histogramming class comes along with the program, for rapid check/debug purposes)
  ```
  Hist pT("top transverse momentum", 100, 0., 200.);
  Hist eta("top pseudorapidity", 100, -5., 5.);
  ```
  before the event loop, store the values
  ```
  pT.fill( pythia.event[iTop].pT() );
  pT.fill( pythia.event[iTop].eta() );
  ```
  for each event, and write out the histograms
  ```
  cout << pT << eta;
  ```
  after the event loop.
- 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 updated whenever the particle `isCharged()` and `isFinal()`. For the histogram, note that it can be trecherous to have bin limits at integers, where rounoff 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.

From what we have discussed so far to the readymade program `main32`, to be used in the next section, a few aspects should be noted.

- Code has been added that will convert the `pythia.event` event information into the HepMC format, and write out these events to a file, for subsequent further analysis.
- On the other hand, there is no code for event analysis inside the main program itself.
- The use of `pythia.readString(...)` commands has been replaced by a `pythia.readFile("filename")` one, where each line in this file can contain one string (without enclosing quotes) of instructions to Pythia.
- A few predefined variables, such as the number of events to generate, can also be set in this file and extracted for use in the main program.
- The name of this input file and of the HepMC output file are read in from the

command line.

5 Generation of top events

In this section the emphasis is on generating files of HepMC events that you can then analyze further with Root. Here Pythia will therefore be considered as a black box. To this end you can use the `main32.cc` program found in the `examples` subdirectory. Assuming that you have properly set up everything locally on your computer, you would need to `make main32` in the `examples` subdirectory to produce an executable.

For the purpose of this school, given the DESY-specific paths to the GENSER-provided libraries, you may instead execute a simple script

```
source /afs/desy.de/user/m/mcpythia/public/pythia8-setup-main32.sh
```

to

- add the HepMC path to `LD_LIBRARY_PATH` and set `PYTHIA8DATA` (the directory where default Pythia settings and particle data are stored),
- copy `main32.cc` from the GENSER repository to your current directory, and
- compile `main32.cc`, linked to Pythia 8.107 and HepMC 2.03.06.

The exectutable `main32.exe` is then run with a command line like

```
./main32.exe infile hepmcfile > outfile
```

Here `hepmcfile` is your chosen name for the output file with HepMC events, and `outfile` your chosen name for information on the run. (If you omit `> outfile` it will appear on the screen.) The rest of this section will be concerned with the `infile`, which is the place where you decide what is to be done during the run.

The `infile` can contain one command per line, of the type

```
variable = value
```

All Pythia variable names begin with a letter or a digit, so if you begin a line with a non-alphanumeric character (such as !, # or $) it will be interpreted as a comment line. You can also add comments on a valid command line, after the value. For readability we recommend it to be preceded by a non-alphanumeric character, as for comment lines. Blanks can be freely inserted, except inside the variable name and the value. Further, the equal sign is optional, but improves readability. The variable names are case-insensitive; the mixing of cases has been chosen purely to improve readability.

All valid variables are listed in the online manual, which you can access if you open a web browser on the file

```
/afs/desy.de/group/alliance/mcg/public/MCGenerators/pythia8/107/
share/htmldoc/Welcome.html
```

(all in one go; the address had to be split here since it did not fit on one line). Use the left-column index to navigate among the topics. Cut-and-paste of variable names can be used to avoid spelling mistakes.

The same pages also exist in an interactive variant, where you semi-automatically can construct a file with all the command lines you wish to have. This requires that somebody installs the `phpdoc` directory in a webserver (i.e. like your homepage is). If you lack a local installation you can use the one at

```
http://home.thep.lu.se/~torbjorn/php8105/Welcome.php
```

This is not a commercial-quality product, however, but requires some user discipline. As explained on the "Save Settings" page, you must initially pick a temporary file, then remember to save changes to this file for each new webpage that you change, and finally determine the ultimate location of the file.

We are now ready to introduce you to some command lines that you should have in your `infile`:

- `Main:numberOfEvents = 100`
  is your choice of how many events will be generated and stored in HepMC format in the `hepmcfile`.
- `Main:timesToShow = 10`
  how many times the program will print how far along the job is (more useful for longer runs).
- `Main:showChangedSettings = on`
  echo back the list of changes you have made, to check you got it right.
- `Beams:idA = 2212`
  first incoming beam is a 2212, i.e. a proton.
- `Beams:idB = 2212`
  second beam is also a proton.
- `Beams:eCM = 14000.`
  the cm energy of collisions.
- `Top:gg2ttbar = on`
  switch on the process $gg \to t\bar{t}$.
- `Top:qqbar2ttbar = on`
  switch on the process $q\bar{q} \to t\bar{t}$.

These commands together would provide a minimal `infile`, necessary to generate some first few events that you could analyze further. As you go along you can increase the number of events and also play with further optional aspects, such as:

- `6:m0 = 175.`
  change the top mass, which by default is 171 GeV.
- `Main:showChangedParticleData = on`
  echo back particle data for those particles you have changed.
- `PartonLevel:FSR = off`
  switch off final-state radiation.
- `PartonLevel:ISR = off`
  switch off initial-state radiation.
- `PartonLevel:MI = off`
  switch off multiple interactions.

6 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.

- Higgs production can proceed through several different production processes. For the Standard Model Higgs some process switches are
  `HiggsSM:ffbar2H` for $f\bar{f} \to H^0$ (f generic fermion, here mainly $b\bar{b} \to H^0$)
  `HiggsSM:gg2H` for $gg \to H^0$
  `HiggsSM:ffbar2HZ` for $f\bar{f} \to H^0Z^0$
  `HiggsSM:ffbar2HW` for $f\bar{f} \to H^0W^{\pm}$
  `HiggsSM:ff2Hff(t:ZZ)` for $f\bar{f} \to H^0f\bar{f}$ via Z^0Z^0 fusion
  `HiggsSM:ff2Hff(t:WW)` for $f\bar{f} \to H^0f\bar{f}$ via W^+W^- fusion
  `HiggsSM:all` for all of the above (and some more) Study the $p_\perp$ and η spectrum of the Higgs in these processes, and compare.
- You can also vary the Higgs mass with a `25:m0 = ...` and switch off FSR/ISR/MI as above for top.
- Z^0 production to lowest order only involves one process, accessible with `WeakSingleBoson:ffbar2gmZ = on`. The problem here is that the process is $f\bar{f} \to \gamma^*/Z^0$ with full γ^*/Z^0 interference and so a signficiant enhancement at low masses. The combined particle is always classified with code 23, however. So generate events and study the γ^*/Z^0 mass and $p_\perp$ distributions. Then restrict to a more "Z^0-like" mass range with `PhaseSpace:mHatMin = 75.` and `PhaseSpace:mHatMax = 120.`
- Using your favourite jet cluster algorithm, study the number of jets found in association with the Z^0 above. You can switch off Z0 decay with `23:mayDecay = no`. If you do not have a jet finder around, to begin with you can use the simple `CellJet` one that comes with PYTHIA, see the "Event Analysis" page in the online manual. A more interesting alternative is provided by FASTJET, see next point. Again check the importance of FSR/ISR/MI.
- If you would like to try out FASTJET, which is on the road to becoming the standard for LHC jet analyses, you can find an example if you copy the file
  `/afs/desy.de/user/m/mcpythia/public/pythia8-setup-fastjetexample.sh`
  to your working directory, then
  `source pythia8-setup-fastjetexample.sh`
  to
  - add the FASTJET installation path (in the DESY GENSER repository) to `LD_LIBRARY_PATH` and set `PYTHIA8DATA`,
  - copy the program `fastjetexample.cc` to the current directory, and
  - compile to produce an executable `fastjetexample.exe` that you can then run.

  Once you have this working, you can modify `fastjetexample.cc` to suit your needs, and change `pythia8-setup-fastjetexample.sh` so that you do not overwrite the former each time you `source` the latter.

Acknowledgements

Thanks to Florian Bechtel for providing suitable scripts for the setup in the DESY environment, and for the FASTJET example.