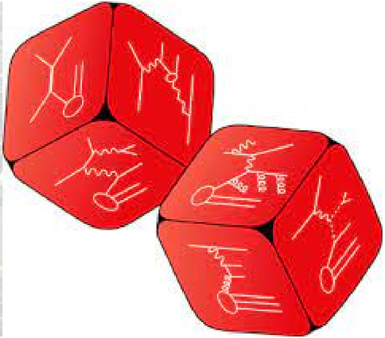
Third HEP Graduate Workshop

Physics Modeling

May 26, 2022

Giancarlo Panizzo Università degli Studi di Udine & INFN - Gruppo collegato di Udine



Hands on session

You can find (similar) material here:

Worksheet link

(there you can find many more details/topics).
Another useful link:

Online Pythia8.3 manual link

I. Event generation

A ”Hello World” program

```
// 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: ttbar production
  pythia.readString("Beams:eCM = 13000."); // 13 TeV CM energy.
  pythia.readString("PartonLevel:all = off"); // swith off showering
  pythia.readString("HadronLevel:all = off"); // switch off hadronization
  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.
```

Compiling and running

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:

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

It is important to remember that you need to compile your code each time that you modify it. If you want to pick another name, or if you need to link to more libraries, you have to edit `examples/Makefile` appropriately.

Questions

By inspection of your output `myout01`, do you understand

1. how many events have you generated?
2. which decay channel does your $t\bar{t}$ event belong to?

The event record

- `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());
- *px*, *py*, *pz* and *e*: the components of the momentum four-vector (px, py, pz , E), in units of GeV with $c = 1$ (methods px(), py(), pz() and e());

Identity codes

A complete specification of the PDG codes is found in the “Review of Particle Physics”. An online listing is available **here**.
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 (where separate entities): negative sign
- Simple meson and baryon codes: constructed from constituent (anti)quark codes+final spin-state-counting digit $2s + 1$ ($K^0_{L,S}$ being exceptions)

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 online manual):

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/decay further its status code is negated
    - notice: it is never removed from the event record!
- only particles in the “final state” remain with positive codes.
    - the `isFinal()` method returns true/false for ± status codes

Questions

By inspection of your output `myout01`, do you understand

1. which particles collided/branched/decayed?
2. if particles in your "final state" have daughters?

Exercise

- Modify your script to produce 100 events
  - Hint: you don't need to initialize `pythia` for each event
- print also on screen the event number every 10 events, to check if your changes are effective

Question: is the event info printed for all your events?

Solution

```
// 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: ttbar production
  pythia.readString("Beams:eCM = 13000."); // 13 TeV CM energy.
  pythia.readString("PartonLevel:all = off"); // swith off showering
  pythia.readString("HadronLevel:all = off"); // switch off hadronization
  pythia.init(); // Initialize; incoming pp beams is default.

  // Generate event(s).
  for (int iEvent = 0; iEvent < 100; ++iEvent) {
     if (iEvent%10==0 ) std::cout << "INFO: event "<< iEvent<< endl;
     pythia.next(); // Generate an(other) event. Fill event record.
  }
  return 0;
} // End main program with error-free return.
```

Answer: no, the event info is printed only for the first event. Use e.g. `readString("Next:numberShowProcess = 5")`.

Hard scattering event analysis

We now would like to use ROOT to plot some “parton level” distributions. We need to link our program to the ROOT library: open your `examples/Makefile` and change

```
# User-written examples for tutorials, without external dependencies.
mymain%: $(PYTHIA) mymain%.cc
        $(CXX) $@.cc -o $@ $(CXX_COMMON)
```

into

```
# User-written examples for tutorials, linked now to ROOT
mymain%: $(PYTHIA) mymain%.cc
ifeq ($(ROOT_USE),true)
        $(CXX) $@.cc -o $@ -w $(CXX_COMMON) $(ROOT_LIB)\
         ‘$(ROOT_CONFIG) --cflags --glibs‘
else
        $(error Error: $@ requires ROOT)
endif
```

Exercise

The example `main91.cc` shows how to interface `ROOT` with a main program. Starting from this

```
cp main91.cc mymain02.cc
```

create a second `mymain02.cc` program drawing a plot of the top quark p_T in the hard scattering.
Hints:

- Replace all `readString()` calls with the ones from your previous example
- We are modeling (until now) only the hard scattering. A good way to iterate on partons in the process is then:

```
// Loop over particles in event. Find last top copy. Fill its pT.
int iTop = -1;
for (int i = 0; i < pythia.process.size(); ++i)
  if (pythia.process[i].id() == 6) iTop = i;

if (iTop>-1) toppt->Fill( pythia.process[iTop].pT() );
```

Solution

```
// Header file to access Pythia 8 program elements.
#include "Pythia8/Pythia.h"

// ROOT, for histogramming.
#include "TH1.h"

// ROOT, for interactive graphics.
#include "TVirtualPad.h"
#include "TApplication.h"

// ROOT, for saving file.
#include "TFile.h"

using namespace Pythia8;

int main(int argc, char* argv[]) {

  // Create the ROOT application environment.
  TApplication theApp("hist", &argc, argv);

  // Create Pythia instance and set it up to generate hard QCD processes
  // above pTHat = 20 GeV for pp collisions at 14 TeV.
  Pythia pythia;
  pythia.readString("Top:gg2ttbar = on"); // Switch on process: ttbar production
  pythia.readString("Beams:eCM = 13000."); // 13 TeV CM energy.
  pythia.readString("PartonLevel:all = off"); // swith off showering
  pythia.readString("HadronLevel:all = off"); // switch off hadronization
  pythia.readString("Next:numberShowProcess = 5"); // print first five events
  pythia.init();
```

Solution

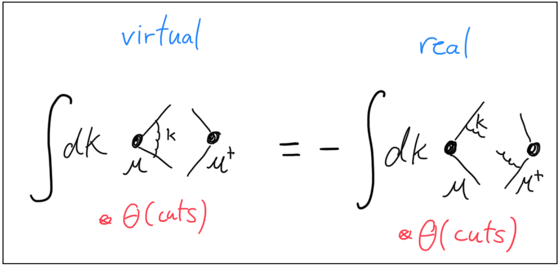
```
  // Create file on which histogram(s) can be saved.
  TFile* outFile = new TFile("hist.root", "RECREATE");
  // Book histogram.
  TH1F *toppt = new TH1F("toppt","top pt", 20, 0, 1000.);

  // Begin event loop. Generate event; skip if generation aborted.
  for (int iEvent = 0; iEvent < 1000; ++iEvent) {
    if (!pythia.next()) continue;

    // Loop over particles in event. Find last top copy. Fill its pT.
    int iTop = -1;
    for (int i = 0; i < pythia.process.size(); ++i)
      if (pythia.process[i].id() == 6) iTop = i;

    if (iTop>-1) toppt->Fill( pythia.process[iTop].pT() );
  }

  // Statistics on event generation.
  pythia.stat();

  // Show histogram. Possibility to close it.
  toppt->Draw();
  std::cout << "\nDouble click on the histogram window to quit.\n";
  gPad->WaitPrimitive();

  // Save histogram on file and close file.
  toppt->Write();
  delete outFile;

  // Done.
  return 0;
}
```

II. Parton showering

Parton showers

Remember the KLN theorem: Infrared singularities arising in real-emission diagrams cancel against alike divergences in virtual corrections.[1]

For the (most) enhanced parts, we can devise a radical interpretation of KLN:



"The rate for # particles remaining the same is (negative) the rate for the # particles increasing at any scale t – even in the presence of cuts/regularization".

This is the first building block of a parton shower.

Sudakov factors

The behavior of partons is similar to that of radioactive elements.

The # particles n can only change $n \to n+1$ (due to decay or splitting) at scale t if it has not already changed at $t' > t$.

The probability to not change in a finite interval Δt is

$$1 - \Delta t P(t)$$

where P is the splitting kernel containing the enhanced parts of the real correction. This is simply statement about unitarity: The rate of no change and the rate of all possible changes add to unity.

The probability not to change in any very small sub-interval $\Delta t/n$ is

$$\left(1 - \frac{\Delta t}{n} P(t)\right)^n \quad \xrightarrow{n\to\infty} \quad \exp\left(-\int_0^{\Delta t} dt P(t)\right)$$

This exponential suppression of not splitting is called the Sudakov factor.

[no splitting] ↔ [fixed # particles]. Thus, the Sudakov introduces virtual corrections.

Exercise

Modify your program to activate showering.
Hints:

- Change all `pythia.process[i]` into `pythia.event[i]`
- Now `pythia.readString("PartonLevel:all = on")`

Question: how many entries has now your event?
Question: how does the top p_T distribution change?

Exercise

Modify your program to activate hadronization. Plot the charged multiplicity distribution of stable particles.

Hints:

- Now also `pythia.readString("HadronLevel:all = on")`

Thanks!