Monte Carlo Generators and Rivet Tutorial

Deepak Kar, MPI@LHC Workshop, Shimla, India

"The predictions of the model are reasonable enough physically that we expect it may be close enough to reality to be useful in designing future experiments and to serve as a reasonable approximation to compare to data. We do not think of the model as a sound physical theory . . ."

– Richard Feynman and Rick Field, 1978

This Tutorial

- How to generate simulated events: Pythia8 (you already learned Herwig7)
- Analysis framework: RIVET for *particle level* analysis.
- Now is a good time to fire up the virtual machine (hope you have it :)

Pythia8

- ... is a leading order PS generator.
- One of the most widely used for many years.
- Relatively easy to install (along with its *friends*: HepMC, and LHAPDF6) and run, online user manual: http://home.thep.lu.se/~torbjorn/pythia82html/Welcome.html
- Run via various *mainXX* programs.

Generating events with Pythia8

- Start the terminal
- Pythia8230 is located at: > Documents/src/pythia8230
- We will use main42, a generic main program. It is inside examples directory.
- Compile: make main42, should result in a main42 executable in the directory.
- Input (which process to generate, how many events, collision energy, ...) are specified via a *runcard (cmnd files)*, we will use main42.cmnd
- > ./main42 main42.cmnd out.hepmc

Run Card

Change to 5000

```
! File: main42.cmnd
! This file contains commands to be read in for a Pythia8 run.
! Lines not beginning with a letter or digit are comments.
! Names are case-insensitive  -  but spellings-sensitive!
! The changes here are illustrative, not always physics-motivated.

! 1) Settings that will be used in a main program.
Main:numberOfEvents = 200          ! number of events to generate
Main:timesAllowErrors = 3          ! abort run after this many flawed events

! 2) Settings related to output in init(), next() and stat().
Init:showChangedSettings = on      ! list changed settings
Init:showAllSettings = off         ! list all settings
Init:showChangedParticleData = on  ! list changed particle data
Init:showAllParticleData = off     ! list all particle data
Next:numberCount = 1000            ! print message every n events
Next:numberShowLHA = 1             ! print LHA information n times
Next:numberShowInfo = 1            ! print event information n times
Next:numberShowProcess = 1         ! print process record n times
Next:numberShowEvent = 1           ! print event record n times
Stat:showPartonLevel = on          ! additional statistics on MPI

! 3) Beam parameter settings. Values below agree with default ones.
Beams:idA = 2212                   ! first beam, p = 2212, pbar = -2212
Beams:idB = 2212                   ! second beam, p = 2212, pbar = -2212
Beams:eCM = 14000.                 ! CM energy of collision
```

Change to 13000

Continued…

Run Card

```
! 4) PDF settings. Default is to use internal PDFs
! some pdf sets examples: cteq61.LHpdf cteq61.LHgrid MRST2004nlo.LHgrid
#PDF:pSet = LHAPDF5:MRST2001lo.LHgrid
! Allow extrapolation of PDF's beyond x and Q2 boundaries, at own risk.
! Default behaviour is to freeze PDF's at boundaries.
#PDF:extrapolate = on

! 5a) Pick processes and kinematics cuts.
! Colour singlet charmonium production of J/psi and chi_c.
Charmonium:gg2ccbar(3S1)[3S1(1)]g    = on,off
Charmonium:gg2ccbar(3PJ)[3PJ(1)]g    = on,on,on
Charmonium:qg2ccbar(3PJ)[3PJ(1)]q    = on,on,on
Charmonium:qqbar2ccbar(3PJ)[3PJ(1)]g = on,on,on
PhaseSpace:pTHatMin = 20.          ! minimum pT of hard process

! 5b) Alternative beam and process selection from a Les Houches Event File.
! NOTE: to use this option, comment out the lines in section 5a above
! and uncomment the ones below. Section 3 is ignored for frameType = 4.
#Beams:frameType = 4                ! read info from a LHEF
#Beams:LHEF = events.lhe             ! the LHEF to read from

! 6) Other settings. Can be expanded as desired.
! Note: may overwrite some of the values above, so watch out.
#Tune:pp = 6                        ! use Tune 4Cx
#ParticleDecays:limitTau0 = on      ! set long-lived particle stable ...
#ParticleDecays:tau0Max = 10        ! ... if c*tau0 > 10 mm
```

Change to process of interest (comment out or remove)

Uncomment

For Monash: Change to: Tune:ee =7 Tune:pp = 14

Example Run Cards

Z-boson production and leptonic decay

```
! 5a) Pick processes and kinematics cuts.
WeakSingleBoson:ffbar2gmZ =on
23:onMode = off
23:onIfAny = 11 13
23:mMin = 60
```

W-boson production and leptonic decay

```
! 5a) Pick processes and kinematics cuts.
WeakSingleBoson:ffbar2W = on
24:onMode = off
24:onIfAny = 11 -11 13 -13
```

ttbar production and semileptonic decay

```
! 5a) Pick processes and kinematics cuts.
Top:gg2ttbar = on
Top:qqbar2ttbar = on
24:onMode = off
24:onPosIfAny = 11 13
24:onNegIfAny = 1 2 3 4 5
```

Minbias Events

```
! 5a) Pick processes and kinematics cuts.
SofQCD:inelastic = on
```

So then …

Analyze the events

- ROOT is used extensively by the experiments
- But unless you are an experimentalist, it is probably too intimidating for you
- Many times, you just want to quickly look at simulated events...

RIVET

Based somewhat on Andy Buckley's LH13 tutorial

- A generator agonistic analysis system for generators (no direct data analysis!) in C++ (now in C++11)
- Physics plots from generator output (in HepMC format)
- Compare MC predictions with *built-in* actual (unfolded) data analyses from different experiments
- Everything defined in terms of stable final state objects
- Details: https://rivet.hepforge.org/

Important!

Analyses intended to be based on physical objects:

- Final state hadrons
- Jets (FastJet)
- Muons, Electrons (dressed)
- Bosons reconstructed from particles (rather than taken from event record)

Version 2.5 contains $\sim$ 350 Analyses (195 LHC)

- Monte Carlo validation and tuning, data preservation
- Lots of code examples to get inspired

Rivet for you!

- Super convenient bootstrap script to install (Rivet and all its dependencies) at rivet.hepforge.org
- Source codes of existing analyses serve as useful examples
- Helping the community by adding your analysis to the official library

Trying out RIVET

- ... it is setup for you, just do > source Documents/bin/activate
- > rivet --help
- > rivet --list-analyses (ATLAS_ or MC_)

Running a Data Analysis

- Since we are looking at Minbias events, lets try
  > rivet -a ATLAS_2016_I1467230 -a ATLAS_2016_I1468167 -a ATLAS_2017_I1509919 out.hepmc
- Output: Rivet.yoda
- Look inside the yoda file
- Plot with rivet-mkhtml Rivet.yoda (--mc-errs)
- View plots by firefox rivet-plots/index.html

Compare Pythia8 and Herwig7

- Run the same rivet command on the Herwig7 output hepmc file.
- Remember to remember the earlier output yoda file to say out_py8.yoda (otherwise it will get overwritten, or you can do -o out_her7.yoda here)
- Plot both yoda files together!

Plot File (example)

```
BEGIN PLOT /ATLAS_2015_I1343107/d18-x01-y01
XLabel=$E^{\rm{miss}}_T$ [GeV]
YLabel=Events
XMin=150
END PLOT
```

Labels, formatting controlled by corresponding .plot file

The data is present in corresponding reference .yoda file

Writing an Analysis

- The analyses named MC_ are pure MC based analysis, no reference data to compare with.
- Useful for testing generator predictions.
- You can make a template: rivet-mkanalysis MC_MyAna
- Find the MC_MyAna.cc file in the directory (also MC_MyAna.info and MC_MyAna.plot)
- Look inside the cc file!

There are many different analyses in Rivet code repository. Usually one or more examples should be close to what you are trying to do.

Walkthrough

1

```
// -*- C++ -*-
#include "Rivet/Analysis.hh"
#include "Rivet/Projections/FinalState.hh"

namespace Rivet {


  /// @brief Add a short analysis description here
  class MC_MyAna : public Analysis {
  public:

    /// Constructor
    DEFAULT_RIVET_ANALYSIS_CTOR(MC_MyAna);


    /// @name Analysis methods
    //@{
```

Basic include stuff, add new headers as required

Continued…

Walkthrough

2

Expect usual C++ init/execute/finalize type loop code structure

```
  /// Book histograms and initialise projections before the run
    void init() {

      // Initialise and register projections
      declare(FinalState(Cuts::abseta < 5 && Cuts::pT > 100*MeV),
"FS");

      // Book histograms
      _h_XXXX = bookProfile1D(1, 1, 1);
      _h_YYYY = bookHisto1D(2, 1, 1);
      _h_ZZZZ = bookCounter(3, 1, 1);

    }
```

Continued…

Walkthrough

2

Expect usual C++ init/execute/finalize type loop code structure

```
  /// Book histograms and initialise projections before the run
    void init() {

      // Initialise and register projections
      declare(FinalState(Cuts::abseta < 5 && Cuts::pT > 100*MeV),
"FS");

      // Book histograms
      _h_XXXX = bookProfile1D(1, 1, 1);
      _h_YYYY = bookHisto1D(2, 1, 1);
      _h_ZZZZ = bookCounter(3, 1, 1);

    }
```

Projections and declaration of two types of histograms

Continued…

Walkthrough

2

Expect usual C++ init/execute/finalize type loop code structure

```
  /// Book histograms and initialise projections before the run
    void init() {

      // Initialise and register projections
      declare(FinalState(Cuts::abseta < 5 && Cuts::pT > 100*MeV),
"FS");

      // Book histograms
      _h_XXXX = bookProfile1D(1, 1, 1);
      _h_YYYY = bookHisto1D(2, 1, 1);
      _h_ZZZZ = bookCounter(3, 1, 1);

    }
```

Projections and declaration of two types of histograms

Continued…

Projections

- Observable calculators - from an event, *project* out the *physical* observables.
- Already defined in the framework
- Registered with a name in *init*
- Applied to the current event in *analyze*
- Avoids unnecessary repetition in the code

Some Details

- ChargedFinalState
- NeutralFinalState
- UnstableFinalState
- IdentifiedFinalState
- VetoedFinalState
- DISFinalState
- VisibleFinalState
- HadronicFinalState

Histogramming

- Declare at *init* by bookHisto1D or bookProfile1D (usual name, binning)
- Can be *autobooked* from reference data!
- Usual fill method in *analyze*
- scale or normalize in *finalize*
- Declare the pointers

Walkthrough

3

```
/// Perform the per-event analysis
    void analyze(const Event& event) {

      /// @todo Do the event by event analysis here

    }


    /// Normalise histograms etc., after the run
    void finalize() {

      normalize(_h_YYYY); // normalize to unity
      scale(_h_ZZZZ, crossSection()/picobarn/sumOfWeights()); // norm to cross
section

    }

    //@}
```

Heart of the code: Fill histograms here

Normalize

Continued…

Some Other Details

`Particle` and **`Jet`** both have a **`momentum()`** method which returns a **`FourMomentum`**.

Some **`FourMomentum`** methods: **`eta()`**, **`pT()`**, **`phi()`**, **`rapidity()`**, **`E()`**, **`px()`** etc., **`mass()`**. Hopefully intuitive!

Walkthrough

4

```
   /// @name Histograms
    //@{
    Profile1DPtr _h_XXXX;
    Histo1DPtr _h_YYYY;
    CounterPtr _h_ZZZZ;
    //@}


  };


  // The hook for the plugin system
  DECLARE_RIVET_PLUGIN(MC_MyAna);


}
```

Histogram pointers declared

Task

- Modify this code to plot number of charged particles and their p_T

Add/Modify

```
#include "Rivet/Projections/ChargedFinalState.hh"
```

Added in headers

```
declare(ChargedFinalState(Cuts::abseta < 2.5 && Cuts::pT > 100*MeV), "CFS");
```

Projection definition changed

```
_h_Nchg = bookHisto1D("Nchg",20,0,100);
_h_pT = bookHisto1D("pT",40,0,200);
```

Histograms declared

```
const FinalState& cfs = apply<FinalState>(event, "CFS");
double mult = cfs.size();
_h_Nchg->fill(mult);
for (const Particle& p : cfs.particles()) {
   h_pT->fill(p.pT()/GeV);
}
```

In event-loop calculate variables and filled histograms

```
normalize(_h_Nchg); // normalize to unity
normalize(_h_pT);
```

Normalised

```
 Histo1DPtr _h_Nchg, _h_pT;
```

Histogram pointers

MC_MyAna

- Compile by: rivet-buildplugin RivetMC_MyAna.so MC_MyAna.cc
- export RIVET_ANALYSIS_PATH=$PWD (or use —pwd switch)
- Run over the same hepmc file and plot.

FIFO

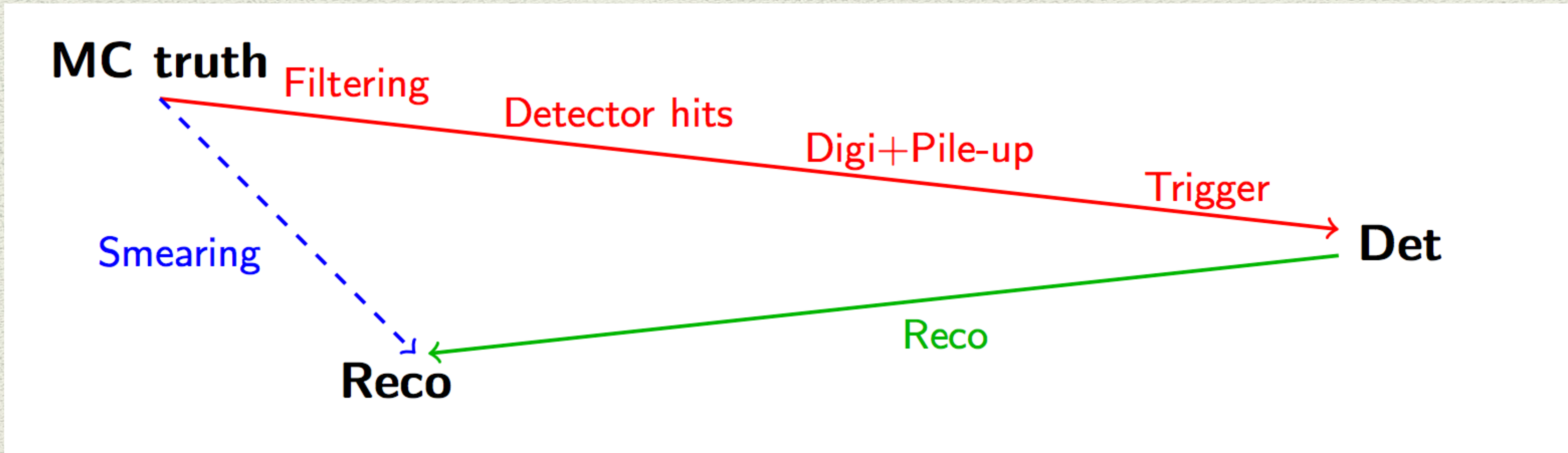
- HepMC files tend to become unmanageably large (5000 events ~ 1 GB)
- Often times, we need millions of events
- We use fifo (file in, file out), which is like a pipe. One event enters, gets processed, only then the second event is generated ...
- Look at Run.sh file (we will *run* that later)

Fifo Script

```
#Simple script to run pythia8 and rivet together via a fifo

export RIVET_ANALYSIS_PATH=$PWD  # Rivet needs to know where the analysis is
export RIVET_REF_PATH=$PWD
rm -rf my.hepmc                  # just a protection
mkfifo my.hepmc                  # create the fifo file

rm *.log                         # make sure log files are new

./main42 main42.cmnd my.hepmc > pythia.log &     # run Pythia, output goes to the pipe, always good to have a log file

rivet -a MC_MYANA  my.hepmc &> rivet.log           # run Rivet over, input comes from the pipe

rm my.hepmc
```

New Feature

- For searches, no unfolded data
- Approximate detector response/efficiencies can be made available
- Smearing of final state objects implemented (from v2.5.0)



Congratulations!