Rivet tutorial

Andy Buckley

LHC Physics Centre at CERN,
TH seminar room, 21 Nov 2013

Contents

First. . . my face

blame Movember

Introduction

The plan

I'm going to try a part work-along / part whistlestop tour.
Please ask questions as we go.

- Some background
- Setting up, and querying available analyses
- From PYTHIA to plots
- Writing and running an analysis

We're using Rivet 2.0.0 here. New release series, *should* be stable! Much nicer than Rivet 1.x, esp. for run merging and NLO/weight vector/etc. possibilities... and "future-proof"

What is Rivet?

A generator-agnostic analysis/validation system for generators.

Co-developed with HepData & HepForge, co-evolved with FastJet, ...

i.e. it's a tool for making physics plots from any generator that can produce events in the HepMC format.

All the "major" generators can do this one way or another: C++ Pythia 8, Sherpa, Herwig++ out of the box, Fortran PYTHIA 6, HERWIG+JIMMY, etc. via AGILe.

Designed (and redesigned...) with usability in mind: analysis code should be able to be concise and clear.
Rivet's become a *de facto* standard for LHC analysis archiving: *many* built-in data analyses.

Used for generator validation, archiving of (LHC) analysis algorithms corresponding to measurement papers. + MC tuning, model development, BSM studies, ...

Design philosophy / pragmatics

Rivet operates on HepMC events. It intentionally doesn't care who made them.

Emphasis on not messing with the implementation details: reconstruct resonances, avoid touching partons, etc. *Most analyses are eventually simpler and better-defined this way.*

New analyses can be picked up at runtime: this is as simple and pleasant as we can make it!
Computations are automatically cached and histograms are automatically synchronised with reference binnings.

Lots of standard analyses are built in, including key ones for pQCD, EW and MPI model testing. Now over 250 built-in analyses! Reference data is also included in the package.

Please write Rivet analyses of your analysis and contribute them. Core manpower is small: new developers welcome!!!

A little philosophy

Ok, so this is mainly for would-be MC tuners...

Generators are the means by which many physics ideas are realised. But not all generator modelling is *ab initio* with formal correctness and predictivity. The non-perturbative QCD parts in particular require phenomenological modelling with (sometimes many) free parameters.

Both kinds of model need to be tested: ensure that perturbative physics has been properly implemented and configured (e.g. NLO matching – it's not obvious, or even documented...), and test that pheno models are viable and params well-tuned (if you're not just about to do the tuning).

A dichotomy: tuning is both for

- understanding/exploring the physics of soft QCD
- data mimicking for best experimental unfolding

Know what you want!

Rivet 2.0 histogramming upgrade

- We *finally* released Rivet 2.0.0!
- Major effort to move from 1.x to 2.x series: the entire histogram system was replaced with a new interface: *YODA*
- YODA: **http://yoda.hepforge.org**
  - Completely new histo library.
  - Fast binning, with gaps and easy irregular binning.
  - Stores all 2nd order weighted moments: $(\sum w, \sum w^2, \sum wx, \sum wx^2, \sum wxy, \ldots)$ for each bin: *full run combination*
  - Natural/default inclusion of overflows and negative weights.
  - Lots more features! Pleasant to use, and more data types being added.
- All 250+ analyses had to be migrated and numerically validated...

More about Rivet 2 histogramming & merging

- YODA allows "simple" automatic run merging. With some heuristics to distinguish homogeneous and heterogeneous run types.
- Not complete: merging (normalised) histograms and profiles is one thing, but what about general objects, e.g. *asymmetries* like $(A - B)/(A + B)$?
- YODA paves the way to a complete treatment:
  - User-accessible histograms will only be temporary copies for the current event group (to allow NLO counter-events, weight vectors, etc.)
  - Synchronised to a less transient copy every time the event number changes in the event loop
  - Periodically, or on `finalize()`, this second copy gets used to make *final* histograms: normalised, scaled, added, etc.
  - "Final" histograms can be written and updated through the run: `finalize()` runs many times
  - And runs can be re-loaded and combined using the pre-finalize copies ⇒ completely general run combination.

Other Rivet developments

- Version 2.0.0 is intentionally a clone of 1.8.3 with much-improved data handling + tweaks
- Releases 1.8.4 and 2.0.1 coming asap to add an extra ~ 10 LHC analyses not yet in a public release
- Physics tool improvements largely waiting for version 2.1
  - A general system for logical combination of kinematic cuts: no more "which `double` arg is the p_{T}?"
  - Extra photon clustering options in W/Z finding (see next slide)
  - Better jet algorithm and object support, with full FastJet compatibility
  - More powerful/flexible flavour tagging, including ghost association with HF hadrons
  - Wishlist: better tools for decay chain analysis (LHCb, others?), tau and top tools

Rivet and truth definitions

- Rivet has become a useful context in which to discuss improvements to truth definitions
  $\Rightarrow$ what is found in current event records is not necessarily the best physics object! Standards are evolving.
- Classic examples: "Born" Z, top mass via `PMAS(6,1)`, ...
- Some interesting discussions about leptonic W/Z defn as the simplest example:
  - no QCD connections between initial and final state: EW effects negligible/important? $\Rightarrow$ analysis-by-analysis
  - how to cluster photons for lepton dressing (and can QED ISR/FSR be "distinguished" via final-state cuts?)
  - optional exclusion of photons from hadron decays is a good start, on the borderline of physical acceptability: appearing in Rivet soon
  - how would/could/should a Born definition work for hadronic W/Z (and Higgs)?
- Top is harder...but definitely a good idea to work on a good defn. (And we need differential top event observables – starting to arrive now.)

Setup

Rivet docs: online at **http://rivet.hepforge.org** – PDF manual, HTML list of existing analyses, and Doxygen.

Instructions:

1. Log in to `lxplus6`: `ssh` ⟨*user*⟩`@lxplus6.cern.ch`
2. Source the setup script: `source /afs/cern.ch/sw/lcg/experimental/rivet/setup.sh`

Test commands:

- `rivet --help`
- `pythia --help`

You should also be able to use **http://rivet.hepforge.org/hg/bootstrap/rawfile/tip/rivet-2-bootstrap** (and install Pythia8+Sacrifice separately)

If you want to run on your own laptop, please use the Rivet wiki installation instructions.

First Rivet runs

Viewing available analyses

Rivet knows all sorts of details about its analyses:

- List available analyses:
  `rivet --list-analyses`
- List ATLAS analyses:
  `rivet --list-analyses ATLAS_`
- Show some pure-MC analyses' full details:
  `rivet --show-analysis MC_`

The PDF and HTML documentation is also built from this info, so is always synchronised.

The analysis metadata is provided via the analysis API and usually read from an `.info` file which accompanies the analysis.

Running a simple analysis (standalone)

To avoid huge files, we get the events from generator to Rivet by writing to a filesystem pipe: **`mkfifo hepmc.fifo`**
You can also just use a file but it'll be *big*.

NB. A FIFO/pipe has to live in a non-AFS directory. On lxplus: **`mkfifo /tmp/$USER/hepmc.fifo`**

I'm going to use the Sacrifice frontend to run Pythia 8 for demonstration – use the same or run any other generator that you like with HepMC output going to the FIFO:

```
pythia -n 2000 -c HardQCD:all=on -o hepmc.fifo &
```

or `agile-runmc Pythia6:426 --beams=LHC:8000 -n 2000 -o hepmc.fifo &`

Now attach Rivet to the other end of the pipe:

```
rivet -a MC_GENERIC -a MC_JETS hepmc.fifo
```

Hopefully that worked. You can use multiple analyses at once, change the output file, etc.: see **`rivet --help`**

Feeding LHEF events into Rivet

If your code outputs LHEF events rather than HepMC, you can't use Rivet directly. Anyway, you're taking a risk that it won't work since Rivet is final-state focused...but you can also get hold of the raw event if you want and just use the histogramming and event loop.

At Les Houches 2011 I made a mini filter program which will convert LHEF files or streams to HepMC ones:

http://rivet.hepforge.org/hg/contrib/file/tip/lhef2hepmc/

Use it like this:

```
./lhef2hepmc lhef.fifo hepmc.fifo
```

or

```
./lhef2hepmc lhef.fifo - | rivet
```

Maybe some help will be needed with building this program – it's not an official part of Rivet so you have to download and build it by hand. Let us know if you need a hand.

Plotting

It's *still* not ROOT. . . we have now replaced the AIDA histograms with a new system called YODA (**http://yoda.hepforge.org**)

We agonised over this, but in the end ROOT's histos have too many restrictions, e.g. bin widths not accounted for, bin gaps not allowed, weights not handled without explicit enabling, etc. YODA is designed from the ground up to be good at what we need to do.

Plotting `.yoda` file is easy:

```
rivet-mkhtml Rivet.yoda
```

or, if you want complete control:

```
rivet-cmphistos Rivet.yoda
make-plots *.dat
```

Then view with a web browser/file browser/evince/gv/xpdf. . .

A `--help` option is available for all Rivet scripts.

Running a data analysis

We're going to use the ATLAS 7 TeV high-p_T jet shapes analysis:

```
rivet --show-analysis ATLAS_2012_I1119557
```

Note that tab completion should work on `rivet` options and analysis names.

Now to run it:

```
pythia -n 20000 -c HardQCD:all=on -c
PhaseSpace:pTHatMin=280 -o hepmc.fifo &

rivet -a ATLAS_2012_I1119557 hepmc.fifo
```

See the Py8 manual: **http://home.thep.lu.se/~torbjorn/pythia81html/Welcome.html**

And plot, much as before:

```
rivet-mkhtml Rivet.yoda:Pythia8
```

or

```
rivet-cmphistos Rivet.yoda:Pythia8
make-plots --pdfpng ATLAS*.dat
```

Writing a first analysis

Writing an analysis

Writing an analysis is of course more involved than just running **`rivet`**! However, the C++ interface is intended to be friendly: most analyses are quite short and simple because the bulk of computation is in the library.

An example is usually the best instruction: take a look at the MC_GENERIC analysis via **http://rivet.hepforge.org/hg/rivet/file/tip/src/Analyses/MC_GENERIC.cc**)

Things to note:

- Analyses are classes and inherit from **`Rivet::Analysis`**
- Usual init/execute/finalize-type event loop structure (certainly familiar from experimental frameworks)
- Weird *projection* things in **`init`** and **`analyze`**
- *Mostly* normal-looking everything else

Projections – registration

Major idea: **projections**. They are just observable calculators: given an **`Event`** object, they *project* out physical observables.

They also automatically cache themselves, to avoid recomputation. This leads to slightly unfamiliar calling code.

They are *registered* with a name in the **`init`** method:

```
void init() {
  ...
  const SomeProjection sp(foo, bar);
  addProjection(sp, "MySP");
  ...
}
```

Projections – applying

Projections were registered with a name. . . they are then applied to the current event, also by name:

```
void analyze(const Event& evt) {
  ...
  const SomeProjectionBase& mysp =
    applyProjection<SomeProjectionBase>(evt, "MySP");
  mysp.foo()
  ...
}
```

We prefer to get a handle to the applied projection as a const reference to avoid unnecessary copying.

It can then be queried about the things it has computed. Projections have different abilities and interfaces: check the Doxygen on the Rivet website, e.g.
http://projects.hepforge.org/rivet/code/dev/hierarchy.html

Final state projections

Rivet is mildly obsessive about only calculating things from final state objects. Accordingly, a *very* important set of projections is those used to extract final state particles: these all inherit from **`FinalState`**.

- The **`FinalState`** projection finds all final state particles in a given η range, with a given p_{T} cutoff.
- Subclasses **`ChargedFinalState`** and **`NeutralFinalState`** have the predictable effect!
- **`IdentifiedFinalState`** can be used to find particular particle species.
- **`VetoedFinalState`** finds particles *other* than specified.
- **`VisibleFinalState`** excludes invisible particles like neutrinos, LSP, etc.

Most FSPs can take another FSP as a constructor argument and augment it. In the near future FSPs should be able to take arbitrary combinations of kinematic cuts as a single argument.

Using FSPs to get final state particles

```
void analyze(const Event& evt) {
  ...
  const FinalState& cfs =
    applyProjection<FinalState>(evt, "ChFS");
  MSG_INFO("Total charged mult. = " << cfs.size());
  foreach (const Particle& p, cfs.particles()) {
    const double eta = p.momentum().eta();
    MSG_DEBUG("Particle eta = " << eta);
  }
  ...
}
```

Note the nice **`foreach`** macro from boost.org. We like the "make simple things simple" philosophy. Please use **`foreach`** when appropriate in any code that you contribute to Rivet. In future we may permit (and prefer) the C++ 11 range-for loop.

Physics vectors

Rivet uses its own physics vectors rather than CLHEP. They are a little nicer to use (we think!), but basically familiar. As usual, check Doxygen: **http://projects.hepforge.org/rivet/code/dev/**

`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!

Histogramming

YODA has Histo1D and Profile1D histograms (and more), which behave as you would expect. See
http://yoda.hepforge.org/doxy/hierarchy.html

Histos are booked via helper methods on the **`Analysis`** base class, which deal with path issues and some other abstractions*:
e.g. **`bookHisto1D("thisname", 50, 0, 100)`**
Histo binnings can also be booked via a vector of bin edges or *autobooked* from a reference histogram.

The histograms have the usual **`fill(value, weight)`** method for use in the **`analyze`** method. There are **`scale()`**, **`normalize()`** and **`integrate()`** methods for use in **`finalize()`**.

The fill weight is important! For kinematic enhancements, systematics, counter-events, etc.Use **`evt.weight()`**.

* The abstractions are key to handling systematics weight vectors, correlated counter-events, completely general run merging, etc.

A first analysis

Let's start with a simple "particle analysis", just plotting some simple particle properties like η, p_{T}, ϕ, etc. Then we'll try jets or W/Z.

To get an analysis template, which you can fill in with an FS projection and a particle loop, run e.g. `rivet-mkanalysis MY_TEST_ANALYSIS` – this will make the required files.

Once you've filled it in, you can either compile directly with `g++`, using the `rivet-config` script as a compile flag helper, or run `rivet-buildplugin MY_TEST_ANALYSIS.cc`

To run, first `export RIVET_ANALYSIS_PATH=$PWD`, then run `rivet` as before... or add the `--pwd` option to the `rivet` command line.

Jets (1)

There are many more projections, but one more important set which we'd like to dwell on is those to construct jets. **`JetAlg`** is the main projection interface for doing this, but almost all jets are actually constructed with FastJet, via the explicit **`FastJets`** projection.

The **`FastJets`** constructor defines the input particles (via a **`FinalState`**), as well as the jet algorithm and its parameters:

```
const FinalState fs(-3.2, 3.2);
addProjection(fs, "FS");
FastJets fj(fs, FastJets::ANTIKT, 0.6);
fj.useInvisibles();
addProjection(fj, "Jets");
```

Remember to **`#include "Rivet/Projections/FastJets.hh"`**

Jets (2)

Then get the jets from the jet projection, and loop over them in decreasing p_{T} order:

```
const Jets jets =
  applyProjection<JetAlg>(evt, "Jets").jetsByPt(20*GeV);
foreach (const Jet& j, jets) {
  foreach (const Particle& p, j.particles()) {
    const double dr =
      deltaR(j.momentum(), p.momentum());
  }
}
```

Check out the **`Rivet/Math/MathUtils.hh`** header for more handy functions like **`deltaR`**.

Jets (3)

For substructure analysis Rivet doesn't provide extra tools: best just to use FastJet directly

```
const PseudoJets psjets = fj.pseudoJets();
const ClusterSequence* cseq = fj.clusterSeq();

Selector sel_3hardest = SelectorNHardest(3);
Filter filter(0.3, sel_3hardest);
foreach (const PseudoJet& pjet, psjets) {
  PseudoJet fjet = filter(pjet);
  ...
}
```

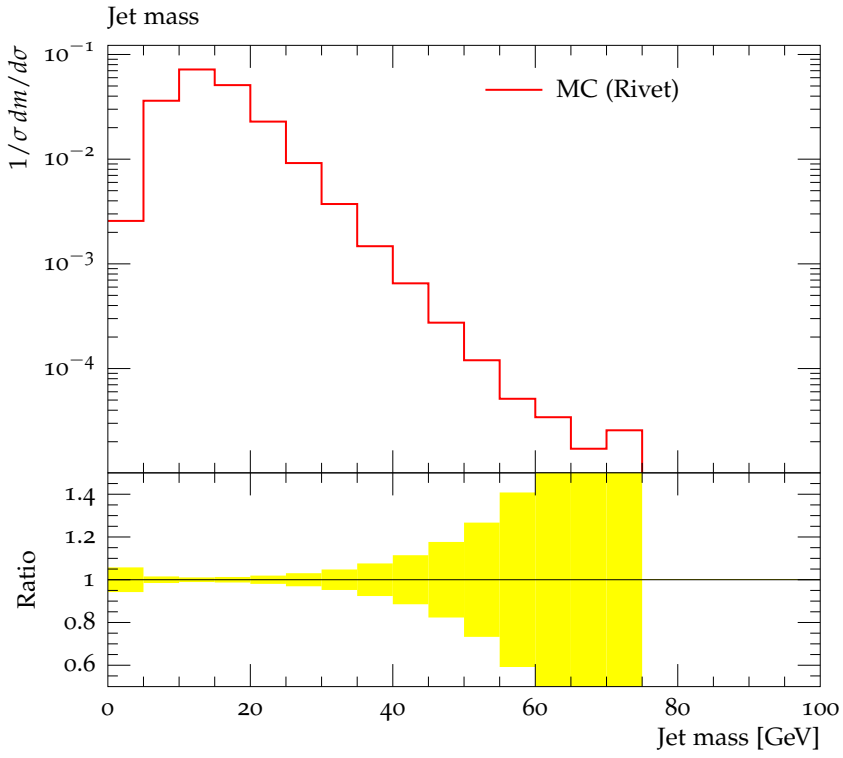
This is an historical design! Rivet 2.1 will include a major overhaul of jets, etc. for much better Fastjet/jet structure integration.

A jetty analysis

I'll walk through making and running a very simple jet + substructure analysis – but no physics input or complexity here!

The ideal result will be found on AFS at

`~abuckley/public/rivet-tutorial/`



Writing a *data* analysis

Starting a data analysis

We'll use the ATLAS 2010 W+jets analysis as an example. Feel free to implement something else: we'll try to troubleshoot.

The SPIRES key for this ATLAS analysis is 8919674 (try "key 8919674" in the SPIRES search box) and it was published in 2010, so in the standard Rivet naming convention it is called **`ATLAS_2010_S8919674`**.

This is a bit outdated: sorry! We prefer Ixxxxxx now, using the *Inspire* key.

There is reference data for this analysis in HepData: running **`rivet --show-analysis ATLAS_2010_S8919674`** supplies this URL: **http://hepdata.cedar.ac.uk/view/irn8919674**

`rivet-mkanalysis ATLAS_2010_S8919674` will download this ref data. NB. the jet multiplicity plots are not output correctly: HepData needs some improvements! Check the **`.info`** and **`.yoda`** files: use **`yoda2flat ATLAS_2010_S8919674.yoda | less`**

The histogram names in this data file can be used for *autobooking*.

Histogram autobooking

The final framework feature to introduce is histogram autobooking. This is a means for getting your Rivet histograms binned with the same bin edges as used in the experimental data that you'll be comparing to.

To use autobooking, just call the booking helper function with only the histogram name (check that this matches the name in the reference `.yoda` file), e.g.

```
_hist1 = bookHisto1D("d01-x01-y01")
```

The "d", "x" and "y" terms are the indices of the HepData dataset, x-axis, and y-axis for this histogram in this paper.

A neater form of the helper function is available and should be used for histogram names in this format:

```
_hist1 = bookHisto1D(1, 1, 1)
```

That's it! If you need to get the binnings without booking a persistent histogram use `refData(name)` or `refData(d,x,y)`.

NB. Extra bool argument for using ref data x vals for `Scatter2Ds`

UnstableFinalState

The **`UnstableFinalState`** projection fetches decayed-but-physical particles (mostly hadrons) from the event record. The HepMC standard declares how these are to be indicated, so the results are reliable and physically safe:

```
const UnstableFinalState ufs(2.5, 6.0);
addProjection(ufs, "UFS");
...
const FinalState& ufs =
  applyProjection<FinalState>(evt, "UFS");
foreach (const Particle& p, j.particles()) {
  const int pid = p.pdgId();
  if (PID::hasBottom(pid)) num_b += 1;
  ...
}
```

HepPDT-type functions are defined in the **`PID`** namespace in the **`Rivet/Tools/ParticleIdUtils.hh`**.

The new **`PrimaryHadrons`** and **`HeavyHadrons`** projections are perhaps better: no duplication. Useful for tagging, etc.

The End