RIVET
Robust Independent Validation of Experiment and Theory
script

André Vieira da Silva

andsilva@ifi.unicamp.br

November 4, 2017

Abstract

The Rivet toolkit is a system for validation of Monte Carlo event generators.

Contents

1 Introduction

RIVET [1]

- The RIVET toolkit (Robust Independent Validation of Experiment and Theory) is a system for validation of Monte Carlo event generators. It provides a large (and ever growing) set of experimental analyses useful for MC generator development, validation, and tuning, as well as a convenient infrastructure for adding your own analyses.
- Rivet is the most widespread way by which analysis code from the LHC and other high-energy collider experiments is preserved for comparison to and development of future theory models. It is used by phenomenologists, MC generator developers, and experimentalists on the LHC and other facilities.

2 Welcome to Rivet!

Rivet is a framework for analysing simulated collider events in HepMC format. As well as providing the infrastructure for such analyses, it provides a set of efficient observable calculators and a large collection of standard analyses from a variety of colliders and experiments. Rivet is designed to be an efficient and simple-to-use system for generator validation, tuning, and regression testing.

2.1 Getting started with Rivet

```
Installation of Rivet

Prerequisite Python header files are required. On Ubuntu,
you can use this command to install the necessary files system-wide:


Terminal Linux:

sudo apt-get install python-dev
sudo apt-get update && sudo apt-get upgrade

Download the bootstrap script into a temporary working directory,
and make it executable:

cd /scratch/rivet
wget http://rivet.hepforge.org/hg/bootstrap/raw-file/2.5.4/rivet-bootstrap
chmod +x rivet-bootstrap

Run the script. By default it will install to $PWD/local, where $PWD
is the current directory.  If you need to change that, specify the
```

```
corresponding values on the command line. Examples:

./rivet-bootstrap


Setting up the environment

After the script grinds away for a while, it will tell you that
it is finished and how to set up a runtime environment (similar
to that used inside the build script) for running Rivet.
A sourceable rivetenv.(c)sh script is provided for (c)sh shell
users to help set up this environment.

Here is how to set up the environment and then test the rivet
program is help feature and analysis listing:

source $PREFIX/rivetenv.sh
rivet --help
rivet --list-analyses

You may wish to add the environment variable settings to your
~/.bashrc shell config file, so that Rivet will work without
needing any special session setup.
```

If that works, everything is installed correctly. If you are using the bash shell in your terminal, then Rivet will offer you programmable tab completion: try typing rivet and pressing the Tab key!

3 Example run

We provide some hepmc events (input for rivet) for download:

http://www.hepforge.org/archive/rivet/Z-hadronic-LEP.hepmc

```
mkdir testRivet && cd testRivet
wget http://www.hepforge.org/archive/rivet/Z-hadronic-LEP.hepmc
```

To run a few analyses on these 10k LEP events, you can do something like this:

```
rivet Z-hadronic-LEP.hepmc -a SLD_2002_S4869273 -a OPAL_1998_S3780481
```

To plot the resulting output file Rivet.yoda the following command suffices:

```
rivet-mkhtml --mc-errs Rivet.yoda:MC Simulation
```

A new folder rivet-plots will have appeared in your current directory on the host system ready for inspection with your browser, e.g.

4 Writing the analysis code

You could make your analysis by copying some example code and then going through a load of search and replace fiddling, but in fact there is a much easier way: the **rivet-mkanalysis script**. This script is installed along with the rest of the Rivet system, and will generate template analysis files for you.

All analysis routines are implemented as sub-classes of the Rivet "Analysis" class: pretty much all the magic that binds the analysis object into the Rivet system is handled in this base class, meaning that your code can really concentrate on implementing the physics goals of the analysis.

Running the rivet-mkanalysis script with the appropriate analysis name will have generated a .cc C++ source file template, and template metadata files for information about the analysis (.info) and specifications of titles, axis labels, etc. for the plots which the analysis will produce (.plot). These templates will include, if possible, extra analysis metadata such as a BibTeX publication entry in the .info file.

4.1 Structure

For simplicity, Rivet analysis classes are usually written in just one .cc file, i.e. no header declaration. This is because classes are almost always not inherited from, and all that the Rivet system needs to know is that it can be treated as an Analysis* pointer: avoiding header files makes everything more compact and removes a source of errors and annoyance.

An analysis has the following components:

- a no-argument constructor;
- three analysis event loop methods: **init**, **analyze** and **finalize**;
- a minimal hook into the plugin system

It is also possible to add some metadata methods which describe the analysis, references to publications, experiment, etc., but we strongly recommend that you put this information into the "YAML" format .info template that the rivet-mkanalysis script generated for you instead: this way the code will remain clean and minimal, and you can update the metadata without needing to recompile. All analyses bundled with Rivet store their metadata in external files.

Useful analyses also contain member variables for the analysis: event weight counters and histograms are the most common of these. Conventionally, we declare the class member variables with a leading underscore: see the CodingStyleGuide for more information on our recommended uniform coding style. Histogram pointer members (for which we use special smart pointers with clever machinery inside) are preferred to start with an "h", e.g. Histo1DPtr hpT.

4.2 Implementation

The constructor and three event loop methods are used for the following:

- Constructor: set whether the generator cross-section is needed. Minimal!
- **init**: book histograms, initialise counters, etc.
- **analyze** : select particles, filter according to cuts, loop over combinations, construct observables, fill histograms. This is where the per-event aspect of the analysis algorithm goes.
- **finalize**: normalize/scale/divide histograms, tuples, etc.

This probably looks similar to every analysis system you've ever used, so hopefully you're not worried about Rivet being weird or difficult to learn ;-)

Rivet provides implementations of many calculational tools, called "projections". These are just observable calculator objects with a silly name, so don't get worried. (They automatically cache their results, to make Rivet automatically efficient, but you don't have to worry about that since it's, well, automatic.) The projections are used by calling the analysis' applyProjection(event) method. This will return a const reference to the completed projection object and takes the type of the reference as a template argument, e.g.

4.3 Projections

Projections are arguably the most important objects in Rivet - they're certainly the part that does most of the work. A projection is an object which calculates a property or properties from an Event. Such properties can be a single number (e.g. Q2), a single complex object (e.g. sphericity tensor) or a list of such objects (e.g. a set of boosted jets).

4.3.1 Specific projections

Here are some examples of projections available in the current release of Rivet:

- Beam - obtain the beam from an event
- FinalState - get the final state particles
- FinalStateHCM - get the final state particles, shifted to the HCM frame
- VetoedFinalState - get the final state particles, minus selected particle types
- FastJets? - jet algorithms accessed via FastJet?
- MissingMomentum? - what it says on the tin

- Multiplicity - count the FS particles of various types
- Sphericity - the sphericity tensor, from which the sphericity, oblateness and planarity can be obtained
- ParisiTensor? - linearized quadratic momentum tensor, from which the Parisi C and D variables can be obtained
- Thrust - the thrust tensor, defining the thrust axes and scalars
- DISKinematics - kinematics of DIS events
- DISLepton - obtain the scattered lepton in DIS events

rivet-tutorial-253

- Andy Buckley
- https://rivet.hepforge.org/rivet-tutorial-253.pdf

4.4 Writing, building running your own analysis

```
Terminal Linux:

rivet-mkanalysis MY_TEST_ANALYSIS

rivet-buildplugin RivetMY_TEST_ANALYSIS.so MY_TEST_ANALYSIS.cc

this plugin is to specify the directory standard analyses RIVET?

$ export RIVET_ANALYSIS_PATH=.../RIVET (location of the files .cc )
$ rivet --list-analyses MY_TEST_ANALYSIS

MY_TEST_ANALYSIS [UNVALIDATED] <Insert short MY_TEST_ANALYSIS description>

$rivet pythia.hepmc -a MY_TEST_ANALYSIS
Rivet 2.5.4 running on machine andre-QBEX-H61H2-M17 (x86_64)
```

4.5 rivet-mkanalysis script

4.5.1 File MY TEST ANALYSIS.cc:

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

namespace Rivet {


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

  /// Constructor
  DEFAULT_RIVET_ANALYSIS_CTOR(MY_TEST_ANALYSIS);


  /// @name Analysis methods
  //@{

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

// Initialise and register projections
//declare(FinalState(Cuts::abseta <5 && Cuts::pT >100*MeV),"FS");
const ChargedFinalState cfs(-2.0, 2.0);
addProjection(cfs, "CFS");

  const FinalState fs_jet(-7., 7.);
  addProjection(fs_jet, "FS_JET");
  FastJets fj02(fs_jet, FastJets::ANTIKT, .4);
  addProjection(fj02, "Jets02");

  // Book histograms
  h_pt_jet02 = bookHisto1D("pt_jet", 20, -2, 2);
  //_h_XXXX = bookProfile1D(1, 1, 1);
  // ( Histogram for the data experimental
  //.yoda see https://hepdata.net/ )
  //_h_YYYY = bookHisto1D(2, 1, 1);
  //_h_ZZZZ = bookCounter(3, 1, 1);

    }
```

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

    const double weight = event.weight();

    /// @todo Do the event by event analysis here
    Jets alljets = applyProjection<FastJets>(event, "Jets02").
    jetsByPt(Cuts::pT > 5*GeV && Cuts::abseta < 2);
    foreach (const Jet &jet, alljets)
    _h_pt_jet02->fill(jet.momentum().eta(), weight);

    }


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


   const double fac =
   crossSection()/(millibarn*sumOfWeights()*2*pi);
   scale(_h_pt_jet02, fac);
   //normalize(_h_YYYY); // normalize to unity
   // norm to cross section
   //scale(_h_ZZZZ, crossSection()/picobarn/sumOfWeights());

   }

   //@}


   /// @name Histograms
   //@{
   Histo1DPtr _h_pt_jet02;
   //Histo1DPtr _h_YYYY;
   //CounterPtr _h_ZZZZ;
   //@}

  };

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


}
```

```
rivet-mkhtml --mc-errs Rivet.yoda:"PYTHIA8"
```

make-plots in LaTeX

Manual for Package pgfplots

http://rivet.hepforge.org/make-plots.html

4.5.2 File MY TEST ANALYSIS.plot:

```
# BEGIN PLOT /MY TEST ANALYSIS/pt_jet
Title= Pseudorapidity
XLabel= $\eta_{jet}$
YLabel= dN/d$\eta$
# + any additional plot settings you might like,
see make-plots documentation
# END PLOT

# ... add more histograms as you need them ...
```

4.5.3 File MY TEST ANALYSIS.info:

```
Name: PYTHIA8_2017_JETS
Year: <Insert year of publication>
Summary: <Insert short PYTHIA8_2017_JETS description>
Experiment: <Insert the experiment name>
Collider: <Insert the collider name>
InspireID: <Insert the Inspire ID>
Status: UNVALIDATED
Authors:
 - Your Name <your@email.address>
#References:
#- '<Example: Eur.Phys.J. C76 (2016) no.7, 392>'
#- '<Example: DOI:10.1140/epjc/s10052-016-4184-8>'
#- '<Example: arXiv:1605.03814>'
RunInfo: <Describe event types, cuts, and other general
generator config tips.>
NeedCrossSection: no
#Beams: <Insert beam pair(s), e.g. [p+, p+] or [[p-, e-], [p-, e+]]>
#Energies: <Run energies or beam energy pairs in GeV, e.g. [13000] or
[[8.0, 3.5]] or [630, 1800]. Order pairs to match "Beams">
Description:
  A fairly long description, including what is measured
  and if possible what it is useful for in terms of MC validation
  and tuning. Use LaTeX for maths like $\pT > 50\;\GeV$.
BibKey:
BibTeX: ''
ToDo: - Implement the analysis, test it, remove this ToDo, and mark
 as VALIDATED :-)
```

5 The Rivet MC analysis system 2.5.4

5.1 Rivet Namespace Reference

Rivet is principally a C++ framework for Monte Carlo event generator validation. It is built on the dual concepts of analyses - routines which implement published experiment analyses - and projections - routines which extract observables from event records. Rivet is designed to be efficient, via automatic caching of projection results for each event.

For more information on Rivet, please see the wiki documentation, managed at http://projects.hepforge.org/rivet/trac/wiki .

6 HEPDATA - High Energy Physics Data Repository/ Data from the LHC

The Durham High Energy Physics Database (HEPData) has been built up over the past four decades as a unique open-access repository for scattering data from experimental particle physics papers. It comprises data points underlying several thousand publications. Over the last two years, the HEPData software has been completely rewritten using modern computing technologies as an overlay on the Invenio v3 digital library framework.

https://arxiv.org/pdf/1704.05473.pdf

References

[1] Andy Buckley et al. "Rivet user manual". In: *Comput. Phys. Commun.* 184 (2013), pp. 2803–2819. DOI: 10.1016/j.cpc.2013.05.021. arXiv: 1003.0694 [hep-ph].