Collider analysis recasting with Rivet & Contur

Andy Buckley, University of Glasgow
Jon Butterworth, University College London

MC4BSM, 20 April 2018

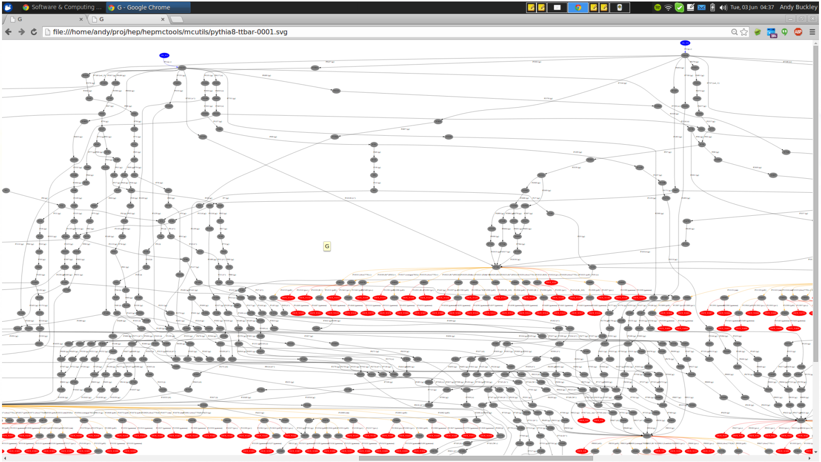
Rivet

Rivet is an analysis system for MC events, and *lots* of analyses

- Easy and powerful way to get physics numbers & plots from *any* MC gen
- LHC standard for preserving data analyses: standard in ATLAS & CMS SM
- Origins in SM, and particularly QCD for MCs – extended for search preservation since v2.5 by adding detector transfer-function features
- C++ library with Python interface, analyses are plugins, code is "clean"

Generator independence

A Pythia8 $t\bar{t}$ event visualised from HepMC output:



Most of this is not standardised: Herwig and Sherpa look *very* different.
But final states and decay chains have to have equivalent meaning.

Rivet setup

Docker

VM-like pre-prepared environments: avoid platform issues, integrates well with host. Instructions at https://rivet.hepforge.org/trac/wiki/Docker

```
docker pull agbuckley/rivet-tutorial
docker run -it agbuckley/rivet-tutorial
```

Local install

Easy to install using our *bootstrap script*:

```
wget http://rivet.hepforge.org/hg/bootstrap/raw-file/2.6.0/rivet-bootstrap
bash rivet-bootstrap
```

Needs valid compiler (C++11), etc. environment

Run from LCG

```
ssh lxplus7.cern.ch
. /cvmfs/sft.cern.ch/lcg/releases/LCG_87/gcc/6.2.0/x86_64-centos7/setup.sh
. /cvmfs/sft.cern.ch/lcg/releases/LCG_87/MCGenerators/rivet/2.5.4/...
   x86_64-centos7-gcc62-opt/rivetenv.sh
```

First Rivet runs

Command-line interface

`rivet` and other command line tools to query and run routines

- List available analyses:
  `rivet --list-analyses`

- List ATLAS analyses:
  `rivet --list-analyses "ATLAS|CMS"`

- Show some pure-MC analyses' full details:
  `rivet --show-analysis MC_`

Same metadata and API docs online at http://rivet.hepforge.org

All Rivet commands start with `rivet-`, so tab-complete lists them all

Running existing analyses

To avoid huge files, we get the events from generator to Rivet by writing HepMC (from Py8) to a filesystem pipe

```
$ mkfifo fifo.hepmc

$ run-pythia -n 200000 -e 8000 -c Top:all=on -o fifo.hepmc &

$ rivet fifo.hepmc -a MC_TTBAR,MC_JETS,MC_GENERIC
    -a ATLAS_2015_I1404878,CMS_2016_I1473674

$ rivet-mkhtml Rivet.yoda:'Pythia 8 $t\bar{t}$'
```

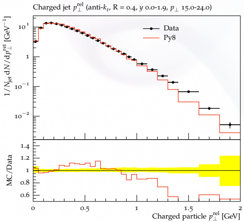
By default *unfinalised* histos are written every 1000 events: monitor progress through the run. Killing with `Ctrl-C` is safe: finalizing is run

Plotting

"YODA" stats library — http://yoda.hepforge.org
Bin-width handling, bin gaps, object ownership, thread-safety $\Rightarrow$ non-ROOT histogramming

- Separation of stats from presentation: plotting via **`make-plots`** script
- Text-based data format with all second-order stat moments: full stat merging up to all means and variances
- YAML metadata and zipped read/write from v1.7.0
- Being gradually extended to handle more complex physics data types

CLI tools: **`yodals`**, **`yodadiff`**, **`yodamerge`**, **`yodascale`**, **`yoda2root`**, etc.

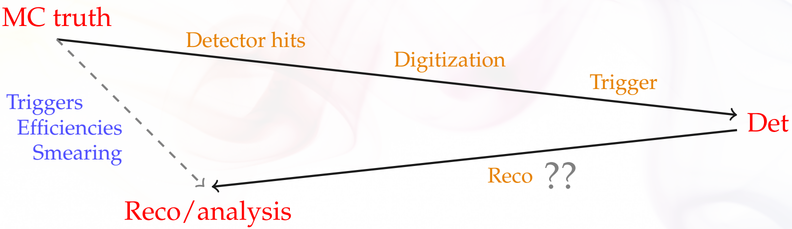


BSM searches and detector effects

Detector effects

Normal in SM, top, etc. measurements to *unfold* detector effects.
Usually "uneconomic" to do that for BSM searches

Explicit fast detector simulation vs. smearing/efficiencies:



- **(Private) reco algorithms already reverse most detector effects**
- Reco calibration to MC truth, so kinematics *usually* subleading
- Efficiency & mis-ID effs dominate – tabulated in all fast-sims
- ⇒ flexible parametrisation: effs change with analysis phase-space, experiment reco-code version, collider run, …

and *need to guarantee stability for preservation*

Using Rivet's fast-sim tools

Smearing is provided as "wrapper projections" on normal particle, jet, and MET finders.

Smearing configuration via efficiency/modifier functions.

To use, first `#include "Rivet/Projections/Smearing.hh"`

Examples:

```
FinalState es1(Cuts::abseta < 5 && Cuts::abspid == PID::ELECTRON);
SmearedParticles es2(es, ELECTRON_EFF_ATLAS_RUN2, ELECTRON_SMEAR_ATLAS_RUN2);
declare(recoes, "Electrons");

FastJets js1(FastJets::ANTIKT, 0.6, JetAlg::DECAY_MUONS);
SmearedJets js2(fj, JET_SMEAR_ATLAS_RUN2, JET_EFF_BTAG_ATLAS_RUN2);
declare(recoj, "Jets");

...

Particles elecs = apply<ParticleFinder>(event, "Electrons").particles(10*GeV);
Jets jets = apply<JetAlg>(event, "Jets").jetsByPt(30*GeV);
```

Standard global functions here, but private fns or inline lambdas better when possible

Selection tools for search analyses

Search analyses typically do a lot more "object filtering" than measurements. Lots of tools to express complex logic neatly:

- Filtering functions: `filter_select(const Particles/Jets&, FN)`, `filter_discard(...)` + `ifilter_*` in-place variants
- *Functors* for common "stateful" filtering criteria: `PtGtr(10*GeV)`, `EtaLess(5)`, `AbsEtaGtr(2.5)`, `DeltaRGtr(mom, 0.4)`, `ParticleEffFilter(FN), ...`
  - Lots of these in `Rivet/Tools/ParticleBaseUtils.hh`, `Rivet/Tools/ParticleUtils.hh`, and `Rivet/Tools/JetUtils.hh`
- `any()`, `all()`, `none()`, etc. – accepting functions/functors
- Cut-flow monitor via `#include "Rivet/Tools/Cutflow.hh"`

BSM hands-on

Look at the source code in `TESTDET.cc`: does it make sense?

- Build & run like:

```
$ rivet-buildplugin TESTDET.cc
$ run-pythia -n 200000 -e 13000 -o fifo.hepmc -c SUSY:all=on
-c SLHA:file=gg_g1500_chi100_g-ttchi.slha &
$ rivet --pwd -a TESTDET -H bsm.yoda fifo.hepmc -lAnalysis=DEBUG
```

- Split and compare the particle- and reco-level observables:

```
$ bash truerecosplit.sh bsm.yoda
$ rivet-mkhtml bsm-*.yoda -m '/TESTDET'
```

- Try adding a constant 70% *b*-tag efficiency to the jets:
  `JET_BTAG_EFFS(0.7)` or
  `(const Jet& j) return j.bTagged() ? 0.7 : 0.0; .`

- Try the same with `CMS_2017_I1594909.cc`; browse the file with `yodals -v` to see the the CMS signal-region counts for recasting

Contur: limit-setting using Rivet analyses

Contur is a layer on top of Rivet to do statistical interpretation of injected BSM signal to “Standard Model” phase spaces.

- **Idea:** make use of the full set of Rivet analyses to constrain new physics models.

  Modelling inclusivity also important: a strength of Herwig 7

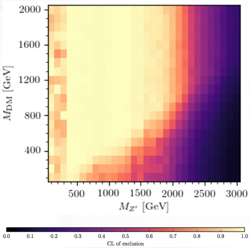
- **Benefits:** model-agnostic and very quick. Can study many possible signatures at the same time

- **Current constraints:** SM modelling is hard, so assume data = SM!

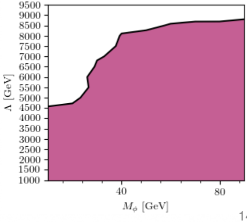
  Single-bin limits within manual analysis groupings in lieu of full correlations.

  Working to include SM predictions and uncertainties

Simplified vector+DM model



Eff-coupling light scalars



Contur hands-on exercise

Finally, let's try running Contur ourselves.

We're going to use a separate Docker image, which adds Herwig 7 & Contur:

```
docker pull agbuckley/contur-tutorial
docker run -it agbuckley/contur-tutorial
```

In docker, do:

```
source setupContur.sh
make
```

Then you can try:

```
contur -s -t CS mY_1500_mX_1000-weak.yoda
```

and

```
contur-mkhtml mY_1500_mX_1000-weak.yoda
```

Look at the plots in the `contur-plots` directory (use `docker cp`)

Other YODA files available, or try generating your own with Herwig/Rivet

That’s all, folks

Summary

- **Rivet is a user-friendly MC analysis system for prototyping and preserving data analyses**
- Allows theorists to use analyses for model development & testing, and BSM recasting
- Also a very useful cross-check: quite a few analysis bugs have been found via Rivet!
- Supports detector simulation for BSM search preservation: more features coming soon: systematics weights, more BSM features …
- Contributions and team membership all very welcome. Twice-annual Rivet hackfests: BSMers welcome!

- **Contur is a BSM limit-setting framework built on Rivet**
- Can operate on any source of YODA files: use the comprehensiveness of Rivet's SM analyses to constrain BSM. Much extension underway…roll up, test your model against the SM!